

DSP System Toolbox™ Release Notes



MATLAB® & SIMULINK®



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

DSP System Toolbox™ Release Notes

© COPYRIGHT 2012–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Spectrum Analyzer with better responsiveness and toolstrip interface for analysis, estimation, and measurement parameters	1-2
Analyzer Tab	1-2
Estimation Tab	1-2
Measurements Tab	1-3
Spectrum Tab	1-3
Spectrogram Tab	1-3
Spectral Mask Tab	1-3
Channel Measurements Tab	1-4
Fourth-Order Section Filter Block: Implement a cascade of fourth-order section filters in Simulink	1-4
New outputDelay Function: Determine delay in multirate filter System objects	1-4
Enhancements to LMS Update block: Multifilter and adaptive linear combiner support	1-6
New CCDF measurement mode in powermeter System object	1-7
Kaiser-window-based design method for FIR halfband decimator and interpolator	1-7
Certain IIR frequency transformation functions now support higher-order section filters	1-7
Farrow Rate Converter block supports variable-size input signal and arbitrary input frame size	1-8
iirnotch, iirpeak, iircomb, and designMultirateFIR functions now support nonconstant inputs during code generation	1-8
Improved speed performance in certain DSP System Toolbox features	1-8
Improved filter response visualization in certain DSP System Toolbox blocks	1-10
Dataflow simulation analysis using the Multicore tab	1-10
Buffer and Unbuffer blocks now support code generation for Simulink Real-Time	1-11

Nonrecursive code generation in DCT and IDCT blocks	1-11
Time Scope MATLAB object and Array Plot now support additional signal statistics	1-11
Configure timescope measurements programmatically	1-11
Configure dsp.ArrayPlot measurements programmatically	1-12
Array Plot block supports multiple frame sizes	1-12
To Workspace block inside a Dataflow Subsystem block supports multithreaded execution	1-13
Functionality being removed	1-13
CCDF measurements will be removed from the dsp.SpectrumAnalyzer object	1-13
'Raised Cosine' and 'Square Root Raised Cosine' response methods will be removed	1-14
DirectFeedthrough property has been removed from dsp.VariableFractionalDelay System object	1-14
dsp.AllpassFilter System object no longer supports cell array coefficients	1-14
Functionality removed in dsp.Delay System object	1-14
Access to Time Scope through Handle Graphics API will be removed ...	1-14
SampleInput parameter will be removed	1-14
Functions being removed	1-14
iirparameq will be removed	1-14
midi functions require Audio Toolbox	1-15
Objects being removed	1-15
dsp.SpectrumAnalyzer object will be removed	1-15
Support for dsp.SpectrumAnalyzer object will be removed from certain object functions	1-15
Certain fdesign objects will be removed	1-16
Certain mfil objects have been removed	1-16
Certain System objects have been removed	1-16
HDL Optimized System objects moved to DSP HDL Toolbox	1-17
Blocks being removed	1-20
Certain blocks moved to Simulink	1-20
HDL Optimized blocks moved to DSP HDL Toolbox	1-20

R2021b

New default filter design for FIR rate conversion objects	2-2
New automatic design for FIR rate conversion objects	2-2

Improved speed performance for certain DSP System Toolbox features	2-4
Specify overlap length in moving statistics objects	2-5
Arbitrary frame size support for dsp.FarrowRateConverter	2-6
SIMD Code Generation: New code replacement library (CRL) customized for DSP System Toolbox features	2-6
Complex support for SOS filter coefficients	2-6
Tunable Filename property for dsp.AudioFileWriter and dsp.AudioFileReader objects in generated code	2-6
POSIX threads (Pthreads) and OpenMP threading support for multicore custom targets using dataflow domain	2-7
Enhancements for Multicore tab analysis results	2-7
dspunfold does not support Xcode 12.0 or later	2-7
Spectrum Analyzer CCDF measurement reference line changed	2-7
dsp.ArrayPlot supports multiple frame sizes	2-7
Functionality being removed	2-7
'Raised Cosine' and 'Square Root Raised Cosine' response methods will be removed	2-7
Objects being removed	2-8
Certain System objects will be removed	2-8
dsp.PeakFinder object has been removed	2-9
Blocks being removed	2-9

R2021a

Multicore tab for Dataflow: Analyze and configure multicore execution for Simulink models using Dataflow	3-2
Rationally oversampled channelizers	3-2
In-Place Memory Optimization: Optimize the memory usage in the generated code for certain DSP System Toolbox features	3-2
Fractional delay FIR filter design	3-2
Power Meter: Measure power of voltage signal in MATLAB and Simulink	3-3

SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox features	3-3
Improved filter response visualization for certain DSP System Toolbox blocks	3-8
Improved Speed Performance in Accelerator Mode for specific blocks in DSP System Toolbox	3-8
Improved display for Array Plot block	3-8
Variable-sized input support for timescope object	3-10
One-Based index support for Peak Finder block	3-10
Removal of the oversampling ratio functionality	3-10
Digital down-converter (DDC) and digital up-converter (DUC) examples for FPGA (requires HDL Coder license for code generation)	3-10
Objects being removed	3-11
Certain System objects will be removed	3-11
Certain System objects have been removed	3-12
dsp.TimeScope will be removed	3-13
Blocks being removed	3-13
Certain blocks will be removed	3-13
Blocks that have been removed	3-13

R2020b

New time scope object: Visualize signals in the time domain	4-2
Scopes Tab	4-2
Measurements Tab	4-2
SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox features	4-3
One-sided short-time Fourier transform in dsp.STFT and dsp.ISTFT objects	4-4
In-place Memory Optimization: Optimize the memory usage in the generated code for certain DSP System Toolbox blocks	4-5
Improved Speed Performance in Accelerator Mode for specific blocks in DSP System Toolbox	4-7
Visualize logged Stateflow states in the Logic Analyzer	4-7

HDL-optimized FIR Decimation block and System object: Downsample signals using a FIR decimation filter with a hardware-friendly interface and architecture (requires HDL Coder for code generation)	4-7
Gigasample-per-second (GSPS) CIC Decimation and CORDIC Algorithm: Increase throughput of HDL-optimized CIC decimation and complex-to-magnitude-angle conversion by using frame-based input (requires HDL Coder for code generation)	4-7
Dataflow domain analysis integrated with Performance Advisor	4-8
MATLAB Compiler support for dsp.ArrayPlot	4-8
Functionality being removed or changed	4-8
dsp.TimeScope will be removed	4-8
Spectrum Analyzer block defaults changed	4-8
HDL Minimum Resource FFT and HDL Streaming FFT blocks have been removed	4-9
Matrix Viewer and Waterfall blocks will be removed	4-9

R2020a

SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox blocks	5-2
FIR Interpolation and FIR Decimation blocks	5-2
LMS Filter block	5-2
Automatically leverage SIMD for multicore dataflow simulations	5-2
New Biquadratic SOS Filter Object	5-2
Multirate processing in FIR Rate Conversion block	5-2
Non-Maximally Decimated Channelizers	5-3
Complex Support for Channelizer and Channel Synthesizer Prototype Coefficients	5-3
Enhancements to designMultirateFIR function	5-3
UDP Sender supports large message sizes	5-3
Variable CIC Decimation Factor: Specify decimation factor as an input to the CIC Decimation HDL Optimized block (requires HDL Coder for code generation)	5-3
Gigasample-per-second (GSPS) NCO: Generate frame-based output from HDL-optimized NCO for high speed applications (requires HDL Coder for code generation)	5-4

Suggestions for optimal model settings in Dataflow Simulation Assistant	5-4
Dataflow subsystems supported in model reference simulation targets	5-4
Functionality being removed or changed	5-4
Removal of DirectFeedthrough property in dsp.VariableIntegerDelay System object	5-4
dsp.AudioPlayer and dsp.AudioRecorder objects removed	5-4
HDL-optimized NCO requires valid input port	5-5
HDL-optimized NCO with floating-point inputs applies phase quantization	5-5
NCO HDL Optimized block now ignores LUTRegisterResetType parameter	5-6
Signal data no longer streams to the Logic Analyzer when signal logging is disabled	5-6

R2019b

SIMD code from Discrete FIR Filter Block: Generate optimized code using Intel AVX2 for FIR Filters in Simulink	6-2
HDL-optimized CIC Decimation block and System object: Downsample signals using a cascade integrator-comb (CIC) filter (requires HDL Coder for code generation)	6-2
Discrete FIR Filter HDL Optimized block: Filter using complex coefficient values (requires HDL Coder for code generation)	6-2
Improved display for dsp.DynamicFilterVisualizer	6-2
Improved display for dsp.ArrayPlot	6-3
dsp.MatrixViewer support for multiple cursor measurements	6-4
Playback control behavior changed for scopes in referenced models ...	6-5
Output of colored noise generator can be bounded	6-5
Blocks with finite states supported for unfolding in Dataflow subsystems	6-5
Simulate Dataflow subsystems using multiple threads in Rapid Accelerator mode	6-5
Virtual bus support at Dataflow subsystem boundaries for heterogeneous signals	6-5
Functionality being removed or changed	6-6
Certain System objects will be removed	6-6

Direct and Inverse Short-Time Fourier Transform: Analyze and process streaming signals in the frequency domain and synthesize them with perfect reconstruction using overlap and add	7-2
Fourth-Order Section Filter: Model and simulate cascaded fourth-order section IIR filters in MATLAB	7-2
Spectrum Analyzer improvements for exponential averaging, mixed-complexity inputs, and MATLAB script generation	7-2
Smooth data with exponential averaging	7-2
Display block inputs with different complexity	7-2
Generate MATLAB script from dsp.SpectrumAnalyzer	7-2
Exponential Spectrum Averaging: Smooth spectrum estimation and analysis efficiently over time using exponential averaging	7-3
Complex Data over UDP: Send and receive complex data directly over UDP in MATLAB and Simulink	7-3
Stream signals only from a defined interval within audio files when using the From Multimedia File block	7-4
New targets supported for multicore code generation from a dataflow subsystem	7-4
Blocks with constant sample times supported in dataflow subsystems ..	7-4
Improve simulation performance of dataflow subsystems using the Dataflow Simulation Assistant	7-4
Identify scopes unsupported for multithreading in dataflow subsystems at edit-time	7-5
Use the Timing Legend to highlight blocks in a dataflow domain	7-6
Discrete FIR Filter HDL Optimized block: Use programmable coefficients with a fully parallel systolic architecture (requires HDL Coder for code generation)	7-7
Discrete FIR Filter HDL Optimized block: Optimize symmetric and antisymmetric coefficients and optional reset port for a partly serial systolic architecture (requires HDL Coder for code generation)	7-7
HDL code generation support for programmable coefficients with frame-based Discrete FIR Filter block (requires HDL Coder for code generation)	7-7
dsp.MatrixViewer System object	7-7
DSP System Toolbox Support Packages for ARM Cortex -A and ARM Cortex -M Processors will be removed	7-8

Functionality being removed or changed	7-8
Certain System objects will be removed	7-8
Parametric EQ Filter block has been removed	7-8
Changes to Discrete FIR Filter HDL Optimized serial filter parameters ...	7-9

R2018b

Dataflow: Accelerate your model using multi-threading and derive frame sizes automatically for multirate signal processing in Simulink	8-2
Programmatic Interface for Spectrum Analyzer Measurements: Configure measurements programmatically and obtain numerical results for further processing or analysis	8-2
Dynamic Filter Visualization: Visualize the magnitude response of time-varying digital filters	8-2
Optimized Multistage Multirate Filters: Design multistage decimation and interpolation FIR filters based on requirements for response and implementation cost	8-3
Sample Range for Audio File Reader: Stream signals only from a defined interval within audio files when using the dsp.AudioFileReader System object	8-3
Faster Channelizer and Channel Synthesizer: Simulate polyphase FFT filters faster by leveraging additional parallel optimizations	8-3
Peek Functionality in dsp.AsyncBuffer System object	8-3
dsp.AudioFileReader System object supports http streams	8-4
Improved Logic Analyzer performance for multichannel signals	8-4
HDL code generation support for complex input signals or complex coefficients of frame-based Discrete FIR Filter and FIR Decimation blocks (requires HDL Coder for code generation)	8-4
Discrete FIR Filter HDL Optimized: Select transposed architecture, optimize symmetric and antisymmetric coefficients, and enable reset port (requires HDL Coder for code generation)	8-4
Functionality being removed or changed	8-5
Vector Scope block has been removed	8-5
Certain linear prediction System objects will be removed	8-5
Cell array support removed for dsp.AllpassFilter coefficients	8-6

Frequency Input Mode for Spectrum Analyzer: Display, measure, and analyze frequency-domain signals in MATLAB and Simulink	9-2
Efficiency-Optimized Digital Filters: Simulate select digital filters faster in MATLAB and Simulink by leveraging additional parallel optimizations	9-2
Complex Bandpass Decimation: Extract a frequency subband using a one-sided (complex) bandpass decimator in MATLAB and Simulink	9-2
Frequency-Domain Adaptive Filter Block: Simulate adaptive FIR filters requiring a large number of taps	9-2
Bit-Natural HDL-Optimized FFT: Return data in bit-natural order from frame-based FFT/IFFT (Requires an HDL Coder license for code generation)	9-2
New partitioned modes in dsp.FrequencyDomainAdaptiveFilter System object	9-3
Obtain section and output word lengths and fraction lengths for dsp.CICDecimator and dsp.CICInterpolator System objects	9-3
Updated info method for dsp.CICDecimator and dsp.CICInterpolator System objects	9-3
Specify coefficients directly in FIR halfband interpolator and decimator	9-3
Frequency-Domain FIR Filter: Specify numerator in frequency domain	9-3
Frequency-Domain FIR Filter: Specify coefficients from input port in Simulink	9-4
Tunable Parameters Through Input Ports: Set values of tunable parameters using input signals for 14 additional Simulink blocks	9-4
Logic Analyzer enhancements	9-4
Additional pipelining of HDL-optimized Complex to Magnitude-Angle	9-4
HDL Channelizer returns data in bit-natural order for both output sizes	9-5
Variable-size signal support for dsp.VariableIntegerDelay System object	9-5
Code generation support for getRateChangeFactors function	9-5

Binary File Reader: Binary file no longer required to exist before code generation	9-5
Log data from Time Scope block as timetable	9-5
Discrete FIR Filter block supports custom state attributes for better customization and efficiency of generated code	9-5
Functionality Being Removed	9-6
Removal of Vector Scope block	9-6
Removal of DirectFeedthrough property in dsp.VariableFractionalDelay System object	9-6
Removal of DirectFeedthrough property in dsp.VariableIntegerDelay System object	9-6
Constraints on the dimensions of InitialConditions in dsp.VariableIntegerDelay System object	9-6
Functionality Removed from dsp.DigitalUpConverter and dsp.DigitalDownConverter System objects	9-7
Functionality Removed from dsp.Delay System object	9-7
Removal of 'linphase' option in firlnorm	9-8

R2017b

Improved Spectrum Analyzer: Analyze signals in the frequency domain using polyphase FFT filter banks, custom windows, dBFS units, and a spectral mask panel	10-2
Zoom FFT: Compute fast Fourier transform (FFT) of a frequency subband at high resolution	10-2
Frequency-Domain FIR Filter: Convolve long sequences while balancing latency and execution efficiency	10-2
Multitap Fractional Delay: Delay signals by multiple sample period values concurrently using variable fractional delay	10-3
Minimum Resource FFT/IFFT: Reduce resource usage with the Burst Radix 2 architecture of the HDL Optimized FFT (requires HDL Coder for code generation)	10-3
Logic Analyzer Improvements: Triggers and bus signal names	10-3
Enhancements to the dsp.Channelizer System object	10-3
Automatic Port Creation: Add inports to scope blocks when routing signals	10-3
Improvements to interactive legend in scope blocks	10-4
Array Plot Improvements: Support for scalar and variable-size inputs, axis scaling at the command line	10-4

dsp.BlockLMSFilter System object supports code generation	10-4
Functionality being removed	10-4
Removal of Overlap-Add FFT Filter block and Overlap-Save FFT Filter block	10-4
Removal of sample-based processing mode from the DSP System Toolbox System objects	10-4
Removal of adaptfilt objects	10-5
Removal of qfft and qformat functions	10-6
Removal of HDL Minimum Resource FFT block	10-7
Removal of Streaming Radix 2 architecture in HDL-optimized FFT blocks and System objects	10-7

R2017a

Improved Spectrum Analyzer: Analyze signals in the frequency domain using additional units, dual visualization, and mask compliance output	11-2
Unified interface for dsp.LogicAnalyzer: Visualize, measure, and analyze signal transitions in MATLAB using the same interface as the Simulink Logic Analyzer	11-2
Channelizer and Channel Synthesizer Blocks: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank in Simulink	11-3
Asynchronous Buffering: Exchange signals at different rates and array sizes with the dsp.AsyncBuffer System object	11-3
HDL Optimized Filters: Model and generate optimized hardware implementations for FIR filters and polyphase filter banks (requires HDL Coder for code generation)	11-3
Discrete FIR Filter	11-3
Polyphase Filter Bank	11-3
Frame Input Support for FIR Decimation	11-3
Remove outliers from streaming signals in MATLAB and Simulink using Hampel filter	11-4
Spectral estimation using filter bank in Simulink	11-4
Tunable UDP port number in generated code	11-4
Filter signals using the dsp.FilterCascade System object	11-4
Use delay and scalar gain in dsp.FilterCascade System object	11-4
Cascade a dsp.FilterCascade System object	11-5
Access the complete history of LMS filter weights in MATLAB	11-5

Tab Completion: Complete parameter names and options in DSP System Toolbox System objects	11-5
Filter Builder and fdesign support IIR halfband filter System objects	11-5
Specify image file icons for MATLAB System block	11-6
Change tunable System object properties before locking	11-6
Support for Time Scope to For Each subsystems	11-6
Copy scope to clipboard	11-6
Interactive legend for scopes	11-6
Stem plot option for Time Scope block	11-7
Time Scope Block: Connect nonvirtual bus and array of buses signals	11-7
Frame-based processing changes	11-7
Input processing parameter set to Inherited	11-7
InputProcessing property set to Inherited errors	11-8
Rate options parameter set to Inherit from input	11-8
Find the histogram over parameter set to Inherited	11-9
Running difference parameter set to Inherit from input	11-9
Save 2-D signals as parameter set to Inherit from input	11-9
Treat Mx1 and unoriented sample-based signals as parameter removed	11-9
Sample-based processing parameter removed	11-9
Functionality being removed	11-10
Running Mode in Statistics Objects and Blocks	11-10
Audio device recorder and player objects	11-10
Radix 2 architecture of HDL-optimized FFT blocks and System objects	11-11

R2016b

Logic Analyzer: Visualize, measure, and analyze transitions and states over time for Simulink signals	12-2
Spectral Mask: Compare a signal spectrum to a spectral mask using Spectrum Analyzer	12-2
Channelizer and Channel Synthesizer: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank	12-2

Moving Statistics: Measure descriptive statistics on streaming signals in MATLAB and Simulink	12-2
Gigasample per Second (GSPS) Signal Processing: Increase the throughput of HDL code generated from Discrete FIR Filter and Integer Delay blocks using frame input	12-2
Stream signals to and from binary files	12-3
Compute LMS adaptive filter weights using LMS Update block	12-3
Allpass Filter block	12-3
Specify coefficients in Farrow Rate Converter block and System object	12-3
Spectral estimation using filter banks	12-3
High-throughput polyphase filter bank for HDL example	12-3
Bit-reversed input order for HDL-optimized FFT	12-3
HDL code generation for reset port on Discrete FIR Filter	12-4
Compiler support for System object scopes	12-4
Custom X-axis data in Array Plot	12-4
Set legend strings and autoscaling programmatically in Time Scope ..	12-4
Simpler way to call System objects	12-4
System objects support for additional inputs, global variables, and enumeration data types	12-5
Functionality being removed	12-5
Removal of sample mode from the DSP System Toolbox System objects	12-5
Digital Filter block and System object	12-6
Removal of adaptfilt objects	12-6
Cell array support removal for dsp.AllpassFilter coefficients	12-7
Inherited option removed from the input processing parameter	12-7
Frame status parameter removed from the Check Signal Attributes block	12-7
qfft object errors	12-8
dspstartup removed	12-8

DSP Unfolding for Mac: Generate multithreaded MEX files from MATLAB functions on Mac OS X	13-2
Faster FIR and Biquad Filters: Run faster simulations for system models that include FIR and biquad filters	13-2
Fixed-Point Farrow Rate Converter: Design and simulate Farrow rate conversion filters using fixed-point data types	13-2
Gigasample per Second (GSPS) Signal Processing: Increase throughput of HDL-optimized FFT and IFFT algorithms using frame input	13-2
HDL Optimizations for Biquad Filter: Reduce critical path or area when generating HDL from a subsystem that includes a Biquad Filter block	13-3
Differentiate a signal using the dsp.Differentiator System object and Differentiator block	13-3
Play audio data using the audioDeviceWriter System object and Audio Device Writer block	13-3
Specify coefficients in IIR Halfband Interpolator and IIR Halfband Decimator Blocks and System objects	13-3
Customize the data limits of the Matrix Viewer block	13-4
Code generation for wave digital filter structure in dsp.AllpassFilter System object	13-4
Generate coefficients for multirate filters	13-4
Select the color of the noise in dsp.ColoredNoise System object	13-4
Full-precision setting for product data type of Biquad Filter	13-4
Code generation for Subband Analysis and Subband Synthesis Filters	13-5
Enhancements to Variable Fractional Delay	13-5
Multiple inputs for Spectrum Analyzer	13-5
Additional axes for Time Scope	13-5
Set legend programmatically in Array Plot	13-5
System object property display	13-5
System object enhancements to MATLAB System block	13-6

Enhanced System Object Development with MATLAB Editor	13-6
Functionality being removed	13-6

R2015b

DSP Unfolding: Generate a multi-threaded MEX File from a MATLAB function	14-2
HDL Optimizations for Discrete FIR Filter: Implement FIR filters in hardware at higher frequencies or using fewer resources	14-2
Array Plot Block: Visualize array and vector data	14-2
Additional Multirate Filters: Design Halfband, CIC compensation, and HDL-optimized FIR rate conversion filters	14-2
Conversion Filter Blocks: Convert the rate of signals in Simulink models	14-3
Implement FIR and IIR filters in Simulink, using the Lowpass Filter and Highpass Filter blocks	14-3
Estimate power spectrum and power spectral density using the Spectrum Estimator block	14-3
Automatic selection of filter coefficients for FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks	14-3
Visualize the frequency response of the underlying filters in the DSP System Toolbox blocks	14-3
Specify the window length and window overlap in Cross-Spectrum Estimator and Discrete Transfer Function Estimator blocks	14-4
Select the color of the noise in Colored Noise block	14-4
New functionality added to the dsp.SpectrumEstimator System object	14-4
Generate C code from dsp.AllpassFilter and import the System object into Simulink using the MATLAB System block	14-5
dsp.CICDecimator and dsp.CICInterpolator System objects support single and double data types	14-5
Frame-based signal logging in structure formats in Time Scope block	14-5
Scientific notation in Time Scope	14-5

Performance improvements for FFT, IFFT and notch peak filters	14-5
Floating-point support and optional valid port for HDL-optimized NCO	14-6
HDL Code Generation from filterbuilder	14-6
Simulink templates for ARM Cortex-A and ARM Cortex-M processors	14-6
ROI processing removed	14-7
Frame-based processing changes	14-7
Inherited Option Removed from the Input Processing Parameter	14-7
Sample-Based Row Vector Processing Changes	14-8
Blocks Emit Sample-Based Signals Only	14-9
Features removed, replaced and renamed	14-10
Blocks removed and replaced	14-10
Removal of adaptfilt objects	14-10
Removal of mfilt objects	14-11
System Object Propagation Mixin Methods Renamed	14-11

R2015a

Audio Latency Reduction: Significantly reduce latency for audio hardware I/O in MATLAB and Simulink	15-2
Filter Design Enhancements: Design high-order IIR parametric EQ filter, variable bandwidth FIR and IIR filters, Digital Down-Converter and Digital Up-Converter blocks	15-2
DSP Simulink Model Templates: Configure the Simulink environment for digital signal processing models	15-3
Streaming Scope Improvements: Plot in stem mode, access log x-axis scaling, customize sample rate, and use infinite data support	15-3
Library for HDL Supported DSP Blocks: Find all blocks that support HDL	15-3
C Code Generation of DSP Algorithms for ARM Cortex-A and Cortex-M processors: Generate optimized and faster performing C code using Embedded Coder	15-3
Performance Improvements	15-4
Updated Time Scope block toolbar and menus	15-4
Specify block filter characteristics through System objects	15-4

Discrete Transfer Function Estimator block	15-5
Specify filter coefficients as an input to the FIR Decimation block	15-5
Enhanced code generation for CIC Decimation and CIC Interpolation filter blocks	15-5
HDL support for 'inherit via internal rule' data type setting on FIR Decimation and Interpolation blocks	15-5
Improvements for creating System objects	15-5
Min/Max logging instrumentation for float-to-fixed-point conversion of DSP System objects	15-6
Provide variable-size input to the Delay System object	15-6
Estimate output coherence of Transfer Function Estimator System object	15-6
Specify filter coefficients as an input to the FIR Decimator System object	15-6
Bit growth to avoid overflow in HDL-optimized FFT and IFFT	15-7
Fixed-point support for FIR Half-band Interpolator and FIR Half-band Decimator System objects	15-7
Updated cost method for filter System objects	15-7
Frame-based processing	15-7
Input processing parameter set to Inherited	15-8
Rate options parameter set to Inherit from input	15-20
Treat Mx1 and unoriented sample-based signals as parameter set to M channels	15-20
Save 2-D signals as parameter set to Inherit from input	15-20
Find the histogram over parameter set to Inherited	15-21
Sample-based processing parameter set to Pass through	15-21
Running difference parameter set to Inherit from input	15-22
Features removed, replaced, and duplicated	15-22
Blocks replaced, removed, and available in additional libraries	15-22
Removal of adaptfilt objects	15-23
Functionality changed or being removed for blocks and System objects	15-24
Removal of sample mode from the DSP System Toolbox System objects	15-24
Option to specify filter coefficients from Digital Up Converter and Digital Down Converter System objects being removed	15-25
Removal of OutputDataType and OverflowAction properties for CIC Compensation Interpolator and Decimator System objects	15-26

Optimized C code generation for ARM Cortex-A Ne10 library from MATLAB and Simulink with DSP System Toolbox Support Package for ARM Cortex-A Processors	16-2
System objects for DSP System Toolbox Support Package for ARM Cortex-M Processors	16-3
Fixed-point support for Biquad Filter on DSP System Toolbox Support Package for ARM Cortex-M Processors	16-3
Multirate filters: Sample and Farrow Rate Converter, CIC Compensation Interpolator/Decimator, and FIR Halfband Interpolator/Decimator System objects	16-3
Tunable coefficients and variable-size input available on FIR Interpolator System object and block	16-3
Variable-size input available on FIR Decimator System object and block	16-4
Min/Max logging instrumentation for float-to-fixed-point conversion of commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter	16-4
HDL-optimized FFT and IFFT System objects and HDL-optimized Complex to Magnitude-Angle System object and block	16-4
Real input, bit-reversed output, reset input available on HDL-optimized FFT and IFFT	16-4
Option to synthesize lookup table to ROM available on HDL-optimized FFT and IFFT blocks	16-5
Reduced latency of HDL-optimized FFT and IFFT	16-5
CIC algorithm and HDL code generation for DC Blocker	16-5
dsp.FilterCascade System object	16-5
Phase Extractor block and dsp.PhaseExtractor System object	16-5
Overflow and underrun reporting on audio device blocks and System objects	16-5
Unsigned input data type in dsp.CICDecimator and dsp.CICInterpolator System Objects	16-6
Logic Analyzer support for vector, enumerated, and complex inputs ...	16-6
System object support in Simulink For Each Subsystem	16-6

Getting Started Tutorials	16-6
Functionality being removed or replaced for blocks and System objects	16-6
Persistence mode in Vector Scope	16-12
Code generation for additional DSP System Toolbox System objects ..	16-12
Tunable amplitude on dsp.SineWave	16-13

R2014a

Up to four-times faster FIR filter simulation in MATLAB System object and Simulink block	17-2
Optimized C code generation for ARM Cortex-M processors from System objects with MATLAB Coder and Embedded Coder	17-2
Notch/peak filter and parametric equalizer filter System objects in MATLAB	17-2
Variable bandwidth FIR and IIR filter System objects in MATLAB	17-2
Pink/Colored noise generation System object in MATLAB	17-3
HDL optimized FFT and IFFT Simulink blocks	17-3
Fixed-point data type support for FIR filter, in ARM Cortex-M support package	17-3
Choice of wrapping or truncating input of FFT, IFFT, and Magnitude FFT in MATLAB and Simulink	17-3
Variable-size input for biquad and LMS filters in MATLAB and Simulink	17-3
More flexible control of dsp.LMSFilter System object fixed-point settings	17-3
DC blocker System object and Simulink block	17-4
dsp.DigitalDownConverter and dsp.DigitalUpConverter now support C code generation	17-4
The isDone method of dsp.AudioFileReader honors PlayCount	17-4
M4A replaced by MPEG4 in dsp.AudioFileWriter	17-4
Spectrogram cursors and CCDF plots in the spectrum analyzer	17-4

Changed dsp.SpectrumAnalyzer property names	17-4
Conversion to/from allpass from/to wave digital filter	17-5
Transfer function estimation in Simulink	17-5
Updates to the Time Scope	17-5
Changed dsp.TimeScope property names	17-5
Time Scope automatically switches to block-based sample time	17-5
dsp.LogicAnalyzer channel selection	17-6
System object templates	17-6
System objects infer number of inputs and outputs from stepImpl method	17-6
System objects setupImpl method enhancement	17-6
System objects infoImpl method allows variable inputs	17-6
System objects base class renamed to matlab.System	17-6
System objects Propagates mixin methods	17-6
Code generation support for additional functions	17-7

R2013b

Support Package for ARM Cortex-M Processors	18-2
Channel and distortion measurement, cursors, and spectrogram visualization using Spectrum Analyzer in MATLAB and Simulink ...	18-2
Channel mapping for multichannel audio devices in MATLAB and Simulink	18-2
Variable-size support for FIR and Allpole filters in MATLAB and Simulink	18-3
Estimation of Power Spectrum, Cross Power Spectrum, and Transfer Function for streaming data in MATLAB	18-3
Data logging and archiving using Time Scope in Simulink	18-3
MIDI control interface support in MATLAB	18-3
Integer support on the output port of the MIDI Controls block	18-3

Kalman filter	18-4
Adaptive filters using Lattice, Fast Transversal, Filtered-X LMS, and Frequency Domain algorithms in MATLAB	18-4
Coupled allpass filter	18-4
Functionality being removed or changed	18-4
Migrate away from fdesign.pulseshaping	18-5
Configuration dialog added to Logic Analyzer	18-5
Complex trigger support in Time Scope	18-5
Default color changes for Array Plot, Time Scope, and Spectrum Analyzer	18-5
MATLAB System Block to include System objects in Simulink models	18-6
Restrictions on modifying properties in System object Impl methods	18-6
System objects matlab.system.System warnings	18-7
Removing HDL Support for NCO Block	18-7

R2013a

Allpass Filter System object	19-2
Adaptive filter System objects using RLS and Affine Projection Filter	19-2
Logic Analyzer System object	19-2
Audio System object support for tunability, variable frame size, variable number of channels, and writing MPEG-4 AAC	19-2
Array Plot System object for displaying vectors or arrays in 2-D and Spectrum Analyzer block with enhanced controls and features such as peak finder	19-3
Time Scope block with triggering and peak finder features	19-7
Triggers Panel	19-8
Peak Finder Features	19-8
Panning Capability	19-8
Programmatic Access	19-8
Scale Axes Limits After 10 Updates	19-8

Change of the default for audio hardware API on Linux	19-9
Change of the default for audio file formats in multimedia blocks and audio file reader and writer System objects	19-9
Change of property default in the audio file reader System object	19-9
Removal of the signalblks package	19-9
Scope Snapshot display of additional scopes in Simulink Report Generator	19-9
Unoriented vector treated as column vector in the Biquad Filter	19-9
NCO HDL Optimized block	19-10
HDLNCO System object	19-10
HDL code generation for NCO HDL Optimized block and System object	19-10
Support for nonpersistent System objects	19-10
New method for action when System object input size changes	19-10
Scaled double data type support for System objects	19-10

R2012b

SpectrumAnalyzer System object	20-2
Cross-platform support for reading and writing WAV, FLAC, OGG, MP3 (read only), MP4 (read only), and M4a (read only)	20-2
Support for code generation for CICDecimator and CICInterpolator System objects	20-3
Support for HDL code generation for multichannel Discrete FIR Filter block	20-3
Time Scope enhancements, including new cursors, embedded simulation controls, and External and Rapid Accelerator modes	20-3
Cursor measurements panel	20-4
Additional embedded simulation controls	20-4
Support for external mode and rapid accelerator mode	20-4
Properties dialog box	20-5
Axes Maximization	20-5
Automatic calculation of Time Span	20-5
ReduceUpdates property	20-6
Support for conditional subsystems	20-6

Source and sink blocks being replaced	20-6
Discrete IIRFilter and AllpoleFilter System objects	20-7
Support for MATLAB Compiler for CICDecimator and CICInterpolator System objects	20-7
Code generation support for SignalSource System object	20-7
Behavior change of locked System objects for loading, saving, and cloning	20-7
Behavior change of statistics blocks for variable-size inputs	20-8
Simulation state save and restore for additional blocks	20-8
For Each subsystem support for additional blocks	20-9
Multi-instance model referencing support for additional blocks	20-9
Expanded analysis support for filter System objects	20-9
Removal of the signalblks package	20-9
Discrete filter block visible in DSP library	20-10
System object tunable parameter support in code generation	20-10
save and load methods for System objects	20-10
Save and restore SimState not supported for System objects	20-10
Map integer delay to RAM on Delay block	20-10
HDL support for System objects	20-10
HDL resource sharing for Biquad Filter block	20-10

R2012a

Frame-Based Processing	21-2
Inherited Option of the Input Processing Parameter Now Warns	21-2
Logging Frame-Based Signals in Simulink	21-3
Model Reference and Using slupdate	21-3
Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay	21-4
Audio Output Sampling Mode Added to the From Multimedia File Block	21-4

System Object Enhancements	21-4
Code Generation for System Objects	21-4
New MAT-File Reader and Writer System Objects	21-4
New System Object Option on File Menu	21-4
Variable-Size Input Support for System Objects	21-4
Data Type Support for System Objects	21-5
New Property Attribute to Define States	21-5
New Methods to Validate Properties and Get States from System Objects	21-5
matlab.system.System changed to matlab.System	21-5
Time Scope Enhancements	21-5
Time Domain Measurements in Time Scope	21-5
Multiple Display Support in Time Scope	21-6
Style Dialog Box in Time Scope	21-6
Sampled Data as Stairs in Time Scope	21-6
Complex Data Support in Time Scope	21-7
Additional Time Scope Enhancements	21-7
ASIO Support in To/From Audio Device Blocks and Objects	21-7
Video Processing Enabled for the DSP System Toolbox Multimedia File Blocks	21-8
System Objects Integrated into Filter Design Workflow	21-8
Integration of System Objects into Filter Design via fdesign, FDATool, and Filterbuilder	21-8
Convert dfilt and mfilt Filter Objects to System Objects	21-8
Filter Analysis and Conversion Methods for System Object Filters	21-8
New Measurement Workflow	21-9
Measurements for Bilevel Pulse Waveforms	21-9
System Objects for Peak-to-RMS and Peak-to-Peak Measurements	21-9
Discrete FIR Filter System Object	21-9
Inverse Dirichlet Sinc-Shaped Passband Design Added to Constrained FIR Equiripple Filter	21-9
Code Generation Support Added to FIR Decimator System Object	21-10
Filter Block Enhancements	21-10
IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function	21-10
Reset Port for Resetting Filter State in Filter Blocks	21-10
Discrete FIR Filter Block Coefficient Port Changes	21-10
Statistics Blocks and Objects Warning for Region of Interest Processing	21-10
New and Updated Demos	21-10

Frame-Based Processing	22-2
General Product-Wide Changes	22-2
Logging Signals in Simulink	22-3
Triggered to Workspace	22-3
Digital Filter Design Block	22-4
Filterbuilder, FDATool and the Filter Realization Wizard Block	22-5
Changes to Row Vector Processing for dsp.Convolver, dsp.CrossCorrelator, and dsp.Interpolator System Objects	22-5
Custom System Objects	22-5
New Allpole Filter Block	22-5
New Audio Weighting Filter Functionality	22-5
Time Scope Enhancements	22-5
New Arbitrary Group Delay Design Support	22-6
Arbitrary Magnitude Responses Now Support Minimum Order and Minimum/Maximum Phase Equiripple Design Options	22-6
Support for Constrained Band Equiripple Designs in MATLAB and Simulink	22-6
New Sinc Frequency Factor and Sinc Power Design Options for Inverse Sinc Filters	22-7
New Inverse Sinc Highpass Filter Designs	22-7
Filterbuilder and dspfdesign Library Blocks Now Support Different Numerator and Denominator Orders for IIR Filters	22-7
New Stopband Shape and Stopband Decay Design Options for Equiripple Highpass Filter Designs	22-7
FFTW Library Support for Non-Power-of-Two Transform Length	22-7
MATLAB Compiler Support for dsp.DigitalDownConverter and dsp.DigitalUpConverter	22-8
Complex Input Support for dsp.DigitalDownConverter	22-8
getFilters Method of dsp.DigitalDownConverter and dsp.DigitalUpConverter Now Return Actual Fixed-Point Settings ...	22-8
dsp.SineWave and dsp.BiquadFilter Properties Not Tunable	22-8
System Object DataType and CustomDataType Properties Changes	22-8

System Objects Variable-Size Input Dimensions	22-9
Conversion of Error and Warning Message Identifiers	22-9
New and Updated Demos	22-10
Blocks Being Removed in a Future Release	22-10

R2011a

Product Restructuring	23-2
Frame-Based Processing	23-2
General Product-Wide Changes	23-2
Blocks with a New Input Processing Parameter	23-3
Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks	23-5
Difference Block Changes	23-5
Signal To Workspace Block Changes	23-6
Spectrum Scope Block Changes	23-6
Sample-Based Row Vector Processing Changes	23-6
New Function for Changing the System Object Package Name from signalblks to dsp	23-8
New Discrete FIR Filter Block	23-9
New Printing Capability from the Time Scope Block	23-9
Improved Display Updates for the Time Scope Block and System Object	23-9
New Implementation Options Added to Blocks in the Filter Designs Library	23-9
New dsp.DigitalDownConverter and dsp.DigitalUpConverter System Objects	23-10
Improved Performance of FFT Implementation with FFTW library ...	23-10
Variable-Size Support for System Objects	23-10
System Objects FullPrecisionOverride Property Added	23-11
'Internal rule' System Object Property Value Changed to 'Full precision'	23-11
MATLAB Compiler Support for System Objects	23-12
Viewing System Objects in the MATLAB Variable Editor	23-12

System Object Input and Property Warnings Changed to Errors	23-12
New and Updated Demos	23-12
Documentation Examples Renamed	23-13
Downsample Block No Longer Has Frame-Based Processing Latency for a Frame Size of One	23-13
SignalReader System Object Accepts Column Input Only	23-13
FrameBasedProcessing Property Removed from the dsp.DelayLine and dsp.Normalizer System Objects	23-14
R2010a MAT Files with System Objects Load Incorrectly	23-14

R2022a

Version: 9.14

New Features

Bug Fixes

Version History

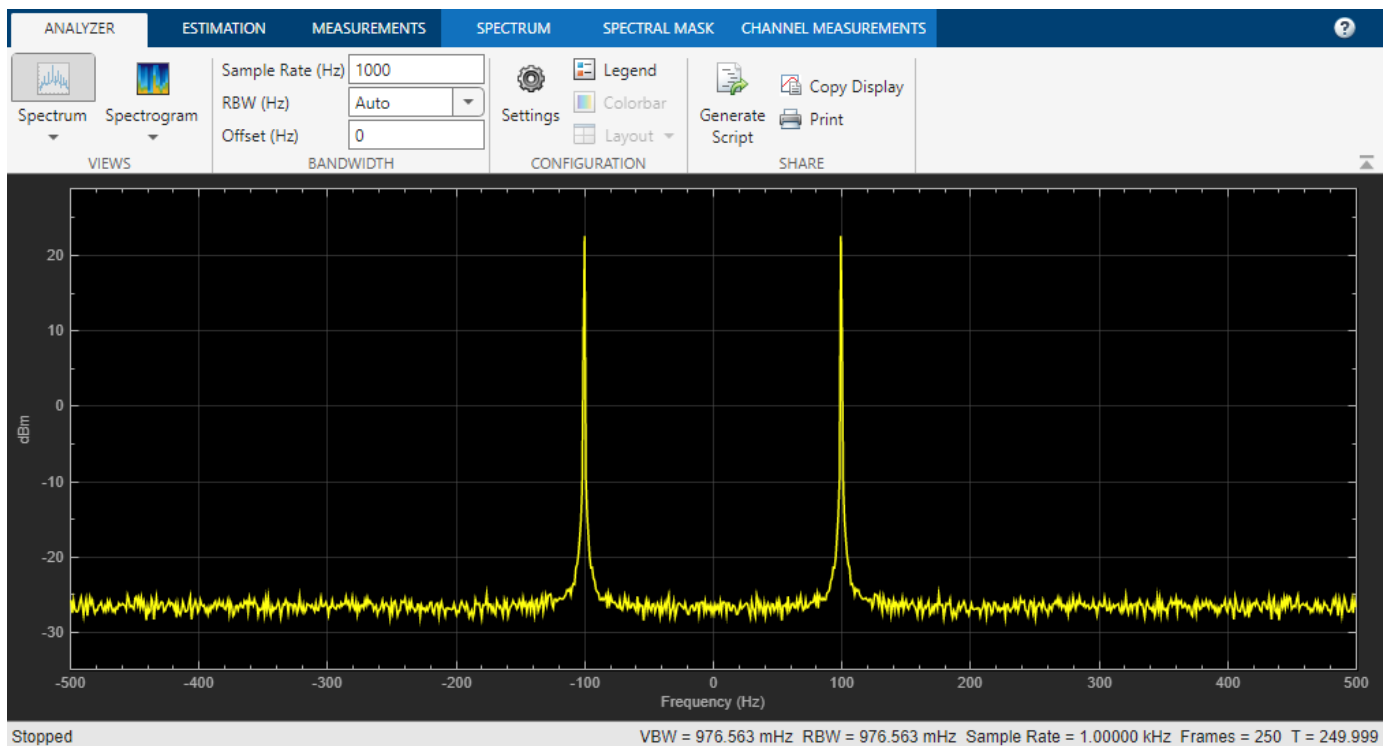
Spectrum Analyzer with better responsiveness and toolstrip interface for analysis, estimation, and measurement parameters

Use the new spectrumAnalyzer MATLAB® object to visualize the frequency spectrum of the time-domain signals. The new object comes with better responsiveness and a toolstrip interface that gives easy access to spectral analysis, estimation, and measurements.

The new Spectrum Analyzer window has the following toolstrip tabs:

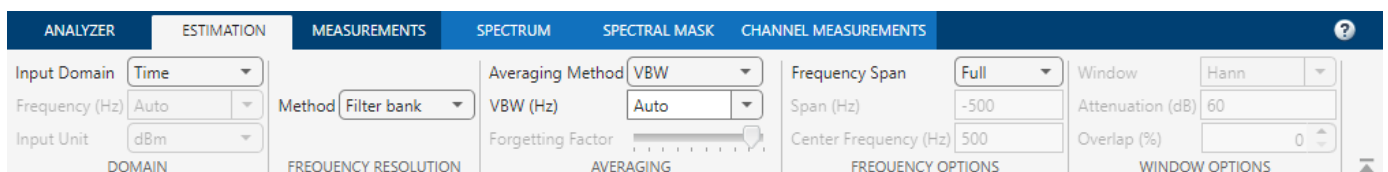
Analyzer Tab

On the **Analyzer** tab, you can control the spectrum and spectrogram display settings, spectral-bandwidth-related settings, and configuration settings of the Spectrum Analyzer. You can also generate script to recreate your spectrum analyzer with the same settings. When doing so, an editor window opens with the code required to recreate your spectrumAnalyzer object.



Estimation Tab

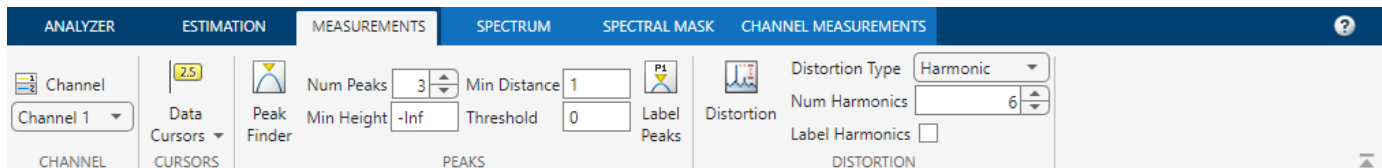
On the **Estimation** tab, you can specify the input domain, frequency resolution method, frequency span options, and the averaging method that is used to estimate the power spectrum. If you specify the frequency resolution method as *Welch*, you can also choose the options of the filter window that is used for spectral estimation.



Measurements Tab

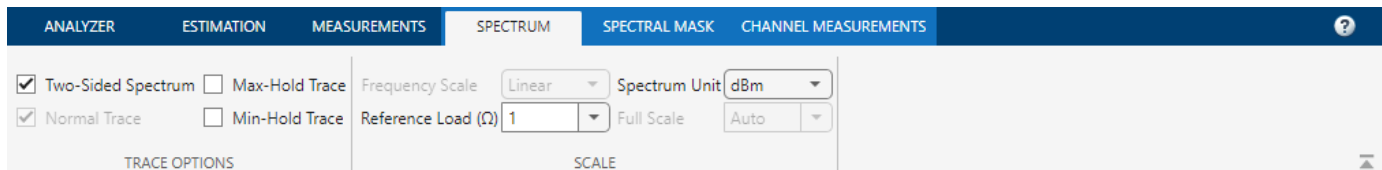
In the **Measurements** tab, all measurements are for a specific channel.

- **Data Cursors** — Display the screen cursors.
- **Peak Finder** — Display peak values for the selected signal.
- **Distortion** — Display harmonic and intermodulation distortion measurements.



Spectrum Tab

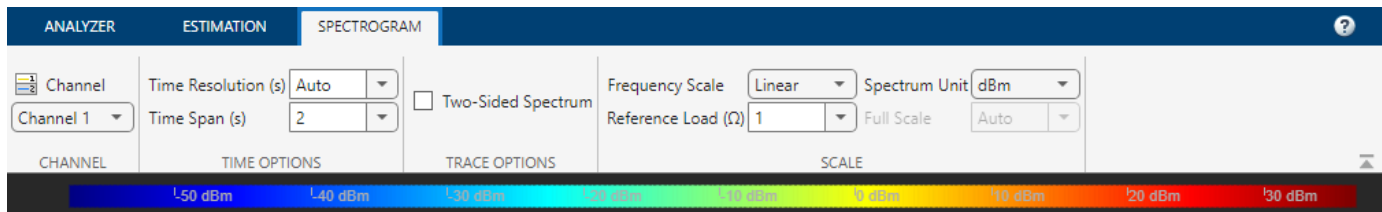
On the **Spectrum** tab, you can specify the trace and scale options of the Spectrum Analyzer. You can choose to show a two-sided spectrum or a one-sided spectrum, and select to show maximum-hold trace, minimum-hold trace, or normal trace. You can also specify the reference load and spectral units.



Spectrogram Tab

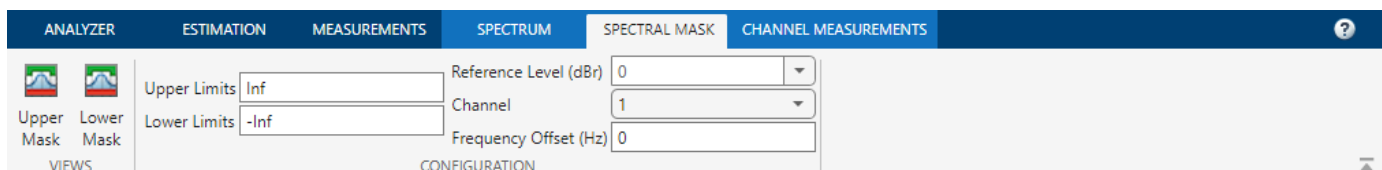
To display the **Spectrogram** tab, click on the **Spectrogram** view in the **Analyzer** tab.

On the **Spectrogram** tab, you can specify the channel over which the spectrogram is computed, time resolution and time span of the spectrogram, and choose to show a two-sided spectrum or a one-sided spectrum. You can also specify the reference load and spectral units.



Spectral Mask Tab

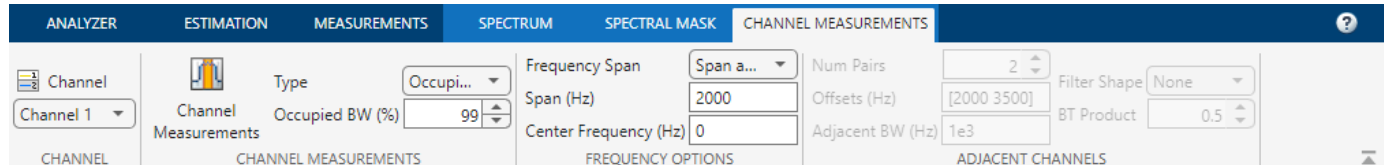
Add upper and lower masks to the Spectrum Analyzer to visualize spectrum limits and compare spectrum values to specification values.



Channel Measurements Tab

In the **Channel Measurements** tab, all measurements are for a specific channel.

- **Channel Measurements** — Select to display occupied bandwidth or adjacent channel power ratio (ACPR) measurements.
- **Frequency Options** — Display frequency span, center frequency, and start and stop frequencies.
- **Adjacent Channels** — Display adjacent channel power ratio (ACPR) measurements.



Fourth-Order Section Filter Block: Implement a cascade of fourth-order section filters in Simulink

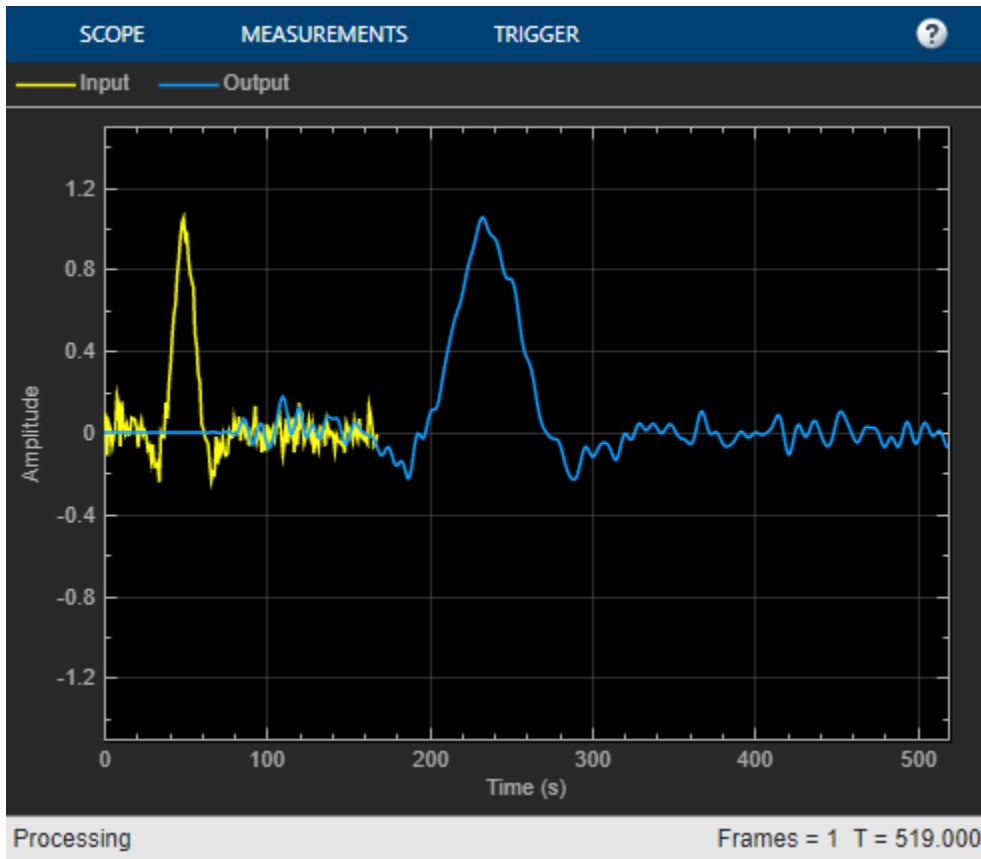
Starting in R2022a, using the Fourth-Order Section Filter block, you can model a cascade of fourth-order section filters in the Simulink® environment. This block accepts the numerator and the denominator coefficients of the filter in the form of fourth-order section matrices.

New outputDelay Function: Determine delay in multirate filter System objects

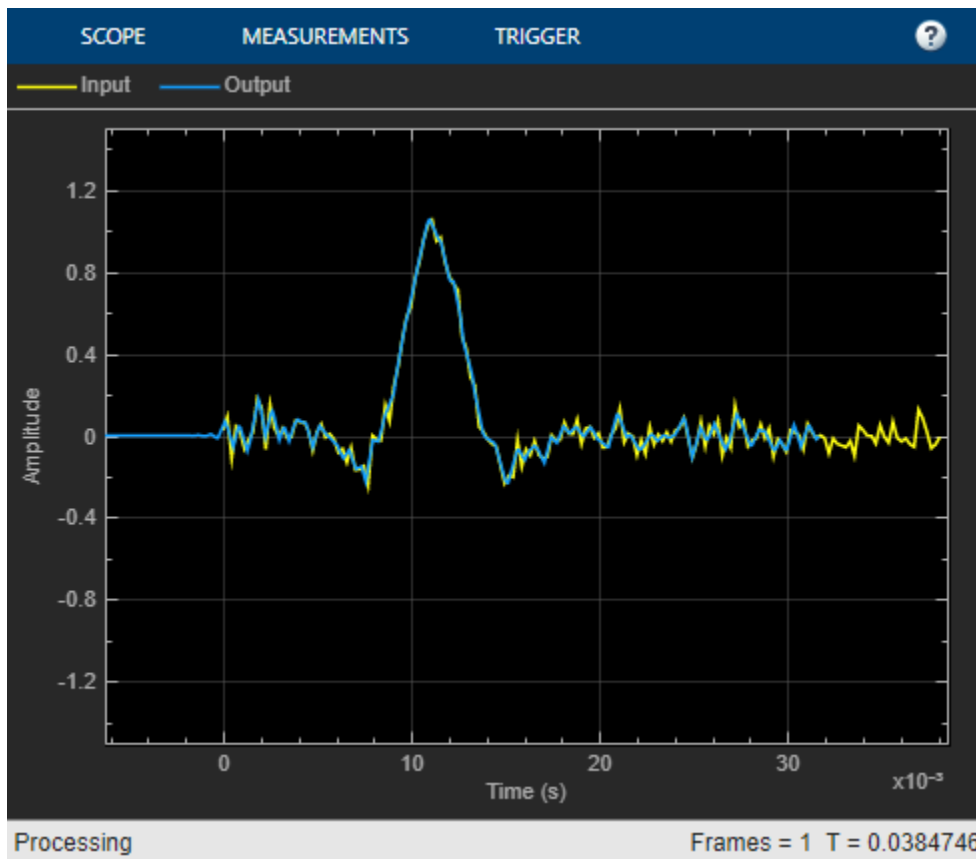
Starting in R2022a, the new `outputDelay` function computes the delay introduced by a multirate filter. This delay is caused by the group delay of the convolution operations within the filter.

```
filterObj = dsp.FIRRateConverter(13,17)
D = outputDelay(filterObj)
```

Due to this delay and the potential time scaling that occurs as a result of rate conversion within the filter, the input and output signals do not appear synchronized on the time domain if plotted against the same sample time.



To plot the input and output signals on the same plot, you must account for the delay and the time scaling.



The `outputDelay` function also returns the output sample rate $FsOut$. You can optionally specify the input sample rate to the function $FsIn$.

```
[D,FsOut] = outputDelay(filterObj,FsIn=44.1e3)
```

Filters with Nonlinear Phase Convolution

Note that the `outputDelay` function also supports multirate filters which have a nonlinear phase convolution (varying group delay) assuming their inputs are narrow-banded. In this case, you can specify the center frequency Fc of the input signal. The function can also return an interval of input frequencies B for which the delay value is valid for up to a specified tolerance.

```
nonlinearObj = cascade(dsp.FIRFilter(designFracDelayFIR(0.25,200)),...  
dsp.FIRDecimator)  
[D,FsOut,B] = outputDelay(nonlinearObj,FsIn=44.1e3,Fc=100,Tol=0.1)
```

Enhancements to LMS Update block: Multifilter and adaptive linear combiner support

Starting in R2022a, you can update the weights of multiple adaptive filters using the same LMS Update block. To enable this support, specify the **Number of adaptive filters** to be a value N greater than 1. Each filter can be chosen to adapt independently during run time by providing the **Adapt** input as a logical vector of length N . All filters are assumed to run on the same input and have a common step size, leakage factor, and reset input.

You can also use the LMS Update block as an adaptive linear combiner where the input to the block is not necessarily obtained from a tapped-delay line. You can enable this mode by setting the **Adaptive**

filter mode parameter to Adaptive linear combiner. In the Adaptive linear combiner mode, the input samples are not buffered internally inside the block, and the input provided to the block must be a column vector of size equal to the **Filter length** parameter.

New CCDF measurement mode in powermeter System object

Starting in R2022a, you can compute and output the CCDF measurements using the powermeter System object™.

Kaiser-window-based design method for FIR halfband decimator and interpolator

You can now specify the Kaiser-window-based filter design method in the following FIR halfband decimators and interpolators:

- `dsp.FIRHalfbandDecimator`
- `dsp.FIRHalfbandInterpolator`
- FIR Halfband Decimator
- FIR Halfband Interpolator

The Kaiser window method is optimal to design filters with very tight specifications such as very high orders and very narrow transition widths. For a given set of design specifications, if the existing Equiripple design method is not able to optimally meet the design constraints, then use the Kaiser window design method.

If you are not sure of which method to use, set the design method to its default value Auto. The Auto mode automatically chooses a design method from the two methods that optimally meets the filter constraints specified in the block parameters.

Certain IIR frequency transformation functions now support higher-order section filters

Starting in R2022a, the following IIR frequency transformation functions in DSP System Toolbox support higher-order section input prototype filters. You can specify the input filter coefficients as second-order or higher-order section matrices. For more details, see these function reference pages.

- `iirftransf`
- `iirlp2bp`
- `iirlp2bpc`
- `iirlp2bs`
- `iirlp2bpc`
- `iirlp2hp`
- `iirlp2lp`
- `iirlp2mb`
- `iirlp2mbc`
- `iirlp2xc`

- `iirlp2xn`

Farrow Rate Converter block supports variable-size input signal and arbitrary input frame size

Starting in R2022a, the Farrow Rate Converter block supports variable-size input signal and arbitrary input frame size.

To enable this behavior, clear the **Force fixed size input and output frames** parameter in the block dialog box. In this mode, the frame size (number of rows) of the input signal can change during simulation while the number of channels (number of columns) must remain constant.

Also, the frame size of the signal no longer needs to be a multiple of the decimation factor. To determine the decimation factor of the rate converter, click the **View Info** button in the block dialog box.

When you select the **Force fixed size input and output frames** parameter, the block expects a fixed-size input signal with a frame size that is a multiple of the decimation factor.

iirnotch, iirpeak, iircomb, and designMultirateFIR functions now support nonconstant inputs during code generation

While generating code from the `iirnotch`, `iirpeak`, `iircomb`, and `designMultirateFIR` functions, you no longer need to specify the function inputs as constants.

```
codegen iircomb -args {10,0.7}
iircomb_mex(10,0.0057)
```

For more details on constant inputs, see the `coder.Constant` object.

In the case of the `designMultirateFIR` function, the inputs to the function must be constants when generating code for transition-width-based filter design. In this design, the polyphase length (and therefore the filter length) is not specified and the function determines this value iteratively. For more details, see the “Code Generation” section in the `designMultirateFIR` page.

Improved speed performance in certain DSP System Toolbox features

In R2022a, these features have an improved speed performance under the specified conditions.

Feature	Conditions	Improved Speed Performance
<code>dsp.Channelizer</code>	<ul style="list-style-type: none"> • Input signal has a data type of <code>single</code> or <code>double</code> • Input signal can be real or complex 	Notice improved speed performance when you: <ul style="list-style-type: none"> • Simulate the object algorithm in MATLAB • Run the MEX file that you generate using the <code>codegen</code> function
<code>dsp.ChannelSynthesizer</code>		
<code>dsp.FIRDecimator</code>		
<code>dsp.FIRRateConverter</code>		
<code>dsp.SpectrumEstimator</code>		
<code>dsp.SubbandAnalysisFilter</code>		

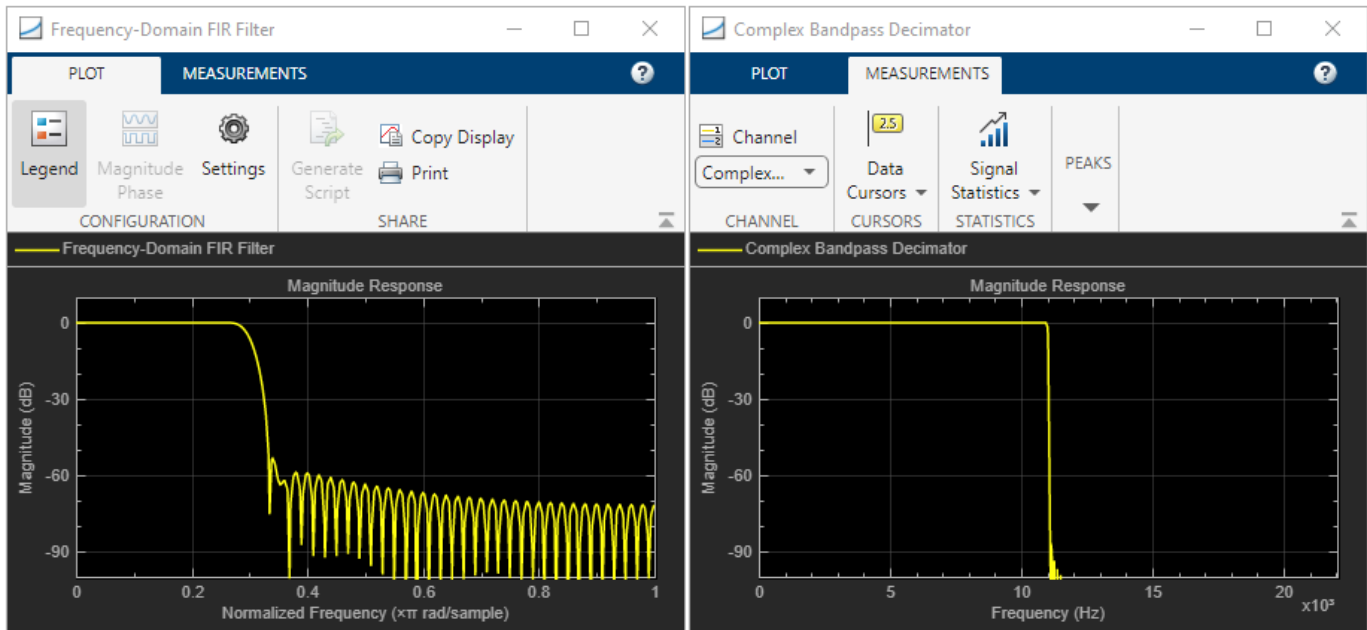
Feature	Conditions	Improved Speed Performance
dsp.SampleRateConverter		<ul style="list-style-type: none"> Run the MEX file that you generate using the <code>dspunfold</code> function
dsp.ZoomFFT		
Spectrum Estimator	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code> Input signal can be real or complex 	<p>Notice improved speed performance when the block:</p> <ul style="list-style-type: none"> Simulates in <code>Normal</code> mode Simulates in <code>Accelerator</code> mode Simulates in <code>Rapid Accelerator</code> mode Belongs to a model reference (Simulink) and operates in <code>Normal</code> mode Belongs to a model reference (Simulink) and operates in <code>Accelerator</code> mode <p>For more details on these modes, see “Choosing a Simulation Mode” (Simulink) and “Choose Simulation Modes for Model Hierarchies” (Simulink).</p>
FIR Decimation	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code> Input signal can be real or complex Rate options set to <code>Enforce single-rate processing</code> or <code>Allow multirate processing</code> When Rate options is set to <code>Allow multirate processing</code>, the input frame length (number of rows) can be of any size and does not have to be a multiple of the decimation factor 	<p>Notice improved speed performance when the block:</p> <ul style="list-style-type: none"> Simulates in <code>Normal</code> mode Simulates in <code>Accelerator</code> mode Simulates in <code>Rapid Accelerator</code> mode Belongs to a model reference (Simulink) and operates in <code>Normal</code> mode Belongs to a model reference (Simulink) and operates in <code>Accelerator</code> mode Belongs to a <code>Dataflow Subsystem</code> block
FIR Rate Conversion		
Two-Channel Analysis Subband Filter		
Channelizer	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>. Input signal can be real or complex. 	<p>For more details on these modes, see “Choosing a Simulation Mode” (Simulink) and “Choose Simulation Modes</p>

Feature	Conditions	Improved Speed Performance
Channel Synthesizer	<ul style="list-style-type: none"> Simulate using is set to Interpreted execution or Code generation. 	for Model Hierarchies” (Simulink).

Improved filter response visualization in certain DSP System Toolbox blocks

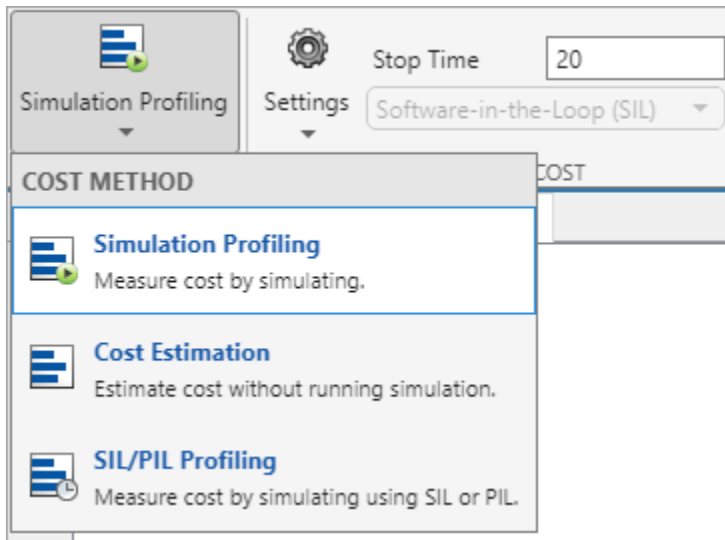
When you click the **View Filter Response** button in the dialog box of the Frequency-Domain FIR Filter and Complex Bandpass Decimator blocks, the dynamic filter visualizer launches and shows the magnitude response of the designed filter. The response is based on the block parameters. To update the magnitude response while the dynamic filter visualizer is running, modify the parameters in the block dialog box and click **Apply**.

Using the dynamic filter visualizer, you can configure the plot settings, measure signal statistics, find peak values, place data cursors, and so on from the interface of the visualizer. For more details on the dynamic filter visualizer interface and the available tools, see `dsp.DynamicFilterVisualizer`.



Dataflow simulation analysis using the Multicore tab

A new **Simulation Profiling** option is available in the **Multicore** tab for simulation profiling. By default, **Simulation Profiling** is the default cost method of the **Multicore** tab. You can use simulation profiling to measure average execution times (cost) of blocks inside the dataflow subsystem for automatic partitioning of the subsystem for multithreaded simulation. For more information, see “Perform Multicore Analysis for Dataflow”.



Buffer and Unbuffer blocks now support code generation for Simulink Real-Time

Starting in R2022a, the Buffer and Unbuffer blocks support code generation for Simulink Real-Time™. The code generated for this target is executed concurrently.

To generate this code, the system target file of the model should be set to `slrealtime.tlc` and the blocks must satisfy certain conditions. For more details on these conditions, see the **Extended Capabilities > C/C++ Code Generation** sections on the Buffer and Unbuffer block reference pages.

Nonrecursive code generation in DCT and IDCT blocks

The DCT and IDCT blocks generate code that is nonrecursive. This is to ensure that the generated code complies with the embedded systems coding standards.

Time Scope MATLAB object and Array Plot now support additional signal statistics

The `timescope` object, `dsp.ArrayPlot` object, and the Array Plot block now support the following additional signal statistics:

- Standard deviation
- Variance
- Mean square

For more details, see “Configure Time Scope MATLAB Object” and “Configure Array Plot”.

Configure timescope measurements programmatically

The following properties configure measurement data programmatically for the `timescope` object:

- `MeasurementChannel`
- `BilevelMeasurements`
- `CursorMeasurements`
- `PeakFinder`
- `SignalStatistics`
- `Trigger`

Using these properties, you can now interact with the scope from the command line. Any changes that you make to the properties in the command line change the corresponding values in the UI. Any changes that you make in the UI change the corresponding property values in the command line.

Configure `dsp.ArrayPlot` measurements programmatically

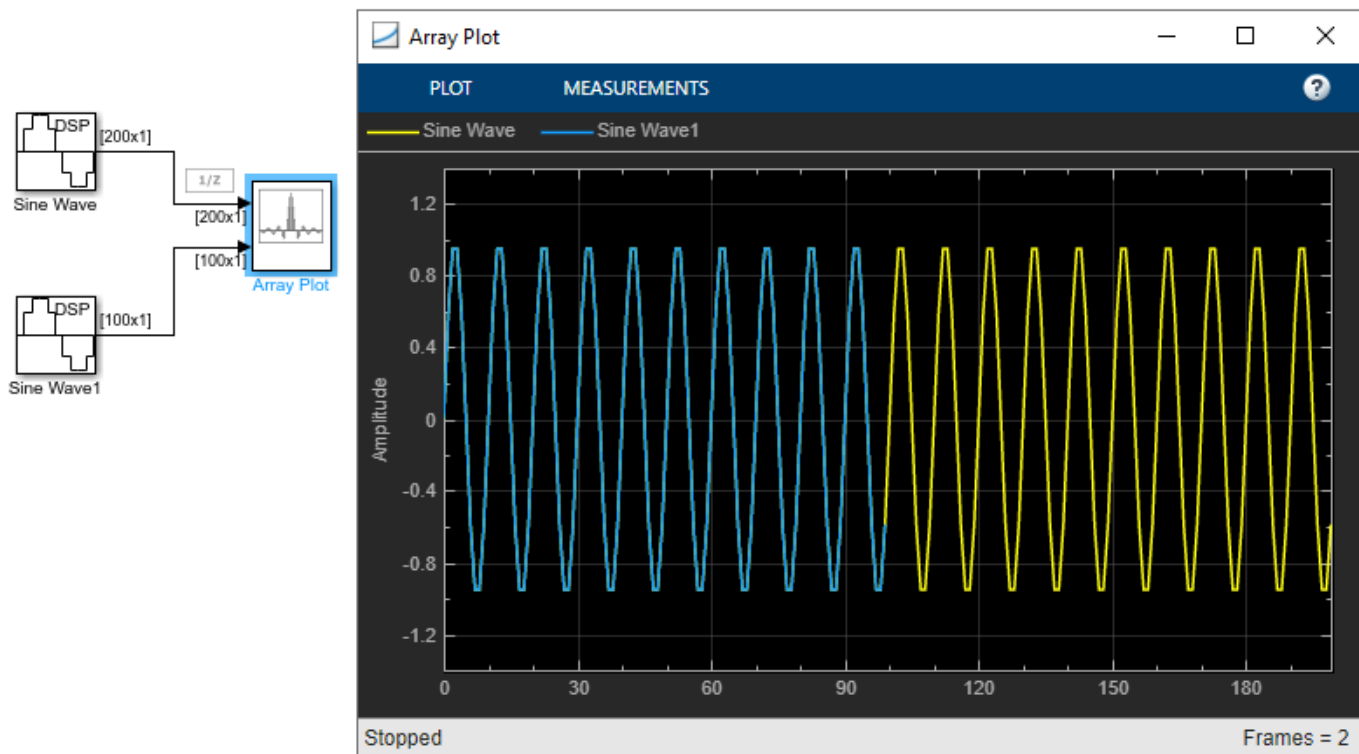
The following properties configure measurement data programmatically for the `dsp.ArrayPlot` object:

- `MeasurementChannel`
- `CursorMeasurements`
- `PeakFinder`
- `SignalStatistics`

Using these properties, you can now interact with the scope from the command line. Any changes that you make to the properties in the command line change the corresponding values in the UI. Any changes that you make in the UI change the corresponding property values in the command line.

Array Plot block supports multiple frame sizes

The Array Plot block now supports inputs with different frame lengths.



To Workspace block inside a Dataflow Subsystem block supports multithreaded execution

Starting in R2022a, the To Workspace block inside a Dataflow Subsystem block supports multithreaded simulation.

Version History

Before R2022a, using a To Workspace block inside a Dataflow Subsystem block forced dataflow domain to single-threaded execution. Starting in R2022a, the To Workspace block supports multi-threaded execution inside a Dataflow Subsystem block. However, during multi-threaded execution, the output of the To Workspace block may be different than the output during the single-threaded execution. The output may differ due to the distribution of the pipeline delays inside the Dataflow Subsystem block for multithreading. These delays impact the output of the To Workspace block due to the latencies in the model. You can visualize these delays and the relationship between the output of the To Workspace block and the delays. For more information, see “Dataflow Domain”.

Functionality being removed

CCDF measurements will be removed from the `dsp.SpectrumAnalyzer` object

Warns

The `CCDFMeasurements` property will be removed from the `dsp.SpectrumAnalyzer` object. If you try to edit these measurements from the command line or from the user interface (UI), the object throws a warning message.

Use the `powermeter` object instead to plot and visualize CCDF measurements.

'Raised Cosine' and 'Square Root Raised Cosine' response methods will be removed

Warns

The 'Raised Cosine' and 'Square Root Raised Cosine' response methods in the `fdesign.decimator` and `fdesign.interpolator` objects will be removed in a future release. Use `rcosdesign`, `comm.RaisedCosineTransmitFilter`, and `comm.RaisedCosineReceiveFilter` instead.

DirectFeedthrough property has been removed from dsp.VariableFractionalDelay System object

The `DirectFeedthrough` property has been removed from the `dsp.VariableFractionalDelay` System object. The object now operates in direct feedthrough mode by default.

dsp.AllpassFilter System object no longer supports cell array coefficients

Cell array support for `AllpassCoefficients`, `WDFCoefficients`, and `LatticeCoefficients` properties in the `dsp.AllpassFilter` object has been removed. Use an N -by-1 or N -by-2 numeric array instead.

Functionality removed in dsp.Delay System object

Starting in R2022a, you can no longer specify the units of input delay through the `Units` property in the `dsp.Delay` System object. The object now delays the input in samples.

The `InitialConditions` property in the `dsp.Delay` object no longer supports a cell array format. Use a `Length-by-numChans` matrix instead, where `numChans` is the number of input channels and `Length` is the value you specify in the `Length` property.

Access to Time Scope through Handle Graphics API will be removed

Still runs

The underlying graphics of the Time Scope block will change in a future release. Once that happens, you will not be able to access or modify the Time Scope block through the handle graphics API.

Update any undocumented use of handle graphics APIs that access the Scope window.

SampleInput parameter will be removed

Still runs

The `SampleInput` parameter of the Time Scope block (currently accessed through the command line API) will be removed in a future release.

Functions being removed

iirparameq will be removed

Warns

Starting in R2022a, the `iirparameq` function will warn. Use `designParamEQ` instead. For details on how to replace your existing code, see the **Compatibility Considerations** section on the `iirparameq` function reference page.

midi functions require Audio Toolbox

Starting in R2022a, to use the following midi functions, you must have the Audio Toolbox™ installed.

- `midicallback`
- `midicontrols`
- `midiid`
- `midiread`
- `midisync`

Objects being removed

dsp.SpectrumAnalyzer object will be removed

Still runs

The `dsp.SpectrumAnalyzer` object will be removed in a future release. Use the new `spectrumAnalyzer` MATLAB object instead. The `spectrumAnalyzer` object has the same properties as the `dsp.SpectrumAnalyzer` object. However, the default value of the `Method` property has changed to `'filter-bank'`, and the default value of the `AveragingMethod` property has changed to `'vbw'`, video bandwidth.

No updates to your code are required except for:

- Replacing instances of `dsp.SpectrumAnalyzer` with `spectrumAnalyzer`.
- Updating the values of `Method` and `AveragingMethod` properties, if required.

For more details on how to update your existing code, see the **Compatibility Considerations** section on the `dsp.SpectrumAnalyzer` object page.

Support for dsp.SpectrumAnalyzer object will be removed from certain object functions

Still runs

The support for `dsp.SpectrumAnalyzer` object will be removed from the following object functions in a future release. Use the new `spectrumAnalyzer` MATLAB object instead in the place of the `dsp.SpectrumAnalyzer` object.

- `show`
- `hide`
- `isVisible`
- `generateScript`
- `getMeasurementsData`
- `getSpectralMaskStatus`
- `getSpectrumData`
- `isNewDataReady`

No updates to your code are required except for replacing instances of `dsp.SpectrumAnalyzer` with `spectrumAnalyzer`.

For more details on how to update your existing code, see the **Compatibility Considerations** section on the individual object function pages.

Certain fdesign objects will be removed

Warns

Starting in R2022a, these `fdesign` objects will warn. Use the equivalent replacements instead.

For details on how to replace your existing code, see the **Compatibility Considerations** section on the respective `fdesign` filter specification object reference page.

Object	Use This Instead
<code>fdesign.parmeq</code>	<code>designParamEQ</code>
<code>fdesign.octave</code>	<code>octaveFilter</code>
<code>fdesign.audioweighting</code>	<code>weightingFilter</code>
<code>fdesign.pulseshaping</code>	<code>rcosdesign, gaussdesign</code>

Certain mfilt objects have been removed

Errors

Starting in R2022a, these `mfilt` objects will error. Use the equivalent replacements instead.

For details on how to replace your existing code, see the **Compatibility Considerations** section on the respective `mfilt` object reference page.

Object	Use This Instead
<code>mfilt.linearinterp</code>	<code>dsp.FIRInterpolator(L,'Linear')</code>
<code>mfilt.cicinterp</code>	<code>dsp.CICInterpolator</code>
<code>mfilt.fftfirinterp</code>	<code>dsp.FIRInterpolator</code>
<code>mfilt.firinterp</code>	<code>dsp.FIRInterpolator</code>
<code>mfilt.holdinterp</code>	<code>dsp.FIRInterpolator(L,'ZOH')</code>
<code>mfilt.iirinterp</code>	<code>dsp.IIRHalfbandInterpolator</code>

Certain System objects have been removed

These System objects have been fully removed in R2022a. You will no longer be able to use these objects in your code. Use the equivalent replacements instead.

System object	Use This Instead
<code>dsp.Histogram</code>	<code>histcounts</code>
<code>dsp.Maximum</code>	<code>max, dsp.MovingMaximum</code>
<code>dsp.Minimum</code>	<code>min, dsp.MovingMinimum</code>
<code>dsp.Mean</code>	<code>mean, dsp.MovingAverage</code>

System object	Use This Instead
dsp.Median	median, dsp.MedianFilter
dsp.RMS	rms, dsp.MovingRMS
dsp.StandardDeviation	std, dsp.MovingStandardDeviation
dsp.Variance	var, dsp.MovingVariance
dsp.DCT	dct
dsp.IDCT	idct
dsp.Normalizer	normalize, vecnorm
dsp.ParametricEQFilter	designParamEQ, multibandParametricEQ
dsp.Buffer	dsp.AsyncBuffer
dsp.LPCToAutocorrelation	poly2ac
dsp.LPCToLSP	cos(poly2lsf)
dsp.LPCToRC	poly2rc
dsp.LSFToLPC	lsf2poly
dsp.RCToAutocorrelation	rc2ac
dsp.RCToLPC	rc2poly
dsp.BurgAREstimator	arburg
dsp.BurgSpectrumEstimator	pburg
dsp.AudioRecorder	audioDeviceReader
dsp.AudioPlayer	audioDeviceWriter
dsp.CepstralToLPC	No replacement
dsp.LPCToCepstral	No replacement
dsp.LSPToLPC	No replacement

HDL Optimized System objects moved to DSP HDL Toolbox

Starting in R2022a, these objects have moved to a new product, DSP HDL Toolbox™. These objects are no longer available with the DSP System Toolbox product. The objects forward to the new name and existing code continues to work.

Old name	New name	Notes
dsp.HDLFFT	dsphdl.FFT	The new object supports FFT length of 4.
dsp.HDLIFFT	dsphdl.IFFT	The new object supports FFT length of 4.

Old name	New name	Notes
dsp.HDLNCO	dsphdl.NCO	When you set the <code>Waveform</code> property to 'Complex exponential' or 'Sine and cosine', the new object implements a 1/8 sine wave lookup table for each of the sine and cosine parts of the waveform, and uses control logic to select and invert the values to generate both sine and cosine waveforms. This optimization means that dual output mode uses similar hardware resources compared to single output mode.
dsp.HDLComplexToMagnitudeAngle	dsphdl.ComplexToMagnitudeAngle	The new object provides the <code>ScalingMethod</code> property to configure the hardware implementation of the CORDIC gain. To use a multiplier, set this property to 'Multipliers'. To use shift-and-add logic, set this property to 'CSD'.
dsp.HDLFIRFilter	dsphdl.FIRFilter	<ul style="list-style-type: none"> • The new object supports high-throughput data. You can apply input data as a N-by-1 vector, where N can be up to 64 values. • The <code>NumberOfCycles</code> property is now named <code>NumCycles</code>. • When you use programmable coefficients with the new object, you must supply the coefficients as a row vector. The old object accepted either a row or a column vector of coefficients.

Old name	New name	Notes									
dsp.HDLCICDecimator	dsphdl.CICDecimator	<ul style="list-style-type: none"> <li data-bbox="1065 300 1453 426">A decimation factor of 1 was invalid on the old object. You can set the decimation factor to 1 on the new object. <li data-bbox="1065 436 1453 1890"> <table border="1"> <thead> <tr> <th data-bbox="1143 443 1243 541">Configuration</th> <th data-bbox="1243 443 1349 541">Before R2022a</th> <th data-bbox="1349 443 1453 541">After 2022a</th> </tr> </thead> <tbody> <tr> <td data-bbox="1143 548 1243 1346">Variable decimation factor</td> <td data-bbox="1243 548 1349 1346">Set the VariableDecimation property to true, and the DecimationFactor property to the maximum expected decimation factor.</td> <td data-bbox="1349 548 1453 1346">Set the DecimationSource property to 'Input', and set the MaxDecimationFactor property to the maximum expected decimation factor.</td> </tr> <tr> <td data-bbox="1143 1352 1243 1890">Fixed decimation factor</td> <td data-bbox="1243 1352 1349 1890">Set the VariableDecimation property to false, and the DecimationFactor property to the desired decima</td> <td data-bbox="1349 1352 1453 1890">Set the DecimationSource property to false, and the DecimationFactor property to the desired decima</td> </tr> </tbody> </table> 	Configuration	Before R2022a	After 2022a	Variable decimation factor	Set the VariableDecimation property to true, and the DecimationFactor property to the maximum expected decimation factor.	Set the DecimationSource property to 'Input', and set the MaxDecimationFactor property to the maximum expected decimation factor.	Fixed decimation factor	Set the VariableDecimation property to false, and the DecimationFactor property to the desired decima	Set the DecimationSource property to false, and the DecimationFactor property to the desired decima
Configuration	Before R2022a	After 2022a									
Variable decimation factor	Set the VariableDecimation property to true, and the DecimationFactor property to the maximum expected decimation factor.	Set the DecimationSource property to 'Input', and set the MaxDecimationFactor property to the maximum expected decimation factor.									
Fixed decimation factor	Set the VariableDecimation property to false, and the DecimationFactor property to the desired decima	Set the DecimationSource property to false, and the DecimationFactor property to the desired decima									

Old name	New name	Notes		
		Configuratio n	Before R2022 a	After 2022a
			tion factor.	tion factor.
		<ul style="list-style-type: none"> The ResetIn property is renamed to ResetInputPort. 		
dsp.HDLFIRDecimator	dsphdl.FIRDecimator	The new object supports vector sizes greater than decimation factor and a partly serial architecture.		
dsp.HDLFIRRateConverter	dsphdl.FIRRateConverter	The dsp.HDLFIRRateConverter System object provided an optional request argument. This argument is not available on the dsphdl.FIRRateConverter System object.		
dsp.HDLChannelizer	dsphdl.Channelizer	The new object supports FFT length of 4.		

Blocks being removed

Certain blocks moved to Simulink

Starting in R2022a, these blocks have moved from the **DSP System Toolbox > Math Functions > Matrices and Linear > Matrix Operations** to **Simulink > Matrix Operations**. These blocks are no longer available in the DSP System Toolbox library.

- Create Diagonal Matrix
- Extract Diagonal Matrix
- Permute Matrix
- Submatrix

All existing models continue to work.

In addition, starting in R2022a, the Identity Matrix block is no longer available in **DSP System Toolbox > Math Functions > Matrices and Linear > Matrix Operations**. Instead, use the Identity Matrix block in **Simulink > Matrix Operations**.

All existing models continue to work.

HDL Optimized blocks moved to DSP HDL Toolbox

Starting in R2022a, these blocks have moved from the **DSP System Toolbox HDL Support** library to the new **DSP HDL Toolbox** library. The blocks forward to the new name and library location and existing models continue to work.

Old name	New name	Notes
FFT HDL Optimized	FFT	The new block supports FFT length of 4.
IFFT HDL Optimized	IFFT	The new block supports FFT length of 4.
NCO HDL Optimized	NCO	When you set the Type of output signal parameter to Complex exponential or Sine and cosine, the new block implements a 1/8 sine wave lookup table for each of the sine and cosine parts of the waveform, and uses control logic to select and invert the values to generate both sine and cosine waveforms. This optimization means that dual output mode uses similar hardware resources compared to single output mode.
Complex To Magnitude-Angle HDL Optimized	Complex To Magnitude-Angle	The new block provides the Scaling method parameter to configure the hardware implementation of the CORDIC gain. To use a multiplier, set this parameter to Multipliers . To use shift-and-add logic, set this property to CSD .
Discrete FIR Filter HDL Optimized	Discrete FIR Filter	<ul style="list-style-type: none"> The new block supports high-throughput data. You can apply input data as a N-by-1 vector, where N can be up to 64 values. When you use programmable coefficients with the new block, you must supply the coefficients as a row vector. The old block accepted either a row or a column vector of coefficients.

Old name	New name	Notes									
CIC Decimation HDL Optimized	CIC Decimator	<ul style="list-style-type: none"> <li data-bbox="1063 298 1466 426">A decimation factor of 1 was invalid on the old block. You can set the decimation factor to 1 on the new block. <li data-bbox="1063 436 1466 1896"> <table border="1" data-bbox="1143 436 1458 1896"> <thead> <tr> <th data-bbox="1143 443 1247 541">Configuration</th> <th data-bbox="1247 443 1351 541">Before R2022a</th> <th data-bbox="1351 443 1458 541">After 2022a</th> </tr> </thead> <tbody> <tr> <td data-bbox="1143 548 1247 1570">Variable decimation factor</td> <td data-bbox="1247 548 1351 1570">Select the Variable decimation parameter and set the Decimation factor (R) parameter to the maximum expected decimation factor.</td> <td data-bbox="1351 548 1458 1570">Set the Decimation factor source parameter to Input port and set the Decimation factor (Rmax) parameter to the maximum expected decimation factor. The decim Factor port is renamed to R.</td> </tr> <tr> <td data-bbox="1143 1577 1247 1896">Fixed decimation factor</td> <td data-bbox="1247 1577 1351 1896">Clear the Variable decimation parameter and set the Decim</td> <td data-bbox="1351 1577 1458 1896">Set the Decimation factor source parameter to Property and set</td> </tr> </tbody> </table> 	Configuration	Before R2022a	After 2022a	Variable decimation factor	Select the Variable decimation parameter and set the Decimation factor (R) parameter to the maximum expected decimation factor.	Set the Decimation factor source parameter to Input port and set the Decimation factor (Rmax) parameter to the maximum expected decimation factor. The decim Factor port is renamed to R .	Fixed decimation factor	Clear the Variable decimation parameter and set the Decim	Set the Decimation factor source parameter to Property and set
Configuration	Before R2022a	After 2022a									
Variable decimation factor	Select the Variable decimation parameter and set the Decimation factor (R) parameter to the maximum expected decimation factor.	Set the Decimation factor source parameter to Input port and set the Decimation factor (Rmax) parameter to the maximum expected decimation factor. The decim Factor port is renamed to R .									
Fixed decimation factor	Clear the Variable decimation parameter and set the Decim	Set the Decimation factor source parameter to Property and set									

Old name	New name	Notes		
		Configuratio n	Before R2022 a	After 2022a
			ation factor (R) parame ter to the desired decima tion factor.	the Decim ation factor (R) to the desired decima tion factor.
FIR Decimation HDL Optimized	FIR Decimator	The new block supports vector sizes greater than decimation factor and partly serial architecture.		
FIR Sample Rate Conversion HDL Optimized	FIR Sample Rate Converter	The FIR Sample Rate Conversion HDL Optimized block provided an optional request port. This port is not available on the FIR Sample Rate Converter block. For an alternate way to control the data rate in your model, see "Control Data Rate Using Ready Signal" (DSP HDL Toolbox)		
Channelizer HDL Optimized	Channelizer	The new block supports FFT length of 4.		

R2021b

Version: 9.13

New Features

Bug Fixes

Version History

New default filter design for FIR rate conversion objects

Starting in R2021b, when you create a `dsp.FIRDecimator`, `dsp.FIRInterpolator`, and `dsp.FIRRateConverter` object, the object designs its filter coefficients based on the rate conversion factor that you specify while creating the object. The objects design the filter using the `designMultirateFIR` function.

As an example, create a `dsp.FIRDecimator` object and specify the decimation factor to be 3. The object designs the filter using `designMultirateFIR(1,3)`. You can access the designed filter through the `Numerator` property.

```
firD3 = dsp.FIRDecimator(3)

firD3 =

    dsp.FIRDecimator with properties:

        DecimationFactor: 3
        NumeratorSource: 'Property'
        Numerator: [1×72 double]
        Structure: 'Direct form'
```

Now create another `dsp.FIRDecimator` object with a decimation factor of 5. The object now designs the filter using `designMultirateFIR(1,5)`.

```
firD5 = dsp.FIRDecimator(5)

firD5 =

    dsp.FIRDecimator with properties:

        DecimationFactor: 5
        NumeratorSource: 'Property'
        Numerator: [1×120 double]
        Structure: 'Direct form'
```

If you want to revert the object to the design mode where the filter is not designed based on the rate conversion factor, pass the keyword `'legacy'` as an input while creating the object.

```
firD5L = dsp.FIRDecimator(5, 'legacy')

firD5L =

    dsp.FIRDecimator with properties:

        DecimationFactor: 5
        NumeratorSource: 'Property'
        Numerator: [1×36 double]
        Structure: 'Direct form'
```

New automatic design for FIR rate conversion objects

When you create a `dsp.FIRDecimator`, `dsp.FIRInterpolator`, and `dsp.FIRRateConverter` object, the object designs its filter coefficients based on the rate conversion factor that you specify while creating the object.

After this initial design, if you want to change the rate conversion factor to a different value and have the filter redesigned automatically based on this new value, set the filter object to operate in the automatic design mode. This mode enables the object to redesign the filter every time there is an update in the rate conversion factor. To set the filter object to operate in the automatic design mode, you can either pass the keyword 'auto' as an input while creating the object, or you can set the 'NumeratorSource' property of the object to 'Auto' explicitly.

```
firI = dsp.FIRInterpolator(3,'auto')  
  
firI =  
  
dsp.FIRInterpolator with properties:  
  
InterpolationFactor: 3  
NumeratorSource: 'Auto'  
DesignMethod: 'Kaiser'
```

When `dsp.FIRInterpolator` and `dsp.FIRRateConverter` objects are in the automatic design mode, you can also specify the underlying D/A signal interpolation model through the `DesignMethod` property. The values of the `DesignMethod` property can be either 'Kaiser', 'Linear', or 'ZOH'.

```
firI.DesignMethod = 'linear'  
  
firI =  
  
dsp.FIRInterpolator with properties:  
  
InterpolationFactor: 3  
NumeratorSource: 'Auto'  
DesignMethod: 'Linear'
```

To access the filter coefficients in the automatic design mode, type `filterobject.Numerator` in the MATLAB command prompt.

```
firI.Numerator  
  
ans =  
  
0.3333 0.6667 1.0000 0.6667 0.3333
```

Now change the interpolation factor to 2 and display the filter coefficients. The automatic design mode redesigns the filter based on the new interpolation factor.

```
firI.InterpolationFactor = 2  
  
firI =  
  
dsp.FIRInterpolator with properties:  
  
InterpolationFactor: 2  
NumeratorSource: 'Auto'  
DesignMethod: 'Linear'  
  
firI.Numerator  
  
ans =  
  
0.5000 1.0000 0.5000
```

Improved speed performance for certain DSP System Toolbox features

In R2021b, these features have an improved speed performance under specified conditions.

Feature	Conditions	Speedup mode
dsp.FIRFilter	Structure is set to 'Direct form' Input signal has a data type of single or double	You can notice a speedup when you: <ul style="list-style-type: none"> • Simulate the object algorithm in MATLAB • Run the MEX file that you generate using the codegen (MATLAB Coder) function • Run the MEX file that you generate using the dspunfold function
dsp.FIRInterpolator	Input signal has a data type of single or double	
dsp.Differentiator		
dsp.VariableBandwidthFIR Filter		
dsp.SubbandSynthesisFilter		
dsp.HighpassFilter	FilterType is set to 'FIR'	
dsp.LowpassFilter	Input signal has a data type of single or double	
Discrete FIR Filter (Simulink)	Filter structure is set to Direct form Input processing parameter is set to Columns as channels (frame based) Input signal has a data type of single or double	You can notice a speedup when the block: <ul style="list-style-type: none"> • Simulates in Normal mode • Simulates in Accelerator mode • Simulates in Rapid Accelerator mode • Belongs to a model reference (Simulink) and operates in Normal mode • Belongs to a model reference (Simulink) and operates in Accelerator mode • Belongs to a Dataflow Subsystem block
FIR Interpolation	Input processing parameter is set to Columns as channels (frame based) Input signal has a data type of single or double	
Two-Channel Synthesis Subband Filter		
Bandpass Filter	Impulse response is set to FIR	
Bandstop Filter	Filter type is set to Single-rate or Interpolator	
Arbitrary Response Filter		
Hilbert Filter		
Nyquist Filter	Structure is set to Direct-form FIR when Filter type is set to Single-rate Input processing is set to Columns as channels (frame based) Input signal has a data type of single or double	

Feature	Conditions	Speedup mode
Inverse Sinc Filter	<p>Filter type is set to Single-rate or Interpolator</p> <p>Structure is set to Direct-form FIR when Filter type is set to Single-rate</p> <p>Input processing is set to Columns as channels (frame based)</p> <p>Input signal has a data type of single or double</p>	
Digital Filter Design	<p>Design Method in Design filter panel panel is set to FIR</p> <p>Filter Structure in Import Filter from Workspace panel is set to Direct-Form FIR</p> <p>Input processing is set to Columns as channels (frame based)</p> <p>Input signal has a data type of single or double</p>	
Highpass Filter	Filter type is set to FIR	<p>You can notice a speedup when the block:</p> <ul style="list-style-type: none"> • Simulates in Normal mode • Simulates in Accelerator mode • Belongs to a model reference (Simulink) and operates in Normal mode
Lowpass Filter	Simulate using is set to Interpreted execution	
Differentiator Filter	Simulate using is set to Interpreted execution	
Variable Bandwidth FIR Filter		

Specify overlap length in moving statistics objects

You can now specify the length of overlap between sliding windows in the following objects:

- `dsp.MovingAverage`
- `dsp.MovingRMS`
- `dsp.MovingVariance`
- `dsp.MovingStandardDeviation`
- `powermeter`

To specify the overlap length, you can either pass the scalar value of the overlap length as a second value argument while creating the object or explicitly set the `OverlapLength` property in the object.

In order to specify the overlap length, the objects must compute the statistic using the sliding window method.

```
mvgAvg = dsp.MovingAverage(4,1)
```

```
mvgAvg =
```

```
    dsp.MovingAverage with properties:
```

```
        Method: 'Sliding window'  
SpecifyWindowLength: true  
        WindowLength: 4  
        OverlapLength: 1
```

Arbitrary frame size support for `dsp.FarrowRateConverter`

The `dsp.FarrowRateConverter` object can now accept inputs of any frame size. You can specify any frame size and are no longer limited to specify frame sizes that are multiples of the decimation factor.

SIMD Code Generation: New code replacement library (CRL) customized for DSP System Toolbox features

You can now further customize the SIMD code that you generate from DSP System Toolbox algorithms by using the new DSP Intel AVX2-FMA (Windows), DSP Intel AVX2-FMA (Linux), DSP Intel AVX2-FMA (Mac) code replacement libraries. These libraries are available if you have the DSP System Toolbox, MATLAB Coder™, Simulink Coder (for Simulink models), and Embedded Coder® installed.

For more details on the process of using these libraries, and the features that support SIMD code generation, see [SIMD Code Generation](#).

Complex support for SOS filter coefficients

The numerator and denominator coefficients of the `dsp.SOSFilter` object now accept complex values. The numerator coefficients are specified through the `Numerator` property and the denominator coefficients are specified through the `Denominator` property.

Tunable Filename property for `dsp.AudioFileWriter` and `dsp.AudioFileReader` objects in generated code

Starting in R2021b, the `Filename` property of the `dsp.AudioFileReader` and `dsp.AudioFileWriter` objects is tunable. With this change, you can pass the name of the audio file as an input parameter while running the code generated from these objects.

To enable the file name tunability in the `dsp.AudioFileReader` object, set the `FilenameIsTunableInCodegen` property to `true`. File attributes such as the audio format, the number of audio channels, the sample rate, and the bit rate are not tunable, and must match the attributes of the prototype audio file specified through the `CodegenPrototypeFile` property. The specified prototype audio file determines the attributes and the type of audio files that can be read by the generated code.

For an example, see Tunable Audio File Name in Generated Code.

POSIX threads (Pthreads) and OpenMP threading support for multicore custom targets using dataflow domain

Starting in R2021b, the new `DataflowThreadingImplementation` parameter allows you to select POSIX threads (Pthreads) or OpenMP threading implementation for multicore custom targets using dataflow domain.

Change the setting using this command, which takes the values `off`, `OpenMP`, or `Pthreads`.

```
set_param(myModel, 'DataflowThreadingImplementation', 'off')
```

Enhancements for Multicore tab analysis results

There are three enhancements for **Multicore** tab analysis results:

- The visualizations for relative weight and speed up are improved.
- The analysis report displays total cost and number of threads values for each Dataflow Subsystem block.
- When you perform SIL/PIL profiling, the results show the units of the profiled cost values.

For more information, see [Perform Multicore Analysis for Dataflow and Multicore Analysis Using a Dataflow Domain](#).

dspunfold does not support Xcode 12.0 or later

Starting in R2021b, if you have a macOS with an Xcode version 12.0 or later, you cannot use the `dspunfold` function on that machine to generate multi-threaded MEX files from your MATLAB algorithms.

Spectrum Analyzer CCDF measurement reference line changed

In the Spectrum Analyzer block and `dsp.SpectrumAnalyzer` object, the Gaussian reference line for the complimentary cumulative distribution function (CCDF) measurement has changed. Previously, the reference line represented the CCDF of the power of a real white Gaussian noise. Starting in R2021b, the reference line represents the CCDF of the power of a complex white Gaussian noise, calculated using a chi-squared distribution.

dsp.ArrayPlot supports multiple frame sizes

The `dsp.ArrayPlot` now supports inputs with different frame lengths.

Functionality being removed

'Raised Cosine' and 'Square Root Raised Cosine' response methods will be removed

Still runs

The 'Raised Cosine' and 'Square Root Raised Cosine' response methods in `fdesign.decimator` and `fdesign.interpolator` objects will be removed in a future release. Use

rcosdesign, comm.RaisedCosineTransmitFilter (Communications Toolbox), and comm.RaisedCosineReceiveFilter (Communications Toolbox) instead.

Objects being removed

Certain System objects will be removed

Warns

Starting in R2021b, these objects will warn. Use the equivalent replacements instead.

For details on how to replace your existing code, see the **Compatibility Considerations** section on the respective System object reference page.

System object	Use This Instead
dsp.ArrayVectorAdder	Use + with array expansion
dsp.ArrayVectorSubtractor	Use – with array expansion
dsp.ArrayVectorMultiplier	Use .* with array expansion
dsp.ArrayVectorDivider	Use ./ with array expansion
dsp.CumulativeProduct	cumprod
dsp.CumulativeSum	cumsum
dsp.Interpolator	dsp.FIRInterpolator, interp1
dsp.Convolver	conv
dsp.Autocorrelator	xcorr
dsp.Crosscorrelator	xcorr
dsp.UniformDecoder	udecode
dsp.UniformEncoder	uencode
dsp.LDLFactor	ldl
dsp.LUFactor	lu
dsp.LevinsonSolver	levinson
dsp.LowerTriangularSolver	mldivide, \ operator
dsp.UpperTriangularSolver	mldivide, \ operator
dsp.Counter	Create a variable in MATLAB and increment the variable by 1
dsp.DelayLine	No direct replacement. dsp.AsyncBuffer object can be used to achieve a delay line
dsp.Window	window
dsp.KalmanFilter	Use Kalman filter functionality in Sensor Fusion and Tracking Toolbox™
dsp.PeakToPeak	peak2peak

System object	Use This Instead
<code>dsp.PulseMetrics</code>	Use the Pulse and Transition Metrics functions. You can use functions like <code>dutycycle</code> , <code>midcross</code> , <code>pulseperiod</code> , <code>pulsesep</code> , and <code>pulsewidth</code> among others
<code>dsp.StateLevels</code>	<code>statelevels</code>
<code>dsp.TransitionMetrics</code>	Use the Pulse and Transition Metrics functions. You can use functions like <code>falltime</code> , <code>overshoot</code> , <code>risetime</code> , <code>settlingtime</code> , <code>slewrates</code> , and <code>undershoot</code> among others
<code>dsp.ScalarQuantizerDecoder</code>	No replacement
<code>dsp.ScalarQuantizerEncoder</code>	No replacement
<code>dsp.VectorQuantizerDecoder</code>	No replacement
<code>dsp.VectorQuantizerEncoder</code>	No replacement

dsp.PeakFinder object has been removed

Errors

The `dsp.PeakFinder` object has been removed in R2021b. To determine the local maxima, use the `findpeaks` function. To determine the local minima, use the `islocalmin` function and find the signal values corresponding to the local minima indices that the function determines. For more details, see the Compatibility Considerations section on the object reference page.

Blocks being removed

These blocks have been removed in R2021b.

- Scalar Quantizer Design
- Vector Quantizer Design

Version History

Existing models using these blocks continue to run in R2021b. For new models created in R2021b, there is no replacement.

R2021a

Version: 9.12

New Features

Bug Fixes

Version History

Multicore tab for Dataflow: Analyze and configure multicore execution for Simulink models using Dataflow

When a subsystem in a model is configured to use a dataflow execution domain, the **Multicore** tab is activated on the Simulink toolstrip. The **Multicore** tab consolidates multicore analysis techniques leveraged in dataflow into an incremental and iterative workflow. For more information, see [Perform Multicore Analysis for Dataflow](#).



Controls on the **Multicore** tab allow you to:

- Estimate the relative cost of blocks using internal Simulink heuristics.
- Measure average execution times (cost) of blocks inside the dataflow subsystems by simulating the model with SIL/PIL. This functionality requires an Embedded Coder license.
- Manually override the block cost values.
- Provide analysis constraints, such as maximum number of threads and threading threshold.
- Run analysis to generate block-to-threads allocation and visualize analysis results.

For an example, see [Multicore Analysis Using a Dataflow Domain](#).

Rationally oversampled channelizers

Design a rationally oversampled channelizer by specifying the decimation factor D such that it is not an integer multiple of the number of frequency bands M . The oversampling ratio M/D in this case is a rational number.

You can now directly specify the decimation factor in the `dsp.Channelizer` object and the Channelizer block using the `DecimationFactor` property and the **Decimation Factor** parameter, respectively. The number of frequency bands is specified using the `NumFrequencyBands` property in the object and the **Number of frequency bands** parameter in the block.

In-Place Memory Optimization: Optimize the memory usage in the generated code for certain DSP System Toolbox features

The Discrete FIR Filter (Simulink) block and the `dsp.FIRFilter` object support in-place memory optimization. Due to in-place optimization, the generated code uses a single buffer to store the input and output data values. Every time there is a new intermediary output, this output buffer is overwritten to store that value. For more details, see [In-Place Memory Optimization](#).

Fractional delay FIR filter design

Design a fractional delay FIR filter using the `designFracDelayFIR` function. The function provides a Kaiser-based FIR approximation of a bandlimited ideal shift filter with a fractional (noninteger) delay value between $[0,1]$.

Power Meter: Measure power of voltage signal in MATLAB and Simulink

Compute the average power, peak power, and peak-to-average power ratio of a voltage signal in MATLAB and Simulink using the `powermeter` System object and the Power Meter block, respectively. While calculating the power, the object and the block account for the reference load.

SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox features

In R2021a, these objects support SIMD code generation using Intel AVX2 technology under the specified conditions.

MATLAB System objects	Conditions
<code>dsp.AnalyticSignal</code>	<ul style="list-style-type: none"> Input signal is real valued. Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.ComplexBandpassDecimator</code>	<ul style="list-style-type: none"> Input signal is complex valued. Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.DCBlocker</code>	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.Differentiator</code>	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.DigitalDownConverter</code>	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.DigitalUpConverter</code>	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.FIRFilter</code>	<ul style="list-style-type: none"> Input signal is complex valued. Structure is set to <code>'Direct form transposed'</code>. Coefficients can be real or complex valued <p>For all other conditions under which the <code>dsp.FIRFilter</code> object generates SIMD code, see the Extended Capabilities > C/C++ Code Generation section on the <code>dsp.FIRFilter</code> reference page.</p>
<code>dsp.FIRHalfbandInterpolator</code>	<ul style="list-style-type: none"> Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.HighpassFilter</code>	<ul style="list-style-type: none"> <code>FilterType</code> is set to <code>'FIR'</code>. Input signal has a data type of <code>single</code> or <code>double</code>.

MATLAB System objects	Conditions
<code>dsp.LowpassFilter</code>	<ul style="list-style-type: none"> • <code>FilterType</code> is set to 'FIR'. • Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.SampleRateConverter</code>	<ul style="list-style-type: none"> • For upsampling, the ratio of output sample rate to input sample rate must be an integer. • For downsampling, the ratio of input sample rate to output sample rate must be an integer. • Input signal has a data type of <code>single</code> or <code>double</code>.
<code>dsp.VariableBandwidthFIRFilter</code>	<ul style="list-style-type: none"> • Input signal has a data type of <code>single</code> or <code>double</code>.

For a full list, see System objects in DSP System Toolbox that Support SIMD Code Generation.

These blocks support SIMD code generation using Intel AVX2 technology under the specified conditions.

Simulink blocks	Conditions
Arbitrary Response Filter	<ul style="list-style-type: none"> • Filter type is set to <code>Single-rate</code>, <code>Decimator</code>, or <code>Interpolator</code>. • For Filter type set to <code>Single-rate</code>, Structure is set to <code>Direct-form FIR</code> or <code>Direct-form FIR transposed</code>. • For Filter type set to <code>Decimator</code>, Structure is set to <code>Direct-form FIR polyphase decimator</code> and Rate options is set to <code>Enforce single-rate processing</code>. • For Filter type set to <code>Interpolator</code>, Rate options is set to <code>Enforce single-rate processing</code>. • Input processing is set to <code>Columns as channels (frame based)</code>. • Input signal has a data type of <code>single</code> or <code>double</code>.
Analytic Signal	<ul style="list-style-type: none"> • Input processing is set to <code>Columns as channels (frame based)</code>. • Input signal has to be real valued. • Input signal has a data type of <code>single</code> or <code>double</code>.

Simulink blocks	Conditions
Bandpass Filter	<ul style="list-style-type: none"> • Impulse response is set to FIR. • Filter type is set to Single-rate. • Structure is set to Direct-form FIR or Direct-form FIR transposed. • Use basic elements to enable filter customization parameter is not selected. • Input processing is set to Columns as channels (frame based). • Input signal has a data type of single or double.
Bandstop Filter	<ul style="list-style-type: none"> • Impulse response is set to FIR. • Filter type is set to Single-rate. • Structure is set to Direct-form FIR or Direct-form FIR transposed. • Use basic elements to enable filter customization parameter is not selected. • Input processing is set to Columns as channels (frame based). • Input signal has a data type of single or double.
Complex Bandpass Decimator	<ul style="list-style-type: none"> • Input signal is complex valued. • Input signal has a data type of single or double.
DC Blocker	<ul style="list-style-type: none"> • Input signal has a data type of single or double.
Differentiator Filter	<ul style="list-style-type: none"> • Input signal has a data type of single or double.
Digital Filter Design	<ul style="list-style-type: none"> • Input processing is set to Columns as channels (frame based). • Filter Structure (in Import Filter from Workspace pane) is set to Direct-Form FIR. You can generate SIMD code even when the filter is a Direct-Form FIR Transposed filter. To create a Direct-Form FIR Transposed filter, select Edit > Convert Structure, and click Direct-Form FIR Transposed. • Input signal has a data type of single or double.

Simulink blocks	Conditions
Discrete FIR Filter (Simulink)	<ul style="list-style-type: none"> • Input signal is complex valued. • Filter structure is set to Direct form transposed. • Coefficients can be real or complex valued <p>For all other conditions under which the Discrete FIR Filter block generates SIMD code, see the Extended Capabilities > C/C++ Code Generation section on the Discrete FIR Filter (Simulink) block reference page.</p>
FIR Halfband Interpolator	<ul style="list-style-type: none"> • Input signal has a data type of single or double.
Highpass Filter	<ul style="list-style-type: none"> • Filter type is set to FIR. • Input signal has a data type of single or double.
Hilbert Filter	<ul style="list-style-type: none"> • Filter type is set to Single-rate, Decimator, or Interpolator. • For Filter type set to Single-rate, Structure is set to Direct-form FIR or Direct-form FIR transposed. • For Filter type set to Decimator, Structure is set to Direct-form FIR polyphase decimator and Rate options is set to Enforce single-rate processing. • For Filter type set to Interpolator: <ul style="list-style-type: none"> • Interpolation Factor cannot be equal to 1. • Rate options is set to Enforce single-rate processing. • Input processing is set to Columns as channels (frame based). • Input signal has a data type of single or double. • Input port dimensions cannot be equal to [1 1].

Simulink blocks	Conditions
Inverse Sinc Filter	<ul style="list-style-type: none"> • Filter type is set to Single-rate, Decimator, or Interpolator. • For Filter type set to Single-rate, Structure is set to Direct-form FIR or Direct-form FIR transposed. • For Filter type set to Decimator, Structure is set to Direct-form FIR polyphase decimator and Rate options is set to Enforce single-rate processing. • For Filter type set to Interpolator, Rate options is set to Enforce single-rate processing. • Input processing is set to Columns as channels (frame based). • Input signal has a data type of single or double.
Lowpass Filter	<ul style="list-style-type: none"> • Filter type is set to FIR. • Input signal has a data type of single or double.
Nyquist Filter	<ul style="list-style-type: none"> • Filter type is set to Single-rate, Decimator, or Interpolator. • For Filter type set to Single-rate, Structure is set to Direct-form FIR or Direct-form FIR transposed. • For Filter type set to Decimator, Structure is set to Direct-form FIR polyphase decimator and Rate options is set to Enforce single-rate processing. • For Filter type set to Interpolator: <ul style="list-style-type: none"> • Interpolation Factor cannot be equal to 1. • Rate options is set to Enforce single-rate processing. • Input processing is set to Columns as channels (frame based). • Input signal has a data type of single or double. • Input port dimensions cannot be equal to [1 1].

Simulink blocks	Conditions
Sample-Rate Converter	<ul style="list-style-type: none"> • For upsampling, the ratio of output sample rate to input sample rate must be an integer. • For downsampling, the ratio of input sample rate to output sample rate must be an integer. • Input signal has a data type of <code>single</code> or <code>double</code>.
Variable Bandwidth FIR Filter	<ul style="list-style-type: none"> • Input signal has a data type of <code>single</code> or <code>double</code>.

For a full list, see Simulink Blocks in DSP System Toolbox that Support SIMD Code Generation.

The SIMD technology significantly improves the performance of the generated code.

Improved filter response visualization for certain DSP System Toolbox blocks

When you click the **View Filter Response** button in the dialog box of the Variable Bandwidth FIR Filter, Variable Bandwidth IIR Filter, and the Notch-Peak Filter blocks, the dynamic filter visualizer launches and shows the magnitude response of the designed filter. The response is based on the block dialog box parameters. To update the magnitude response while the dynamic filter visualizer is running, modify the dialog box parameters and click **Apply**.

Using the dynamic filter visualizer, you can configure the plot settings, measure signal statistics, find peak values, place data cursors, and so on from the interface of the visualizer. For more details on the dynamic filter visualizer interface and the tools that are available, see `dsp.DynamicFilterVisualizer`.

Improved Speed Performance in Accelerator Mode for specific blocks in DSP System Toolbox

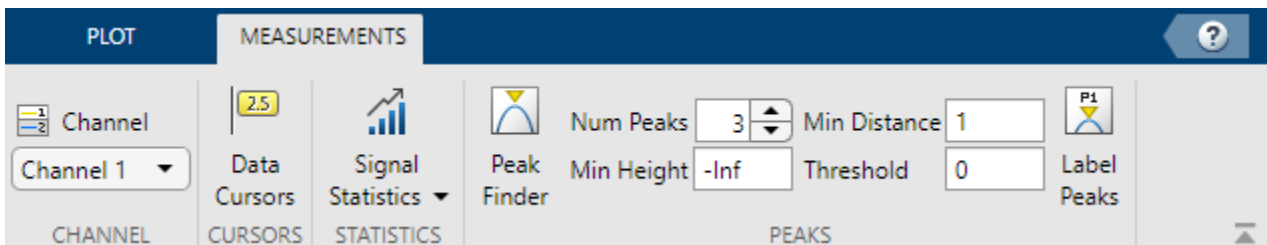
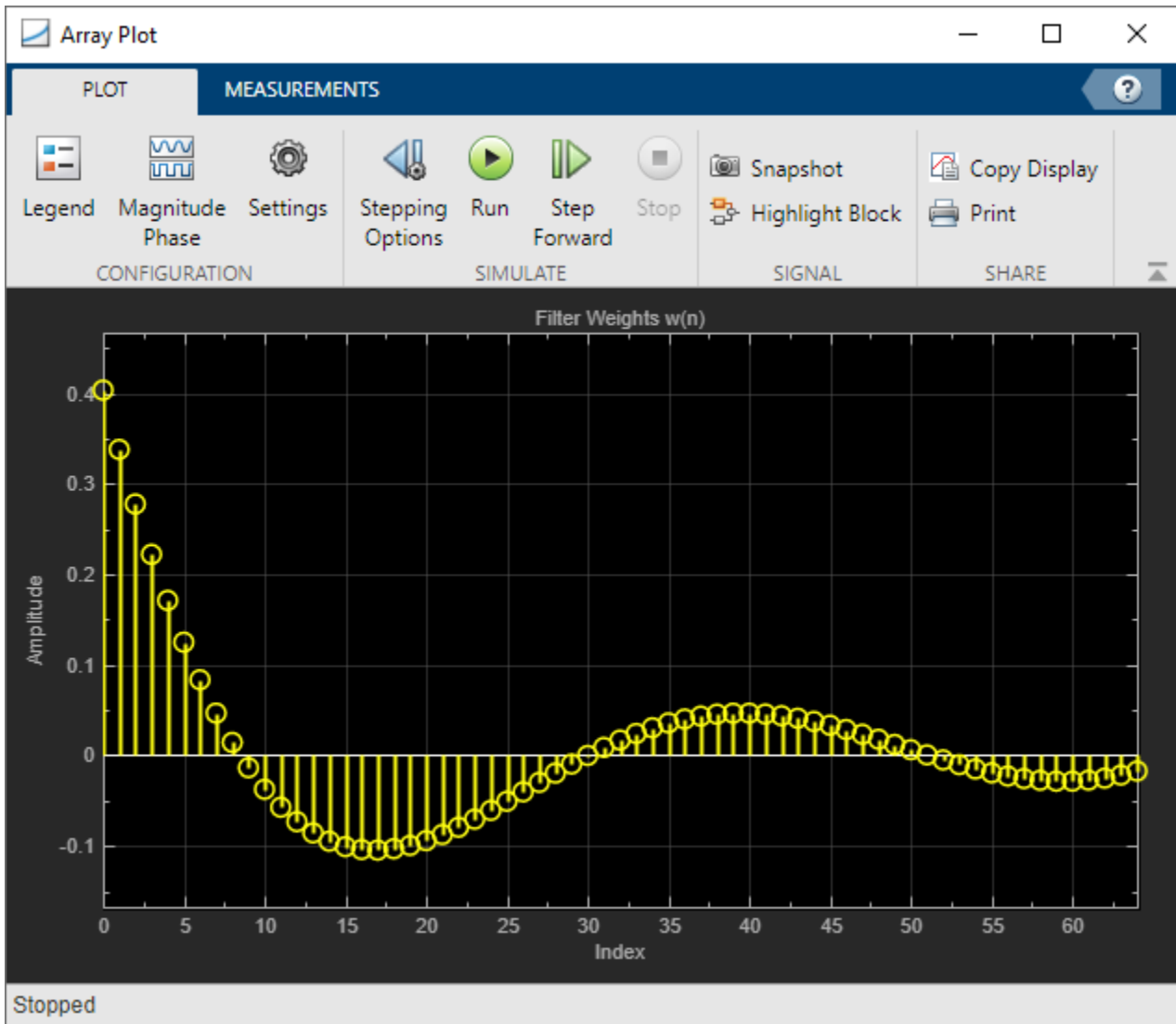
When you run these blocks in the accelerator mode, the blocks now run faster.

- FIR Decimation
- FIR Interpolation
- LMS Filter

To run the model containing these blocks in the accelerator mode, on the **Simulation** tab of the model, in the **Simulate** section, select **Accelerator** from the drop-down list. Build an executable for the model by clicking **Run**. The acceleration (Simulink) mode uses this executable in simulations as long as the model remains structurally unchanged. For more details, see [Perform Acceleration \(Simulink\)](#).

Improved display for Array Plot block

In Simulink, the Array Plot block has a new and improved interface with two toolstrip tabs.



From the toolbar, you can run the model, modify the settings, turn on measurements, and share an image of the plot.

For more information, see Configure Array Plot.

Variable-sized input support for timescope object

The timescope object allows you to visualize scalar or variable-sized input signals. If the signal is variable sized, the number of channels (columns) cannot change.

One-Based index support for Peak Finder block

The Peak Finder block can now output one-based index values for input peaks when you set the **Index base** parameter to **One**. With this setting, if the input vector is [-1.5, 0.5, 0], the peak value is 0.5 and the index of the peak value is 2. The default value for the **Index base** parameter is **Zero**. In this setting, for an input vector of [-1.5, 0.5, 0], the peak value is 0.5 and the index of the peak value is 1.

Version History

The default index base for the block continues to be zero-based. Existing models created in a previous release continue to work as expected. If a model created in R2021a has **Index base** set to **One**, and you export the model to a previous release, then the **Index base** parameter changes to **Zero** in the exported model.

Removal of the oversampling ratio functionality

The **Oversampling ratio** parameter in the Channelizer block has been removed. The **OversamplingRatio** property in the `dsp.Channelizer` object will be removed in a future release. You can now set the oversampling ratio M/D by specifying the number of frequency bands M and the decimation factor D .

Version History

Existing code using the **OversamplingRatio** property and existing models using the **Oversampling ratio** parameter continue to work.

If you create a new model containing the Channelizer block in R2021a, and if the **Decimation factor** parameter value in the block equals the value of the **Number of frequency bands** parameter, then the model containing the block can be exported to a release prior to R2021a. The exported model will have the **Oversampling ratio** parameter set to 1.

If the **Decimation factor** parameter value is less than and is a divisor of the value of the **Number of frequency bands** parameter, and the model is exported to a prior release that is not older than R2020a, the Channelizer will continue to work and the decimation factor will be replaced with the appropriate oversampling ratio. For a release older than R2020a, the block will be replaced with any empty subsystem.

If the specified **Decimation factor** parameter is not equal to and not a divisor of the **Number of frequency bands** parameter, the block will be replaced with an empty subsystem.

Digital down-converter (DDC) and digital up-converter (DUC) examples for FPGA (requires HDL Coder license for code generation)

The HDL Implementation of a Digital Down-Converter for LTE and HDL Implementation of a Digital Up-Converter for LTE examples show how to design filter chains for communications systems. The

examples show how to model the behavioral algorithm in MATLAB and then compare that reference result against a Simulink model that uses HDL-optimized blocks to implement the algorithm for HDL code generation.

Objects being removed

Certain System objects will be removed

Still runs

These objects will be removed in a future release. Use the equivalent replacements instead.

For details on how to replace your existing code, see the **Compatibility Considerations** section in the respective System object reference page.

System object	Use This Instead
<code>dsp.ArrayVectorAdder</code>	Use <code>+</code> with array expansion.
<code>dsp.ArrayVectorSubtractor</code>	Use <code>-</code> with array expansion.
<code>dsp.ArrayVectorMultiplier</code>	Use <code>.*</code> with array expansion.
<code>dsp.ArrayVectorDivider</code>	Use <code>./</code> with array expansion.
<code>dsp.CumulativeProduct</code>	<code>cumprod</code>
<code>dsp.CumulativeSum</code>	<code>cumsum</code>
<code>dsp.Interpolator</code>	<code>dsp.FIRInterpolator</code> , <code>interp1</code>
<code>dsp.Convolver</code>	<code>conv</code>
<code>dsp.Autocorrelator</code>	<code>xcorr</code>
<code>dsp.Crosscorrelator</code>	<code>xcorr</code>
<code>dsp.UniformDecoder</code>	<code>udecode</code>
<code>dsp.UniformEncoder</code>	<code>uencode</code>
<code>dsp.LDLFactor</code>	<code>ldl</code>
<code>dsp.LUFactor</code>	<code>lu</code>
<code>dsp.LevinsonSolver</code>	<code>levinson</code>
<code>dsp.LowerTriangularSolver</code>	<code>mldivide</code> , <code>\</code> operator
<code>dsp.UpperTriangularSolver</code>	<code>mldivide</code> , <code>\</code> operator
<code>dsp.Counter</code>	Create a variable in MATLAB and increment the variable by 1.
<code>dsp.DelayLine</code>	No direct replacement. <code>dsp.AsyncBuffer</code> object can be used to achieve a delay line.
<code>dsp.Window</code>	<code>window</code>
<code>dsp.KalmanFilter</code>	Use Kalman filter functionality in Sensor Fusion and Tracking Toolbox.
<code>dsp.PeakToPeak</code>	<code>peak2peak</code>

System object	Use This Instead
dsp.PulseMetrics	Use the Pulse and Transition Metrics functions. You can use functions like <code>dutycycle</code> , <code>midcross</code> , <code>pulseperiod</code> , <code>pulsesep</code> , and <code>pulsewidth</code> among others.
dsp.StateLevels	<code>statelevels</code>
dsp.TransitionMetrics	Use the Pulse and Transition Metrics functions. You can use functions like <code>falltime</code> , <code>overshoot</code> , <code>risetime</code> , <code>settlingtime</code> , <code>slewrates</code> , and <code>undershoot</code> among others.
dsp.ScalarQuantizerDecoder	No replacement
dsp.ScalarQuantizerEncoder	No replacement
dsp.VectorQuantizerDecoder	No replacement
dsp.VectorQuantizerEncoder	No replacement

Certain System objects have been removed

Errors

These objects have been removed in R2021a. Use the equivalent replacements instead.

For details on how to replace your existing code, see the **Compatibility Considerations** section in the respective System object reference page.

System object	Use This Instead
dsp.Histogram	<code>histcounts</code>
dsp.Maximum	<code>max</code> , <code>dsp.MovingMaximum</code>
dsp.Minimum	<code>min</code> , <code>dsp.MovingMinimum</code>
dsp.Mean	<code>mean</code> , <code>dsp.MovingAverage</code>
dsp.Median	<code>median</code> , <code>dsp.MedianFilter</code>
dsp.RMS	<code>rms</code> , <code>dsp.MovingRMS</code>
dsp.StandardDeviation	<code>std</code> , <code>dsp.MovingStandardDeviation</code>
dsp.Variance	<code>var</code> , <code>dsp.MovingVariance</code>
dsp.DCT	<code>dct</code>
dsp.IDCT	<code>idct</code>
dsp.Normalizer	<code>normalize</code> , <code>vecnorm</code>
dsp.ParametricEQFilter	<code>designParamEQ</code> function, <code>MultibandParametricEQ</code> object from Audio Toolbox
dsp.Buffer	<code>dsp.AsyncBuffer</code>
dsp.LPCToAutocorrelation	<code>poly2ac</code>
dsp.LPCToLSP	<code>cos(poly2lsf)</code>
dsp.LPCToRC	<code>poly2rc</code>
dsp.LSFToLPC	<code>lsf2poly</code>

System object	Use This Instead
<code>dsp.RCToAutocorrelation</code>	<code>rc2ac</code>
<code>dsp.RCToLPC</code>	<code>rc2poly</code>
<code>dsp.BurgAREstimator</code>	<code>arburg</code>
<code>dsp.BurgSpectrumEstimator</code>	<code>pburg</code>
<code>dsp.CepstralToLPC</code>	No replacement
<code>dsp.LPCToCepstral</code>	No replacement
<code>dsp.LSPToLPC</code>	No replacement

dsp.TimeScope will be removed

Warns

`dsp.TimeScope` will be removed in a future release. Use `timescope` instead. The `timescope` object has the same properties as the `dsp.TimeScope` System object. In your code, replace instances of `dsp.TimeScope` with `timescope`.

Starting in R2021a, `dsp.TimeScope` uses the new `timescope` interface.

Blocks being removed

Certain blocks will be removed

Still runs

These blocks will be removed in a future release.

- Scalar Quantizer Design
- Vector Quantizer Design

Blocks that have been removed

These blocks have been removed in R2021a. Use the equivalent replacements instead.

Simulink Blocks	Use This Instead
Wavelet Analysis	DWT
Wavelet Synthesis	IDWT

R2020b

Version: 9.11

New Features

Bug Fixes

Version History

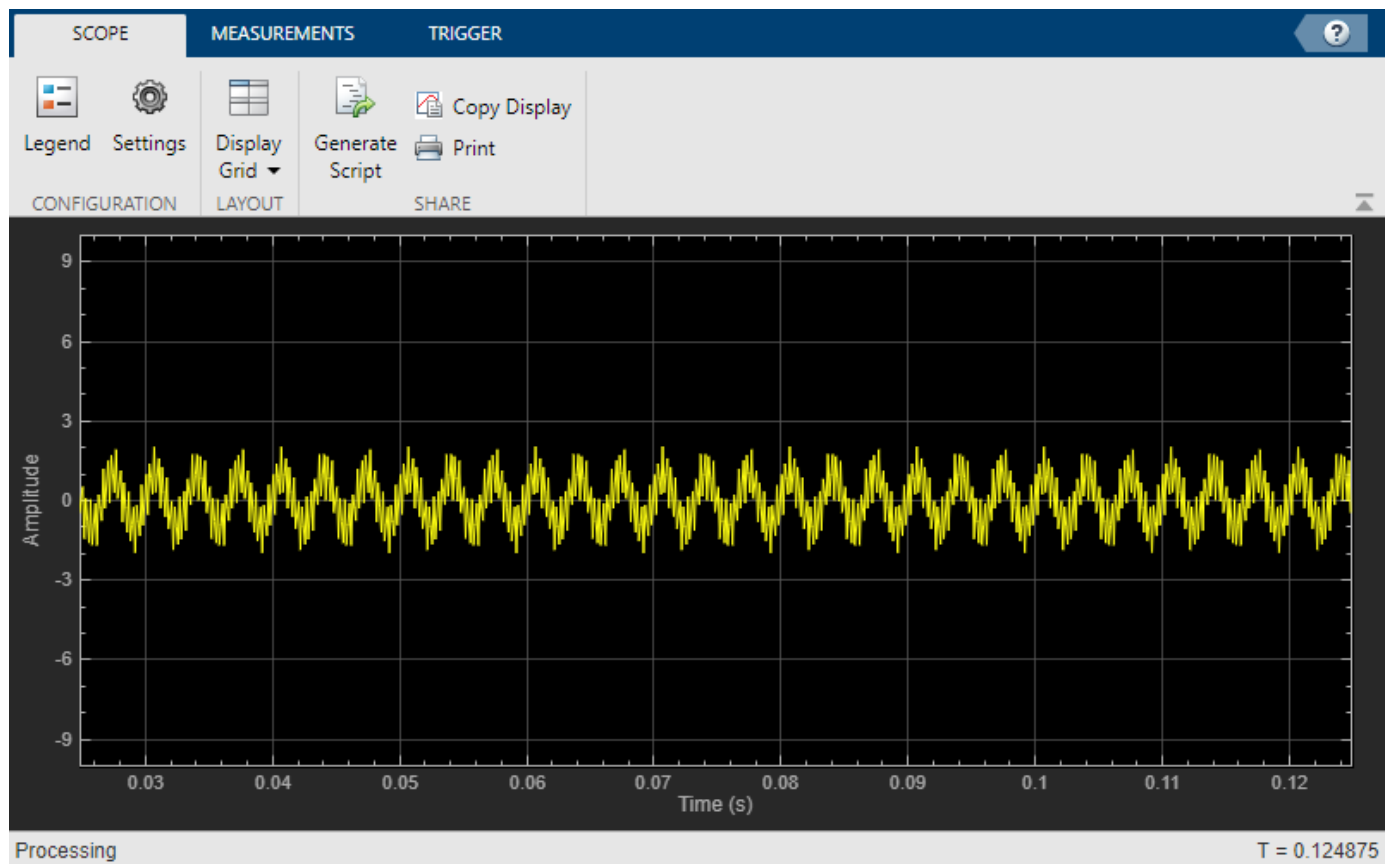
New time scope object: Visualize signals in the time domain

Use the `timescope` object to visualize real- and complex-valued floating-point and fixed-point signals in the time domain.

The Time Scope window has two toolstrip tabs:

Scopes Tab

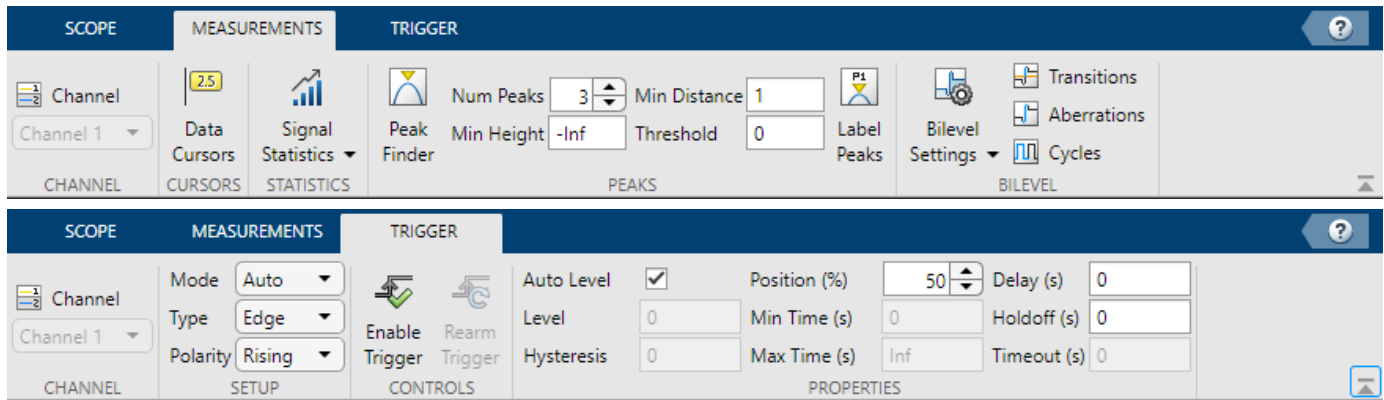
On the **Scopes** tab, you can control the layout and configuration settings, and set the display settings of the Time Scope. You can also generate script to recreate your time scope with the same settings. When doing so, an editor window opens with the code required to recreate your `timescope` object.



Measurements Tab

In the **Measurements** tab, all measurements are made for a specified channel.

- **Data Cursors** — Display the screen cursors.
- **Signal Statistics** — Display various statistics of the selected signal, such as maximum or minimum values, peak-to-peak values, mean, median, and RMS.
- **Peak Finder** — Display peak values for the selected signal.
- **Bilevel** — Measure transitions, overshoots, undershoots, and cycles.
- **Triggers** — Set triggers to sync repeating signals and pause the display when events occur.



SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox features

The following features support SIMD code generation using Intel AVX2 technology under these conditions:

dsp.FIRFilter System object

- Filter structure is set to 'Direct form' or 'Direct form transposed'.
- Input signal is real-valued with real filter coefficients.
- When filter structure is set to 'Direct form', the input signal can also be complex-valued with real or complex filter coefficients.
- Input signal has a data type of single or double.

For more details, see `dsp.FIRFilter` System object.

dsp.FIRDecimator System object

- Filter structure is set to 'Direct form'.
- Input signal is real-valued with real filter coefficients.
- Input signal is complex-valued with real or complex filter coefficients.
- Input signal has a data type of single or double.

For more details, see `dsp.FIRDecimator` System object.

dsp.FIRInterpolator System object

- Input signal is real-valued with real filter coefficients.
- Input signal is complex-valued with real or complex filter coefficients.
- Input signal has a data type of single or double.

For more details, see `dsp.FIRInterpolator` System object.

dsp.LMSFilter System object

- Method is set to 'LMS' or 'Normalized LMS'.

- `WeightsOutput` is set to 'None' or 'Last'.
- Input signal is real-valued.
- Input signal has a data type of `single` or `double`.

For more details, see `dsp.LMSFilter` System object.

FIR Interpolation block

- **Input processing** is set to `Columns as channels (frame based)`.
- **Rate options** is set to `Enforce single-rate processing`.
- Input signal is real-valued with real filter coefficients.
- Input signal is complex-valued with real or complex filter coefficients.
- Input signal has a data type of `single` or `double`.

For more details, see FIR Interpolation block.

FIR Decimation block

- **Filter structure** is set to `Direct form`.
- **Input processing** is set to `Columns as channels (frame based)`.
- **Rate options** is set to `Enforce single-rate processing`.
- Input signal is real-valued with real filter coefficients.
- Input signal is complex-valued with real or complex filter coefficients.
- Input signal has a data type of `single` or `double`.

For more details, see FIR Decimation block.

Discrete FIR Filter block

- **Filter structure** is set to `Direct form` or `Direct form transposed`.
- **Input processing** is set to `Columns as channels (frame based)`.
- Input signal is real-valued with real filter coefficients.
- When **Filter structure** is set to `Direct form`, the input signal can also be complex-valued with real or complex filter coefficients.
- Input signal has a data type of `single` or `double`.

For more details, see Discrete FIR Filter block.

The SIMD technology significantly improves the performance of the generated code.

One-sided short-time Fourier transform in `dsp.STFT` and `dsp.ISTFT` objects

When you set the `FrequencyRange` property to 'onesided', the `dsp.STFT` object computes the one-sided short-time Fourier transform of the real input signal. Correspondingly, the one-sided inverse short-time FFT is computed when the `FrequencyRange` property of the `dsp.ISTFT` object is set to 'onesided'. To compute the two-sided short-time FFT and inverse short-time FFT, set the `FrequencyRange` property to 'twosided'.

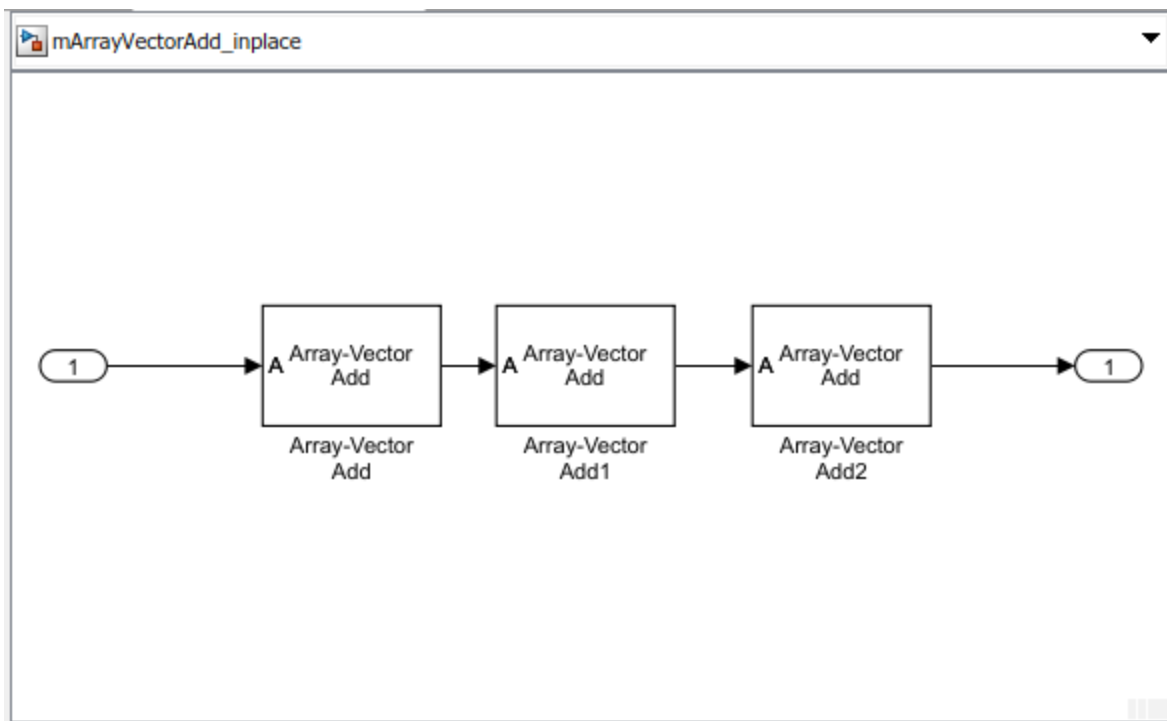
Using the `getFrequencyVector` function, you can obtain the vector of frequencies at which the short-time FFT is computed.

In-place Memory Optimization: Optimize the memory usage in the generated code for certain DSP System Toolbox blocks

The Array-Vector Add, Array-Vector Subtract, Array-Vector Multiply, and the Array-Vector Divide blocks in DSP System Toolbox support in-place memory optimization when the input to the block is real.

Due to in-place optimization, the generated code uses a single buffer for storing the output data values. Every time there is a new intermediary output in the model, this output buffer is overwritten to store that value.

For example, in this model, there is a sequence of Array-Vector Add blocks connected to each other.



Each block adds a vector to its input data and generates an output that is of the same size as the input.

This image shows a sequence of three s-functions generated in the in-place optimized generated code. Each block of s-function corresponds to the respective Array-Vector Add block in the Simulink model. In each s-function, the output buffer `rtb_ArrayVectorAdd2[]` rewrites its value every time the Array-Vector Add block has an updated output.

```

/* S-Function (sdspdmult2): '<Root>/Array-Vector Add' incorporates:
 * Inport: '<Root>/In1'
 */
idxS = 0;
for (i = 0; i < 3; i++) {
    rtb_ArrayVectorAdd2[idxS] = mArrayVectorAdd_inplace_U.In1[idxS] +
        mArrayVectorAdd_inplace_P.ArrayVectorAdd_V_VectFromMask[0];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] = mArrayVectorAdd_inplace_U.In1[idxS] +
        mArrayVectorAdd_inplace_P.ArrayVectorAdd_V_VectFromMask[1];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] = mArrayVectorAdd_inplace_U.In1[idxS] +
        mArrayVectorAdd_inplace_P.ArrayVectorAdd_V_VectFromMask[2];
    idxS++;
}

/*End of S-Function (sdspdmult2): '<Root>/Array-Vector Add' */

/* S-Function (sdspdmult2): '<Root>/Array-Vector Add1' */
idxS = 0;
for (i = 0; i < 3; i++) {
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd1_V_VectFromMask[0];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd1_V_VectFromMask[1];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd1_V_VectFromMask[2];
    idxS++;
}

/* End of S-Function (sdspdmult2): '<Root>/Array-Vector Add1' */

/* S-Function (sdspdmult2): '<Root>/Array-Vector Add2' */
idxS = 0;
for (i = 0; i < 3; i++) {
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd2_V_VectFromMask[0];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd2_V_VectFromMask[1];
    idxS++;
    rtb_ArrayVectorAdd2[idxS] +=
        mArrayVectorAdd_inplace_P.ArrayVectorAdd2_V_VectFromMask[2];
    idxS++;
}

/* End of S-Function (sdspdmult2): '<Root>/Array-Vector Add2' */

```

The generated code is efficient and uses less memory.

Improved Speed Performance in Accelerator Mode for specific blocks in DSP System Toolbox

When you run the following blocks in the accelerator mode, the blocks now run faster.

- Discrete FIR Filter
- FIR Rate Conversion -- To see the speedup in the FIR Rate Conversion block, set **Rate options** to **Allow multirate processing**.
- Arbitrary Response Filter
- Bandpass Filter
- Bandstop Filter
- Hilbert Filter
- Inverse Sinc Filter
- Nyquist Filter
- Digital Filter Design
- Analytic Signal

To run the model containing these blocks in the accelerator mode, on the **Simulation** tab of the model, in the **Simulate** section, select **Accelerator** from the drop-down list. Build an executable for the model by clicking **Run**. The acceleration (Simulink) mode uses this executable in simulations as long as the model remains structurally unchanged. For more details, see [Perform Acceleration \(Simulink\)](#).

Visualize logged Stateflow states in the Logic Analyzer

When you log signals in Stateflow® charts, you can use the **Logic Analyzer** to visualize the state changes. To log Stateflow states, in the **Simulation** tab, under **Prepare**, select a state logging option. In the Logic Analyzer, you'll see your states marked for logging in the left column.

For more details about how to log Stateflow signals, see [Log Simulation Output for States and Data \(Stateflow\)](#).

HDL-optimized FIR Decimation block and System object: Downsample signals using a FIR decimation filter with a hardware-friendly interface and architecture (requires HDL Coder for code generation)

The FIR Decimation HDL Optimized block downsamples signals using a transposed or systolic filter architecture. The block provides an efficient hardware implementation and uses hardware-friendly control signals. The block supports HDL code generation with HDL Coder™.

This algorithm is also available with the `dsp.HDLFIRDecimation` System object.

Gigasample-per-second (GSPS) CIC Decimation and CORDIC Algorithm: Increase throughput of HDL-optimized CIC decimation and complex-to-

magnitude-angle conversion by using frame-based input (requires HDL Coder for code generation)

You can now generate frame-based waveforms from the CIC Decimation HDL Optimized and Complex to Magnitude-Angle HDL Optimized blocks. Each block accepts and returns a column vector of elements that represent samples in time. The input vector can contain up to 64 samples. When you use frame-based input with the CIC Decimation HDL Optimized block, you must use a fixed decimation factor.

This capability increases throughput in hardware designs. For a list of all blocks that support frame-based input and output for HDL code generation, see High Throughput HDL Algorithms.

This feature is also available with the `dsp.HDLCICDecimation` and `dsp.HDLComplexToMagnitudeAngle` System objects.

To generate HDL code, you must have the HDL Coder product.

Dataflow domain analysis integrated with Performance Advisor

You can now use the Performance Advisor to find the optimal latency settings for all Dataflow subsystems in your model in a single step.

To find the optimal latency settings for the Dataflow subsystems in your model, open the Performance Advisor. In the **Performance Advisor > Simulation > Checks that Require Simulation to Run** folder, run the **Check Dataflow Domain Settings** check.

MATLAB Compiler support for `dsp.ArrayPlot`

You can use the `mcc` function to compile MATLAB code containing a `dsp.ArrayPlot`.

Functionality being removed or changed

`dsp.TimeScope` will be removed

Still runs

`dsp.TimeScope` will be removed in a future release. Use `timescope` instead. The `timescope` object has the same properties as the `dsp.TimeScope`, therefore, no updates to your code are required except replacing instances of `dsp.TimeScope` with `timescope`.

Spectrum Analyzer block defaults changed

Behavior change

Starting in R2020b, by default, the Spectrum Analyzer block uses the filter bank algorithm as the spectrum estimation method and exponential averaging as the averaging method:

- The default value of the **Method** parameter is now `Filter Bank`.
- The default value of the **Averaging method** parameter is now `Exponential`.

Version History

For existing models with a Spectrum Analyzer, the default values are not changed. For new visualizations with the Spectrum Analyzer block, you may see some changes in the output because of

the spectrum estimation method and averaging method. If you want to use the previous default values, set:

- **Method** to Welch
- **Averaging method** to Running

HDL Minimum Resource FFT and HDL Streaming FFT blocks have been removed

Errors

The HDL Minimum Resource FFT and HDL Streaming FFT blocks have been removed. Use the FFT HDL Optimized block instead.

- When replacing the HDL Minimum Resource FFT block, set the **Architecture** parameter of the FFT HDL Optimized block to `Burst Radix 2`.
- When replacing the HDL Streaming FFT block, set the **Architecture** parameter of the FFT HDL Optimized block to `Streaming Radix 2^2`.

For more information, see [Implement FFT for FPGA Using FFT HDL Optimized Block](#).

Matrix Viewer and Waterfall blocks will be removed

Still runs

The Matrix Viewer and Waterfall blocks will be removed in a future release.

R2020a

Version: 9.10

New Features

Bug Fixes

Version History

SIMD Code Generation: Use Intel AVX2 to generate optimized code for certain DSP System Toolbox blocks

FIR Interpolation and FIR Decimation blocks

The FIR Interpolation and FIR Decimation blocks now support SIMD code generation using Intel AVX2 technology when **Input processing** is set to Columns as channels (frame based), **Rate options** is set to Enforce single-rate processing, and the signal is real-valued with a data type of single or double. The SIMD technology significantly improves the performance of the generated code.

LMS Filter block

The LMS Filter block supports SIMD code generation using Intel AVX2 technology when the block's **Algorithm** parameter is set to LMS or Normalized LMS and the signal is real-valued with a data type of single or double.

Automatically leverage SIMD for multicore dataflow simulations

When your host CPU supports AVX2 technology, and you are using a supported mex compiler, multithreaded simulations of supported Dataflow subsystems automatically leverage SIMD. This can significantly improve the performance of multithreaded simulations of dataflow subsystems.

Supported compilers include

- Windows®: Microsoft® Visual C++ compilers.
- Linux®: gcc compilers

New Biquadratic SOS Filter Object

The `dsp.SOSFilter` System object implements a biquadratic second-order section IIR filter in MATLAB. The numerator and denominator coefficients and the filter section scale values are tunable which means their values can change even after the System object is locked. These values are specified using the `Numerator`, `Denominator`, and `ScaleValues` properties, respectively.

Multirate processing in FIR Rate Conversion block

The **Rate options** parameter now allows the block to operate in either a single-rate or a multirate processing mode.

- **Enforce single-rate processing** -- The output frame size is L/K times the input frame size, where L is the interpolation factor and K is the decimation factor.

The output signal rate in Simulink equals the input signal rate in Simulink.

- **Allow multirate processing** -- The output frame size equals the input frame size.

The output signal rate in Simulink is L/K times the input signal rate.

All blocks connected to the output operate at the output signal rate and all blocks connected to the input operate at the input signal rate.

Non-Maximally Decimated Channelizers

You can now specify the oversampling ratio in the `dsp.Channelizer` System object and the Channelizer block. Oversampling ratio O is a positive divisor of the number of frequency bands M .

The decimation factor D of the channelizer is determined using the ratio $D = M/O$.

When the oversampling ratio is an integer greater than 1, the channelizer is known as the non-maximally decimated channelizer or oversampled channelizer. Non-maximally decimated channelizers offer increased design freedom, but at the expense of increased computational cost.

Complex Support for Channelizer and Channel Synthesizer Prototype Coefficients

The lowpass prototype coefficients of the channelizer and the channel synthesizer objects and blocks can be complex.

If you specify complex coefficients, the channelizer and channel synthesizer design a prototype filter that is centered at a nonzero frequency, also known as a bandpass filter. The modulated versions of the prototype bandpass filter appear with respect to the prototype filter, and are wrapped around the frequency range $[-F_s, F_s]$. For an example, see Channelizer with Complex Coefficients.

Enhancements to designMultirateFIR function

You can now specify the transition width (TW) as a design parameter to the function as an alternative to the half polyphase length (P) parameter.

The function also supports a 'SystemObject' flag, which when set to `true` returns one of the following multirate System objects:

- `dsp.FIRInterpolator` -- When $L > 1$ and $M = 1$, where L is the interpolation factor and M is the decimation factor.
- `dsp.FIRDecimator` -- When $L = 1$ and $M > 1$.
- `dsp.FIRRateConverter` -- When $L > 1$ and $M > 1$.

When the 'SystemObject' flag is set to `false`, the function returns a vector of filter coefficients.

For an example, see Design a `dsp.FIRInterpolator` System object.

UDP Sender supports large message sizes

The UDP Sender object and block now support message sizes up to 67108864 bytes. The size can be specified through the `SendBufferSize` property in the object, and the **Send buffer size (bytes)** parameter in the UDP Send block dialog box.

Variable CIC Decimation Factor: Specify decimation factor as an input to the CIC Decimation HDL Optimized block (requires HDL Coder for code generation)

You can specify the decimation factor for the CIC Decimation HDL Optimized block as an input port. You can also now optionally enable automatic gain correction.

These features are also available when using the `dsp.HDLCICDecimation` System object.

Gigasample-per-second (GSPS) NCO: Generate frame-based output from HDL-optimized NCO for high speed applications (requires HDL Coder for code generation)

You can now generate frame-based waveforms from the NCO HDL Optimized block. The block returns a vector where each element represents a sample in time. Set the **Samples per frame** parameter to the desired output vector size.

This capability increases throughput in hardware designs. For a list of all blocks that support frame-based input and output for HDL code generation, see High Throughput HDL Algorithms.

This feature is also available when using the `dsp.HDLNCO` System object. Set the `SamplesPerFrame` property to the desired output vector size.

To generate HDL code, you must have the HDL Coder product.

Suggestions for optimal model settings in Dataflow Simulation Assistant

The Dataflow Simulation Assistant now recommends optimal model settings to improve simulation performance of your model. To open the Dataflow Simulation Assistant, in the **Execution** tab of the Property Editor, click **Dataflow assistant**.

To improve simulation performance, in the Dataflow Simulation Assistant, next to **Change model settings for simulation performance** click **Accept all**. For more information on the optimal model settings, see Simulation of Dataflow Domains.

Dataflow subsystems supported in model reference simulation targets

A Dataflow Subsystem inside a referenced model that executes in accelerator mode (uses a simulation target) can now simulate using multiple threads.

Functionality being removed or changed

Removal of `DirectFeedthrough` property in `dsp.VariableIntegerDelay` System object

The `DirectFeedthrough` property in the `dsp.VariableIntegerDelay` System object has been removed.

Version History

Make sure you remove all references to this property from your MATLAB code. The System object will operate in direct feedthrough mode only.

`dsp.AudioPlayer` and `dsp.AudioRecorder` objects removed

Errors

Starting in R2020a, using the following System objects throws an error message. Use the equivalent replacements instead.

System object	Use This Instead
<code>dsp.AudioRecorder</code>	<p><code>audioDeviceReader</code> object in Audio Toolbox.</p> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box has been removed. You can specify the driver in the <code>audioDeviceReader</code> object by using the <code>Driver</code> property.</p>
<code>dsp.AudioPlayer</code>	<p><code>audioDeviceWriter</code> object.</p> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in the <code>audioDeviceWriter</code> object by using the <code>Driver</code> property. This property is enabled only if you have the Audio Toolbox.</p>

For more details, see the **Compatibility Considerations** sections in the System object reference pages.

HDL-optimized NCO requires valid input port

Behavior change

In previous releases, the input **validIn** port of the NCO HDL Optimized block was optional. It is now required, and renamed **valid**. If you are using no other input ports, the block uses the **valid** signal as an enable signal.

When you use the `dsp.HDLNCO` System object, you must specify an input `validIn` argument. If you are using no other input arguments, the object uses the `validIn` argument as an enable signal.

HDL-optimized NCO with floating-point inputs applies phase quantization

Behavior change

The output waveform returned from floating-point input values has changed. The output waveform now matches that returned from the same input values specified in fixed-point types.

In previous releases, when using floating-point input types, the NCO HDL Optimized block did not quantize the phase internally. The block expected floating-point phase increment and phase offset inputs specified in radians. Now, the block quantizes the phase internally, and you must specify the input phase increment and offset in terms of the quantized size, for both floating-point and fixed-point input types.

This change also applies to the `dsp.HDLNCO` System object.

For example, in previous releases, for a floating-point HDL NCO to generate output samples with a desired output frequency of F_0 and sample frequency of F_s , you had to specify the phase increment as $2\pi(F_0/F_s)$ and phase offset as $\pi/2$.

Now, you must specify the phase increment and phase offset in terms of the quantized size, N . These input values are the same as the input values you use with fixed-point types. Specify the phase increment as $(F_0 \times 2^N)/F_s$, and the phase offset as $(\pi/2)2^N/2\pi$, or $2^N/4$.

NCO HDL Optimized block now ignores LUTRegisterResetType parameter

Behavior change

In previous releases, you could choose from two options for the **LUTRegisterResetType** parameter on the **HDL Block Properties** dialog of the NCO HDL Optimized block. The two options were **default**, and **none**. Starting in R2020a, the block ignores the parameter setting and uses **none** for this parameter value. This option does not connect a reset signal to the LUT registers. This configuration enables the synthesis tool to determine whether to implement the lookup tables with LUTs or BRAM.

Signal data no longer streams to the Logic Analyzer when signal logging is disabled

Behavior change

Previously, signals marked for logging have streamed to the **Logic Analyzer**, regardless of the setting for **Signal logging** in the model configuration parameters. Starting in R2020a, signals marked for logging stream to the Logic Analyzer *only* when signal logging is enabled for a model.

To view data in the Logic Analyzer, you must enable signal logging for the model. (Logging is on by default.) To enable signal logging, open **Model Settings** from the toolstrip, navigate to the **Data Import/Export** pane, and select **Signal logging**.

R2019b

Version: 9.9

New Features

Bug Fixes

Version History

SIMD code from Discrete FIR Filter Block: Generate optimized code using Intel AVX2 for FIR Filters in Simulink

When **Filter structure** is set to `Direct` form, and the signal is real-valued with a data type of single or double, the Discrete FIR Filter block supports SIMD code generation using Intel AVX2 technology. The SIMD technology significantly improves the performance of the generated code, in most cases meeting or exceeding the simulation performance.

HDL-optimized CIC Decimation block and System object: Downsample signals using a cascade integrator-comb (CIC) filter (requires HDL Coder for code generation)

The CIC Decimation HDL Optimized block downsamples signals using a CIC filter. The block provides an efficient hardware implementation and uses hardware-friendly control signals. The block supports HDL code generation with HDL Coder.

This algorithm is also available as a System object, `dsp.HDLCICDecimation`.

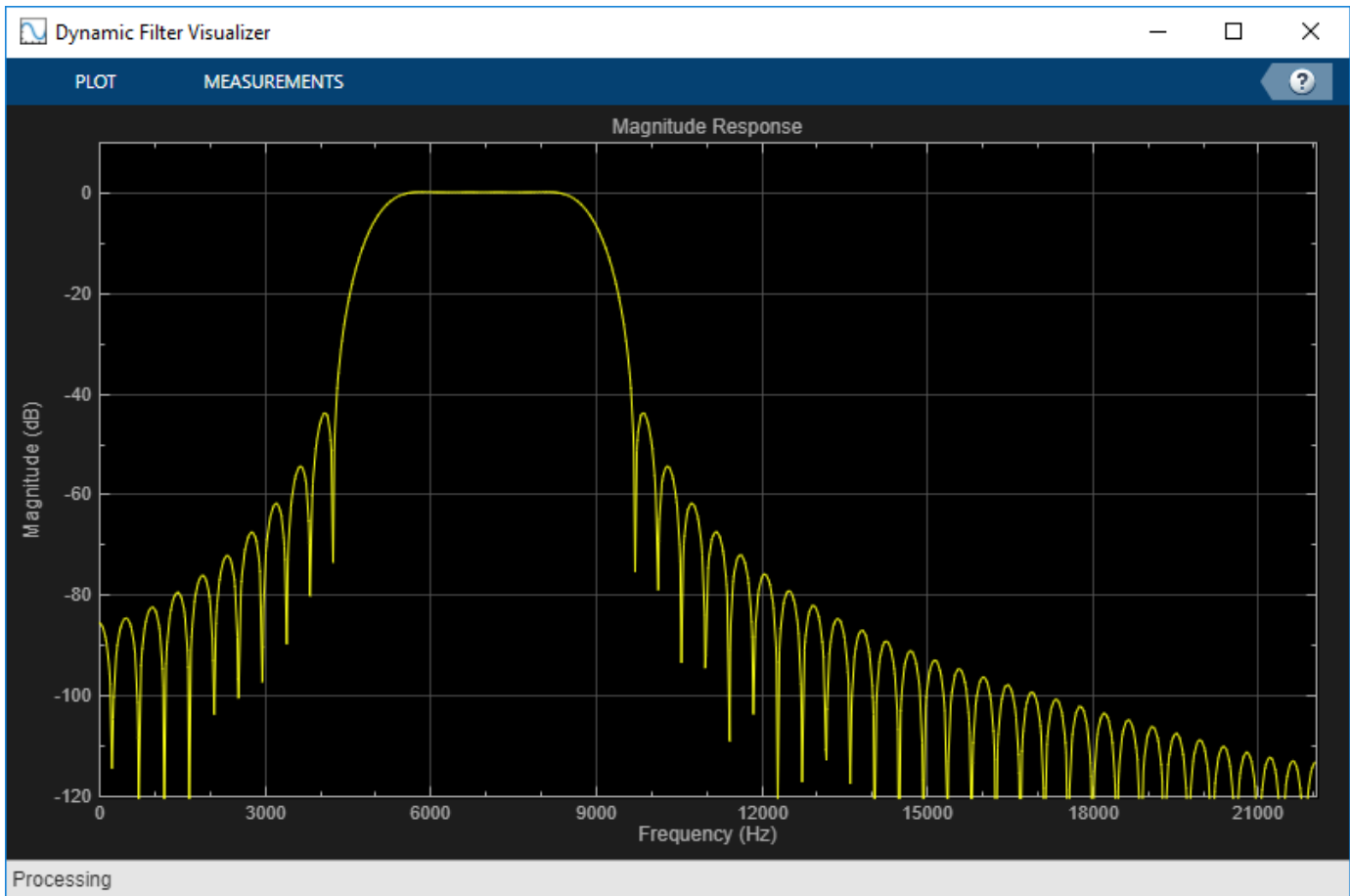
Discrete FIR Filter HDL Optimized block: Filter using complex coefficient values (requires HDL Coder for code generation)

The Discrete FIR Filter HDL Optimized block now supports complex-valued coefficients. If both coefficients and input data are complex, the block implements each filter tap with three multipliers. If either data or coefficients are complex but not both, the block uses two multipliers for each filter tap. You can use complex coefficients with all architectures and with programmable coefficients.

This feature is also available with the `dsp.HDLFIRFilter` System object.

Improved display for `dsp.DynamicFilterVisualizer`

In MATLAB, the dynamic filter visualizer object, `dsp.DynamicFilterVisualizer`, has a new and improved interface:

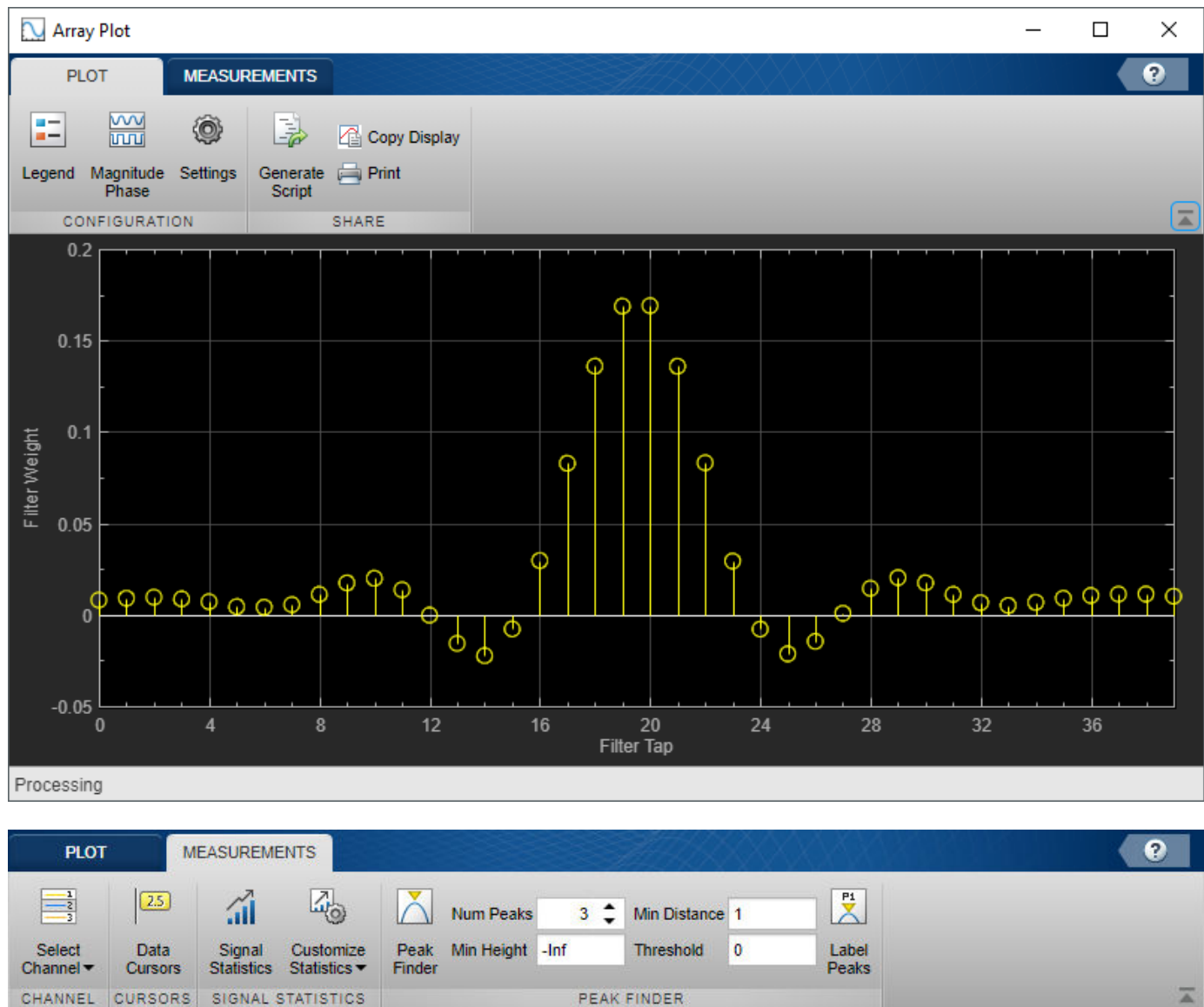


You can visualize dynamic filters with improved graphics, more responsiveness, and better interactivity. You can access settings and measurements via tabs and toolstrips.

For more information, see [Configure Array Plot MATLAB Object](#).

Improved display for `dsp.ArrayPlot`

In MATLAB, the Array Plot object `dsp.ArrayPlot` has a new and improved interface with two toolstrip tabs:



By clicking different buttons in the toolstrip, you can configure settings, turn on measurements, and share an image of the plot.

For more information, see [Configure Array Plot MATLAB Object](#).

Version History


- The `dsp.ArrayPlot` no longer supports `mcc` compilation.
- The `ReduceUpdates` property is no longer applicable. The new Array Plot interface continually tries to improve performance with update reduction as needed.

`dsp.MatrixViewer` support for multiple cursor measurements

When you activate cursors in the `dsp.MatrixViewer`, two horizontal and two vertical cursors appear. As you move the cursors around, the `dsp.MatrixViewer` shows the values at the cursor line

intersections and the difference between two intersection points. For more information, see [Cursor Measurements](#).

Playback control behavior changed for scopes in referenced models

When you use a scope in a referenced model, the playback controls in the scope now match the playback controls of the last model you interacted with that contains the scope. For example, if you opened your scope from a model referenced by another model with the Model block, the run button  in the scope runs the top level model. If the referenced model is opened as a root model, the run button runs the referenced model in isolation.

This change affects the Time Scope, Spectrum Analyzer, and Array Plot blocks, as well as the **Logic Analyzer** app.

For more information, see [Scopes in Referenced Models \(Simulink\)](#).

Output of colored noise generator can be bounded

The output generated from the `dsp.ColoredNoise` object can be bounded between +1 and -1 when you set the `BoundedOutput` property to `true`. Similarly in Simulink, the output of the Colored Noise block can be bounded between +1 and -1 by selecting the **Guarantee the output is bounded (+/-1)** parameter.

Blocks with finite states supported for unfolding in Dataflow subsystems

When cost analysis identifies a single block in a Dataflow subsystem that is computationally dominant, the system uses unfolding technology to improve the throughput through parallelization. In past releases, only stateless blocks were supported for unfolding. In 19b, blocks with finite states, such as the FIR Filter block, are also supported for unfolding.

For more information, see [Types of Parallelism](#).

Simulate Dataflow subsystems using multiple threads in Rapid Accelerator mode

Dataflow subsystems now support multithreaded simulation in Rapid Accelerator mode. For more information, see [Multicore Simulation and Code Generation of Dataflow Domains](#).

Virtual bus support at Dataflow subsystem boundaries for heterogeneous signals

You can now use virtual buses with mixed signal dimensions and data types at the boundaries of Dataflow subsystems for single-threaded simulation. For multithreaded simulation of Dataflow subsystems, replace virtual buses at the boundaries with non-virtual buses.

Functionality being removed or changed

Certain System objects will be removed

Warns

Starting in R2019b, using the following System objects throw a warning message. These objects will be removed in a future release. Use the equivalent replacements instead.

System object	Use This Instead
dsp.Buffer	dsp.AsyncBuffer
dsp.Histogram	histcounts
dsp.PeakFinder	findpeaks, islocalmin
dsp.Maximum	max, dsp.MovingMaximum
dsp.Minimum	min, dsp.MovingMinimum
dsp.Mean	mean, dsp.MovingAverage
dsp.Median	median, dsp.MedianFilter
dsp.RMS	rms, dsp.MovingRMS
dsp.StandardDeviation	std, dsp.MovingStandardDeviation
dsp.Variance	var, dsp.MovingVariance

For more details, see the **Compatibility Considerations** sections in the System object reference pages.

R2019a

Version: 9.8

New Features

Bug Fixes

Version History

Direct and Inverse Short-Time Fourier Transform: Analyze and process streaming signals in the frequency domain and synthesize them with perfect reconstruction using overlap and add

The `dsp.STFT` and `dsp.ISTFT` objects compute the short-time Fourier transform (STFT) and inverse short-time Fourier transform (ISTFT) of streaming data with time-varying spectral characteristics. Examples of such signals include audio, seismic data, and ECG. To analyze the frequency content of these signals, the `dsp.STFT` object windows the data into shorter segments with relatively nonvarying spectral characteristics and applies FFT on the segmented data. To reconstruct the original time-domain signal, the `dsp.ISTFT` object performs an ISTFT on the individual transformed subbands followed by overlap-add operations.

Fourth-Order Section Filter: Model and simulate cascaded fourth-order section IIR filters in MATLAB

The `dsp.FourthOrderSectionFilter` object creates a cascade of fourth-order IIR filter sections in MATLAB.

Spectrum Analyzer improvements for exponential averaging, mixed-complexity inputs, and MATLAB script generation

The Spectrum Analyzer block and `System` object now allow mixed-complexity inputs, exponential averaging, and generation of MATLAB scripts.

Smooth data with exponential averaging

The Spectrum Analyzer block and `dsp.SpectrumAnalyzer System` object now have two averaging modes for smoothing input samples: running averages (existing) and exponential averages (new). To specify the smoothing options for your input data, use these properties:

- **Averaging method** (`AveragingMethod`) — Choose `Running` or `Exponential`.
- **Averages** (`SpectralAverages`) — For running averages, choose the number of spectrum estimates to include in the running average.
- **Forgetting factor** (`ForgettingFactor`) — For exponential averaging, weigh previous spectrum estimates with a forgetting factor in the range $(0,1]$.

For more details about the averaging methods, see [Spectrum Analyzer Algorithms](#).


Display block inputs with different complexity

In the Spectrum Analyzer block, you can now visualize frequencies of inputs with different complexities. The signals must have the same sample rate and frame size, but can have both real and complex values.

Mixed-complexity inputs are already supported by the `dsp.SpectrumAnalyzer System` object.

Generate MATLAB script from `dsp.SpectrumAnalyzer`

After you modify the `dsp.SpectrumAnalyzer` from the UI, you can generate a MATLAB script to automatically set up the Spectrum Analyzer with your modified properties. To generate the script, use one of these methods:

-
- In the Spectrum Analyzer window, select **File > Generate MATLAB Script** or the Generate MATLAB script button .
 - From the command line, run the `generateScript` object function. For example:

```
sa = dsp.SpectrumAnalyzer
show(sa)
% change settings
generateScript(sa)
```

Note The script only generates commands for settings that are available from the command line, applicable to the current visualization, and changed from the default value.

Exponential Spectrum Averaging: Smooth spectrum estimation and analysis efficiently over time using exponential averaging

You can now smooth the power spectral density (PSD) data computed by the following System objects and blocks:

- `dsp.SpectrumEstimator`
- `dsp.CrossSpectrumEstimator`
- `dsp.TransferFunctionEstimator`
- Spectrum Estimator
- Cross-Spectrum Estimator
- Discrete Transfer Function Estimator

These objects and blocks now offer two averaging modes to smooth the PSD data: running mode and exponential weighting mode. In the running mode, the PSD data is averaged over the last N spectral data vectors. In the exponential weighting mode, the algorithm computes the average over the current and the previous PSD vector weighted by an exponentially decaying forgetting factor.

These settings control the smoothing options:

- **Averaging method** (`AveragingMethod`) — Choose `Running` or `Exponential`.
- **Number of spectral averages** (`SpectralAverages`) — For running averages, choose the number of spectrum estimates to include in the running average.
- **Forgetting factor** (`ForgettingFactor`) — For exponential averaging, weigh previous spectrum estimates with a forgetting factor in the range $(0,1]$.

For more details about the averaging methods, see the Algorithms section in `dsp.CrossSpectrumEstimator`.

Complex Data over UDP: Send and receive complex data directly over UDP in MATLAB and Simulink

You can now transmit complex data over a UDP network using the following UDP sender and receiver objects and blocks:

- `dsp.UDPSEnder`

- `dsp.UDPReceiver`
- UDP Send
- UDP Receive

Receive data coming from a UDP network as complex data by setting the `IsMessageComplex` property to `true` in the `dsp.UDPReceiver` object or by selecting the **Message is complex** parameter in the UDP Receive block.

Stream signals only from a defined interval within audio files when using the From Multimedia File block

You can specify a range of samples to stream from an audio file using the **Read range** parameter in the From Multimedia File block. When you run a model that contains this block, the From Multimedia File block streams data only from the interval defined by the **Read range** parameter.

New targets supported for multicore code generation from a dataflow subsystem

The following targets are now supported for multicore code generation from dataflow subsystems. In past releases, code generated from a dataflow subsystem for these targets was single-threaded.

- **Mac OS desktop targets**- Code generated for a Mac OS desktop target is now multithreaded using OpenMP.
- **Embedded Coder targets using Linux and VxWorks® operating systems** - Code generated for these Embedded Coder targets is now multithreaded using POSIX threads.

For more information on multicore code generation from dataflow subsystems, see Multicore Simulation and Code Generation of Dataflow Domains.

Code generation for these targets requires a Simulink Coder or an Embedded Coder license.

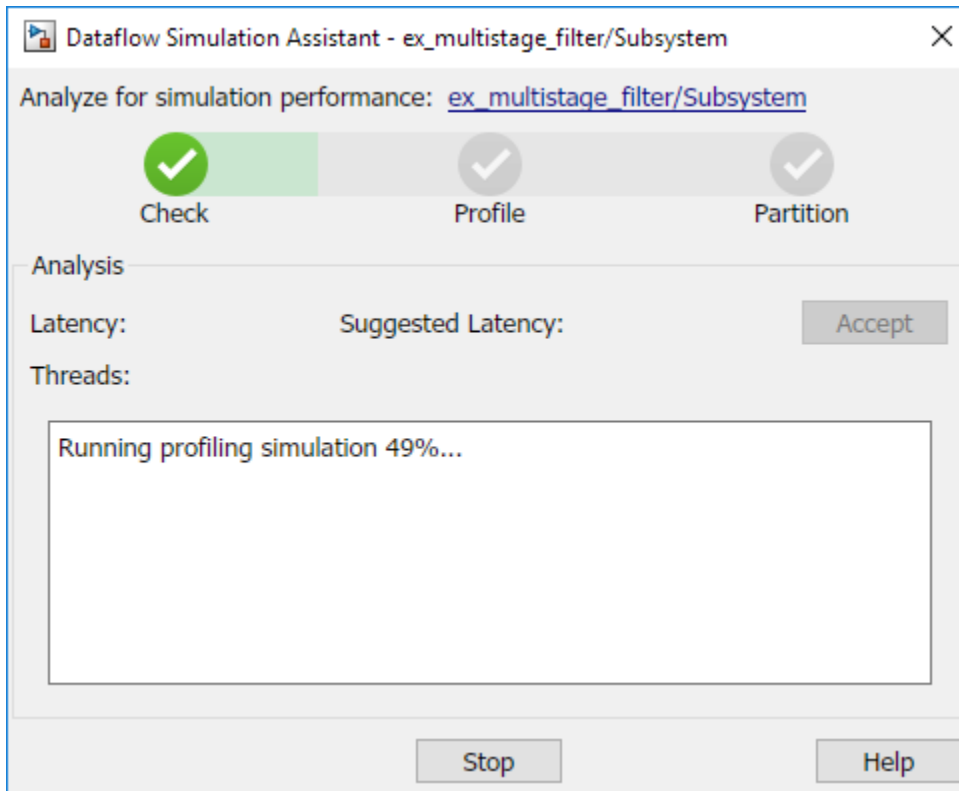
Blocks with constant sample times supported in dataflow subsystems

Blocks using constant (`inf`) sample times are now supported in dataflow subsystems. In previous releases, only blocks specifying inherited sample times were supported.

Improve simulation performance of dataflow subsystems using the Dataflow Simulation Assistant

To improve simulation performance of a system, it can be advantageous to increase the latency of the system. To find the optimal latency value for a dataflow subsystem, use the Dataflow Simulation Assistant. Starting in 19a, the Dataflow Simulation Assistant notifies you when a simulation with profiling or model compilation is required for the dataflow analysis. It also displays the progress of the dataflow analysis.

When the analysis completes, click **Accept** to apply the suggested latency to the subsystem.

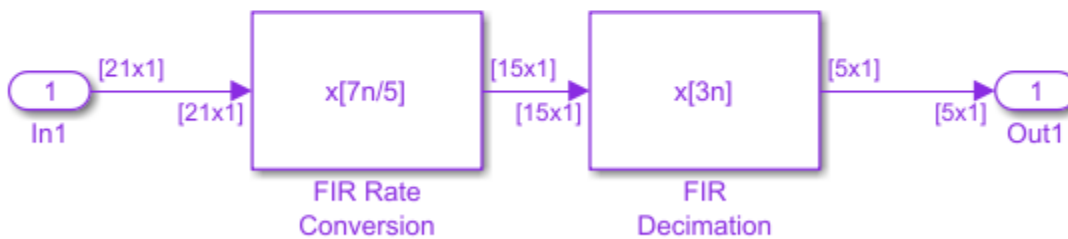


For more information, see [Dataflow Domain](#)

Identify scopes unsupported for multithreading in dataflow subsystems at edit-time

When you add one of the scope blocks inside a dataflow subsystem, the software highlights the block and displays a warning that the Scope block does not support multithreaded execution. Place the Scope outside the dataflow subsystem to simulate the subsystem using multiple threads.

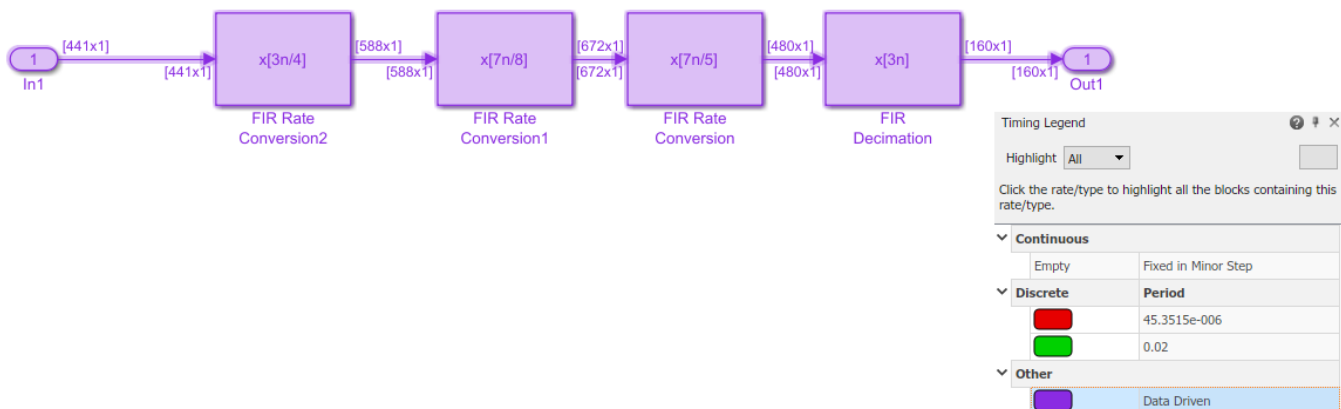
Dataflow domain will execute using a single thread.



Use the Timing Legend to highlight blocks in a dataflow domain

You can now highlight blocks in a dataflow domain using the Timing Legend. To view the highlighting,

- 1 In the Simulink menu, select **Display** > **Sample Time** > **Colors**.
- 2 Update the model diagram by selecting **Simulation** > **Update Diagram**.
- 3 To open the Timing Legend, select **Display** > **Sample Time** > **Timing Legend**.
- 4 In the Timing Legend, set **Highlight** to **All** or **Origin**.
- 5 In the legend, select **Data Driven** to highlight blocks in the dataflow domain.



Discrete FIR Filter HDL Optimized block: Use programmable coefficients with a fully parallel systolic architecture (requires HDL Coder for code generation)

The Discrete FIR Filter HDL Optimized block now provides the option to specify coefficients using an input port when you select the `Direct` form systolic architecture. You cannot use programmable coefficients with transposed or partly serial systolic architectures.

This feature is also available with the `dsp.HDLFIRFilter` System object.

Discrete FIR Filter HDL Optimized block: Optimize symmetric and antisymmetric coefficients and optional reset port for a partly serial systolic architecture (requires HDL Coder for code generation)

The Discrete FIR Filter HDL Optimized block now provides optimization of symmetric and antisymmetric coefficients and an optional reset port for any architecture, including a serial systolic architecture with resource sharing. This optimization reduces the number of multipliers and makes efficient use of FPGA DSP resources. The `reset` port provides a local synchronous reset of the data path registers.

These features are also available with the `dsp.HDLFIRFilter` System object.

The options for configuring a serial filter architecture have changed. For details, see “Changes to Discrete FIR Filter HDL Optimized serial filter parameters” on page 7-9.




HDL code generation support for programmable coefficients with frame-based Discrete FIR Filter block (requires HDL Coder for code generation)


The Discrete FIR Filter block now supports specifying coefficients from an input port when you use frame-based input.

`dsp.MatrixViewer` System object

The new `dsp.MatrixViewer` System object visualizes matrices by mapping matrix values to a color spectrum. You can customize the axes, labels, color scheme, and scope appearance by setting `Name,Value` pairs.

In the scope, zooming and panning buttons appear when you hover over the image:

-  Cursor measurements - Drag the cursor to see the x, y, and matrix value of individual data points. To customize the label of the cursor data, use the `setCursorDataLabels` object function.
-  Fit to view - After zooming or panning, select the fit to view button to see the full matrix image.
-  Zooming - Zoom in or out by selecting the magnifying glass button or by scrolling with your mouse.

-  Panning - Pan around the image by selecting the hand button or by clicking and dragging the matrix image.

DSP System Toolbox Support Packages for ARM Cortex -A and ARM Cortex -M Processors will be removed

Starting in R2019a, the DSP System Toolbox Support Package for ARM® Cortex®-A Processors and DSP System Toolbox Support Package for ARM Cortex-M Processors will no longer be available for download. This functionality has been moved to Embedded Coder Support Package for ARM Cortex-A Processors and Embedded Coder Support Package for ARM Cortex-M Processors, respectively. For more details on installing and getting started with these support packages, see Setup and Configuration (Embedded Coder Support Package for ARM Cortex-A Processors) and Setup and Configuration (Embedded Coder Support Package for ARM Cortex-M Processors).

Functionality being removed or changed

Certain System objects will be removed

Warns

Starting in R2019a, using the following System objects throws a warning message. These objects will be removed in a future release. Use the equivalent replacements instead.

System object	Use This Instead
dsp.CepstralToLPC	No replacement
dsp.LPCToAutocorrelation	poly2ac
dsp.LPCToCepstral	No replacement
dsp.LPCToLSF	poly2lsf
dsp.LPCToLSP	cos(poly2lsf)
dsp.LPCToRC	poly2rc
dsp.LSFToLPC	lsf2poly
dsp.LSPToLPC	No replacement
dsp.RCToAutocorrelation	rc2ac
dsp.RCToLPC	rc2poly
dsp.BurgAREstimator	arburg
dsp.BurgSpectrumEstimator	pburg
dsp.DCT	dct
dsp.IDCT	idct
dsp.Normalizer	normalize, vecnorm
dsp.ParametricEQFilter	designParamEQ from Audio Toolbox

For more details, see the **Compatibility Considerations** sections in the System object reference pages.

Parametric EQ Filter block has been removed

The Parametric EQ Filter block has been removed. For new models, use the Parametric EQ Filter block from Audio Toolbox instead.

Version History

Existing models using this block continue to run.

Changes to Discrete FIR Filter HDL Optimized serial filter parameters

Behavior change

Prior to R2019a, you specified the serial implementation by setting a requirement for input timing. Now, you can specify the serialization requirement based on either input timing or resource usage.

For a filter with L coefficients, the block implements a serial filter with not more than M multipliers and requires input samples that are at least N cycles apart, such that $L = N \times M$.

Serial Filter Requirement	Configuration Prior to R2019a	Configuration in R2019a
Specify a serialization rule based on input timing, that is, N cycles.	<ul style="list-style-type: none"> • Set Filter structure to Direct form systolic. • Select Share DSP resources. • Set Sharing factor to N. 	<ul style="list-style-type: none"> • Set the Filter structure to Partly serial systolic. • Set Specify serialization factor as to Minimum number of cycles between valid input samples. • Set Number of cycles to N.
Specify a serialization rule based on resource usage, that is, M multipliers.	<p>Serialization by resource usage is not supported prior to R2019a. However, you can calculate N based on your multiplier requirement.</p> <ul style="list-style-type: none"> • Set Filter structure to Direct form systolic. • Select Share DSP resources. • Set Sharing factor to $\text{ceil}(\text{NumCoeffs}/M)$. 	<ul style="list-style-type: none"> • Set the Filter structure to Partly serial systolic. • Set Specify serialization factor as to Maximum number of multipliers. • Set Number of multipliers to M.

R2018b

Version: 9.7

New Features

Bug Fixes

Version History

Dataflow: Accelerate your model using multi-threading and derive frame sizes automatically for multirate signal processing in Simulink

Using a dataflow domain, you can model and simulate a computationally intensive signal processing or multirate signal processing system. Dataflow domains simulate using computation synchronous dataflow, which is data-driven and statically scheduled. Simulation of dataflow domains in Normal and Accelerator modes leverages the multicore CPU architecture of the host computer. It automatically partitions your model and simulates the subsystem using multiple threads. The software can also automatically calculate the frame sizes needed for each block in a frame-based signal processing system, and insert buffers where needed.

For more information, see Dataflow Domain.

Programmatic Interface for Spectrum Analyzer Measurements: Configure measurements programmatically and obtain numerical results for further processing or analysis

The following properties configure measurement data programmatically for the `dsp.SpectrumAnalyzer` System object and the Spectrum Analyzer block:

- `MeasurementChannel`
- `PeakFinder`
- `CursorMeasurements`
- `ChannelMeasurements`
- `DistortionMeasurements`
- `CCDFMeasurements`

For the Spectrum Analyzer block, these properties belong to the `Spectrum Analyzer Configuration` object. To configure measurements for the block, select the Spectrum Analyzer block in the Simulink model, create a `Spectrum Analyzer Configuration` object for the selected block in the MATLAB command prompt, and edit the measurement properties.

```
% Select the Spectrum Analyzer block in your Simulink model
% and run the following:
cfg = get_param(gcf, 'ScopeConfiguration');
cfg.ChannelMeasurements.Enable = true;
cfg.ChannelMeasurements.PercentOccupiedBW = 95;
cfg.Visible = true;
```

Using the `getMeasurementData` function, you can obtain the measurement data programmatically for the `dsp.SpectrumAnalyzer` System object and the Spectrum Analyzer block.

Dynamic Filter Visualization: Visualize the magnitude response of time-varying digital filters

Using the `dsp.DynamicFilterVisualizer` object, you can now visualize the magnitude response of time-varying digital filters or time-varying filter coefficients. Consider an example where the `CenterFrequency` property of the `dsp.VariableBandwidthFIRFilter` System object changes with time, there by changing the magnitude response of the filter. You can view this time-varying magnitude response using the `dsp.DynamicFilterVisualizer` object.

```

% Create a dsp.DynamicFilterVisualizer object.
dfv = dsp.DynamicFilterVisualizer('YLimits', [-120 10]);

% Define a bandpass variable bandwidth FIR filter.
Fs = 44100;
vbw = dsp.VariableBandwidthFIRFilter('FilterType','Bandpass',...
    'FilterOrder',100,...
    'SampleRate',Fs,...
    'CenterFrequency',5e3,...
    'Bandwidth',4e3);

% Visualize the filter response as the CenterFrequency property
% changes.
for idx = 1:100
    dfv(vbw);
    vbw.CenterFrequency = vbw.CenterFrequency + 20;
end

```

Optimized Multistage Multirate Filters: Design multistage decimation and interpolation FIR filters based on requirements for response and implementation cost

Design optimal multistage decimation and interpolation FIR filters using the `designMultistageDecimator` and `designMultistageInterpolator` functions, respectively. You can design a filter with the lowest possible number of filter coefficients and multiplications per input sample by setting the `'MinTotalCoeffs'`, `'NumStages'`, and `'CostMethod'` arguments.

Sample Range for Audio File Reader: Stream signals only from a defined interval within audio files when using the `dsp.AudioFileReader` System object

You can specify a range of samples to stream from an audio file using the `ReadRange` property in `dsp.AudioFileReader` System object. When you run this object, the audio file reader streams data only from the interval defined by `ReadRange`.

Faster Channelizer and Channel Synthesizer: Simulate polyphase FFT filters faster by leveraging additional parallel optimizations

Simulation speed has improved for the following System objects and blocks:

- `dsp.Channelizer` and `dsp.ChannelSynthesizer` System objects.
- `Channelizer` and `ChannelSynthesizer` blocks.

The simulation speed improves for the blocks only when the **Simulate using** parameter in each block is set to `Interpreted` execution.

Peek Functionality in `dsp.AsyncBuffer` System object

The `peek` function outputs unread samples in the async buffer without changing the number of unread samples in the buffer.

Consider an async buffer that contains a column vector of 100 samples, `[1:100]'`. Peek at the first three samples using the `peek` function. The output is `[1; 2; 3]`. After peeking, read the first 50 samples using the `read` function. The output is `[1:50]'`. This shows that the `peek` function has not changed the number of unread samples. Now peek again at the first three samples. Now, the output is `[51; 52; 53]`. Read 50 samples again. The output now contains the sequence `[51:100]`, and the `NumUnreadSamples` is set to 0.

```
Q = dsp.AsyncBuffer;
signal = (1:100).';
write(Q,signal);
out1 = peek(Q,3); % out1 = [1; 2; 3];
out2 = read(Q,50); % out2 = (1:50).';
out3 = peek(Q,3); % out3 = [51; 52; 53];
out4 = read(Q); % out4 = (51:100).';
```

dsp.AudioFileReader System object supports http streams

The `dsp.AudioFileReader` System object can now read data from an http web address, such as `'http://audio.wgbh.org:8000/'`.

Improved Logic Analyzer performance for multichannel signals

When you use the **Logic Analyzer** to visualize signals with over 100 channels, the Logic Analyzer is now up to 80% faster. If you expand the signal to view individual channels, you may experience a short loading time as you scroll through the channels.

HDL code generation support for complex input signals or complex coefficients of frame-based Discrete FIR Filter and FIR Decimation blocks (requires HDL Coder for code generation)

You can generate HDL code from a frame-based filter that uses either complex input signals and real coefficients or complex coefficients and real input signals. See the "Frame-Based Input Support" sections of Discrete FIR Filter and FIR Decimation.

Discrete FIR Filter HDL Optimized: Select transposed architecture, optimize symmetric and antisymmetric coefficients, and enable reset port (requires HDL Coder for code generation)

The Discrete FIR Filter HDL Optimized block now provides:

- Option to select a direct form transposed architecture.
- Optimization of symmetric and antisymmetric coefficients when you select `Direct form systolic` (without **Share DSP resources** enabled) or select `Direct form transposed`. This optimization reduces the number of multipliers and makes efficient use of FPGA DSP resources.
- Optional **reset** input port to provide a local synchronous reset of the data path registers. By default the block connects the global HDL reset to only the control path registers. The reset parameters are not supported when you select **Share DSP resources**.

These features are also available on the `dsp.HDLFIRFilter` System object.

Version History

Starting in R2018b:

- The **validIn** port is mandatory. The **Enable valid input port** parameter is no longer available.
- The **ready** port is enabled when you select **Share DSP resources** and disabled when you clear **Share DSP resources**. The **Enable ready output port** parameter is no longer available.
- When you select `Direct form systolic` without **Share DSP resources** enabled, the block implements an improved fully-parallel architecture compared to previous releases. This architecture may have different latency than previous versions. Use the **validOut** signal to align with parallel delay paths. When using this architecture, the default global HDL reset now clears only the control path registers. Previous releases connected the global HDL reset to the data path registers and the control path registers. This change improves hardware performance and lowers the resources used. To implement the same fully parallel architecture as previous releases, select **Share DSP resources** and set **Sharing factor** to 1.
- When you select `Direct form systolic`, select **Share DSP resources**, and use any **Sharing factor**, the implemented filter has the same latency and uses the same hardware resources as in previous releases. The reset behavior for this architecture is also the same as previous releases.

Functionality being removed or changed

Vector Scope block has been removed

The Vector Scope block has been removed. When you open a model in R2018b or later, any Vector Scope blocks are automatically replaced with one of these blocks:

- Time Scope — Visualize time domain signals
- Spectrum Analyzer — Visualize frequency domain signals
- Array Plot — Visualize other input domain signals.

Version History

For visualizing frequency data, you may need to add a MATLAB Function block to perform a `fftshift`. For an example, see [Transform Time-Domain Data into Frequency Domain](#).

Certain linear prediction System objects will be removed

Still runs

The following System objects will be removed in a future release. Use the equivalent functions from Signal Processing Toolbox™ instead.

System object	Equivalent function
<code>dsp.CepstralToLPC</code>	No replacement
<code>dsp.LPCToAutocorrelation</code>	<code>poly2ac</code>
<code>dsp.LPCToCepstral</code>	No replacement
<code>dsp.LPCToLSF</code>	<code>poly2lsf</code>
<code>dsp.LPCToLSP</code>	<code>cos(poly2lsf)</code>

System object	Equivalent function
<code>dsp.LPCToRC</code>	<code>poly2rc</code>
<code>dsp.LSFToLPC</code>	<code>lsf2poly</code>
<code>dsp.LSPToLPC</code>	No replacement
<code>dsp.RCToAutocorrelation</code>	<code>rc2ac</code>
<code>dsp.RCToLPC</code>	<code>rc2poly</code>

For more details, see the **Compatibility Considerations** sections in the System object reference pages.

Cell array support removed for `dsp.AllpassFilter` coefficients

Errors

The following properties of the `dsp.AllpassFilter` System object have been removed in R2018b:

- `LatticeCoefficients`
- `AllpassCoefficients`
- `WDFCoefficients`

Use an N -by-1 or N -by-2 numeric array instead. For details, see the **Compatibility Considerations** section of the `dsp.AllpassFilter` System object.

R2018a

Version: 9.6

New Features

Bug Fixes

Version History

Frequency Input Mode for Spectrum Analyzer: Display, measure, and analyze frequency-domain signals in MATLAB and Simulink

Use the Spectrum Analyzer to visualize and analyze frequency signals. In MATLAB, use the `dsp.SpectrumAnalyzer` System object with the `InputDomain` property set to 'Frequency'. In Simulink, use the Spectrum Analyzer block and set the **Input Domain** to Frequency.

Efficiency-Optimized Digital Filters: Simulate select digital filters faster in MATLAB and Simulink by leveraging additional parallel optimizations

Simulation speed has improved for the following System objects and blocks:

- `dsp.FIRRateConverter`, `dsp.FarrowRateConverter`, `dsp.LMSFilter` System objects.
- FIR Rate Conversion and LMS Filter blocks.

Complex Bandpass Decimation: Extract a frequency subband using a one-sided (complex) bandpass decimator in MATLAB and Simulink

Extract a subband of frequencies from an input signal using the `dsp.ComplexBandpassDecimator` System object in MATLAB and Complex Bandpass Decimator block in Simulink.

Frequency-Domain Adaptive Filter Block: Simulate adaptive FIR filters requiring a large number of taps

Simulate FIR adaptive filters having a long impulse response using the Frequency-Domain Adaptive Filter block in Simulink. In partitioned mode, the latency of the filter decreases by an amount commensurate with the partitioned block length, rather than the entire impulse response length.

Bit-Natural HDL-Optimized FFT: Return data in bit-natural order from frame-based FFT/IFFT (Requires an HDL Coder license for code generation)

You can now select bit-natural output order, with any input order, when using the frame-based mode of the HDL-optimized FFT and IFFT. Before R2018a, input and output data were required to be in opposite orders. The orders of the input and output data are no longer restricted. This feature is added to these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Version History

Before R2018a, the output order of the Channelizer HDL Optimized block was bit-reversed when you set **Output vector size** to `Same as input size`. The output order is now bit-natural for both output sizes. This change also affects the `dsp.HDLChannelizer` System object.

New partitioned modes in `dsp.FrequencyDomainAdaptiveFilter` System object

Two new partitioned modes - `Partitioned constrained FDAF` and `Partitioned unconstrained FDAF` - have been added to the `dsp.FrequencyDomainAdaptiveFilter` System object. In these modes, filter latency decreases by an amount commensurate with the partitioned impulse response length.

Obtain section and output word lengths and fraction lengths for `dsp.CICDecimator` and `dsp.CICInterpolator` System objects

Using the `getFixedPointInfo` function, you can now obtain the word lengths and fraction lengths of the fixed-point sections and the output for the CIC filter System objects, `dsp.CICDecimator` and `dsp.CICInterpolator`.

Updated info method for `dsp.CICDecimator` and `dsp.CICInterpolator` System objects

When `dsp.CICDecimator` and `dsp.CICInterpolator` objects are locked in fixed-point configuration, the `info` method in 'long' format shows the word lengths and fraction lengths of the fixed-point filter sections and the filter output.

Specify coefficients directly in FIR halfband interpolator and decimator

When you set the filter specification to `Coefficients`, you can specify the FIR halfband filter coefficients directly in `dsp.FIRHalfbandDecimator` and `dsp.FIRHalfbandInterpolator` System objects in MATLAB and in FIR Halfband Decimator and FIR Halfband Interpolator blocks in Simulink. The coefficients you specify must comply with the FIR halfband filter format.

Frequency-Domain FIR Filter: Specify numerator in frequency domain

When you set the numerator domain of the frequency-domain FIR filter to `Frequency`, you can specify the frequency response of the filter directly. For details, see the `dsp.FrequencyDomainFIRFilter` System object and Frequency-Domain FIR Filter block.

Frequency-Domain FIR Filter: Specify coefficients from input port in Simulink

When you select the **Specify coefficients from input port** or **Specify frequency response from input port** parameter in the Frequency-Domain FIR Filter block, you can specify the time-domain filter coefficients or the frequency-domain filter coefficients, respectively, through an input port.

Tunable Parameters Through Input Ports: Set values of tunable parameters using input signals for 14 additional Simulink blocks

Specify tunable parameters through input ports for the following Simulink blocks in DSP System Toolbox:

- Moving Average
- Moving Variance
- Moving RMS
- Moving Standard Deviation
- Hampel Filter
- Channelizer
- Channel Synthesizer
- Allpass Filter
- IIR Halfband Interpolator
- IIR Halfband Decimator
- Variable Bandwidth FIR Filter
- Variable Bandwidth IIR Filter
- Notch-Peak Filter
- Parametric EQ Filter

Logic Analyzer enhancements

- The hexadecimal display now supports larger signal values. Integer-valued signals are supported up to 64 bits, and fixed-point signals are supported up to 128 bits.
- When displaying signed floating-point numbers, the Logic Analyzer rounds the number to four decimal places when the whole number cannot fit in the display.

Additional pipelining of HDL-optimized Complex to Magnitude-Angle

To improve synthesized clock frequency and make better use of DSP blocks on FPGAs, the Complex to Magnitude-Angle HDL Optimized block has additional pipelining. This change also affects the `dsp.HDLComplexToMagnitudeAngle` System object.

Version History

The latency of the block and System object is three cycles longer than previous releases. You must adjust the delay balancing of parallel data paths. As before, the latency is displayed on the block.

HDL Channelizer returns data in bit-natural order for both output sizes

Version History

Before R2018a, the output order of the Channelizer HDL Optimized block was bit-reversed when you set **Output vector size** to `Same as input size`. The output order is now bit-natural for both output sizes. This change also affects the `dsp.HDLChannelizer` System object.

Variable-size signal support for `dsp.VariableIntegerDelay` System object

The `dsp.VariableIntegerDelay` System object now supports variable-size input signals. When the input is a variable-size signal, the number of input rows can change during run time without having to call the `release` method between two calls to the algorithm. The number of channels must remain fixed.

See [Code Generation for Variable-Size Arrays](#) for information on variable-size signals.

Code generation support for `getRateChangeFactors` function

The `getRateChangeFactors` function which returns the overall interpolation and decimation factors of the rate converter in `dsp.FarrowRateConverter` and `dsp.SampleRateConverter` System objects now supports C and C++ code generation.

Binary File Reader: Binary file no longer required to exist before code generation

The `dsp.BinaryFileReader` object and Binary File Reader block no longer require the binary file to exist before generating code from the corresponding MATLAB code and the Simulink model, respectively.

Log data from Time Scope block as timetable

When logging signals using the Time Scope block, you can log data as a `timetable`.

If you set the configuration parameter **Dataset signal format** to `timetable` and the scope parameter **Save format** to `Dataset`, the data is saved as a `timetable` instead of a `timeseries` object.

Discrete FIR Filter block supports custom state attributes for better customization and efficiency of generated code

The Discrete FIR Filter block now supports custom state attributes to customize and generate code more efficiently. To access or set these attributes, in the Simulink editor, select **View > Model Data Editor** or press **Ctrl+Shift+E**. For an example, see [Custom State Attributes in Discrete FIR Filter block \(Simulink\)](#).

Functionality Being Removed

Removal of Vector Scope block

The Vector Scope block will be removed in a future release. Use one of these blocks instead:

- Time Scope — Visualize time domain signals
- Spectrum Analyzer — Visualize frequency domain signals
- Array Plot — Visualize other input domain signals.

Removal of DirectFeedthrough property in dsp.VariableFractionalDelay System object

Modifying the `DirectFeedthrough` property in the `dsp.VariableFractionalDelay` System object errors in R2018a. This property will be removed in a future release. Remove all references to this property from your MATLAB code. System objects will operate in direct feedthrough mode only.

Version History

If this property is currently set to `true`, no further change is required. If this property is currently set to `false`, modify your code to work in direct feedthrough mode.

Removal of DirectFeedthrough property in dsp.VariableIntegerDelay System object

Modifying the `DirectFeedthrough` property in the `dsp.VariableIntegerDelay` System object warns in R2018a. This property will be removed in a future release. Make sure you remove all references to this property from your MATLAB code. System objects will operate in direct feedthrough mode only.

Version History

If this property is currently set to `true`, no further change is required. If this property is currently set to `false`, modify your code to work in direct feedthrough mode.

Constraints on the dimensions of InitialConditions in dsp.VariableIntegerDelay System object

If the `InitialConditions` property of the `dsp.VariableIntegerDelay` System object is non-scalar, the property must be a 1-by-`numChans`-by-`MaximumDelay` matrix, where `numChans` is the number of input channels.

Running the following code gives an error message:

```
vid = dsp.VariableIntegerDelay;  
vid.InitialConditions = ones(1,10);  
vid(randn(100,10),1);
```

```
Error using VariableIntegerDelay/parenReference  
Specifying initial conditions in this format will be removed  
in a future release. Specify the initial conditions as a  
1xnumChansxMaximumDelay matrix instead, where numChans is the  
number of input channels.
```

When you change the dimensions of the initial conditions vector, the error message disappears. In this example, `vid.MaximumDelay = 100`.

```
vid = dsp.VariableIntegerDelay;  
vid.InitialConditions = ones(1,10,100);  
vid(randn(100,10),1);
```

Version History

In your existing code, set the `InitialConditions` property to a scalar or a matrix of size 1-by-numChans-by-MaximumDelay.

Functionality Removed from `dsp.DigitalUpConverter` and `dsp.DigitalDownConverter` System objects

The following properties have been removed from the `dsp.DigitalDownConverter` System object:

- `FilterSpecification`
- `SecondFilterCoefficients`
- `SecondFilterCoefficientsDataType`
- `CustomSecondFilterCoefficientsDataType`
- `ThirdFilterCoefficients`
- `ThirdFilterCoefficientsDataType`
- `CustomThirdFilterCoefficientsDataType`

The following properties have been removed from the `dsp.DigitalUpConverter` System object:

- `FilterSpecification`
- `FirstFilterCoefficients`
- `FirstFilterCoefficientsDataType`
- `CustomFirstFilterCoefficientsDataType`
- `SecondFilterCoefficients`
- `SecondFilterCoefficientsDataType`
- `CustomSecondFilterCoefficientsDataType`

The System objects do not allow the data type of coefficients to be specified for individual stages.

Version History

If these properties are currently set in your existing MATLAB code, modify your code to remove instances of these properties.

Functionality Removed from `dsp.Delay` System object

Setting delay 'Units' to 'Frames' is no longer supported

Starting in R2018a, setting the delay 'Units' to 'Frames' errors. To resolve the error, set 'Units' to 'Samples' and specify Length in samples instead of frames. Calculate the number of samples by multiplying the frame delay value by the frame size.

Running the following code gives an error message:

```
d = dsp.Delay;  
d.Units = 'Frames'  
d(1)
```

Error using Delay/parenReference
Specifying the delay in frame units will be removed in a future release. Set Units to 'Samples' and specify Length in samples instead by multiplying the frame delay value by the frame size.

Initial conditions are not supported in a cell array format

The `InitialConditions` vector no longer supports cell array format. Specify the initial conditions as a Length-by-numChans matrix instead, where numChans is the number of input channels.

Running the following code gives an error message:

```
d = dsp.Delay;  
d.InitialConditionsPerChannel = true;  
d.InitialConditionsPerSample = true;  
d.InitialConditions = {[1 3],5}  
d(1)
```

Error using Delay/parenReference
Support for cell-array initial conditions will be removed in a future release. Specify the initial conditions as a Length-by-numChans matrix instead, where numChans is the number of input channels.

The InitialConditionsPerChannel and InitialConditionsPerSample properties must be set to the same option

The `InitialConditionsPerChannel` and `InitialConditionsPerSample` properties must both be set to either true or false.

Removal of 'linphase' option in firlnorm

The 'linphase' option is no longer available in the `firlnorm` function.

R2017b

Version: 9.5

New Features

Bug Fixes


Version History

Improved Spectrum Analyzer: Analyze signals in the frequency domain using polyphase FFT filter banks, custom windows, dBFS units, and a spectral mask panel

Spectrum Analyzer blocks and `dsp.SpectrumAnalyzer` System objects have been improved.

- Analyze frequency signals using a polyphase FFT filter bank spectral estimation. The filter bank method has a lower noise floor and better frequency resolution. Filter bank also requires fewer samples per update compared to the Welch method. To use filter bank estimation in the Spectrum Analyzer block, in the Spectrum Setting panel, set **Method** to Filter Bank. For the System object, use `dsp.SpectrumAnalyzer('Method','Filter Bank')`.

Two new properties allow you to control the spectral estimation method and the filter bank estimation:

- Method** - Choose the spectral estimation method.
- NumTapsPerBand** - Specify the number of taps per frequency band.
- Save the spectrum data plotted in the Spectrum Analyzer using two new functions:
 - getSpectrumData** - Save the spectrum or spectrogram displayed in the Spectrum Analyzer along with simulation time, frequency vector, min hold, and max hold.
 - isNewDataReady** - Check if spectrum is updated to avoid saving duplicate data.
- Window your spectral analysis with two new **Window** options: Blackman-Harris and Custom. For more information, see `Window` and `CustomWindow`.
- Analyze spectrum data in dBFS. When using `dBFS SpectrumUnits`, you can customize the full scale with the new properties `FullScale` and `FullScaleSource`.
- Customize axis scaling at the command line with the `AxesScaling` property for auto scaling and the `ColorLimits` and `YLimits` properties for hard-coded limits.
- Within the Spectrum Analyzer window, set up spectral masks and monitor how often the mask fails. In the toolbar, select the  button to show the spectral mask settings and statistics panel.

Zoom FFT: Compute fast Fourier transform (FFT) of a frequency subband at high resolution

Analyze a subband of frequencies at a high resolution using the zoom FFT algorithm in the `dsp.ZoomFFT` System object and Zoom FFT block.

Frequency-Domain FIR Filter: Convolve long sequences while balancing latency and execution efficiency

Using the `dsp.FrequencyDomainFIRFilter` System object and the Frequency-Domain FIR Filter block, you can filter a streaming input signal using FFT based filtering methods. To mitigate the latency for a long filter, you can partition the impulse response into shorter blocks, and apply the filtering on these blocks.

Multitap Fractional Delay: Delay signals by multiple sample period values concurrently using variable fractional delay

Concurrently delay signals by multiple fractional delay values using the `dsp.VariableFractionalDelay` System object and the Variable Fractional Delay block.

Minimum Resource FFT/IFFT: Reduce resource usage with the Burst Radix 2 architecture of the HDL Optimized FFT (requires HDL Coder for code generation)

You can now choose a minimum resource architecture for the HDL-optimized FFT blocks and System objects. To use this feature, select the `Burst Radix 2` architecture in these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Logic Analyzer Improvements: Triggers and bus signal names

- In the **Logic Analyzer** and `dsp.LogicAnalyzer`, you can now use triggers to display signal data when certain conditions are met. Once you attach a signal to a trigger, you can trigger on:
 - Rising or falling edges
 - Bit pattern matching
 - Less than or greater than a value
 - Equal to a value
- If you log a bus signal in the **Logic Analyzer**, you can now view the bus element names. In the Logic Analyzer Settings window, select **Display bus element names**.

Enhancements to the `dsp.Channelizer` System object

You can now compute and visualize the frequency response of an individual or a combination of the filters in the `dsp.Channelizer` System object, using the `freqz` and the `fvtool` functions. You can also obtain the band edge frequencies and the center frequencies of the bandpass filters in the channelizer using the `bandedgeFrequencies` and the `centerFrequencies` functions. The `getFilters` function returns a matrix of filter coefficients, with each row containing the coefficients of the corresponding bandpass filter.

Automatic Port Creation: Add inports to scope blocks when routing signals

For Spectrum Analyzer and Array Plot blocks in Simulink models, dragging a line to connect another signal to the scope automatically adds a new input port. For an example, see [Build and Edit a Model in the Simulink Editor \(Simulink\)](#).

Improvements to interactive legend in scope blocks

For scope blocks and System objects, use the scope legend to toggle signal visibility. In the scope legend, click a signal name to hide the signal in the scope. To show the signal, click the signal name again. To show only one signal, right-click the signal name, which hides all other signals.

Array Plot Improvements: Support for scalar and variable-size inputs, axis scaling at the command line

For Array Plot blocks and `dsp.ArrayPlot` system objects, you can now:

- Visualize scalar or variable-sized input signals. If the signal is variable sized, the number of channels (columns) cannot change.
- Customize array plot axis scaling at the command line with the `AxisScaling` property.

`dsp.BlockLMSFilter` System object supports code generation

Generate C and C++ code from MATLAB code that contains the `dsp.BlockLMSFilter` System object using the MATLAB Coder.

Functionality being removed

Removal of Overlap-Add FFT Filter block and Overlap-Save FFT Filter block

Overlap-Add FFT Filter and Overlap-Save FFT Filter blocks have been replaced with the Frequency-Domain FIR Filter block. Existing instances of these blocks continue to run. For new models, use the Frequency-Domain FIR Filter block instead.

Removal of sample-based processing mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects has been removed. All the System objects listed now only work in frame-based processing mode. See [What Is Frame-Based Processing?](#) for more information.

- `dsp.AllpoleFilter`
- `dsp.AnalyticSignal`
- `dsp.BiquadFilter`
- `dsp.Buffer`
- `dsp.CumulativeProduct`
- `dsp.CumulativeSum`
- `dsp.Delay`
- `dsp.FIRFilter`
- `dsp.IIRFilter`
- `dsp.MatFileReader`
- `dsp.MatFileWriter`
- `dsp.Maximum`
- `dsp.Mean`

- `dsp.Minimum`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseUnwrapper`
- `dsp.RMS`
- `dsp.SignalSink`
- `dsp.StandardDeviation`
- `dsp.VariableFractionalDelay`
- `dsp.VariableIntegerDelay`
- `dsp.Variance`
- `dsp.ZeroCrossingDetector`

Version History

In existing code, if this property is set to `true`, no further change to the input is required. If this property is set to `false` and the input is a column vector or an N -D array, reshape the input such that each column in the input is an independent channel.

Removal of `adaptfilt` objects

All `adaptfilt` objects have been removed. Use the corresponding System object instead.

adaptfilt Object	Replacement System Object
<code>adaptfilt.lms</code>	<code>dsp.LMSFilter</code>
<code>adaptfilt.nlms</code>	
<code>adaptfilt.se</code>	
<code>adaptfilt.sd</code>	
<code>adaptfilt.ss</code>	
<code>adaptfilt.blms</code>	<code>dsp.BlockLMSFilter</code>
<code>adaptfilt.rls</code>	<code>dsp.RLSFilter</code>
<code>adaptfilt.qdrls</code>	
<code>adaptfilt.swrls</code>	
<code>adaptfilt.hrls</code>	
<code>adaptfilt.hswrls</code>	
<code>adaptfilt.ftf</code>	<code>dsp.FastTransversalFilter</code>
<code>adaptfilt.swftf</code>	

adaptfilt Object	Replacement System Object
adaptfilt.ap adaptfilt.apru adaptfilt.bap	dsp.AffineProjectionFilter
adaptfilt.gal adaptfilt.lsl adaptfilt.qrdlsl	dsp.AdaptiveLatticeFilter
adaptfilt.filtxlms adaptfilt.fdaf adaptfilt.ufdaf	dsp.FilteredXLMSFilter dsp.FrequencyDomainAdaptiveFilter
adaptfilt.blmsfft	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.adjlms adaptfilt.dlms	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.pbfdaf adaptfilt.pbufdaf	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.tdafdct adaptfilt.tfafdft	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter

Removal of qfft and qformat functions

The functions qfft and qformat have been removed. Use the dsp.FFT System object instead.

Removal of HDL Minimum Resource FFT block

The HDL Minimum Resource FFT block will be removed in a future release. In R2017b, the HDL Minimum Resource FFT block returns a warning. Use the `Burst Radix 2` architecture of the FFT HDL Optimized block instead. This block is in the Transforms (`dspxfm3`) library.

Removal of Streaming Radix 2 architecture in HDL-optimized FFT blocks and System objects

The `Streaming Radix 2` architecture has been removed from HDL-optimized FFT blocks and System objects for this release. Use the `Streaming Radix 22` architecture instead, which results in better hardware performance.

When you open a model containing an FFT HDL Optimized or IFFT HDL Optimized block that uses the `Streaming Radix 2` architecture, the block is automatically converted to use the `Streaming Radix 22` architecture. This change affects the latency of the block, so you must adjust the delay balancing of parallel data paths. The new latency is displayed on the block.

`dsp.HDLFFT` and `dsp.HDLIFFT` System objects that use the `Streaming Radix 2` architecture now return errors.

R2017a

Version: 9.4

New Features

Bug Fixes

Version History

Improved Spectrum Analyzer: Analyze signals in the frequency domain using additional units, dual visualization, and mask compliance output

Signal analysis with the Spectrum Analyzer block and System object has been improved:

- Visualize your signal spectrum as the root-mean squares (RMS) by using **Type > RMS**. When you select RMS as your spectrum type, you can choose from two spectrum units **Vrms** and **dBV**. Previously, these units were called power units.
- View the signal spectrum and spectrogram at the same time with the **View** setting. Use the **Axes layout** setting to set the dual view mode orientation: vertically stacked or side by side. You can also still view the spectrum and spectrogram individually.
- When you add a spectral mask, see when the spectrum values are inside or outside the limits. When the spectrum is inside the limits, the mask is green. When the spectrum is outside the limits, the mask is red.

Also, you can get statistics about what percentage of time the spectral mask passed or failed using the `getSpectralMaskStatus` function or the status bar tooltip.

- On the **Distortion Measurements** pane, measure up to 99 harmonics using the **Num. Harmonics** option.

For more information, see the `dsp.SpectrumAnalyzer` System object or the Spectrum Analyzer block.

Version History

Changed System Object Properties

Pre-R2017a Functionality	Use Instead	Considerations
PowerUnits	SpectrumUnits	The property has been renamed. Update any existing code with the new property name.
SpectrumType = 'Spectrogram'	ViewType = 'Spectrogram'	The property value Spectrogram is now a view type. Update any existing code with the new property name.

Unified interface for dsp.LogicAnalyzer: Visualize, measure, and analyze signal transitions in MATLAB using the same interface as the Simulink Logic Analyzer

The `dsp.LogicAnalyzer` System object has improved runtime and interaction performance, memory usage, and a unified interface with the Simulink **Logic Analyzer**.

Version History

The `DisplayChannelColor` property now supports customizable colors. String specifications continue to be supported and are converted to [R G B] values.

The default `DisplayChannelFormat` property is now 'auto'.

The `DisplayChannelHeight` and `DisplayChannelSpacing` properties are now defined in terms of pixels.

The `ReduceUpdates` and `MaxNumTimeSteps` properties are no longer used and will be removed in a future release.

Channelizer and Channel Synthesizer Blocks: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank in Simulink

The Channelizer block implements an analysis filter bank that splits a broadband input signal into multiple narrowband signals. The Channel Synthesizer block merges multiple narrowband signals to form a single broadband signal. These filter banks are implemented using an FFT based polyphase structure.

Asynchronous Buffering: Exchange signals at different rates and array sizes with the `dsp.AsyncBuffer` System object

You can write and read data from a FIFO buffer at different rates and frame sizes using the `dsp.AsyncBuffer` System object. The data that you write occupies the next available empty space in the buffer. After the last space in the buffer is used, the object overwrites the oldest data. The buffer does not erase the data when the object reads it, so you can reread data from the past (overlap reading).

HDL Optimized Filters: Model and generate optimized hardware implementations for FIR filters and polyphase filter banks (requires HDL Coder for code generation)

Discrete FIR Filter

This Discrete FIR Filter HDL Optimized block and `dsp.HDLFIRFilter` System object model FIR filter structures optimized for HDL code generation with HDL Coder. The filter is sample-based and includes control signals for flow control. Resource sharing options allow for tradeoffs between throughput and resource utilization. The block and object provide cycle-accurate models of the generated HDL code.

Polyphase Filter Bank

The Channelizer HDL Optimized block and `dsp.HDLChannelizer` System object model a polyphase filter bank and fast Fourier transform and support HDL code generation with HDL Coder. The algorithm provides an efficient hardware implementation and hardware-friendly control signals. You can achieve giga-sample-per-second (GSPS) throughput using vector input.

Frame Input Support for FIR Decimation

You can now generate HDL code from the FIR Decimation block when using frame input. The block accepts a column vector of input data, where each element of the vector represents a sample in time. The coder implements a parallel HDL architecture for the filter. This capability increases throughput in hardware designs. To configure the block for frame input:

- 1 Connect a column vector signal to the FIR Decimation block input port.
- 2 Set **Input processing** to `Columns as channels (frame based)`.
- 3 Set **Rate options** to `Enforce single-rate processing`.
- 4 Right-click the block and select **HDL Code > HDL Block Properties**. Set the **Architecture** to `Frame Based`. The block implements a parallel HDL architecture. See `Frame-Based Architecture (HDL Coder)`.

To generate HDL code, you must have an HDL Coder license. For information on HDL support for this block, see `FIR Decimation`.

Remove outliers from streaming signals in MATLAB and Simulink using Hampel filter

The `dsp.HampelFilter` System object in MATLAB and the Hampel Filter block in Simulink remove outliers from streaming signals by using the Hampel identifier.

Spectral estimation using filter bank in Simulink

To estimate the spectrum of a streaming signal in Simulink by using an analysis filter bank, set the **Method** parameter of the Spectrum Estimator block to `Filter bank`. For a given signal length, the filter bank approach of spectral estimation provides lower spectral leakage, higher frequency resolution, and a more accurate noise floor compared to the Welch's method.

Tunable UDP port number in generated code

The following System object properties and block parameters are now tunable in C/C++ generated code:

- `LocalIPPort` property in `dsp.UDPReceiver` System object.
- `RemoteIPPort` property in `dsp.UDPSender` System object.
- **Local IP port** parameter in UDP Receive block.
- **Remote IP port** parameter in UDP Send block.

These properties and parameters are not tunable during simulation. They are tunable only when you execute the generated code.

Filter signals using the `dsp.FilterCascade` System object

You can filter streaming signals with a cascade of filters using the `step` method of the `dsp.FilterCascade` System object. For a list of System objects that can be used as stages of the filter cascade, at the MATLAB command prompt, enter `dsp.FilterCascade.helpSupportedSystemObjects`.

Use delay and scalar gain in `dsp.FilterCascade` System object

You can now use the `dsp.Delay` System object and a numeric scalar value as stages in the `dsp.FilterCascade` System object.

Cascade a dsp.FilterCascade System object

The dsp.FilterCascade System object can now cascade another dsp.FilterCascade System object.

For example:

```
cicdecim = dsp.CICDecimator('DecimationFactor', 6, ...  
                           'NumSections', 6);  
decimcasc = dsp.FilterCascade(cicdecim, 1/gain(cicdecim));  
ciccomp = dsp.CICCompensationDecimator;  
filtchain = dsp.FilterCascade(decimcasc, ciccomp)  
  
filtchain =
```

```
    dsp.FilterCascade with properties:
```

```
    Stage1: [1x1 dsp.FilterCascade]  
    Stage2: [1x1 dsp.CICCompensationDecimator]
```

Access the complete history of LMS filter weights in MATLAB

When you set the `WeightsOutput` property of the dsp.LMSFilter System object to 'All', the object outputs a *FrameLength*-by-*Length* matrix of weights. *FrameLength* is the frame size of the input. *Length* is the length of the LMS filter. This matrix of weights corresponds to the full sample-by-sample history of weights values for all *FrameLength* samples of the input values.

Tab Completion: Complete parameter names and options in DSP System Toolbox System objects

When creating DSP System Toolbox System objects, you can use the Tab key to complete parameter names and options. For example, if you type `dsp.FIRFilter('` and press **Tab**, MATLAB displays a list of possible parameter names and options for the `dsp.FIRFilter` object.

Completion of parameters and options is not available for DSP System Toolbox functions.

Filter Builder and fdesign support IIR halfband filter System objects

The **Filter Builder** app and the `fdesign` function now support the `dsp.IIRHalfbandDecimator` and `dsp.IIRHalfbandInterpolator` System objects.

For example:

- **Filter Builder** — When you choose the filter response as **Halfband**, select the **Impulse response** as **IIR**, and set **Filter Type** to either **Decimator** or **Interpolator**, the `filterBuilder` designs a `dsp.IIRHalfbandDecimator` and `dsp.IIRHalfbandInterpolator` System objects, respectively.
- `fdesign` — 'SystemObject' flag set to `true` is supported in the design method of `fdesign.decimator`, `fdesign.interpolator`, `fdesign.halfband`, and `fdesign.nyquist` objects while designing an IIR halfband signal decimator or interpolator filter.

```
Fs = 2000; % Sampling frequency of input signal  
M = 2; % Decimation factor  
TW = 100; % Transition width of filter to be designed, 100 Hz
```

```

Ast = 60; % Stopband attenuation of filter to be designed, 80 db
% Store decimator design specs
f = fdesign.decimator(2,'halfband','TW,Ast',TW,Ast,Fs);
iirhalfbanddecim = design(f,'iirlinphase','FilterStructure','iirdecim',...
'SystemObject',true)

iirhalfbanddecim =

    dsp.IIRHalfbandDecimator with properties:

        Specification: 'Coefficients'
        Structure: 'Minimum multiplier'
        HasPureDelayBranch: true
        Delay: 14
        AllpassCoefficients2: [7x2 double]
        HasTrailingFirstOrderSection: false

```

Specify image file icons for MATLAB System block

You can specify a MATLAB System block icon as an image file using a new option in the MATLAB Editor. While editing your System object, specify the image file by selecting **System Block > Add Image Icon**. After specifying the image file, this code is added to the System object class:

```

function icon = getIconImpl(~)
    % Define icon for System block
    icon = matlab.system.display.Icon('image.png');
end

```

For more information, see [Customize System Block Appearance](#).

Change tunable System object properties before locking

For independent tunable properties, you can now change the value at any time.

If the tunable property has a dependent datatype property, you must lock the object before changing the property.

Support for Time Scope to For Each subsystems

You can place a Time Scope block within a For Each subsystem block. The scope follows the same behavior as the Display block. For example, within a For Each subsystem, the Time Scope block displays only the last iteration of the For Each subsystem.

Copy scope to clipboard

To share the output of a signal simulation, copy the scope graphic to your clipboard by selecting **File > Copy to Clipboard**. The scope colors are converted to a print-friendly coloring. See [Share Scope Image \(Simulink\)](#).

Interactive legend for scopes

If you show a legend on your scope block or System object, you can use the legend to filter which signals are shown. Left-clicking a signal in the legend hides all other signals in the scope. Right-clicking a signal in the legend toggles whether the scope shows or hides the signal.

Stem plot option for Time Scope block

In the Time Scope block, you can visualize your signal as a stem plot. From the **View > Style** menu, select **Plot type > Stem**.

Time Scope Block: Connect nonvirtual bus and array of buses signals

You can connect nonvirtual bus signals and array of buses signals to a Time Scope block. To display the signals in the Time Scope block, use the normal or accelerator simulation mode. For details, see Nonvirtual Bus and Array of Buses Signals and Save Simulation Data Using a Scope Block.

Frame-based processing changes

As part of the changes in how DSP System Toolbox handles frame-based processing, certain block options have been removed and certain function options error. The following sections provide more detailed information about the specific R2017a DSP System Toolbox software changes for frame-based processing:

- “Input processing parameter set to Inherited” on page 11-7
- “InputProcessing property set to Inherited errors” on page 11-8
- “Rate options parameter set to Inherit from input” on page 11-8
- “Find the histogram over parameter set to Inherited” on page 11-9
- “Running difference parameter set to Inherit from input” on page 11-9
- “Save 2-D signals as parameter set to Inherit from input” on page 11-9
- “Treat Mx1 and unoriented sample-based signals as parameter removed” on page 11-9
- “Sample-based processing parameter removed” on page 11-9

Input processing parameter set to Inherited

The **Inherited** option has been removed from the **Input processing** parameter of the following blocks:

- Biquad Filter
- Arbitrary Response Filter — To see the change, select **Use basic elements to enable filter customization**.
- Bandpass Filter — To see the change, select **Use basic elements to enable filter customization**.
- Bandstop Filter — To see the change, select **Use basic elements to enable filter customization**.
- CIC Filter — To see the change, select **Use basic elements to enable filter customization**.
- Comb Filter — To see the change, select **Use basic elements to enable filter customization**.
- Hilbert Filter — To see the change, select **Use basic elements to enable filter customization**.
- Inverse Sinc Filter — To see the change, select **Use basic elements to enable filter customization**.
- Nyquist Filter — To see the change, select **Use basic elements to enable filter customization**.
- Octave Filter — To see the change, select **Use basic elements to enable filter customization**.

- Digital Filter Design — To see the change, select **Use basic elements to enable filter customization**.
- CIC Interpolation
- Downsample
- Upsample
- Repeat
- Variable Fractional Delay
- Mean — To see the change, select **Running mean**.
- Variance — To see the change, select **Running variance**.
- RMS — To see the change, select **Running RMS**.
- Standard Deviation — To see the change, select **Running standard deviation**.
- Minimum — To see the change, set **Mode** to Running.
- Maximum — To see the change, set **Mode** to Running.
- Cumulative Sum
- Cumulative Product
- Edge Detector
- Zero Crossing
- Unwrap

Version History

Blocks using the Inherited option in models created in previous releases can no longer use this option in R2017a. When you open these models in R2017a, the block chooses `Columns as channels` (frame based).

InputProcessing property set to Inherited errors

Setting Input processing property to Inherited now causes an error in these functions:

- `block`
- `realizemdl`

It is recommended that you set this property to one these options:

- `'columnsaschannels'` — Treat each column of the input signal as an independent channel.
- `'elementsaschannels'` — Treat each element of the input signal as an independent channel.

Rate options parameter set to Inherit from input

The `Inherit from input` option has been removed from the **Rate options** parameter of the CIC Decimation block when the input to the block is a scalar.

Version History

If the **Rate options** parameter in the existing model is set to `Inherit from input`, the model errors in R2017a. It is recommended that you update this parameter to either `Allow multirate processing` or `Enforce single-rate processing`.

For information on which **Rate options** parameter to choose, see *Rate options parameter set to Inherit from input* under *Frame-based processing* in the R2015a DSP System Toolbox release notes.

Find the histogram over parameter set to Inherited

The **Inherited** option has been removed from the **Find the histogram over** parameter of the Histogram block.

Version History

In existing models, Histogram blocks using the **Inherited** option can no longer use this option in R2017a. When you open these models in R2017a, the block chooses **Each column**.

Running difference parameter set to Inherit from input

The **Inherit from input** option has been removed from the **Running difference** parameter of the Difference block.

Version History

In existing models, Difference blocks using the **Inherit from input** option can no longer use this option in R2017a. When you open these models in R2017a, the block chooses **No**.

Save 2-D signals as parameter set to Inherit from input

The **Inherit from input** option has been removed from the **Save 2-D signals as** parameter of the Triggered To Workspace block.

Version History

Triggered To Workspace blocks using the **Inherit from input** option in existing models can no longer use this option in R2017a. When you open these models in R2017a, the block chooses **2-D array (concatenate along first dimension)**.

Treat Mx1 and unoriented sample-based signals as parameter removed

The **Treat Mx1 and unoriented sample-based signals as** parameter has been removed from these blocks:

- Buffer
- Delay Line

Version History

In existing models, Buffer and Delay Line blocks using the **Treat Mx1 and unoriented sample-based signals as** parameter can no longer use this parameter in R2017a. The blocks treat these inputs as single channels.

Sample-based processing parameter removed

The **Sample-based processing** parameter has been removed from the Unbuffer block.

Version History

In existing models, Unbuffer blocks using the **Sample-based processing** parameter can no longer use this parameter in R2017a. The block automatically unbuffers an M -by- N matrix input into a 1-by- N output vector.

Functionality being removed

- “Running Mode in Statistics Objects and Blocks” on page 11-10
- “Audio device recorder and player objects” on page 11-10
- “Radix 2 architecture of HDL-optimized FFT blocks and System objects” on page 11-11

Running Mode in Statistics Objects and Blocks

The running mode in the following System objects and blocks will be removed in a future release.

System Object	Use This Instead
dsp.Maximum	dsp.MovingMaximum
dsp.Minimum	dsp.MovingMinimum
dsp.Mean	dsp.MovingAverage
dsp.RMS	dsp.MovingRMS
dsp.StandardDeviation	dsp.MovingStandardDeviation
dsp.Variance	dsp.MovingVariance

Blocks	Use This Instead
Maximum	Moving Maximum
Minimum	Moving Minimum
Mean	Moving Average
RMS	Moving RMS
Standard Deviation	Moving Standard Deviation
Variance	Moving Variance

Audio device recorder and player objects

The following DSP features warn in R2017a and will be removed in a future release.

System object	Use This Instead
dsp.AudioRecorder	<p>audioDeviceReader object in Audio Toolbox.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box has been removed. You can specify the driver in the audioDeviceReader object by using the Driver property.</p>

System object	Use This Instead
dsp.AudioPlayer	audioDeviceWriter object. Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in the audioDeviceWriter object by using the Driver property.

Version History

Existing instances of these System objects continue to run. For new instances of the functionality, use the replacement feature.

Radix 2 architecture of HDL-optimized FFT blocks and System objects

The Radix 2 architecture, used in the FFT HDL Optimized and IFFT HDL Optimized blocks, and in the dsp.HDLFFT and dsp.HDLIFFT System objects, will be removed in a future release. Use of this architecture returns a warning in R2017a. Use the Radix 2² architecture instead, which results in better hardware performance.

Architecture	Use This Instead
Streaming Radix 2	Streaming Radix 2 ²

Version History

The latency of the two architectures is different. The new latency is displayed on the blocks, or can be accessed using the getLatency method of the System object. When you change the architecture, adjust any delay balanced paths in your model.

R2016b

Version: 9.3

New Features

Bug Fixes

Version History

Logic Analyzer: Visualize, measure, and analyze transitions and states over time for Simulink signals

The Logic Analyzer visualization tool enables you to view the transitions of signals. You can use the **Logic Analyzer** to:

- Debug and analyze models.
- Trace and correlate many signals simultaneously.
- Detect and analyze timing violations.
- Trace system execution.

See *Inspect and Measure Transitions Using the Logic Analyzer* to explore some of its key functionality.

Spectral Mask: Compare a signal spectrum to a spectral mask using Spectrum Analyzer

The Spectrum Analyzer block and `SpectrumAnalyzer` System object support defining and overlaying a spectral mask on spectrum plots. Spectral masks are useful for verifying that the signal spectrum is within an area of defined spectral limitations.

Channelizer and Channel Synthesizer: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank

The `dsp.Channelizer` System object implements an analysis filter bank that splits a broadband input signal into multiple narrowband signals. The `dsp.ChannelSynthesizer` System object merges multiple narrowband signals to form a single broadband signal. These filter banks are implemented using an FFT based polyphase structure.

Moving Statistics: Measure descriptive statistics on streaming signals in MATLAB and Simulink

Compute moving statistics such as the average, RMS, standard deviation, variance, minimum, maximum, and median of a streaming input signal in MATLAB code and Simulink models.

Gigasample per Second (GSPS) Signal Processing: Increase the throughput of HDL code generated from Discrete FIR Filter and Integer Delay blocks using frame input

You can now generate HDL code from the Discrete FIR Filter block when using frame input. First set **Input processing** to `Columns as channels (frame based)`. Then right-click the block, open **HDL Code > HDL Block Properties**, and set the **Architecture** to `Frame Based`. The block accepts vector input data, where each element of the vector represents a sample in time. The coder implements a parallel HDL architecture for the filter. For information on HDL support for this block, see *Discrete FIR Filter*.

The Delay block also supports HDL code generation with frame input data. Set **Input processing** to `Columns as channels (frame based)`. The block accepts vector input data, where each element of the vector represents a sample in time.

This capability increases throughput in hardware designs. To generate HDL code, you must have an HDL Coder license.

Stream signals to and from binary files

You can now read and write binary files in the MATLAB environment using the `dsp.BinaryFileReader` and `dsp.BinaryFileWriter` System objects. In the Simulink environment, you can use the corresponding Binary File Reader and Binary File Writer blocks. The reader can read any binary file and does not depend on how a binary file is created.

Compute LMS adaptive filter weights using LMS Update block

LMS Update block estimates adaptive filter weights using an LMS adaptive algorithm. The block accepts the data and error as inputs and computes the filter weights based on the specified LMS algorithm. Using this block, you can model and simulate variants of LMS adaptive filtering algorithm, including the filtered-X LMS.

Allpass Filter block

The Allpass Filter block filters each channel of a streaming input signal using a single-section or a multiple-section allpass filter in Simulink. You can implement the allpass filter in minimum multiplier, wave digital filter, or lattice form.

Specify coefficients in Farrow Rate Converter block and System object

You can now specify the coefficients of Farrow rate converters directly. To specify the coefficients, set the `Specification` property of the `dsp.FarrowRateConverter` System object to `Coefficients`. In the Farrow Rate Converter block, set the **Specification method** parameter to `Coefficients`.

Spectral estimation using filter banks

To estimate the spectrum of a signal using an analysis filter bank, set the `Method` property of the `dsp.SpectrumEstimator` System object to `'Filter bank'`. The filter bank approach provides low spectral leakage, high frequency resolution, and an accurate noise floor.

High-throughput polyphase filter bank for HDL example

The Generate HDL Code for High Throughput Signal Processing model example shows how to design a polyphase filter bank to achieve gigasample per second data rates in the generated HDL implementation. The model uses the FFT HDL Optimized block with vector input.

Bit-reversed input order for HDL-optimized FFT

For vector input data, the HDL-optimized FFT algorithms now support bit-reversed input with natural order output. For scalar input data, you can select any input order with any output order. The default is natural order input with bit-reversed output.

This change affects these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

HDL code generation for reset port on Discrete FIR Filter

You can now generate HDL code from the Discrete FIR Filter block when you configure the block to have an external reset port.

Compiler support for System object scopes

You can now compile MATLAB code containing calls to `dsp.ArrayPlot`, `dsp.SpectrumAnalyzer`, or `dsp.TimeScope` System objects by using the `mcc` MATLAB compiler command. You use compiled MATLAB code to create a standalone application.

Custom X-axis data in Array Plot

You can now customize X-axis data in the Array Plot block and `dsp.ArrayPlot` System object. Use this option to specify the axis for arbitrarily spaced data.

Set legend strings and autoscaling programmatically in Time Scope

In the `dsp.TimeScope` System object, you can control the appearance of the scope from the MATLAB command line or from within MATLAB code. Use the `ChannelNames` property to specify a cell array of names to use in the plot legend. When `ShowLegend` is `true`, the legend pulls the names from the cell array. Use the `AxesScaling` property to enable autoscaling of the plot. The default scaling setting is to scale the plot when the simulation stops.

Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object, you can call the object with arguments, as if it were a function. The `step` method continues to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `dsp.FFT` System object named `fft1024`, then you call the System object as a function with that name.

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...  
    'FFTLength',1024);  
fft1024(x)
```

The equivalent operation using the `step` method is:

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...  
    'FFTLength',1024);  
step(fft1024,x)
```

When the `step` method has the System object as its only argument, the function equivalent has no arguments. You must call this function with empty parentheses. For example, `step(sysobj)` and `sysobj()` perform equivalent operations.

System objects support for additional inputs, global variables, and enumeration data types

- System objects in code generated using MATLAB Coder can have up to 1024 inputs.
- You can use global variables declared in System objects to exchange data with the Data Store Memory block in Simulink. You can use these variables in generated code.
- Enumeration data types for System objects included in Simulink using the MATLAB System block is supported. Enumerations restrict data to a finite set of data values that inherit from `int8`, `uint8`, `int16`, `uint16`, `int32`, or `Simulink.IntEnumType` data types, or a data type you define using `Simulink.defineIntEnumType`.

Functionality being removed

Removal of sample mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects will be removed in a future release. System objects containing this property will then work only in frame-based processing mode. See [What Is Frame-Based Processing?](#) for more information. If this property is set to `true`, no further change to the input is required. If this property is set to `false` and the input is a column vector or an N -D matrix, reshape the input such that each column in the input is an independent channel.

Effective R2016b, modifying this property throws an error for these System objects:

- `dsp.AllpoleFilter`
- `dsp.AnalyticSignal`
- `dsp.BiquadFilter`
- `dsp.Buffer`
- `dsp.CumulativeProduct`
- `dsp.CumulativeSum`
- `dsp.Delay`
- `dsp.FIRFilter`
- `dsp.IIRFilter`
- `dsp.MatFileReader`
- `dsp.MatFileWriter`
- `dsp.Maximum`
- `dsp.Mean`
- `dsp.Minimum`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseUnwrapper`
- `dsp.RMS`

- `dsp.SignalSink`
- `dsp.StandardDeviation`
- `dsp.VariableFractionalDelay`
- `dsp.VariableIntegerDelay`
- `dsp.Variance`
- `dsp.ZeroCrossingDetector`

Digital Filter block and System object

The `dsp.DigitalFilter` System object has been removed and the Digital Filter block will be removed in a future release. The table lists the recommended replacement blocks and System objects.

Filter to Implement	Replacement Block	Replacement System Object
FIR filter structures	Discrete FIR Filter	<code>dsp.FIRFilter</code>
Biquad (SOS) structures	Biquad Filter	<code>dsp.BiquadFilter</code>
Non-SOS IIR filter structures	Discrete Filter	<code>dsp.IIRFilter</code>
Allpole structures	Allpole Filter	<code>dsp.AllpoleFilter</code>

Version History

Existing instances of the Digital Filter block continue to run. For new models, use the replacement block listed in the table.

Removal of `adaptfilt` objects

Starting in R2016b, using `adaptfilt` objects throws an error. In a future release, these objects will be removed. Use the corresponding System object instead.

<code>adaptfilt</code> Object	Replacement System Object
<code>adaptfilt.lms</code>	<code>dsp.LMSFilter</code>
<code>adaptfilt.nlms</code>	
<code>adaptfilt.se</code>	
<code>adaptfilt.sd</code>	
<code>adaptfilt.ss</code>	
<code>adaptfilt.blms</code>	<code>dsp.BlockLMSFilter</code>
<code>adaptfilt.rls</code>	<code>dsp.RLSFilter</code>
<code>adaptfilt.qdrls</code>	
<code>adaptfilt.swrls</code>	
<code>adaptfilt.hrls</code>	
<code>adaptfilt.hswrls</code>	

adaptfilt Object	Replacement System Object
adaptfilt.ftf	dsp.FastTransversalFilter
adaptfilt.swftf	
adaptfilt.ap	dsp.AffineProjectionFilter
adaptfilt.apru	
adaptfilt.bap	
adaptfilt.gal	dsp.AdaptiveLatticeFilter
adaptfilt.lsl	
adaptfilt.qrdls1	
adaptfilt.filtxlms	dsp.FilteredXLMSFilter
adaptfilt.fdaf	dsp.FrequencyDomainAdaptiveFilter
adaptfilt.ufdaf	
adaptfilt.blmsfft	Will be removed in a future release
adaptfilt.adj1ms	Will be removed in a future release
adaptfilt.dlms	
adaptfilt.pbfdaf	Will be removed in a future release
adaptfilt.pbufdaf	
adaptfilt.tdafdct	Will be removed in a future release
adaptfilt.tfafdft	

Cell array support removal for dsp.AllpassFilter coefficients

Cell array support for the LatticeCoefficients, AllpassCoefficients, and WDFCoefficients properties of dsp.AllpassFilter System object will be removed in a future release. Use an N -by-1 or N -by-2 numeric array instead.

Inherited option removed from the input processing parameter

The Inherited option has been removed from the **Input processing** parameter in the Analytic Signal block.

Version History

In models created before R2016b that have **Input processing** set to Inherited, the parameter changes to Columns as channels (frame based) if you open the model in R2016b or after. If the block input is sample-based, nonscalar, and not a column vector, change **Input processing** to Elements as channels (sample based).

Frame status parameter removed from the Check Signal Attributes block

The **Frame status** parameter has been removed from the Check Signal Attributes block.

qfft object errors

Starting in R2016b, using the `qfft` object throws an error. Use the `dsp.FFT System` object instead.

dspstartup removed

`dspstartup` has been removed. Use the DSP Simulink Model Templates instead.

R2016a

Version: 9.2

New Features

Bug Fixes

Version History

DSP Unfolding for Mac: Generate multithreaded MEX files from MATLAB functions on Mac OS X

DSP unfolding is a technique to improve throughput through parallelization. In R2016a, `dspunfold` function implements DSP unfolding on Mac OS X platform as well as Windows and Linux. The multithreaded MEX file that `dspunfold` generates from the specified MATLAB function leverages the multicore CPU architecture of the host computer, and can improve speed significantly.

Faster FIR and Biquad Filters: Run faster simulations for system models that include FIR and biquad filters

These filters have significantly faster simulation speeds for multichannel inputs:

- `dsp.FIRFilter` System object and Discrete FIR Filter block in the `Direct form structure`
- `dsp.BiquadFilter` System object and Biquad Filter block in the `Direct form II transposed structure`

Fixed-Point Farrow Rate Converter: Design and simulate Farrow rate conversion filters using fixed-point data types

The `dsp.FarrowRateConverter` System object and Farrow Rate Converter block now support fixed-point data types.

Gigasample per Second (GSPS) Signal Processing: Increase throughput of HDL-optimized FFT and IFFT algorithms using frame input

You can increase the throughput of the FFT and IFFT calculation by using vector input and output ports. The internal algorithm computes the FFT or IFFT of each vector element in parallel.

The FFT implementation is now a Radix 2^2 architecture which improves performance for vector input. The table compares hardware implementation resources between the old Radix 2 architecture and the new Radix 2^2 architecture.

Architecture	Multipliers	Adders	Memory	Control Logic For Vector Input
Radix 2 Hybrid	$\log_4(N-1)$	$3 \times \log_4(N)$	$17N/16 - 1$	Complicated
Radix 2^2 (SDF)	$\log_4(N-1)$	$4 \times \log_4(N)$	$N - 1$	Simple

This change affects these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

HDL Optimizations for Biquad Filter: Reduce critical path or area when generating HDL from a subsystem that includes a Biquad Filter block

The Biquad Filter block is now included in subsystem optimizations for speed and area of the generated HDL. To specify resource sharing, streaming, and pipeline options, right-click the subsystem containing the Biquad Filter block and open the **HDL Code > HDL Properties** dialog box. To use these optimizations you must set the **Architecture** of the Biquad Filter block to **Fully parallel**. This feature requires an HDL Coder license.

The optimizations work the same way as the optimizations for the Discrete FIR Filter block. You can share resources between Biquad Filter and Discrete FIR Filter blocks in the same subsystem. See *Subsystem Optimizations for Filters*.

Differentiate a signal using the dsp.Differentiator System object and Differentiator block

`dsp.Differentiator` System object applies a direct form FIR full band differentiator filter on an input signal. The object uses an FIR equiripple filter design to design the differentiator filter. The Differentiator Filter block imports the functionality of this object into the Simulink environment.

Play audio data using the audioDeviceWriter System object and Audio Device Writer block

Play audio data on your computer's audio device using the `audioDeviceWriter` System object and Audio Device Writer block. Audio Toolbox provides enhanced functionality. For instance, you can customize the channel-to-speaker mapping and monitor the dropouts in the audio data. Audio Toolbox also adds support for low-latency ASIO drivers on Windows.

Specify coefficients in IIR Halfband Interpolator and IIR Halfband Decimator Blocks and System objects

You can now specify the coefficients of IIR halfband filters directly. To specify the coefficients, set the `Specification` property of the `dsp.IIRHalfbandInterpolator` and `dsp.IIRHalfbandDecimator` System objects to `Coefficients`. In the blocks, set the **Filter specification** parameter to `Coefficients`.

In this mode:

- You can specify the structure as `Minimum multiplier` or `Wave Digital Filter`. In both forms, you specify the coefficients through the applicable properties.
- The coefficients in the `Minimum multiplier` form are tunable, that is, you can change them even after the object is locked or during the model simulation.
- The first branch of the polyphase filter structure can be modeled as a pure delay.
- The last section of the second branch of the polyphase filter structure can contain a first-order section.

Customize the data limits of the Matrix Viewer block

You can now customize the x-axis and y-axis limits of the Matrix Viewer block. Changing the limits on the block dialog box updates the block axes in real time.

Code generation for wave digital filter structure in dsp.AllpassFilter System object

With the Structure property of the dsp.AllpassFilter System object set to Wave Digital Filter, you can:

- Generate C code from the System object.
- Import this System object into Simulink using the MATLAB System block.
- Specify the last section of the filter as first order by setting the TrailingFirstOrderSection property to true.
- Assign array data to the WDFCoefficients property.

Version History

Old MATLAB code with WDFCoefficients property set to a cell array continues to run in R2016a. However, it is recommended that you set this property to an array value.

Generate coefficients for multirate filters

Design an FIR interpolator, decimator, and rate converter using the designMultirateFIR function. To compute the coefficients of the multirate filter, this function uses an FIR Nyquist filter. Coefficients are computed based on the rate conversion factor provided as the input to the function.

Select the color of the noise in dsp.ColoredNoise System object

In the dsp.ColoredNoise System object, you can now specify the color of the noise to generate by setting the Color property to:

- White
- Pink
- Brown
- Blue
- Purple
- Custom

When you choose Custom, you can specify the power density of the noise through the InverseFrequencyPower property.

Full-precision setting for product data type of Biquad Filter

You can now specify biquad filters to use full-precision rules for the product data type. In the dsp.BiquadFilter System object, set the NumeratorProductDataType and

DenominatorProductDataType properties can be set to Full precision. In the Biquad Filter block, set the **Product output** parameter to Inherit via internal rule.

In the full-precision setting, the filter computes all internal arithmetic and output data types using full-precision rules. These rules provide more accurate fixed-point numerics that prevent quantization from occurring within the filter. Bits are added as needed so that no roundoff or overflow occurs.

Code generation for Subband Analysis and Subband Synthesis Filters

The dsp.SubbandAnalysisFilter and dsp.SubbandSynthesisFilter System objects can generate C code. When these objects run in single precision mode, you can also generate C code optimized for ARM Cortex-A and ARM Cortex-M processors. To generate code on ARM Cortex processors, you must have an Embedded Coder license.

Enhancements to Variable Fractional Delay

- The Variable Fractional Delay block and dsp.VariableFractionalDelay System object accept variable-size input signals. During the block simulation or when the object is locked, you can vary the number of samples per channel of the input signal. However, you cannot vary the number of channels.
- Interpolation precision has improved. To improve the precision, set the Interpolation method is set to FIR.

Multiple inputs for Spectrum Analyzer

The Spectrum Analyzer block and dsp.SpectrumAnalyzer System object accept more than one input port. In the Spectrum Analyzer block, set the number of inputs by selecting **File > Number of Input Ports**. For dsp.SpectrumAnalyzer, set the NumInputPorts property to the number of ports you want. For the block and System object, all inputs must have the same frame size. For the block only, all inputs must also have the same sample time.

Additional axes for Time Scope

The Time Scope block and dsp.TimeScope System object now default to 4 x 4 separate axes and allow up to 16 x 16 axes. To set the axes select **View > Layout**. Then, drag the axes grid to the desired number of axes and positions of those axes.

Set legend programmatically in Array Plot

In the Array Plot block and dsp.ArrayPlot System object, you can control the channel names from the MATLAB command line or from within MATLAB code. Define the channel names in a cell array. The plot legend pulls the names from this cell array. For the block, set the names in the ChannelNames parameter. For the System object, set the names in the ChannelNames property.

System object property display

How System objects properties are displayed at the MATLAB command line has changed.

- Fixed-point properties are displayed only if you click a link at the end of a System object property display.

- Properties are grouped as defined in the `getPropertyGroupsImpl` method.
- If `getPropertyGroupsImpl` defines multiple sections, only properties from the first section group are displayed. To display additional properties, click the link at the end of a System object property display. Section groups are defined using `matlab.system.display.SectionGroup`. Group titles are also displayed. To omit the “Main” title for the first group of properties, in the `matlab.system.display.SectionGroup` class, set `TitleSource` to `'Auto'`.

Version History

The `matlab.system.showFixedPointProperties` and `matlab.system.hideFixedPointProperties` functions have been removed. These functions controlled the display of fixed-point properties. If your code uses either of these functions, such as in a startup script, you now receive a warning. The **System Objects Preferences** panel in the MATLAB Preferences dialog box has also been removed. This panel was another way to set the fixed-point properties display.

System object enhancements to MATLAB System block

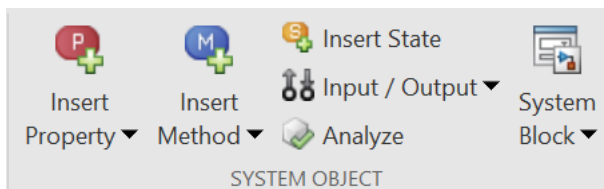
To implement these classes and methods for defining your own System objects, add them to your object's class definition file.

- Fixed-point data tab — The `showFiSettingsImpl` method adds a Data Types tab to the MATLAB System block dialog box. This tab includes options for fixed-point data settings.
- Model reference discrete sample time inheritance — The `allowModelReferenceDiscreteSampleTimeInheritanceImpl` method lets you specify whether a System object in a referenced model can inherit the sample time of the parent model. If your object uses discrete sample time in its algorithm, you set this method to `true` to allow inheritance.

Enhanced System Object Development with MATLAB Editor

Create System objects in the MATLAB Editor using code insertion and visualization options.

- Define your System object with options to insert properties, methods, states, inputs, and outputs.
- View and navigate the System object code with the Analyzer.
- Develop System block and preview block dialog box interactively (with Simulink only).



These coding tools are available when you open an existing System object or create a new System object with **New > System object**.

Functionality being removed

The following DSP features will be removed in a future release

Functions	Use This Instead
<code>fdesign.parmeq</code>	<code>fdesign.parmeq</code> function in Audio Toolbox.
<code>fdesign.octave</code>	<code>fdesign.octave</code> function in Audio Toolbox.
<code>fdesign.audioweighting</code>	<code>fdesign.audioweighting</code> function in Audio Toolbox.
<code>iirparameq</code>	<code>designParamEQ</code> function in Audio Toolbox.
<code>midicallback</code>	<code>midicallback</code> function in Audio Toolbox.
<code>midicontrols</code>	<code>midicontrols</code> function in Audio Toolbox.
<code>midiid</code>	<code>midiid</code> function in Audio Toolbox.
<code>midiread</code>	<code>midiread</code> function in Audio Toolbox.
<code>midisync</code>	<code>midisync</code> function in Audio Toolbox.

Blocks	Use This Instead
Audio Weighting Filter block	Audio Weighting Filter block in Audio Toolbox.
Octave Filter block	Octave Filter block in Audio Toolbox.
MIDI Controls block	MIDI Controls block in Audio Toolbox.
Parametric EQ Filter block	Parametric EQ Filter block in Audio Toolbox.
From Audio Device block	Audio Device Reader block in Audio Toolbox. Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in Audio Device Reader block using the Driver parameter.
To Audio Device block	Audio Device Writer block. Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in Audio Device Writer block using the Driver parameter.

System object	Use This Instead
<code>dsp.ParametricEQFilter</code>	<code>multibandParametricEQ</code> object in Audio Toolbox.
<code>dsp.AudioRecorder</code>	<code>audioDeviceReader</code> object in Audio Toolbox. Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in <code>audioDeviceReader</code> object using the <code>Driver</code> property.

System object	Use This Instead
dsp.AudioPlayer	<p>audioDeviceWriter object .</p> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in audioDeviceWriter object using the Driver property.</p>
mfilt object	Use One of The Following
mfilt.linearinterp	<ul style="list-style-type: none"> • dsp.FarrowRateConverter object with PolynomialOrder property set to 1. This value achieves linear interpolation. • dsp.FIRInterpolator object with InterpolationFactor property set to 2. • dsp.CICInterpolator object with NumSections and InterpolationFactor properties set to 2.
qfft object	Use This Instead
qfft	dsp.FFT

Version History

Existing instances of these functions, System objects, and blocks continue to run. For new instances of the functionality, use the replacement feature.

R2015b

Version: 9.1

New Features

Bug Fixes

Version History

DSP Unfolding: Generate a multi-threaded MEX File from a MATLAB function

Generate a multi-threaded MEX file from a MATLAB function, using the `dspunfold` function. DSP unfolding is a technique to improve throughput through parallelization. The multi-threaded MEX file leverages the multicore CPU architecture of the host computer and can improve speed significantly.

HDL Optimizations for Discrete FIR Filter: Implement FIR filters in hardware at higher frequencies or using fewer resources

You can now optimize speed and area of the generated HDL for the Discrete FIR Filter block. Right-click the subsystem containing the Discrete FIR Filter block, and open the **HDL Code > HDL Properties** dialog to specify resource sharing, streaming, and pipeline options. You can use these optimizations when the **Architecture** is Fully parallel. This feature requires an HDL Coder license. See Discrete FIR Filter.

Array Plot Block: Visualize array and vector data

An Array Plot block for plotting arrays and vectors has been added to the `dspnks4` library.

Additional Multirate Filters: Design Halfband, CIC compensation, and HDL-optimized FIR rate conversion filters

Implement FIR and IIR halfband interpolator and decimator using these new design features:

- The `dsp.IIRHalfbandInterpolator` and `dsp.IIRHalfbandDecimator` System objects use efficient polyphase IIR halfband structure to interpolate and decimate an input signal, by a factor of two. Allpass filters, used in the polyphase branches, use elliptic or quasilinear phase design methods. You can use these System objects to implement the synthesis portion and the analysis portion of a two-band filter bank. For a Simulink implementation, use the IIR Halfband Interpolator and IIR Halfband Decimator blocks
- FIR Halfband Interpolator and FIR Halfband Decimator blocks use an FIR equiripple design to construct the halfband filter; they use an efficient polyphase implementation to filter the input. These blocks implement the functionality of the `dsp.FIRHalfbandInterpolator` and `dsp.FIRHalfbandDecimator` System objects.

CIC Compensation Interpolator and CIC Compensation Decimator blocks compensate for the passband droop and wide transition region of CIC filters. These blocks implement the functionality of the `dsp.CICCompensationInterpolator` and `dsp.CICCompensationDecimator` System objects.

FIR Rate Conversion HDL Optimized block upsamples, filters, and downsamples a signal using an efficient polyphase FIR structure. The block operates on one sample at a time and provides hardware control signals to pace the flow of samples in and out of the block. The `dsp.HDLFIRRateConverter` System object provides equivalent MATLAB functionality. Both the block and System object support HDL code generation, when used with HDL Coder.

Conversion Filter Blocks: Convert the rate of signals in Simulink models

- The Farrow Rate Converter block uses a farrow structure to implement a polynomial-based filter that does the sample rate conversion. This converter can handle arbitrary rate change factors efficiently. This block implements the functionality of `dsp.FarrowRateConverter` System object in Simulink.
- The Sample-Rate Converter block uses polyphase filters, which are adapted to interpolation and decimation with an integer factor, and to fractional rate conversions with a low conversion factor. This block implements the functionality of the `dsp.SampleRateConverter` System object.

Implement FIR and IIR filters in Simulink, using the Lowpass Filter and Highpass Filter blocks

- The Lowpass Filter block designs FIR or IIR low pass filters with minimum-order or specified order options. This block implements the functionality of the `dsp.LowpassFilter` System object.
- The Highpass Filter block designs FIR or IIR high pass filters with minimum-order or specified order options. This block implements the functionality of the `dsp.HighpassFilter` System object.

Estimate power spectrum and power spectral density using the Spectrum Estimator block

The Spectrum Estimator block combines the functionality of `dsp.SpectrumEstimator` System object, and the functionality included in the Spectrum Analyzer scope block, such as the ability to specify resolution bandwidth, automatic buffering with custom overlap, max-hold and min-hold spectra, and power units.

Automatic selection of filter coefficients for FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks

The FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks can now choose their filter coefficients automatically. In the block dialog box, under **Coefficient source**, select **Auto**. The block then uses the coefficients of an FIR Nyquist filter, designed for the rate conversion factor of the block. For more information, see the 'Choose Filter Coefficients Automatically' section in the block reference page.

This feature is supported for HDL code generation from the FIR Interpolation and FIR Decimation blocks. HDL code generation requires an HDL Coder license.

Visualize the frequency response of the underlying filters in the DSP System Toolbox blocks

You can now use FVTool to visualize the frequency response of the filter in these filter blocks:

- DC Blocker
- Variable Bandwidth FIR Filter and Variable Bandwidth IIR Filter
- Notch-Peak Filter

- Parametric EQ Filter
- Digital Down-Converter and Digital Up-Converter
- Farrow Rate Converter
- Sample-Rate Converter
- FIR Halfband Interpolator and FIR Halfband Decimator
- IIR Halfband Interpolator and IIR Halfband Decimator
- CIC Compensation Interpolator and CIC Compensation Decimator
- Lowpass Filter and Highpass Filter

In the block dialog box, click the **View Filter Response** button. FVTool opens and displays the filter frequency response, which is computed based on the block dialog box parameters. Changes made to these parameters update the FVTool response. For more information, see the 'View Filter Response' section in the block reference page.

Specify the window length and window overlap in Cross-Spectrum Estimator and Discrete Transfer Function Estimator blocks

Cross-Spectrum Estimator block and Discrete Transfer Function Estimator block now accept **Window length** and **Window Overlap** parameters. These parameters buffer the input data into overlapping segments, enabling you to implement the Welch spectrum estimation algorithm.

Select the color of the noise in Colored Noise block

In the Colored Noise block, you can now specify the color of the noise to generate by setting the **Noise color** parameter to:

- White
- Pink
- Brown
- Blue
- Purple
- Custom

When you choose Custom, you can specify the power density of the noise through the **Power of inverse frequency** parameter.

New functionality added to the `dsp.SpectrumEstimator` System object

`dsp.SpectrumEstimator` System object now has these properties, which are also found in the Spectrum Analyzer:

- `PowerUnits` — Units in which the `dsp.SpectrumEstimator` displays the power values, specified as dBm, dBW, or Watts.
- `ReferenceLoad` — Reference load, in ohms, that `dsp.SpectrumEstimator` uses as a reference to compute the power values.

-
- `OutputMaxHoldSpectrum` — The maximum-hold spectrum at each frequency bin. To compute this value, `dsp.SpectrumEstimator` keeps the maximum value of all the power spectrum estimates.
 - `OutputMinHoldSpectrum` — The minimum-hold spectrum at each frequency bin. To compute this value, `dsp.SpectrumEstimator` keeps the minimum value of all the power spectrum estimates.

Generate C code from `dsp.AllpassFilter` and import the System object into Simulink using the MATLAB System block

When the `Structure` property of `dsp.AllpassFilter` System object is set to `Minimum multiplier` or `Lattice`, you can:

- Generate C code from the System object.
- Import this System object into Simulink using MATLAB System block.
- Specify the last section of the filter as first order by setting the `TrailingFirstOrderSection` property to `true`.
- Assign array data to the `AllpassCoefficients` and `LatticeCoefficients` properties.

When the `Structure` property is set to `Wave Digital Filter`, the `WDFCoefficients` property accepts cell arrays only. This configuration does not support code generation and cannot interface with the MATLAB System block.

Version History

Old MATLAB code with `AllpassCoefficients` and `LatticeCoefficients` properties set to a cell array, continues to run in R2015b. However, it is recommended that you set these properties to an array value. The option to set the `Structure` property to `Wave Digital Filter` will be removed in a future release.

`dsp.CICDecimator` and `dsp.CICInterpolator` System objects support single and double data types

Inputs and outputs to the `dsp.CICDecimator` and `dsp.CICInterpolator` System objects can now be single or double data types, in addition to the fixed-point data type.

Frame-based signal logging in structure formats in Time Scope block

For frame-based, single-port signals and multiport signals, the Time Scope block now supports logging in the `Structure with Time` format and `Structure` formats.

Scientific notation in Time Scope

The Time Scope block and System object now uses scientific E notation in the measurement panels. Previously they displayed SI units.

Performance improvements for FFT, IFFT and notch peak filters

Simulation speed has improved for:

- `dsp.FFT`, `dsp.IFFT`, and `dsp.NotchPeakFilter` System objects.
- FFT and IFFT blocks.

Floating-point support and optional valid port for HDL-optimized NCO

The NCO HDL Optimized block and the `dsp.HDLNCO` System object now support floating-point (double or single) input data for use with Fixed-Point Designer™ tools.

When a data input is fixed point, or when no data input ports are enabled, the block computes a fixed-point output waveform based on the fixed-point parameters. When a data input is floating-point, the block ignores the fixed-point parameters, and computes a double-precision output waveform.

The `validIn` port on the NCO HDL Optimized block and the `validIn` argument to the `step` method of the `dsp.HDLNCO` System object are now optional. This port or argument is enabled by default.

HDL Code Generation from filterbuilder

Using Filter Design HDL Coder™, you can generate HDL code from the filter objects designed in `filterbuilder`. On the **Code Generation** tab, click **Generate HDL** to set HDL code generation options and generate code. You can generate HDL code from the following filter types:

Filter	Structure	System object
FIR	Direct form Direct form transposed Direct form symmetric Direct form antisymmetric	<code>dsp.FIRFilter</code>
FIR Decimator	Direct form Direct form transposed	<code>dsp.FIRDecimator</code>
FIR Interpolator	Direct form Direct form transposed	<code>dsp.FIRInterpolator</code>
IIR	Direct form I Direct form I transposed Direct form II Direct form II transposed	<code>dsp.BiquadFilter</code>
CIC Decimator		<code>dsp.CICDecimator</code>
CIC Interpolator		<code>dsp.CICInterpolator</code>

Simulink templates for ARM Cortex-A and ARM Cortex-M processors

Simulink model templates for ARM Cortex-A and ARM Cortex-M processors have been added to the template gallery. These templates enable the reuse of settings, including configuration parameters for models with optimized code generation for ARM Cortex-A or ARM Cortex-M processors. For successful code generation, the provided model requires DSP System Toolbox Support Package for ARM Cortex-A Processors or DSP System Toolbox Support Package for ARM Cortex-M Processors. Instead of using the new model default canvas, select a template model to get started. The templates create models that use best practices and previous solutions to common problems. To avoid having to reconfigure your model for your environment or application, use a provided template or create templates from your existing models.

For more information, see [Configure the Simulink Environment for Signal Processing Models](#).

ROI processing removed

The **Enable ROI processing** parameter has been removed from the following blocks in the `dspstat3` library. In addition, the `ROIProcessing` property has been removed from the corresponding System objects.

- Maximum block and `dsp.Maximum` System object
- Minimum block and `dsp.Minimum` System object
- Mean block and `dsp.Mean` System object
- Standard Deviation block and `dsp.StandardDeviation` System object
- Variance block and `dsp.Variance` System object

If you have Computer Vision System Toolbox™ software installed, use the equivalent block from the `visionstatistics` library. To select the rectangular region, use the Simulink Selector block.

Version History

Blocks in old models implementing ROI processing no longer use this functionality in R2015b. To disable ROI processing from these models, at the MATLAB command prompt, enter `set_param(blockpath, 'roiEnable', 'off')`.

Frame-based processing changes

As part of the changes in how DSP System Toolbox handles frame-based processing, certain block options have been removed.

The following sections provide more detailed information about the specific R2015b DSP System Toolbox software changes for frame-based processing:

- “Inherited Option Removed from the Input Processing Parameter” on page 14-7
- “Sample-Based Row Vector Processing Changes” on page 14-8
- “Blocks Emit Sample-Based Signals Only” on page 14-9

Inherited Option Removed from the Input Processing Parameter

Inherited option has been removed from the **Input processing** parameter in these blocks:

- FIR Interpolation
- FIR Decimation
- Two-Channel Synthesis Subband Filter
- Two-Channel Analysis Subband Filter

Version History

In old models that have the **Input processing** parameter set to `Inherited`, the behavior has not changed. However, it is recommended that you update this parameter to either `Columns as channels` (frame based) or `Elements as channels` (sample based). For information on

how to choose the **Input processing** parameter, see “Input processing parameter set to Inherited” on page 15-8.

Sample-Based Row Vector Processing Changes

In previous releases, some of the blocks that treated sample-based row vector inputs as columns had a **Treat sample-based row input as column** check box which explicitly enabled this behavior. Other blocks automatically processed sample-based row vector inputs as column vectors.

In R2015b, these blocks now treat sample-based row vector inputs as an n channel input, where n is the number of samples in the input signal.

- **Treat sample-based row input as column** check box has been removed from these blocks.
 - Maximum
 - Mean
 - Median
 - Minimum
 - Normalization
 - RMS
 - Standard Deviation
 - Variance
- These blocks no longer treat sample-based row vectors as single-channel column vectors:
 - Autocorrelation
 - Autocorrelation LPC
 - Burg AR Estimator
 - Burg Method
 - Complex Cepstrum
 - Convolution
 - Correlation
 - Covariance AR Estimator
 - Covariance Method
 - DCT
 - FFT
 - IDCT
 - IFFT
 - Levinson-Durbin
 - LPC to LSF/LSP Conversion
 - LPC to/from Cepstral Coefficients
 - LPC to/from RC
 - LPC/RC to Autocorrelation
 - LSF/LSP to LPC Conversion
 - Modified Covariance AR Estimator

-
- Modified Covariance Method
 - Polynomial Stability Test
 - Real Cepstrum
 - Yule-Walker AR Estimator
 - Yule-Walker Method

Version History

In R2015b, old models that treat sample-based row vector inputs as columns, might produce unexpected results. To ensure consistent results, place a Math Function block, with the **Function** parameter set to `transpose`, in front of the affected block. The Math Function block transposes the sample-based row vector into a column vector, which is then input into the affected block.

Blocks Emit Sample-Based Signals Only

- 1 Source blocks:** In previous releases, the following source blocks emitted frame-based signals (double lines) when `samples per frame` was greater than 1 and sample-based signals (single lines), when `samples per frame` was 1. In R2015b, the source blocks now emit sample-based signals (single lines), irrespective of the value of `samples per frame`.
 - Sine Wave
 - Chirp
 - From Audio Device
 - Random Source
 - Discrete Impulse
 - NCO
 - Signal From Workspace
 - Triggered Signal From Workspace
- 2 Nonsource blocks:** In previous releases, the following blocks emitted frame-based signals even when the input was sample-based. In R2015b, these blocks emit only sample-based signals.
 - Buffer
 - CIC Interpolation
 - Dyadic Analysis Filter Bank
 - Dyadic Synthesis Filter Bank
 - DWT
 - IDWT
 - Inverse Short-Time FFT
 - Delay Line

Version History

Blocks emitting frames in models created in previous releases continue to emit frames in R2015b. To update these blocks to emit samples, use Simulink Upgrade Advisor. To ensure consistent results with a previous release, use the Frame Conversion block after the affected block.

Features removed, replaced and renamed

Blocks removed and replaced

DSP Block Removed	Use This Instead	Backward Compatibility
Pulse Shaping Filter (with Filter Type set to Decimator and Pulse shape set to Raised Cosine or Squared Root Raised Cosine).	Raised Cosine Receive Filter in Communications Toolbox™.	None Old models using the Pulse Shaping Filter block still run.
Pulse Shaping Filter (with Filter Type set to Interpolator and Pulse shape set to Raised Cosine or Squared Root Raised Cosine).	Raised Cosine Transmit Filter in Communications Toolbox.	None Old models using the Pulse Shaping Filter block still run.

DSP Block Replaced	Use This Instead	Backward Compatibility
CIC Compensator (with Filter Type set to Decimator)	CIC Compensation Decimator	None Old models using the CIC Compensator block still run.
CIC Compensator (with Filter Type set to Interpolator)	CIC Compensation Interpolator	None Old models using the CIC Compensator block still run.
Halfband Filter (with Impulse response set to FIR, and Filter Type set to Decimator)	FIR Halfband Decimator	None Old models using the Halfband Filter block still run.
Halfband Filter (with Impulse response set to FIR, and Filter Type set to Interpolator)	FIR Halfband Interpolator	None Old models using the Halfband Filter block still run.
Halfband Filter (with Impulse response set to IIR, and Filter Type set to Decimator)	IIR Halfband Decimator	None Old models using the Halfband Filter block still run.
Halfband Filter (with Impulse response set to IIR, and Filter Type set to Interpolator)	IIR Halfband Interpolator	None Old models using the Halfband Filter block still run.

The Lowpass Filter and Highpass Filter blocks have been modified to match the functionality and interface of the `dsp.LowpassFilter` and `dsp.HighpassFilter` System objects.

Removal of `adapfilt` objects

`adapfilt` objects are removed and cause a warning message. In a future release, these objects will be removed entirely, so you should now use the corresponding System object. See “Removal of `adapfilt` objects” on page 15-23 for a list of replacement objects.

Removal of mfilt objects

mfilt objects will be removed in a future release. Instead, use their System object counterparts, which are more powerful and support code generation.

mfilt Object	Use This System object Instead
mfilt.cascade	dsp.FilterCascade
mfilt.cicdecim	dsp.CICDecimator
mfilt.cicinterp	dsp.CICInterpolator
mfilt.farrowsrc	dsp.FarrowRateConverter
mfilt.fftfirinterp	dsp.FIRInterpolator (approximates mfilt.fftfirinterp)
mfilt.firdecim	dsp.FIRDecimator
mfilt.firtdecim	dsp.FIRDecimator
mfilt.firinterp	dsp.FIRInterpolator
mfilt.firsrc	dsp.FIRRateConverter
mfilt.holdinterp	dsp.CICInterpolator (with NumSections = 1, approximates mfilt.holdinterp)
mfilt.iirdecim	dsp.CICInterpolator dsp.IIRHalfbandDecimator
mfilt.iirinterp	dsp.CICInterpolator dsp.IIRHalfbandInterpolator
mfilt.iirwdfdecim	dsp.IIRHalfbandDecimator (approximates mfilt.iirwdfdecim)
mfilt.iirwdfinterp	dsp.IIRHalfbandInterpolator (approximates mfilt.iirwdfinterp)
mfilt.linearinterp	dsp.CICInterpolator (with NumSections = 2, approximates mfilt.linearinterp)

System Object Propagation Mixin Methods Renamed

System object propagation mixin methods have been renamed. You use propagation methods to control System object data specifications in Simulink. If you use an old method name, an error occurs. You can use `sobjupdate` to update the following renamed methods to the new methods.

Renamed Propagation Method	New Propagation Method
inputComplexity	propagatedInputComplexity
outputComplexity	propagatedOutputComplexity
inputDataType	propagatedInputDataType
outputDataType	propagatedOutputDataType
inputSize	propagatedInputSize

Renamed Propagation Method	New Propagation Method
outputSize	propagatedOutputSize
inputFixedSize	propagatedInputFixedSize
outputFixedSize	propagatedOutputFixedSize

R2015a

Version: 9.0

New Features

Bug Fixes

Version History

Audio Latency Reduction: Significantly reduce latency for audio hardware I/O in MATLAB and Simulink

Improvements to the audio I/O infrastructure significantly reduce latency.

To determine audio latency on your system, see [Measuring Audio Latency](#).

Filter Design Enhancements: Design high-order IIR parametric EQ filter, variable bandwidth FIR and IIR filters, Digital Down-Converter and Digital Up-Converter blocks

Implement FIR and IIR filters conveniently with minimal design options, using the following new filter design features:

- `iirparameq` function designs IIR biquad parametric equalizer filters.
- `dsp.LowpassFilter` System object designs FIR or IIR lowpass filters with minimum-order or with specified order options.
- `dsp.HighpassFilter` System object designs FIR or IIR highpass filters with minimum-order or with specified order options.

Implement in Simulink several new filters, a power spectrum estimator, signal operations and generate colored noise:

- The Variable Bandwidth FIR Filter and Variable Bandwidth IIR Filter blocks implement the functionality of `dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter` System objects. These blocks allow you to vary the passband while filtering. Also, they enable you to tune the filter in a computationally efficient way while preserving the filter structure.
- The Parametric EQ Filter block implements a parametric equalizer with tunable gain, bandwidth, and center frequency. These filters are widely used in audio processing applications. The Parametric EQ Filter block implements the functionality of `dsp.ParametricEQFilter` System object. This block replaces the Param EQ block, found in `dspfdesign` library.
- The Notch-Peak Filter block implements a notching or peaking IIR filter in Simulink. Notch-Peak filters are used in many signal processing applications, including tone removal and removal of power line interference. With the Notch-Peak filter block, you can use tunable parameters to control the center frequencies and 3-dB bandwidths of the notches and peaks. The Notch-Peak Filter block implements the functionality of the `dsp.NotchPeakFilter` System object. This block replaces the Peak-Notch Filter block, found in `dspfdesign` library.
- The Cross Spectrum Estimator block uses Welch's modified periodogram method to compute the cross-power spectrum of inputs. This block implements the functionality of the `dsp.CrossSpectrumEstimator` System object.
- The Digital Down-Converter and Digital Up-Converter blocks provide tools to design decimation and interpolation filters, and simplify the steps required to implement down conversion and up conversion, in Simulink. These blocks implement the functionality of the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects.
- The Colored Noise block generates a colored noise signal in Simulink. With this block, you can generate noise with a $1/f^\alpha$ power spectral density. Set α equal to any value in the interval $[-2,2]$. Specifying $\alpha = 1$ results in pink noise. Specifying $\alpha = 2$ produces Brownian noise. This block implements the functionality of the `dsp.ColoredNoise` System object.

DSP Simulink Model Templates: Configure the Simulink environment for digital signal processing models

DSP Simulink model templates enable reuse of settings, including configuration parameters. Create models from templates to encourage best practices and take advantage of previous solutions to common problems. Instead of the default canvas of a new model, select a template model to help you get started. Choose `blank`, `basic` or `audio` model templates to create a skeletal model using settings recommended for DSP System Toolbox.

You can use built-in templates or create templates from models that you already configured for your environment or application.

For more information, see [Configure the Simulink Environment for Signal Processing Models](#).

Streaming Scope Improvements: Plot in stem mode, access log x-axis scaling, customize sample rate, and use infinite data support

The following improvements have been made to streaming scopes:

- Legend settable programmatically in Spectrum Analyzer block and `dsp.SpectrumAnalyzer` System object.
- Stem plot mode for Spectrum Analyzer block and System object.
- Log frequency axis for Spectrogram mode in Spectrum Analyzer block and System object.
- Vector-tunable frequency offset in Spectrum Analyzer block and System object.
- Custom sample rate in Spectrum Analyzer block.
- Variable-size support in Spectrum Analyzer System object.
- Infinite data support in Time Scope block.
- Log x-axis scaling in the `dsp.ArrayPlot` System object.

Library for HDL Supported DSP Blocks: Find all blocks that support HDL

In the Simulink Library Browser, use the **DSP System Toolbox HDL Support** library to find all the DSP System Toolbox blocks that support HDL code generation.

Alternatively, at the MATLAB command prompt, enter `dsphdllib` to open this library.

All blocks in this library have their parameters configured for HDL code generation. To generate HDL code, you must have an HDL Coder license.

C Code Generation of DSP Algorithms for ARM Cortex-A and Cortex-M processors: Generate optimized and faster performing C code using Embedded Coder

In R2015a, using Embedded Coder, you can generate C code optimized for ARM processors using these DSP blocks and System objects:

ARM Cortex-A processors

- `dsp.LowpassFilter` System object, with `FilterType` set to FIR
- `dsp.HighpassFilter` System object, with `FilterType` set to FIR
- Variable Bandwidth FIR Filter block

ARM Cortex-M processors

- `dsp.LowpassFilter` System object, with `FilterType` set to FIR or IIR
- `dsp.HighpassFilter` System object, with `FilterType` set to FIR or IIR
- `dsp.VariableBandwidthIIRFilter` System object, with `FilterType` set to Lowpass or Highpass
- Variable Bandwidth IIR Filter block, with `FilterType` property set to Lowpass and Highpass
- Variable Bandwidth FIR Filter block

Performance Improvements

Simulation speed has improved for:

- `dsp.FIRDecimator`, `dsp.AllpassFilter`, and `dsp.CoupledAllpassFilter` System objects
- FIR Decimation and Downsample blocks

Updated Time Scope block toolbar and menus

The following Time Scope block menu and toolbar items have been added or updated:

- **Zoom Out** option added to **Tools** menu and toolbar
- Axes scaling options grouped into an **Axes Scaling** submenu of the **Tools** menu
- **Scale Axes Limits** option indicates axes to be scaled
- **Save** and **Restore Axes Limits** added to **Axes Scaling** submenu
- Warnings appear in drop-down area above the plot, instead of in a separate dialog box

Specify block filter characteristics through System objects

You can now specify the filter characteristics of FIR Decimation, FIR Interpolation, FIR Rate Conversion, CIC Decimation, CIC Interpolation, and Biquad Filter blocks using System objects.

In the block dialog box, under **Coefficient source**, select `Filter` object . You can now specify the name of the System object in the filter object variable. See the **Specify Multirate Filter Object** section in each of these block reference pages.

Block	System object to specify
FIR Decimation	<code>dsp.FIRDecimator</code>
FIR Interpolation	<code>dsp.FIRInterpolator</code>
FIR Rate Conversion	<code>dsp.FIRRateConverter</code>
CIC Decimation	<code>dsp.CICDecimator</code>

Block	System object to specify
CIC Interpolation	<code>dsp.CICInterpolator</code>

See the **Specify Discrete-Time Filter Object** section in this block reference page.

Block	System object to specify
Biquad Filter	<code>dsp.BiquadFilter</code>

This feature is supported for HDL code generation from the FIR Decimation, FIR Interpolation, CIC Decimation, CIC Interpolation, and Biquad Filter blocks. HDL code generation requires a HDL Coder license.

Discrete Transfer Function Estimator block

You can now configure the Discrete Transfer Function Estimator block to estimate the output coherence, or spectral coherence, of the input and output signal of the estimated transfer function.

In the dialog box, select the `Output magnitude squared coherence estimate` parameter to compute the spectral coherence between the input and output signals. Spectral coherence is a useful metric for estimating transfer functions, especially for the purposes of system identification.

Specify filter coefficients as an input to the FIR Decimation block

You can now provide filter coefficients to the FIR Decimation block. In the block dialog box, under **Coefficient Source**, select `Input Port`. Coefficient values are tunable (can change during simulation), while their properties must remain constant.

See the **Provide Filter Coefficients through Input port** section in the FIR Decimation block reference page.

Enhanced code generation for CIC Decimation and CIC Interpolation filter blocks

Code generated from CIC Decimation and CIC Interpolation filter blocks can now support data types that have word lengths greater than 32 bits.

HDL support for 'inherit via internal rule' data type setting on FIR Decimation and Interpolation blocks

FIR Decimation and FIR Interpolation blocks now support HDL code generation with data types specified by **Inherit via internal rule**. HDL code generation requires a HDL Coder license.

Improvements for creating System objects

The following improvements have been made to creating your own System objects:

- Number of allowable code generation inputs increased to 32
- Code generation support for unbounded variable-size vectors

- `isInputSizeLockedImpl` method for specifying whether the input port dimensions are locked
- `matlab.system.display.Action` class, used in the `getPropertyGroupsImpl` method, to define a MATLAB System block button that can call a System object method
- `getSimulateUsingImpl` and `showSimulateUsingImpl` methods to set the value of the `SimulateUsing` parameter and specify whether to show the `SimulateUsing` parameter in the MATLAB System block

Min/Max logging instrumentation for float-to-fixed-point conversion of DSP System objects

Convert the following DSP System Toolbox System objects to fixed-point using the Fixed-Point Converter app (requires a Fixed-Point Designer license):

- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.FIRFilter` (direct form transposed)
- `dsp.LUFactor`
- `dsp.VariableFractionalDelay`
- `dsp.Window`

Propose and apply data types for these System objects based on simulation range data. During the conversion process, you can view simulation minimum and maximum values and proposed data types for these System objects. You can also view whole number information and histogram data. You cannot propose data types for these System objects based on static range data.

Provide variable-size input to the Delay System object

The `dsp.Delay` System object now supports variable-size input signals. When the input is a variable-size signal, the number of input rows can change during run time without having to call `release` method between two calls to the `step` method, while the number of channels must remain fixed.

See [What Is Variable-Size Data?](#) for information on variable-size signals .

Estimate output coherence of Transfer Function Estimator System object

You can now use `dsp.TransferFunctionEstimator` System object to estimate the magnitude squared coherence, or spectral coherence, of the input and output signals of the estimated transfer function. To compute the spectral coherence, specify the `OutputCoherence` property of the System object as `true`. Spectral coherence is a useful metric for estimating transfer functions, especially for the purposes of system identification.

Specify filter coefficients as an input to the FIR Decimator System object

Provide filter coefficients to the `dsp.FIRDecimator` System object as an input argument. Specify `NumeratorSource` property of the System object as one of `Property` (default) or `Input port`.

When you specify `Input port`, the filter object requires the numerator coefficients to be specified as the third argument at every step.

See the `NumeratorSource` property of `dsp.FIRDecimator` for details.

Bit growth to avoid overflow in HDL-optimized FFT and IFFT

When $1/N$ scaling is disabled, the FFT algorithm grows the internal word length by 1 bit after each butterfly stage. This adjustment avoids overflow. This change affects the following blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Fixed-point support for FIR Half-band Interpolator and FIR Half-band Decimator System objects

Implement HDL focused fixed-point operations on `dsp.FIRHalfbandInterpolator` and `dsp.FIRHalfbandDecimator` System objects.

Updated cost method for filter System objects

The `cost` method for filter System objects is now a structure with the following fields:

- `NumCoefficients`
- `NumStates`
- `MultiplicationsPerInputSample`
- `AdditionsPerInputSample`

Frame-based processing

As part of general product-wide changes pertaining to Frame-Based processing, certain block options that use the `frame` attribute of the input signal now cause an error.

The following sections provide more detailed information about the specific R2015a DSP System Toolbox software changes for frame-based processing:

- “Input processing parameter set to Inherited” on page 15-8
- “Rate options parameter set to Inherit from input” on page 15-20
- “Treat $M \times 1$ and unoriented sample-based signals as parameter set to M channels” on page 15-20
- “Save 2-D signals as parameter set to Inherit from input” on page 15-20
- “Find the histogram over parameter set to Inherited” on page 15-21
- “Sample-based processing parameter set to Pass through” on page 15-21
- “Running difference parameter set to Inherit from input” on page 15-22

Input processing parameter set to Inherited

Setting **Input processing** parameter to **Inherited** now causes an error in these blocks:

- Biquad Filter
- FIR Interpolation
- FIR Decimation
- Edge Detector
- Analytic Signal
- Variable Integer Delay
- Variable Fractional Delay
- Zero Crossing
- Two-Channel Analysis Subband Filter
- Two-Channel Synthesis Subband Filter
- CIC Interpolation
- Upsample
- Downsample
- Repeat
- Unwrap
- Minimum, when you set **Mode** to **Running**
- Maximum, when you set **Mode** to **Running**
- Mean, when you select **Running mean** check box
- Standard Deviation, when you select **Running standard deviation** check box
- Variance, when you select **Running variance** check box
- RMS, when you select **Running RMS** check box
- Cumulative Sum, when you set **Sum input along** to **Channels** (running sum)
- Cumulative Product, when you set **Multiply input along** to **Channels** (running product)

Version History

To ensure consistent results for models created in previous releases, set **Input processing** to:

- **Columns as channels (frame based)**, for frame-based input signals (double-line)
- **Elements as channels (sample based)**, for sample-based input signals (single-line)

After compiling the model, frame-based signals appear as double lines. Sample-based signals appear as single lines.

For models created in R2015a:

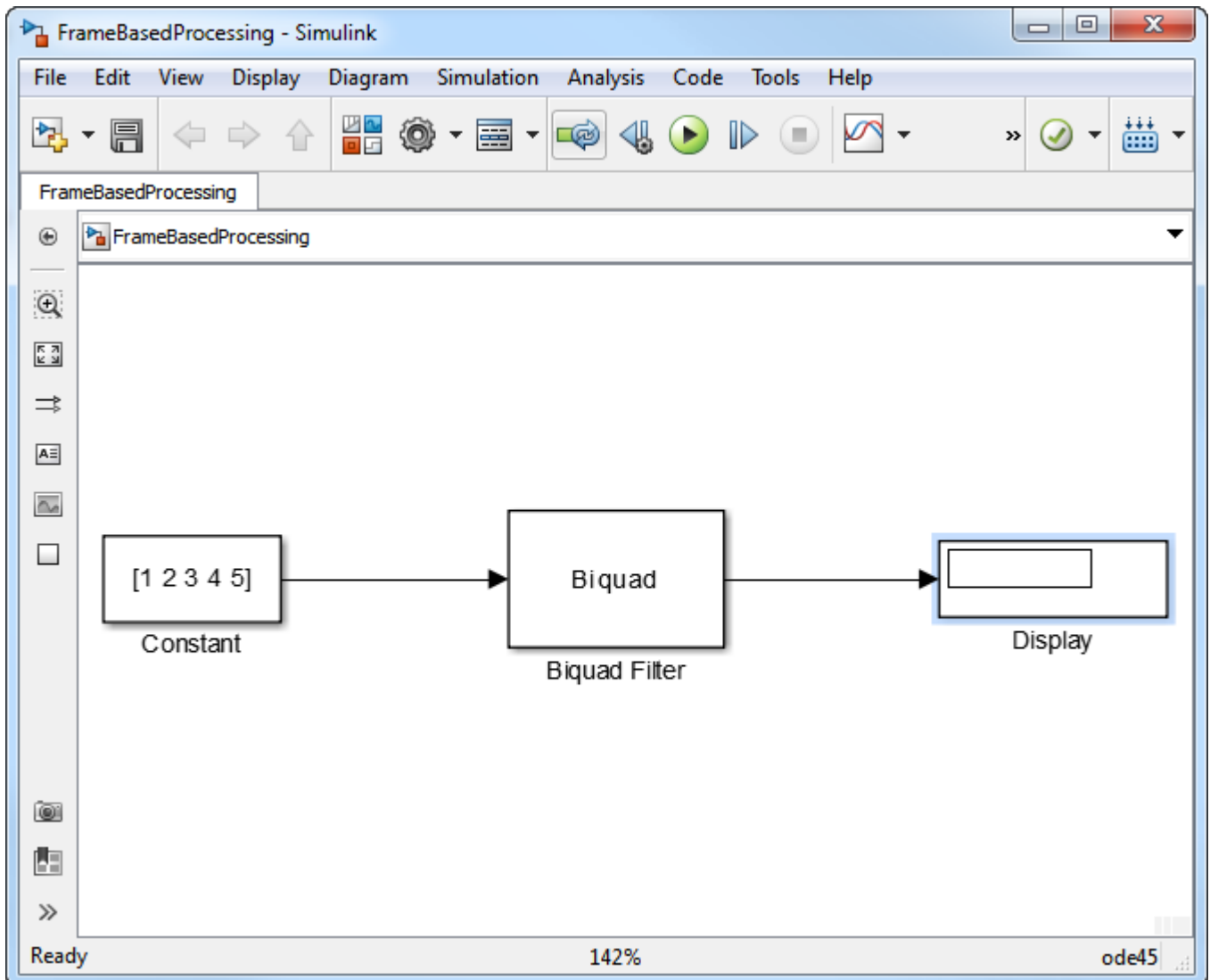
- To treat each column of the input signal as an independent channel, set **Input processing** to **Columns as channels (frame based)**.
- To treat each element of the input signal as an independent channel, set **Input processing** to **Elements as channels (sample based)**.

Simulink Upgrade Advisor

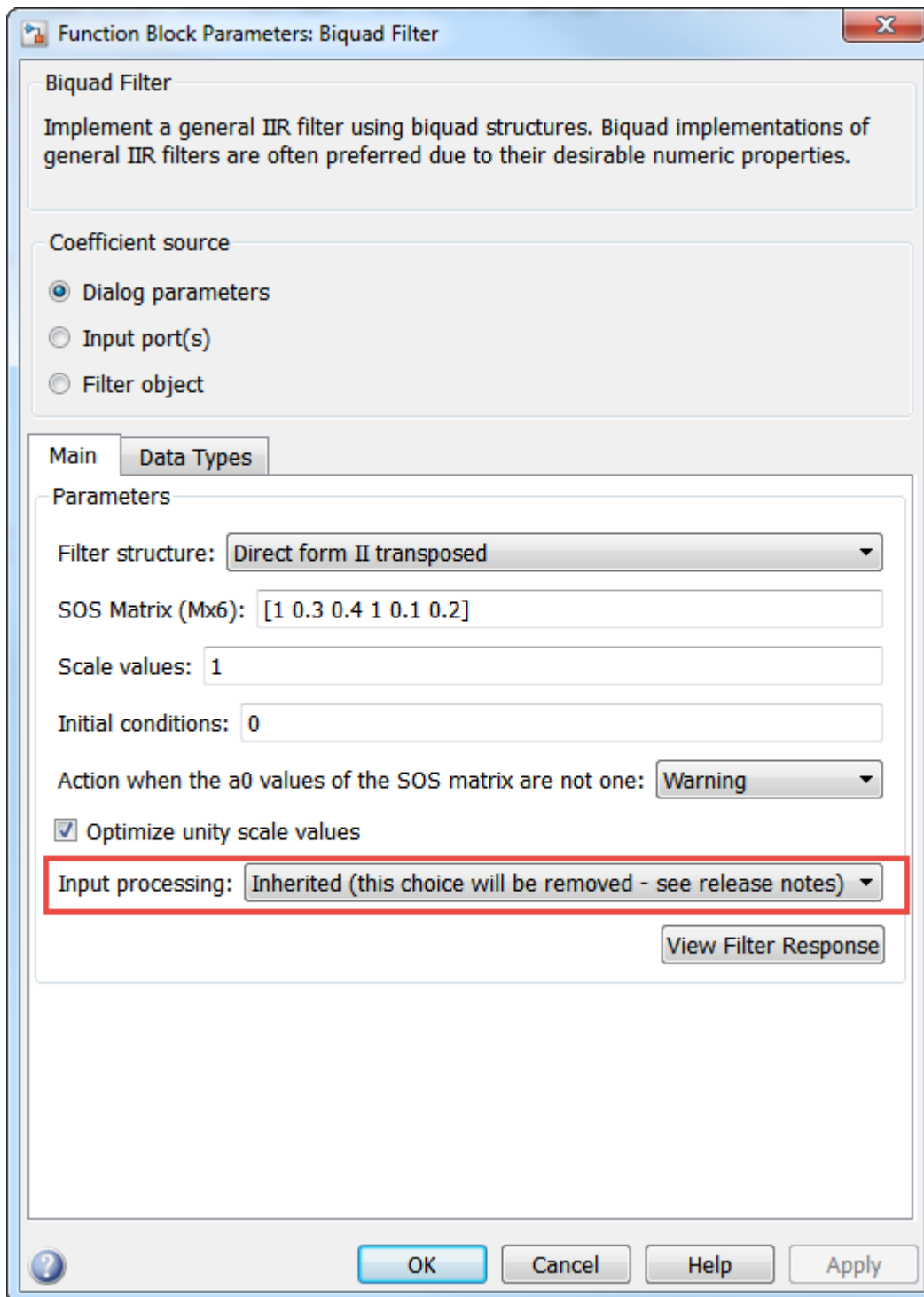
If you are not sure which **Input processing** option applies to your model and choose Inherited instead, use the Simulink Upgrade Advisor to update your model.

1 Update blocks in a model

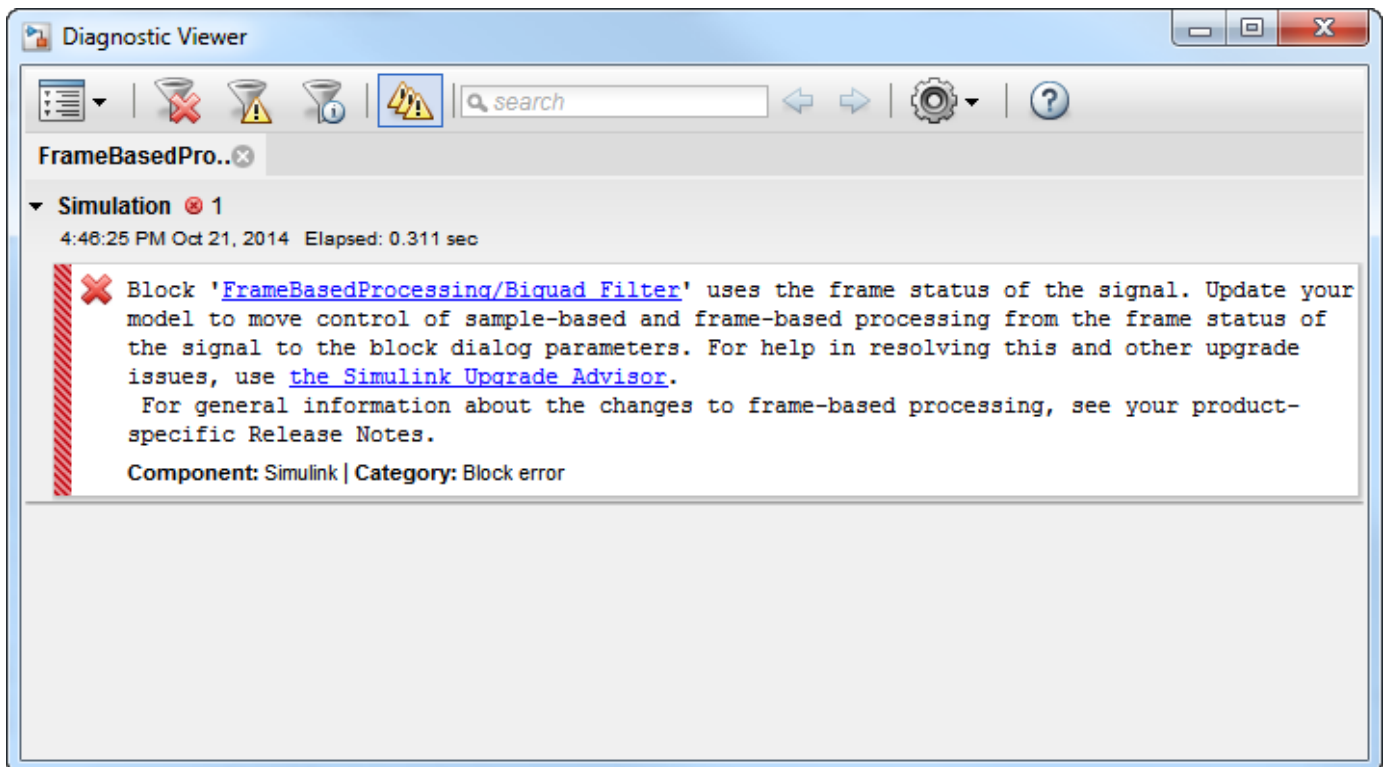
In this model, the Biquad Filter block uses frame-based processing.



In the Biquad Filter dialog box, **Input processing** is set to Inherited.

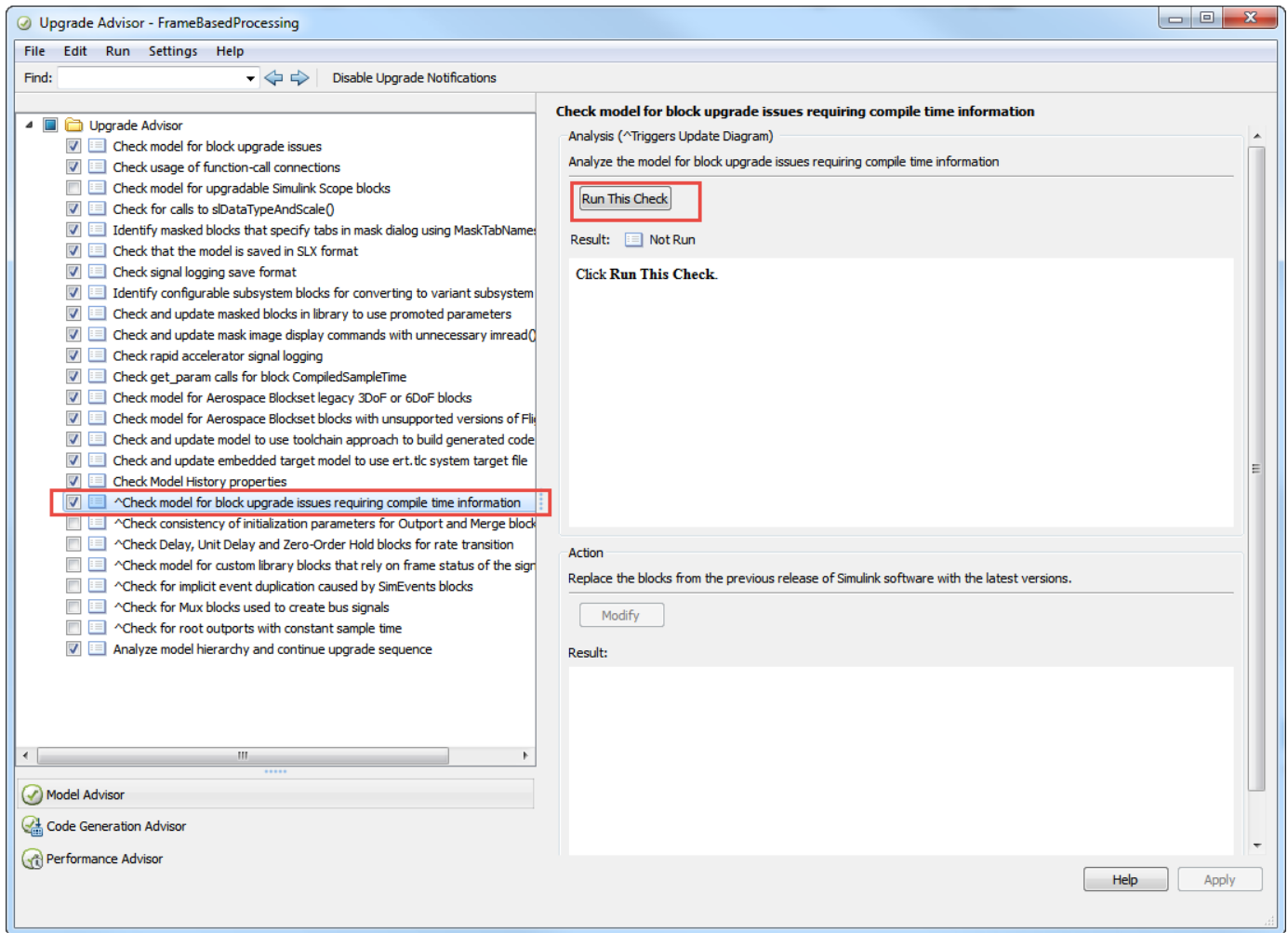


When you run the model, Simulink issues an error message.

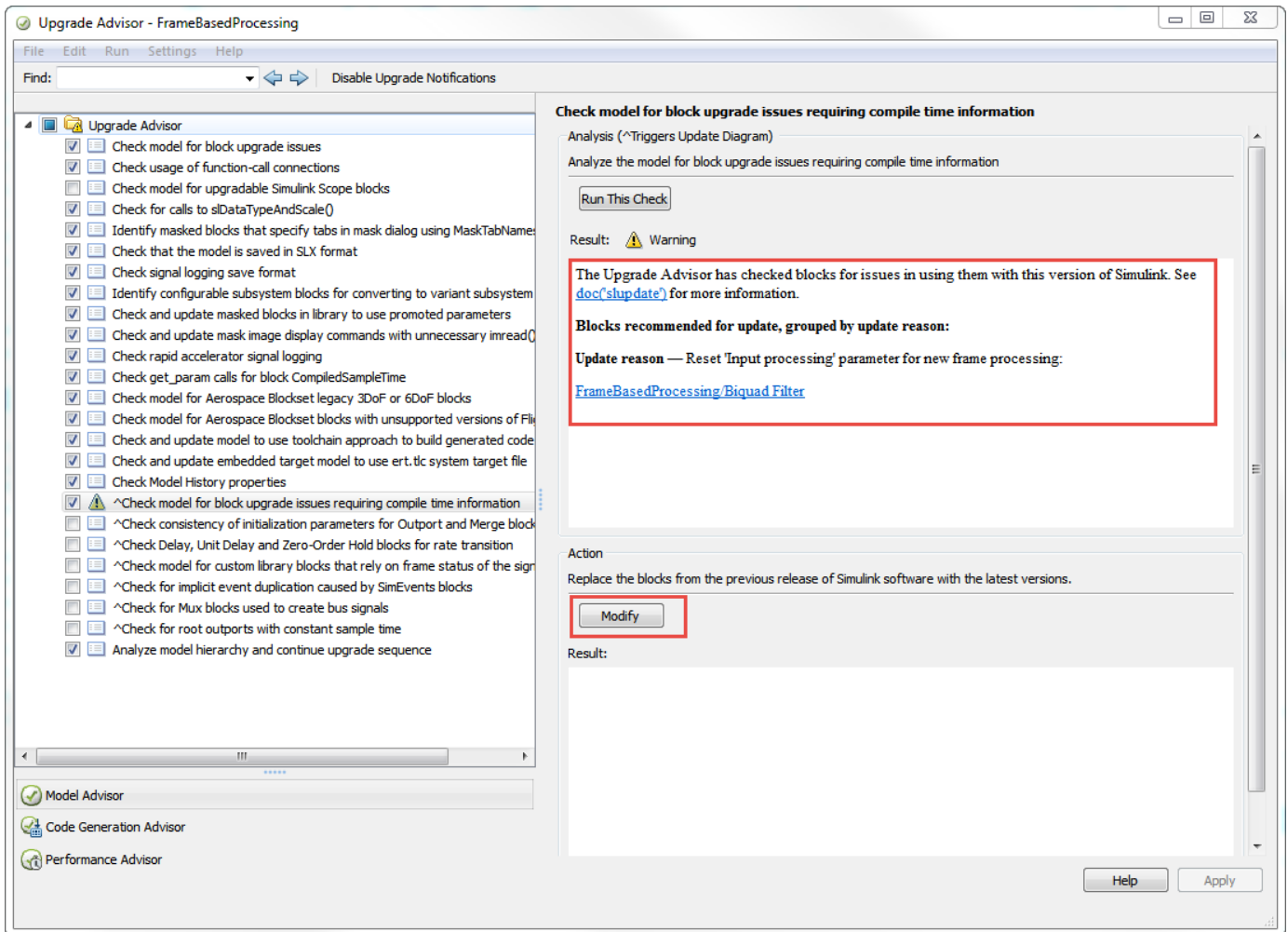


From the error message, you can open the Simulink Upgrade Advisor by clicking the hyperlink.

Then you can select the Check model for block upgrade issues requiring compile time information check and click **Run this Check**.



The Upgrade Advisor runs the check on all the blocks in the model, recommends an update for the needed blocks, and gives the reason for the update.



Click **Modify** to update the FrameBasedProcessing/Biquad Filter block.

Because the input signal is sample-based, in the Biquad Filter block, the Upgrade Advisor changes **Input processing** to Elements as Channels (Sample based).

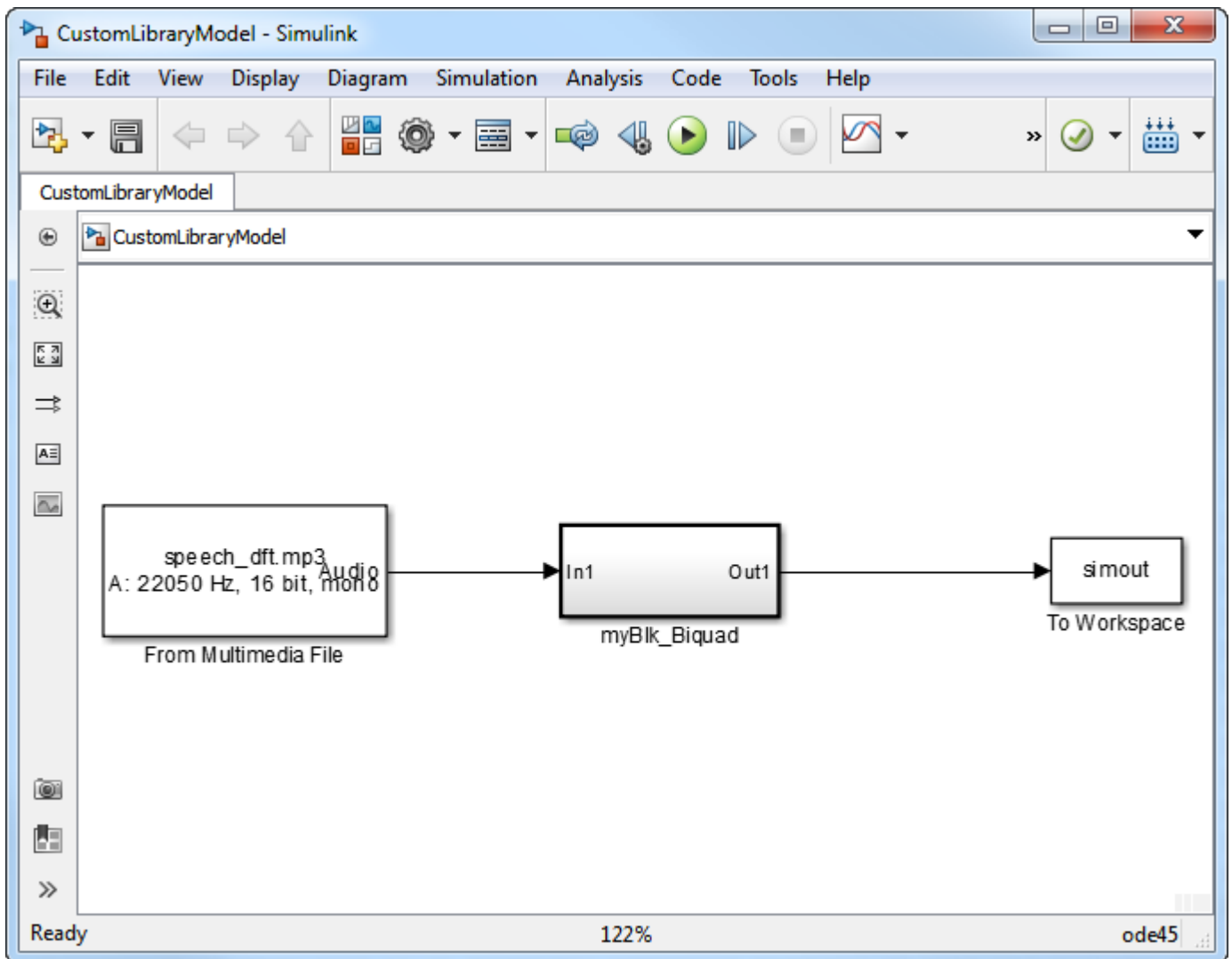
2 Update blocks in a custom library

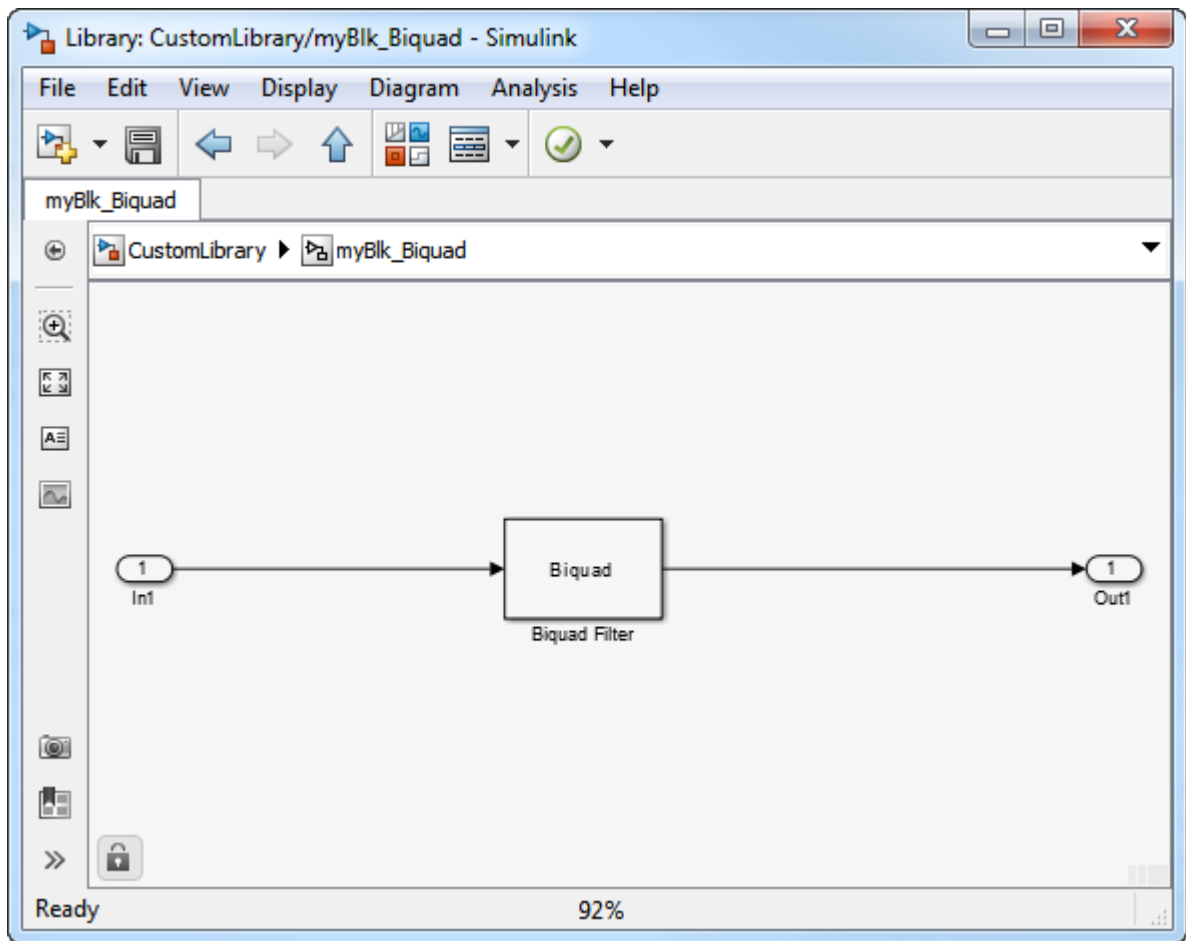
When you select the Check model for block upgrade issues requiring compile time information check box, the Upgrade Advisor does not update blocks in custom libraries.

Custom libraries are Simulink block libraries that you can create to reuse your blocks and subsystems in one or more models. For more information, see Create Custom Block Libraries.

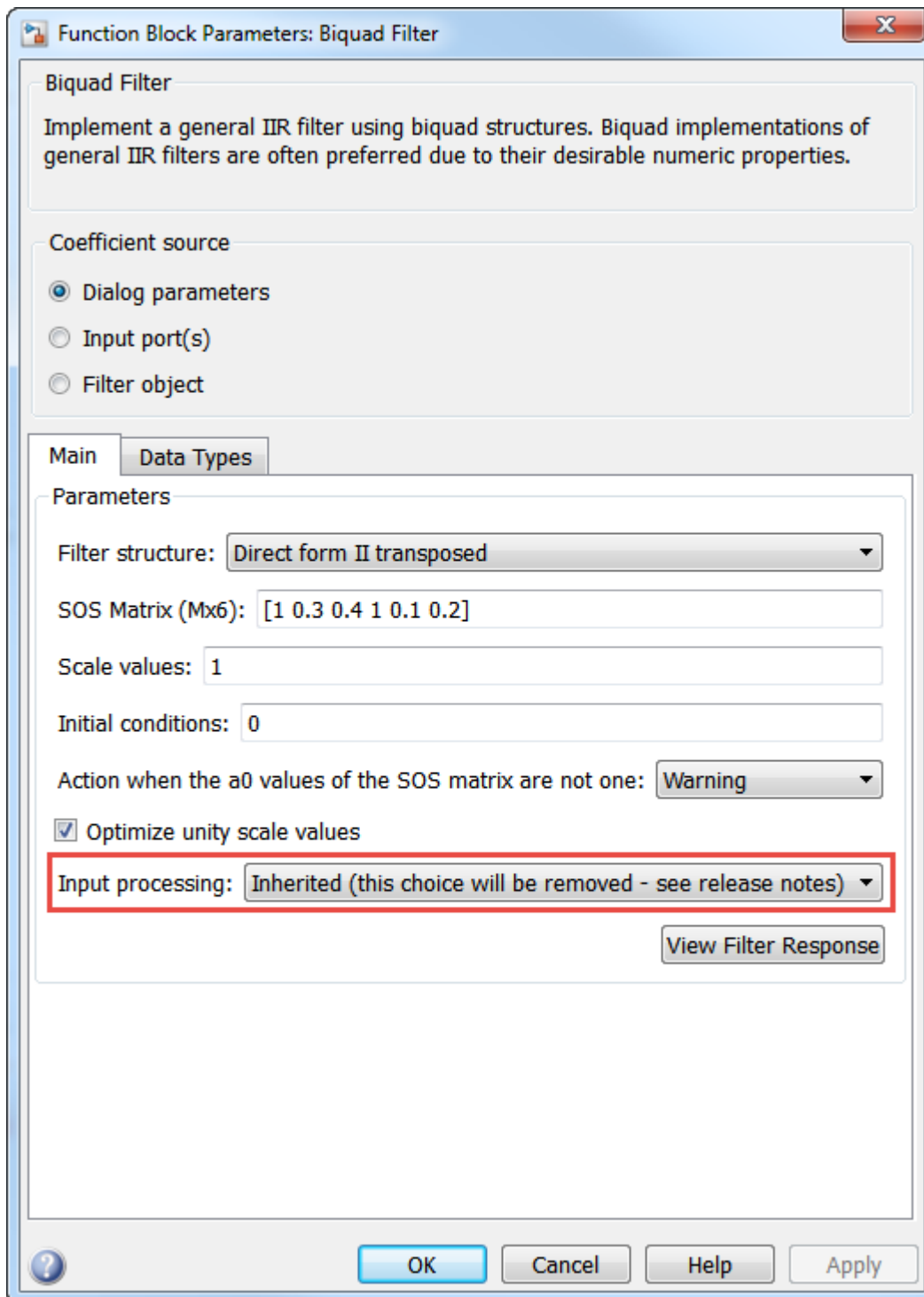
To analyze frame-based processing related errors in custom library blocks, open the Simulink Upgrade Advisor and select the Check model for custom library blocks that rely on frame status of the signal, and click **Run This Check**. This check does not update the library blocks. It analyzes the blocks, recommends fixes, and gives reasons for the fixes. You must make the fixes manually.

In this model, myBlk_Biquad is a block from the custom library. It contains a Biquad Filter block, which is one of the blocks that causes an error when you set **Input processing** to Inherited.

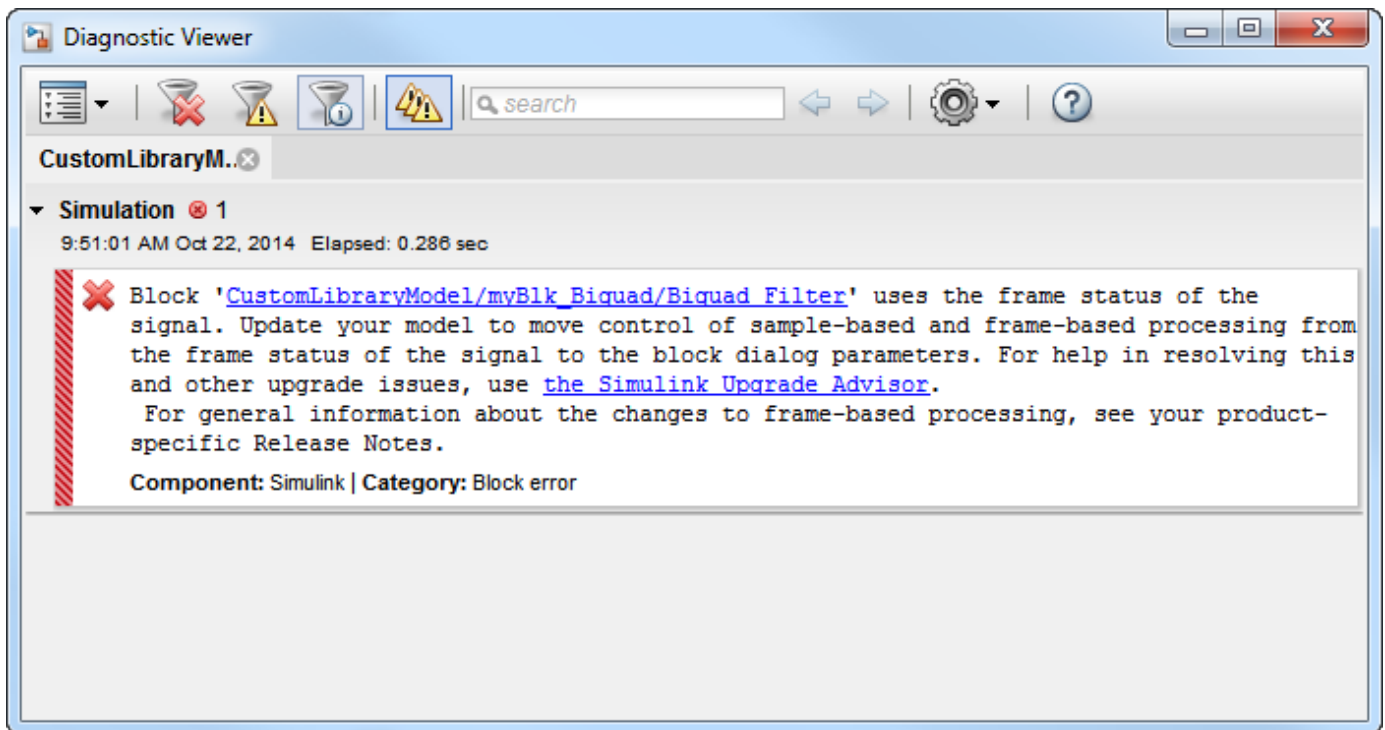




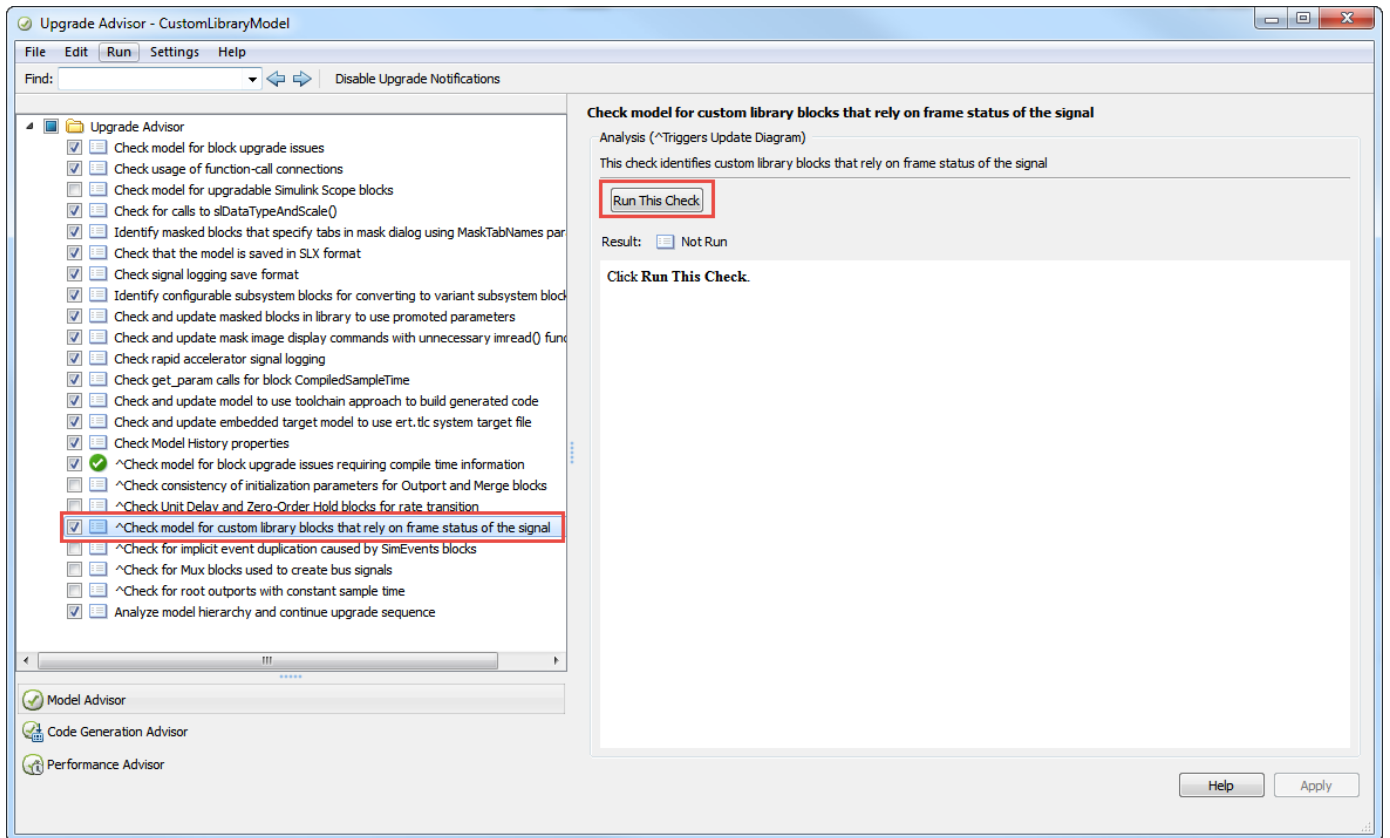
In the Biquad Filter dialog box, **Input processing** is set to Inherited.



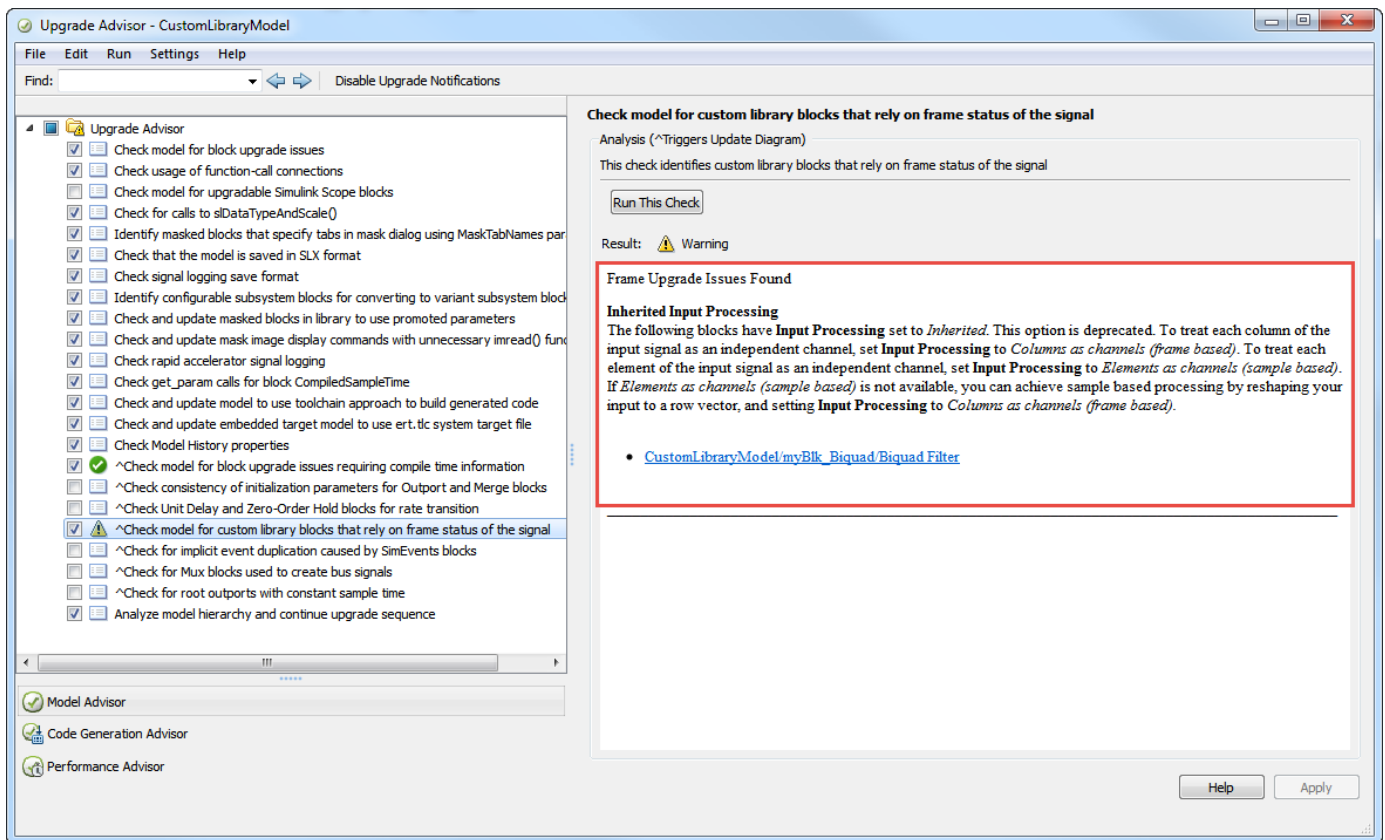
When you run the model, Simulink issues an error message.



To resolve this error message, reopen Simulink Upgrade Advisor, select the Check model for custom library blocks that rely on frame status of the signal check and click **Run This Check**.



The Upgrade Advisor runs the check on all the blocks in the custom library subsystem block, recommends an update for the needed blocks, and gives the reason for the update.



For the CustomLibraryModel/myBlk_Biquad/Biquad Filter block, the Upgrade Advisor recommends changing **Input Processing** to either **Elements as channels (sample based)** or **Columns as channels (frame based)**. To make the change, you update the parameter in the library block, myBlk_Biquad/Biquad Filter, not in the specific instance of the block in the model. The model now runs successfully.

If the custom library block (in this case, myBlk_Biquad block) is used for frame-based processing in all the models that use it, set the **Input Processing** parameter to **Columns as channels (frame based)** in the library block. Else, set this parameter to **Elements as channels (sample based)**.

If the library block is used for frame-based processing in some models and sample-based processing in others, you can add a parameter to the mask of the library block and configure this parameter to choose one of **Columns as channels (frame based)** or **Elements as channels (sample based)**. Choosing the appropriate option in the block mask should set the **Input Processing** parameter of underlying block to **Columns as channels (frame based)** or **Elements as channels (sample based)**.

Another approach is to promote the option chosen for **Input Processing** parameter from underlying block (in this case, Biquad Filter block) to the mask of the library block. This is known as parameter promotion. For details on parameter promotion, see Promote Underlying Block Parameters to Mask.

Rate options parameter set to Inherit from input

In the CIC Decimation block, setting the **Rate options** parameter to Inherit from input causes an error.

Version History

To ensure consistent results for models created in previous releases, set **Rate options** to:

- Allow multirate processing, for sample-based input signals
- Enforce single-rate processing, for frame-based input signals

For models created in R2015a:

- To run the block in single-rate mode, set **Rate options** to Enforce single-rate processing.
- To run the block in multirate mode, set **Rate options** to Allow multirate processing.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Treat Mx1 and unoriented sample-based signals as parameter set to M channels

Setting **Treat Mx1 and unoriented sample-based signals as** parameter to M channels now causes an error in these blocks:

- Buffer
- Delay Line
- Overlap-Save FFT Filter
- Overlap-Add FFT Filter
- Short-Time FFT

Version History

To ensure consistent results for models created in previous releases, reshape the input to be a 1-by-M vector, and set **Treat Mx1 and unoriented sample-based signals as** to One channel.

Save 2-D signals as parameter set to Inherit from input

In the Triggered To Workspace block, setting the **Save 2-D signals as** parameter to Inherit from input now causes an error.

Version History

To ensure consistent results for models created in previous releases, set **Save 2-D signals as** to

- **3-D array (concatenate along third dimension)**, for sample-based input signals

-
- **2-D array (concatenate along first dimension)**, for frame-based input signals

For models created in R2015a:

- For frame-based processing, set **Save 2-D signals as** to **2-D array (concatenate along first dimension)**.
- For sample-based processing, set **Save 2-D signals as** to **3-D array (concatenate along third dimension)**.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Find the histogram over parameter set to Inherited

In the Histogram block, setting the **Find the histogram over** to Inherited now causes an error.

Version History

To ensure consistent results for models created in previous releases, set **Find the histogram over** to:

- **Entire input**, for sample-based input signals
- **Each column**, for frame-based input signals

For models created in R2015a:

- To compute the histogram for each column of the input independently, set **Find the histogram over** to **Each column**.
- To compute the histogram over the entire input, set the **Find the histogram over** to **Entire input**.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Sample-based processing parameter set to Pass through

In the Unbuffer block, setting the **Sample-based processing** to Pass through now causes an error.

Version History

Unbuffer the M -by- N matrix input into a 1-by- N output vector by setting **Sample-based processing** to Same as frame based.

If you want the input to pass through, remove the Unbuffer block from the model.

Running difference parameter set to Inherit from input

In the Difference block, setting the **Running difference** to Inherit from input now causes an error.

Version History

To ensure consistent results for models created in older releases, set **Running difference** to:

- No, for sample-based input signals
- Yes, for frame-based input signals

For models created in R2015a:

- To compute the difference between adjacent elements in the current input, set **Running difference** to No.
- To compute the running difference across consecutive inputs, set **Running difference** to Yes.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Features removed, replaced, and duplicated

Blocks replaced, removed, and available in additional libraries

DSP Block Replaced	Replaced With	Backward Compatibility — What Happens When You Run Models Containing This Block?
Delay	Delay block in Simulink library	When there is an exact match in functionality, the Simulink Delay block replaces the DSP Delay block automatically.
Param EQ	Parametric EQ Filter	None Old models using the Param EQ block still run.
Peak-Notch Filter	Notch-Peak Filter	None Old models using Peak-Notch Filter block still run.

DSP Block Removed	Use This Block Instead	Backward Compatibility
Transpose	Math Function block in Simulink library	None The Math Function block replaces the Transpose block automatically.
Complex Exponential	Trigonometric Function block in Simulink library	None The Trigonometric Function block replaces the Complex Exponential block automatically.
Constant Diagonal Matrix	Constant block in Simulink library	None The Constant block replaces the Constant Diagonal Matrix block automatically.

DSP Block Available in New Library	
NCO and NCO HDL Optimized block	Blocks are now available in <code>dsprcs4</code> library in addition to <code>dpsigops</code> library.

Removal of adaptfilt objects

adaptfilt objects will be removed in a future release. Instead, use their System object counterparts, which are more powerful and support code generation.

adaptfilt Object	Use This System object Instead
adaptfilt.lms	dsp.LMSFilter
adaptfilt.nlms	
adaptfilt.se	
adaptfilt.sd	
adaptfilt.ss	
adaptfilt.blms	dsp.BlockLMSFilter
adaptfilt.rls	dsp.RLSFilter
adaptfilt.qdrls	
adaptfilt.swrls	
adaptfilt.hrls	
adaptfilt.hswrls	
adaptfilt.ftf	dsp.FastTransversalFilter
adaptfilt.swftf	

adaptfilt Object	Use This System object Instead
adaptfilt.ap adaptfilt.apru adaptfilt.bap	dsp.AffineProjectionFilter
adaptfilt.gal adaptfilt.lsl adaptfilt.qrdlsl	dsp.AdaptiveLatticeFilter
adaptfilt.filtxlms	dsp.FilteredXLMSFilter
adaptfilt.fdaf adaptfilt.ufdaf	dsp.FrequencyDomainAdaptiveFilter
adaptfilt.blmsfft	Will be removed in a future release
adaptfilt.adjlm adaptfilt.dlms	Will be removed in a future release
adaptfilt.pbfdaf adaptfilt.pbufdaf	Will be removed in a future release
adaptfilt.tdafdct adaptfilt.tfafdft	Will be removed in a future release

Functionality changed or being removed for blocks and System objects

Removal of sample mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects will be removed in a future release. System objects containing this property will then work only in frame-based processing mode. See [What Is Frame-Based Processing?](#) for more information. Effective R2015a, modifying this property throws a warning for these System objects:

- `dsp.AllpoleFilter`
- `dsp.AnalyticSignal`
- `dsp.BiquadFilter`
- `dsp.Buffer`
- `dsp.CumulativeProduct`
- `dsp.CumulativeSum`
- `dsp.Delay`
- `dsp.FIRFilter`
- `dsp.IIRFilter`
- `dsp.MatFileReader`

- `dsp.MatFileWriter`
- `dsp.Maximum`
- `dsp.Mean`
- `dsp.Minimum`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseUnwrapper`
- `dsp.RMS`
- `dsp.SignalSink`
- `dsp.StandardDeviation`
- `dsp.VariableFractionalDelay`
- `dsp.VariableIntegerDelay`
- `dsp.Variance`
- `dsp.ZeroCrossingDetector`

In the `dsp.CumulativeProduct` and `dsp.CumulativeSum` System objects, the default value of `FrameBasedProcessing` property is true.

Option to specify filter coefficients from Digital Up Converter and Digital Down Converter System objects being removed

`FilterSpecification` and its related properties will be removed in a future release from the `dsp.DigitalUpConverter` and `dsp.DigitalDownConverter` System objects. The System objects then will not allow you to specify coefficients for individual stages.

System object	Properties Being Removed
<code>dsp.DigitalUpConverter</code>	<ul style="list-style-type: none"> • <code>FilterSpecification</code> • <code>FirstFilterCoefficients</code> • <code>SecondFilterCoefficients</code> • <code>FirstFilterCoefficientsDataType</code> • <code>SecondFilterCoefficientsDataType</code> • <code>CustomFirstFilterCoefficientsDataType</code> • <code>CustomSecondFilterCoefficientsDataType</code>
<code>dsp.DigitalDownConverter</code>	<ul style="list-style-type: none"> • <code>FilterSpecification</code> • <code>SecondFilterCoefficients</code> • <code>ThirdFilterCoefficients</code> • <code>SecondFilterCoefficientsDataType</code> • <code>ThirdFilterCoefficientsDataType</code> • <code>CustomSecondFilterCoefficientsDataType</code> • <code>CustomThirdFilterCoefficientsDataType</code>

Removal of OutputDataType and OverflowAction properties for CIC Compensation Interpolator and Decimator System objects

The `OutputDataType` and `OverflowAction` properties for `dsp.CICCompensationInterpolator` and `dsp.CICCompensationDecimator` System objects have been removed. The `OutputDataType` property of these System objects is now always `Same word length as input`. For these System objects, the word length matches the input word length, and the fraction length is computed to give the best possible precision to the data. You no longer have access to this parameter.

R2014b

Version: 8.7

New Features

Version History

Optimized C code generation for ARM Cortex-A Ne10 library from MATLAB and Simulink with DSP System Toolbox Support Package for ARM Cortex-A Processors

This release adds code-generation support for ARM Cortex-A processors in MATLAB for select blocks and System objects. With the supported blocks and System objects, you can generate optimized C code that calls the Ne10 library function and compiles to provide an executable to run on ARM Cortex-A processors. To use the DSP System Toolbox Support Package for ARM Cortex-A Processors, you must have the following products:

- DSP System Toolbox
- Embedded Coder
- MATLAB Coder

To design in Simulink, you must also have these products:

- Simulink
- Simulink Coder

The following DSP System Toolbox blocks and System objects support the Ne10 library:

- Discrete FIR Filter
- FIR Decimation
- FIR Interpolation
- FFT
- IFFT
- `dsp.FIRFilter`
- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.FFT`
- `dsp.iFFT`
- `dsp.VariableBandwidthFIRFilter`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`
- `dsp.CICCompensationDecimator`
- `dsp.CICCompensationInterpolator`
- `dsp.DigitalDownConverter`
- `dsp.DigitalUpConverter`
- `dsp.SampleRateConverter`

For more information, see [Support Package for ARM Cortex-A Processors](#).

System objects for DSP System Toolbox Support Package for ARM Cortex-M Processors

This release adds support to generate optimized C code for the following System objects on ARM Cortex-M processors

- `dsp.VariableBandwidthFIRFilter`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`
- `dsp.CICCompensationDecimator`
- `dsp.CICCompensationInterpolator`
- `dsp.DigitalDownConverter`
- `dsp.DigitalUpConverter`
- `dsp.SampleRateConverter`

Fixed-point support for Biquad Filter on DSP System Toolbox Support Package for ARM Cortex-M Processors

This release adds fixed-point support to generate optimized C code for the Biquad Filter block and `dsp.BiquadFilter` System object on ARM Cortex-M processors. The supported data formats are Q15 and Q31.

Multirate filters: Sample and Farrow Rate Converter, CIC Compensation Interpolator/Decimator, and FIR Halfband Interpolator/Decimator System objects

This release adds the following multirate filter System objects:

- `dsp.SampleRateConverter`
- `dsp.FarrowRateConverter`
- `dsp.CICCompensationInterpolator`
- `dsp.CICCompensationDecimator`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`

Tunable coefficients and variable-size input available on FIR Interpolator System object and block

The FIR Interpolation block and the `dsp.FIRInterpolator` System object now support tunable coefficients and variable-size input, enabling you to specify filter coefficients from the input port.

Variable-size input available on FIR Decimator System object and block

The FIR Decimation block and the `dsp.FIRDecimator` System object now support variable-size input.

Min/Max logging instrumentation for float-to-fixed-point conversion of commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter

You can now convert the following DSP System Toolbox System objects to fixed point using the Fixed-Point Converter app (requires a Fixed-Point Designer license).

- `dsp.BiquadFilter`
- `dsp.FIRFilter`, direct form only
- `dsp.FIRRateConverter`
- `dsp.LowerTriangularSolver`
- `dsp.UpperTriangularSolver`
- `dsp.ArrayVectorAdder`

You can propose and apply data types for these System objects based on simulation range data. During the conversion process, you can view simulation minimum and maximum values and proposed data types for these System objects. You can also view whole number information and histogram data. You cannot propose data types for these System objects based on static range data.

HDL-optimized FFT and IFFT System objects and HDL-optimized Complex to Magnitude-Angle System object and block

This release introduces:

- `dsp.HDLFFT` and `dsp.HDLIFFT` System objects for the fast Fourier transform and inverse FFT, optimized for HDL code generation
- Complex to Magnitude-Angle HDL Optimized block and `dsp.ComplexToMagnitudeAngle` System object for converting complex inputs to magnitude and phase angle, optimized for HDL code generation using the CORDIC algorithm

Real input, bit-reversed output, reset input available on HDL-optimized FFT and IFFT

The following blocks and System objects now support real input, enable you to select or disable bit-reversed output, and provide an optional reset input:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Option to synthesize lookup table to ROM available on HDL-optimized FFT and IFFT blocks

To enable this feature, right-click the block, select **HDL Code > HDL Block Properties** and set **LUTRegisterResetType** to none.

The option to synthesize LUT to a ROM is not available on System objects.

Reduced latency of HDL-optimized FFT and IFFT

The FFT HDL Optimized and IFFT HDL Optimized blocks take fewer cycles to compute one frame of output than in previous releases. For instance, for the default 1024-point FFT, the latency in R2014a was 1589 cycles. In R2014b, the latency is 1148 cycles. The latency is displayed on the block icon.

Version History

If you have manually matched latency paths in models using the R2014a version of the FFT HDL Optimized and IFFT HDL Optimized block, rebalance those paths with the new latency.

CIC algorithm and HDL code generation for DC Blocker

This release adds an option to implement the DC Blocker using the CIC algorithm. You can generate HDL code from DC Blocker and `dsp.DCBlocker`. CIC mode is not yet supported for HDL code generation.

dsp.FilterCascade System object

This release introduces a new System object, `dsp.FilterCascade`, that constructs a cascade of filter System objects.

Phase Extractor block and dsp.PhaseExtractor System object

This release introduces a new block, Phase Extractor, and a new System object, `dsp.PhaseExtractor`, that extract the unwrapped phase from complex input signals.

Overrun and underrun reporting on audio device blocks and System objects

The following blocks and System objects now provide a count of samples lost to queue underrun/overrun since the last transfer of a frame to or from an audio device. You can use this information to debug throughput problems.

- To Audio Device
- From Audio Device
- `dsp.AudioPlayer`
- `dsp.AudioRecorder`

For an example of how to measure and tune audio throughput see [Measuring Audio Latency](#) example. You can open this example by typing `audiolatencymeasurement` at the MATLAB command line.

Unsigned input data type in `dsp.CICDecimator` and `dsp.CICInterpolator` System Objects

`dsp.CICDecimator` and `dsp.CICInterpolator` System objects now support unsigned input data type.

Logic Analyzer support for vector, enumerated, and complex inputs

The `dsp.logicAnalyzer` System object now supports vector, enumerated, and complex input signals.

System object support in Simulink For Each Subsystem

The new `supportsMultipleInstanceImpl` method enables the use of System objects in Simulink For Each Subsystem blocks. Include this method in your System object class definition file when you define a new kind of System object.

Getting Started Tutorials

This release adds 15 new tutorials, which illustrate a broad range of applications supported by the DSP System Toolbox software. There are new tutorials on the following topics:

- Streaming signal processing
- Filter design in MATLAB and Simulink
- Real-time audio processing and latency measurements
- Signal visualization in time and frequency
- Algorithm acceleration using code generation
- Multistage-multirate filtering for sample-rate conversion
- Authoring System objects
- Deploying MATLAB code and applications

See [Getting Started with DSP System Toolbox](#) for links to the new tutorials.

Functionality being removed or replaced for blocks and System objects

The Signal To Workspace block is now called To Workspace.

Certain functionality in the following blocks and System objects will be removed in future releases:

- Digital Filter block
- Variable Integer Delay block
- Delay block
- `dsp.DigitalFilter`

- `dsp.VariableIntegerDelay`
- `dsp.Delay`

These features will trigger a warning in R2014b. For most functionality, you can automatically update your model by running the Upgrade Advisor and selecting 'Check model for known block upgrade issues requiring compile time information'. See Consult the Upgrade Advisor.

Version History

Digital Filter and `dsp.DigitalFilter`

Use of the Digital Filter block and `dsp.DigitalFilter` System object in future releases is not recommended. Existing instances will continue to operate, but certain functionality will be disabled. If your model includes the functionality listed in the table below, you must update your model.

For future designs, choose from Discrete FIR Filter, Discrete Filter, Biquad Filter, or Allpole Filter blocks, and `dsp.FIRFilter`, `dsp.IIRFilter`, `dsp.BiquadFilter`, or `dsp.AllpoleFilter` System objects.

The functionality in the following table will be removed in future releases. The table describes the changes for the Digital Filter block. For `dsp.DigitalFilter`, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Updating filter coefficients once per sample	Coefficient source is Input port(s), Input processing is Columns as channels (frame based), and Coefficient update rate is One filter per sample	Set Coefficient update rate to One filter per frame, insert the filter block in a For Iterator subsystem block, and use Variable Selector blocks to apply one set of coefficients to each input sample in a loop.	Yes
Column-based vector filter coefficients from input ports	Coefficient source is Input port(s)	Transpose your filter coefficients into a row vector by using a Transpose block.	Yes
Nonunity denominator coefficients from input ports	Coefficient source is Input port(s), Transfer function type is either IIR (poles & zeros) or IIR (all poles), and the First denominator coefficient = 1, remove a0 term in the structure check box is cleared.	Ensure the First denominator coefficient = 1, remove a0 term in the structure check box is selected, and scale your coefficients and initial values accordingly.	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Complex Biquad scale values	Transfer function type is IIR (poles and zeros) and the Filter structure is any SOS form	Use real scale values. Alternatively, for non-fixed-point input data types and zero initial conditions, set the scale value to 1, and add a Gain block at the filter's input port, where the gain value is equal to the product of the complex scale values.	Yes
State data type different from Accumulator data type	Transfer function type is FIR (all zeros) and the Filter structure is Direct form I transposed	Set the State data type to Same as accumulator.	No

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Variable Integer Delay and `dsp.VariableIntegerDelay`

The DSP Variable Integer Delay block has been replaced with the Simulink Variable Integer Delay block. Existing instances of the DSP block will continue to operate, but certain functionality in the DSP block and `dsp.VariableIntegerDelay` System object will be disabled in future releases. If your model includes the functionality listed in the table below, you must update your model.

The functionality in the following table will be removed in future releases. The table describes the changes for the Variable Integer Delay block. For `dsp.VariableIntegerDelay`, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Nonscalar delay	Using the block only. Nonscalar delay remains supported in the <code>dsp.VariableIntegerDelay</code> System object.	Insert your block in a For Each subsystem and partition the data and delay inputs to apply each delay value to the corresponding data channel.	Yes
Initial conditions specified as a vector	Input processing is Columns as channels (frame based), and the input has multiple channels (columns)	Specify the Initial conditions as a $1 \times \text{NumChans} \times R$ matrix, where <i>NumChans</i> is the number of input channels, and <i>R</i> is the maximum delay value.	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
	Input processing is Elements as channels (sample based), and the input has multiple channels (samples)	Specify the initial conditions as a $(dim1 \times dim2 \times \dots \times dimN) \times R$ array instead, where $dimM$ is the M th input dimension and R is the maximum delay value.	Yes

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Delay and dsp.Delay

The functionality in the following table will be removed in future releases. The table describes the changes for the Delay block. For dsp.Delay, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Nonscalar delay	Using the block only. Nonscalar delay remains supported in dsp.Delay System object.	Use a Variable Integer Delay block in a For Each subsystem. Partition the data and delay inputs to apply each delay value to the corresponding data channel.	Yes
Delay specified in units of frames	Delay units is Frames	Set Delay units to Samples and set your new delay as the delay in frames multiplied by the frame length.	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Specification of different initial conditions for each channel but same conditions within a channel	The Specify different initial conditions for each channel check box is selected, and the Specify different initial conditions within a channel check box is cleared.	<p>Clear both check boxes and specify the initial condition as a scalar. Alternatively, you can select the Specify different initial conditions within a channel check box and specify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify the initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), change input processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Specification of different initial conditions within a channel but same conditions for each channel	The Specify different initial conditions within a channel check box is selected, and the Specify different initial conditions for each channel check box is cleared.	<p>Clear both check boxes and specify the initial condition as a scalar. Alternatively, you can select the Specify different initial conditions for each channel check box and specify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify the initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), set Input Processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Initial conditions specified as cell array		<p>Clear the Specify different initial conditions within a channel and Specify different initial conditions for each channel check boxes and specify the initial condition as a scalar. Alternatively, select both check boxes and modify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), set Input Processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	Yes

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Persistence mode in Vector Scope

The Vector Scope block no longer supports Persistence mode, which retained historical data on a single plot.

Version History

You do not need to update any existing model that had Persistence mode set. As of R2014b, you will not see historical data on your Vector Scope.

Code generation for additional DSP System Toolbox System objects

In R2014b, you can generate code from the following additional DSP System Toolbox System objects. Code generation from MATLAB code requires a MATLAB Coder license.

-
- `dsp.CICCompensationDecimator`
 - `dsp.CICCompensationInterpolator`
 - `dsp.FarrowRateConverter`
 - `dsp.FilterCascade`

You cannot generate code directly from `dsp.FilterCascade`. Instead, first use the `dsp.FilterCascade.generateFilteringCode` method to generate a MATLAB function from the System object. Then generate C/C++ code from the MATLAB function.

- `dsp.FIRDecimator` for transposed structure
- `dsp.FIRHalfbandDecimator`
- `dsp.FIRHalfbandInterpolator`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseExtractor`
- `dsp.SampleRateConverter`
- `dsp.StateLevels`

See System Objects in MATLAB Code Generation and Functions and System Objects Supported for C Code Generation.

Tunable amplitude on `dsp.SineWave`

The `Amplitude` property of `dsp.SineWave` is now tunable when the `Method` property is `Differential` or `Trigonometric` function.

R2014a

Version: 8.6

New Features

Version History

Up to four-times faster FIR filter simulation in MATLAB System object and Simulink block

This release introduces a refactoring of the Discrete FIR Filter block and `dsp.FIRFilter` System object to significantly improve simulation speed in multi-core processors. The refactored FIR simulation in MATLAB and Simulink leverages the Intel® Threading Building Blocks (TBB) library to optimize multi-core parallelism at the channel and frame level.

Optimized C code generation for ARM Cortex-M processors from System objects with MATLAB Coder and Embedded Coder

This release adds code-generation support for ARM Cortex-M processors in MATLAB for select System objects. With the supported System objects, you can generate C code that can be linked with the CMSIS library and compiled to provide an executable to run on ARM Cortex-M processors. To use the DSP System Toolbox Support Package for ARM Cortex-M Processors, you must have the following products in addition to the DSP System Toolbox: Simulink, Simulink Coder, Embedded Coder and MATLAB Coder. The following DSP System Toolbox System objects support the CMSIS library:

- `dsp.FIRFilter`
- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.LMSFilter`
- `dsp.BiquadFilter`
- `dsp.FFT`
- `dsp.IFFT`
- `dsp.Convolver`
- `dsp.CrossCorrelator`
- `dsp.Mean`
- `dsp.RMS`
- `dsp.StandardDeviation`
- `dsp.Variance`

Notch/peak filter and parametric equalizer filter System objects in MATLAB

This release introduces new second-order IIR notching/peaking and parametric equalizer filters. Use `dsp.NotchPeakFilter` to implement a peaking or notching filter. With `dsp.NotchPeakFilter`, you can control the center frequencies and 3-dB bandwidths of the peaks/notches with tunable properties.

Use `dsp.ParametricEQFilter` to implement a parametric equalizer with tunable gain, bandwidth, and center frequency.

Variable bandwidth FIR and IIR filter System objects in MATLAB

This releases introduces two new System objects, `dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter`, which allow you to vary the passband while filtering.

`dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter` enable you to tune the filter in a computationally efficient way while preserving your filter structure.

Pink/Colored noise generation System object in MATLAB

This release introduces the ability to generate noise with a $1/f^\alpha$ power spectral density. You can set α equal to any value in the interval $[-2,2]$. Specifying $\alpha=1$ results in pink noise, while setting $\alpha=2$ produces Brownian noise. See `dsp.ColoredNoise` for details.

HDL optimized FFT and IFFT Simulink blocks

This release introduces FFT HDL Optimized and IFFT HDL Optimized blocks for the discrete Fourier transform (DFT) and inverse DFT optimized for HDL code generation.

Fixed-point data type support for FIR filter, in ARM Cortex-M support package

The Discrete FIR Filter from the Simulink workflow, and the `dsp.FIRFilter` from the MATLAB workflow, support fixed-point data types defined in the CMSIS library.

Choice of wrapping or truncating input of FFT, IFFT, and Magnitude FFT in MATLAB and Simulink

In the FFT, IFFT, and Magnitude FFT blocks, a boolean parameter has been added that is by default checked. This widget reads: **Wrap input data when FFT length is shorter than input length**, and it gives you the choice of wrapping or truncating the input, depending on the FFT length. If this parameter is checked, modulo-length data wrapping occurs before the FFT operation, given FFT length is shorter than the input length. If this property is unchecked, truncation of the input data to the FFT length occurs before the FFT operation.

In the `dsp.FFT` and `dsp.IFFT` System objects, a boolean property is added that is by default `true`. If this property is set to `true`, modulo-length data wrapping occurs before the FFT operation, given FFT length is shorter than the input length. If this property is set to `false`, truncation of the input data to the FFT length occurs before the FFT operation.

Variable-size input for biquad and LMS filters in MATLAB and Simulink

The Biquad Filter and LMS Filter blocks and the corresponding System objects, `dsp.BiquadFilter` and `dsp.LMSFilter`, now support variable-size input. In Simulink, this support means that the frame size (number of rows) can change during simulation. In a System object, this support allows the `step` method to handle an input that is changing in size.

More flexible control of dsp.LMSFilter System object fixed-point settings

In this release you can specify independent fixed-point data types for all `dsp.LMSFilter` System object fixed-point settings.

DC blocker System object and Simulink block

This release adds a new System object and Simulink block to remove the DC component of a signal. Use `dsp.DCBlocker` in MATLAB and the corresponding block, DC Blocker, in Simulink.

`dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` now support C code generation

In this release you can generate C code for both digital down and up converters.

The `isDone` method of `dsp.AudioFileReader` honors `PlayCount`

The behavior of `isDone` has changed for this object. `isDone` returns True when EOF is reached `PlayCount` number of times. The default of `PlayCount` has changed from `Inf` to 1.

Version History

If you had a while loop controlled by `isDone`, and you had set the `PlayCount` to `Inf`, you should now set it to the number of times you want the loop to be executed, otherwise you will have an infinite loop.

M4A replaced by MPEG4 in `dsp.AudioFileWriter`

In `dsp.AudioFileWriter`, the M4A file format has been changed to MPEG4.

Spectrogram cursors and CCDF plots in the spectrum analyzer

This release introduces cursors in `dsp.SpectrumAnalyzer` when the `SpectrumType` is set to 'Spectrogram'. Additionally, you can now obtain complementary cumulative distribution function (CCDF) plots for your data.

Changed `dsp.SpectrumAnalyzer` property names

This table lists the `dsp.SpectrumAnalyzer` property name changes.

Old Property Name	New Property Name
Grid	ShowGrid
LegendSource	ShowLegend
MaxHoldTrace	PlotMaxHoldTrace
MinHoldTrace	PlotMinHoldTrace
NormalTrace	PlotNormalTrace
TwoSidedSpectrum	PlotAsTwoSidedSpectrum

Version History

Update all instances of old names in your code to the new names.

Conversion to/from allpass from/to wave digital filter

This release introduces two new conversion functions, `allpass2wdf` and `wdf2allpass`, which enable you to convert from an allpass filter to a wave digital filter and from a wave digital filter to an allpass filter.

Transfer function estimation in Simulink

This release introduces a new Simulink block for transfer function estimation. You can find the Discrete Transfer Function Estimator block in the Power Spectrum Estimation library.

Updates to the Time Scope

The following updates have been made to the Time Scope.

- Array, structure, structure with time, and MAT-file logging formats
- Compact toolbar to allow more space for data display
- Scale X-axis , Y-axis, and XY-axes options, autoscaling and panning
- Default display does not show time-axis label. You can, optionally, turn the label on.
- Sampling option
- Model Configuration properties now honored by the Time Scope
- Snap-to-data cursors option to force cursors to data points
- Open Measurements panels saved when you save a model. Those panels reopen when you open the saved model.

Changed `dsp.TimeScope` property names

This table lists the `dsp.TimeScope` property name changes.

Old Property Name	New Property Name
Grid	ShowGrid
LegendSource	ShowLegend
MagnitudePhase	PlotAsMagnitudePhase
TimeSpanOvrerrunMode	TimeSpanOvrerrunAction

Version History

Update all instances of old names in your code to the new names.

Time Scope automatically switches to block-based sample time

The Time Scope uses port-based sample time, except in Simulink External or Rapid-Accelerator modes. In External and Rapid-Acceleration modes, the Time Scope switches to block-based sample time. Port-based sample time uses individual sample times for each input port. Block-based sample time uses the same sample time for the whole block.

dsp.LogicAnalyzer channel selection

In the `dsp.LogicAnalyzer` System object, you now can select individual or multiple channels. Then, you can find the next and previous transitions for the selected channel. Also, you can copy, paste, and move selected channels.

System object templates

The MATLAB **New > System object** menu now has three new class-definition file templates. The **Basic** template sets up a simple System object. The **Advanced** template includes additional features of System objects. The **Simulink Extension** template provides additional customization of the System object for use in the MATLAB System block.

System objects infer number of inputs and outputs from `stepImpl` method

When you create a new kind of System object that has a fixed number of inputs or outputs specified in the `stepImpl` method, you no longer need to include `getNumInputsImpl` or `getNumOutputsImpl` in your class definition file. The correct number of inputs and outputs are inferred from the `stepImpl` inputs and outputs, respectively.

System objects `setupImpl` method enhancement

When you create a new kind of System object and include the `setupImpl` method, you do not have to match the `setupImpl` method inputs to the `stepImpl` method inputs. If your `setupImpl` method does not use any input characteristics, such as, data type or size), you can include only the System object as the input argument.

System objects `infoImpl` method allows variable inputs

When you create a new kind of System object, you can use the `info` method to provide information specific to that object. The `infoImpl` method, which you include in your class-definition file, now allows `varargin` as an input argument.

System objects base class renamed to `matlab.System`

The System object base class, `matlab.system.System` has been renamed to `matlab.System`. If you use `matlab.system.System` when defining a new System object, an error message results.

Version History

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

System objects Propagates mixin methods

Four new methods have been added to the Propagates mixin class. You use this mixin when creating a new kind of System object for use in the MATLAB System block in Simulink. You use these methods to query the input and specify the output of a System object.

-
- `propagatedInputComplexity`
 - `propagatedInputDataType`
 - `propagatedInputFixedSize`
 - `propagatedInputSize`

Code generation support for additional functions

This release introduces code generation support for the following functions. You must have the MATLAB Coder software to generate code.

- `ca2tf`
- `cl2tf`
- `firceqrip`
- `fireqint`
- `firgr`
- `firhalfband`
- `firminphase`
- `firnyquist`
- `firpr2chfb`
- `ifir`
- `iircomb`
- `iirgrpdelay`
- `iirlpnorm`
- `iirlpnormc`
- `iirnotch`
- `iirpeak`
- `tf2ca`
- `tf2cl`

R2013b

Version: 8.5

New Features

Version History

Support Package for ARM Cortex-M Processors

The DSP System Toolbox Support Package for ARM Cortex-M Processors allows you to model your signal processing algorithm in Simulink and generate C code. The generated code can be linked with the CMSIS library, and compiled to provide an executable to run on ARM Cortex-M processors. To use the DSP System Toolbox Support Package for ARM Cortex-M Processors, you must have the following products in addition to the DSP System Toolbox: Simulink, Simulink Coder, Embedded Coder, and MATLAB Coder. The following DSP System Toolbox blocks, which support this library, make it optimal for use in the ARM Cortex-M processors.

- Discrete FIR Filter
- FIR Decimation
- FIR Interpolation
- LMS Filter
- Biquad Filter
- FFT
- IFFT
- Correlation
- Convolution
- Mean
- RMS
- Variance
- Standard Deviation

To download and install this feature, select **Add-Ons > Get Hardware Support Packages** on the MATLAB Toolstrip. Then, use Support Package Installer to install the DSP System Toolbox Support Package for ARM Cortex-M Processors. For more information, see Support Package for ARM Cortex-M Processors.

Channel and distortion measurement, cursors, and spectrogram visualization using Spectrum Analyzer in MATLAB and Simulink

To enhance visualizing your data, channel measurements, distortion measurements, cursor measurements, and a spectrogram view have been added to the Spectrum Analyzer. Channel measurements show occupied bandwidth and adjacent channel power ratio (ACPR) measurements. Distortion measurements show harmonic distortion and intermodulation distortion measurements. Cursor measurements show measurements between two cursors. A spectrogram is a display of the frequencies in a signal over time.

Channel mapping for multichannel audio devices in MATLAB and Simulink

The `dsp.AudioPlayer` and `dsp.AudioRecorder` System objects, and the To Audio Device and From Audio Device blocks, now support channel mapping. The term channel mapping is used to refer to a 1-1 mapping that associates channels on the selected audio device to channels of the data.

When you play audio, channel mapping allows you to specify on which channel of the audio device to output a specific channel of audio data. You can specify channel mapping as a vector of output channel indices corresponding to each output channel of data being written.

When you record audio, channel mapping allows you to specify on which channel of the audio data to input a specific channel of audio device. You can specify channel mapping as a vector of audio channel indices corresponding to each channel of data being read.

Variable-size support for FIR and Allpole filters in MATLAB and Simulink

In this release, `dsp.AllpoleFilter`, `dsp.FIRFilter`, Discrete FIR Filter block, and Allpole Filter block support variable-size input. Thus the number of rows (size of the frame) can vary.

Estimation of Power Spectrum, Cross Power Spectrum, and Transfer Function for streaming data in MATLAB

This release introduces `dsp.TransferFunctionEstimator`, `dsp.SpectrumEstimator`, and `dsp.CrossSpectrumEstimator`. The transfer function estimator, estimates the complex frequency-domain transfer function from time-domain data, based on the Welch averaged periodogram method. `dsp.TransferFunctionEstimator` provides functionality similar to the Signal Processing Toolbox function `tffestimate`, albeit in a streaming-friendly manner. The PSD and Cross-PSD estimators, are also provided using the Welch averaged periodogram method, functionality similar to the Periodogram block in DSP System Toolbox. These three System objects support double- and single-precision floating point inputs. They also support C code generation.

Data logging and archiving using Time Scope in Simulink

Data logging and external mode data archiving have been added to the Time Scope block. You can now log scope data to a MAT-file.

MIDI control interface support in MATLAB

R2013b introduces these five functions that together provide the same functionality as that of the MIDI Controls block:

- `midiid` — Interactively identify a MIDI control.
- `midicontrols` — Open a group of MIDI controls for reading.
- `midiread` — Read the most recent values of the group of MIDI controls.
- `midisync` — Send values to update the group of MIDI controls.
- `midicallback` — Invoke a callback when an open control changes.

Among these functions, only `midicontrols`, `midiread`, and `midisync` support code generation.

Integer support on the output port of the MIDI Controls block

The MIDI Controls block now supports the `uint8` data type on the output port, for the range of 0 to 127, selecting the **raw MIDI** mode.

Kalman filter

This release introduces `dsp.KalmanFilter`. This filter supports C/C++ code generation, single- and double-precision floating point, MIMO, and optional control input. It also includes a subset of functionality in the corresponding block, Kalman Filter, including:

- Initial condition for estimated state
- Initial condition for estimated error covariance
- State transition matrix
- Process noise covariance
- Measurement matrix
- Measurement matrix noise covariance
- Output estimated measurement
- Output estimated state
- Multiple parallel filters
- Disable update on a subset of filters

Adaptive filters using Lattice, Fast Transversal, Filtered-X LMS, and Frequency Domain algorithms in MATLAB

The following adaptive filters are introduced in this release:

- `dsp.AdaptiveLatticeFilter`
- `dsp.FastTransversalFilter` (no code generation)
- `dsp.FilteredXLMSFilter` (no code generation)
- `dsp.FrequencyDomainAdaptiveFilter`

They have the following features:

- C/C++ code generation support
- Floating-point data type support (double and single) for inputs
- Variable frame-size for inputs
- Real and complex inputs

Coupled allpass filter

This release introduces the `dsp.CoupledAllpass` filter, which implements IIR filters as the sum of two allpass filters operating in parallel. This filter allows you to use complex coefficients. It supports Variable-size input, floating-point filter analysis, and the filter coefficients are tunable. You can integrate this filter with filter design workflows such as `fdesign` and `filterbuilder`.

Functionality being removed or changed

Using `fdesign.pulseshaping` is discouraged because it is being removed in the future. However, it still runs when you try to use this functionality, but you are encouraged to use `rcosdesign` and `gaussdesign` instead.

Migrate away from `fdesign.pulseshaping`

Using `fdesign.pulseshaping` is not recommended. Use `rcosdesign` or `gaussdesign`. This table shows examples of how to replace the old function with the new functions.

<code>fdesign.pulseshaping</code>	<code>rcosdesign</code> and <code>gaussdesign</code>
<pre>sps=6; span=4; Beta=fdesign.pulseshaping(sps,... 'Square Root Raised Cosine',... 'Nsym,Beta',span,Beta); d1=design(f1); n1=d1.Numerator</pre>	<pre>n1n=rcosdesign(Beta,span,sps); n1n=n1n/max(n1n)*(-1/(pi*sps)... *(pi*(Beta-1)-4*Beta))</pre>
<pre>g1=fdesign.pulseshaping(sps,... 'Square Root Raised Cosine',... 'N,Beta',sps*span,Beta); h1=design(g1); k1=h1.Numerator</pre>	<pre>k1n=rcosdesign(Beta,span,sps); k1n=k1n/max(k1n)*(-1/(pi*sps)... *(pi*(Beta-1)-4*Beta))</pre>
<pre>f2=fdesign.pulseshaping(sps,... 'Raised Cosine',... 'Nsym,Beta',span,Beta); d2=design(f2); n2=d2.Numerator</pre>	<pre>n2n=rcosdesign(Beta,span,sps,'normal'); n2n=n2n/max(abs(n2n))/sps</pre>
<pre>g2=fdesign.pulseshaping(sps,... 'Raised Cosine',... 'N,Beta',sps*span,Beta); h2=design(g2); k2=h2.Numerator</pre>	<pre>k2n=rcosdesign(Beta,span,sps,'normal'); k2n=k2n/max(abs(k2n))/sps</pre>
<pre>BT=0.3; f3=fdesign.pulseshaping(sps,...'Gaussian',... 'Nsym,BT',span,BT); d3=design(f3); n3=d3.Numerator</pre>	<pre>n3n=gaussdesign(BT,span,sps)</pre>

Configuration dialog added to Logic Analyzer

A Visuals — Logic Analyzer Properties dialog box has been added to the `dsp.LogicAnalyzer`. This dialog box allows you to change the appearance of the scope. To open this dialog box, select **View > Configuration Properties**.

Complex trigger support in Time Scope

The Time Scope can plot signals as real/imaginary or magnitude/phase. In addition to using triggers from the real or imaginary data, you can now use magnitude or phase values from complex signals as triggers.

Default color changes for Array Plot, Time Scope, and Spectrum Analyzer

The following scopes use a new default color scheme to help emphasize the data being visualized:

- `dsp.ArrayPlot`
- `dsp.SpectrumAnalyzer` and Spectrum Analyzer block
- `dsp.TimeScope` and Time Scope block

MATLAB System Block to include System objects in Simulink models

The MATLAB System block is a new block in the Simulink User-Defined Functions library. Use this block to create a Simulink block that includes a System object™ in your model. This capability is useful for including your algorithm in your model.

Restrictions on modifying properties in System object Impl methods

When defining a new System object, certain restrictions affect your ability to modify a property.

You cannot use any of the following methods to modify the properties of an object:

- `cloneImpl`
- `getDiscreteStateImpl`
- `getDiscreteStateSpecificationImpl`
- `getNumInputsImpl`
- `getNumOutputsImpl`
- `getOutputDataTypeImpl`
- `getOutputSizeImpl`
- `isInputDirectFeedthroughImpl`
- `isOutputComplexImpl`
- `isOutputFixedSizeImpl`
- `validateInputsImpl`
- `validatePropertiesImpl`

This restriction is required by code generation, which assumes that these methods do not change any property values. These methods are validation and querying methods that are expected to be constant and should not impact the algorithm behavior.

Also, if either of the following conditions exist:

- You plan to generate code for the object
- The object will be used in the MATLAB System block

you cannot modify tunable properties for any of the following runtime methods:

- `outputImpl`
- `processTunedPropertiesImpl`
- `resetImpl`
- `setupImpl`
- `stepImpl`
- `updateImpl`

This restriction prevents tunable parameter updates within the object from interfering with updates from outside the generated code. Tunable parameters can only be changed from outside the generated code.

Version History

If any of your class definition files contain code that changes a property in one of the above `Impl` methods, move that property code into an allowable `Impl` method. Refer to the System object `Impl` method reference pages for more information.

System objects `matlab.system.System` warnings

The System object base class, `matlab.system.System`, has been replaced by `matlab.System`. If you use `matlab.system.System` when defining a new System object, a warning message results.

Version History

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

Removing HDL Support for NCO Block

HDL support for the NCO block will be removed in a future release. Use the NCO HDL Optimized block instead.

Version History

In the current release, if you generate HDL code for the NCO block, a warning message appears. In a future release, any attempt to generate HDL code for the NCO block will cause an error.

R2013a

Version: 8.4

New Features

Version History

Allpass Filter System object

This release introduces an Allpass Filter System object™, `dsp.AllpassFilter`, which unifies functionality already existing in a number of `dfilt` objects under various names. `dsp.AllpassFilter` supports three different allpass structures and it can handle both single-section and cascaded configurations. It also supports double and single floating point, multichannel and variable-length input, tunability of filter coefficients and filter analysis.

Adaptive filter System objects using RLS and Affine Projection Filter

This release introduces two adaptive filter System objects, `dsp.RLSFilter` and `dsp.AffineProjectionFilter`. These System objects both support double and single floating point, and code generation.

Logic Analyzer System object

As of R2013a, DSP System Toolbox software provides a new Logic Analyzer System object that enables you to view the transitions of signals. To create a Logic Analyzer System object variable called `h`, at the MATLAB command prompt, type `h=dsp.LogicAnalyzer`. The Logic Analyzer has several graphical features:

- Multiple signals in a single window — The y-axis of the display can contain a number of channels, vertically tiled on top of each other.
- Ability to vary the display style — You can modify the name, height, color, and font size of each wave.
- Analog and Digital display formats — Both discrete and continuous signals can appear as waves or be tiled vertically in the display.
- Data numerical display options — You can display numerical values in various numeric systems, including unsigned decimal, signed decimal, hexadecimal, octal, and binary form. Such flexibility is especially useful for visualizing fixed-point signals.
- Cursors to mark points of interest and view values — By placing a cursor at a discrete time stamp, you can position a solid vertical line on the display and observe the values of each channel at that time stamp.
- Dividers to delineate groups of waves in a channel — By placing a divider on the display, you can separate waves by horizontal dashed lines.

For more information, see the `dsp.LogicAnalyzer` System object reference topic.

Audio System object support for tunability, variable frame size, variable number of channels, and writing MPEG-4 AAC

Effective 13a, variable frame size and variable number of channels are supported by `dsp.AudioPlayer`. If you use variable-size signals with this System object, you may experience sound dropouts when the size of the input changes. With the added support, you can avoid this behavior. Before you start the simulation, call `setup` for a signal with maximum dimensions.

This release also enhances `dsp.AudioPlayer` and `dsp.AudioRecorder` System objects to make sample rate, buffer size, and queue duration tunable. Tuning these properties stops and restarts the sound card, which creates a pause. The length of the pause depends on your buffer size and queue duration.

In addition, the `dsp.AudioFileWriter` System object and To Multimedia File block are enhanced to support MPEG-4 AAC audio files on Windows 7, and Mac OS X. You can use both M4A and MP4 extensions. The following platform specific restrictions apply when writing these files:

Windows 7	Mac OS X
<ul style="list-style-type: none"> Only sample rates of 44100 and 48000 Hz are supported. 	<ul style="list-style-type: none"> Only mono or stereo outputs are allowed.
<ul style="list-style-type: none"> Only mono or stereo outputs are allowed. 	
<ul style="list-style-type: none"> Output data is padded on both front and back of the signal, with extra samples of silence. <p>Windows AAC encoder places sharp fade-in and fade-out on an audio signal, causing the signal to be slightly longer in samples when written to disk.</p>	<ul style="list-style-type: none"> Not all sampling rates are supported, although the Mac Audio Toolbox API do not explicitly specify a restriction.
<ul style="list-style-type: none"> A minimum of 1025 samples must be written to the MPEG-4 AAC file. 	

Array Plot System object for displaying vectors or arrays in 2-D and Spectrum Analyzer block with enhanced controls and features such as peak finder

As of R2013a, DSP System Toolbox software provides a new Array Plot System object that enables you to visualize streaming data in two dimensions. Using Array Plot, you can visualize any set of data on the y-axis, opposite another set of data on the x-axis, labeling the x- and y-axes anything you choose. To create an Array Plot System object variable called *h*, at the MATLAB command prompt, type `h=dsp.ArrayPlot`. Array Plot contains the following panels.

- **Cursor Measurements** — shows cursors on all the displays. In the **Settings** pane, you can choose either waveform cursors, which are always attached to the signal data, or screen cursors, which may be placed anywhere on the axes. In the **Measurements** pane, you can see the x-axis value, y-axis value, and other calculated values at the locations of the cursors.
- **Signal Statistics** — displays the maximum, minimum, peak-to-peak difference, mean, median, and RMS values of a selected signal. It also shows the corresponding x-axis values at which the maximum and minimum values occur.
- **Peak Finder** — displays y-axis maxima and the corresponding x-axis values at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

For more information, see the `dsp.ArrayPlot` System object reference topic.

As of R2013a, DSP System Toolbox software provides a new Spectrum Analyzer block to replace the Spectrum Scope block in the Sinks library. Using Spectrum Analyzer, you can view the power spectrum or power spectral density of signals. The graphical interface of the Spectrum Analyzer block resembles that of the `dsp.SpectrumAnalyzer` System object. Spectrum Analyzer contains the following panels.

- **Spectrum Settings** — enables you to modify settings to control how the spectrum is calculated. You can modify such parameters as frequency span, resolution bandwidth, number of spectral

averages, and number of FFT points. You can also choose between a one-sided or two-sided spectrum and toggle normal, maximum hold, and minimum hold trace views.

- **Peak Finder** — displays spectral maxima and the corresponding frequencies at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

You can programmatically modify parameters of the Spectrum Analyzer block using MATLAB code. To do so, you can first use the Simulink `get_param` function to get an instance of the `spbscopes.SpectrumAnalyzerConfiguration` class. Then, you can use dot notation or the `get` and `set` commands to modify properties of the Spectrum Analyzer block.

For more information, see the Spectrum Analyzer block reference topic.

Version History

All Simulink models containing Spectrum Scope blocks load with Spectrum Analyzer blocks in R2013a or later. The Spectrum Scope block had several dialog box parameters that do not appear in any Spectrum Analyzer block settings.

- Several options that were available on the Parameters dialog box of the Spectrum Scope block are no longer available or have changed. The parameters of Spectrum Scope map to Spectrum Analyzer parameters in the following manner.

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Scope Properties	Buffer input check box	R2013a Spectrum Analyzer does not require that input signals are buffered. Spectrum Analyzer determines the number of samples needed using the value of the RBW parameter. Regardless of whether the input is a frame-based or sample-based signal, Spectrum Analyzer calculates the spectrum once it has acquired the requisite number of samples.	<p>For Spectrum Scope blocks in R2012b or earlier models, the equivalent R2013a Spectrum Analyzer RBW value is given by the equation:</p> $RBW = \frac{K \times F_s}{L}$ <p>In the preceding equation, K is the window constant calculated for a segment length of 1000, F_s is the sample rate of the block, and L is the buffer length. If the input signal to the R2012b Spectrum Scope block was frame-based and the Buffer input check box was cleared, then the R2013a Spectrum Analyzer computes the RBW value with L set to the frame size of the input signal.</p>
Scope Properties	Buffer size parameter	R2013a Spectrum Analyzer uses the RBW parameter to determine the requisite number of samples to calculate the spectrum, instead of using the buffer size or frame length.	For Spectrum Scope blocks in R2012b or earlier models, if the input signal was frame-based and the Buffer input check box was selected, then the R2013a Spectrum Analyzer computes the RBW value with L set to the value of the Buffer size parameter.
Scope Properties	Buffer Overlap parameter	R2013a Spectrum Analyzer has an Overlap % parameter that is directly related to buffer overlap.	<p>R2013a Spectrum Analyzer will compute its Overlap % using the equation:</p> $O_p = O_l / L \times 100$ <p>In the preceding equation, O_p is Overlap % parameter value, O_l is the R2012b Spectrum Scope Buffer overlap parameter value, and L is the buffer length.</p>

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Scope Properties	Window parameter	R2013a Spectrum Analyzer does not have the Bartlett, Blackman, Triang, or Hanning settings.	Spectrum Scope blocks in R2012b or earlier models with a window parameter set to any of these values will have their Window parameter set to Hann in the R2013a Spectrum Analyzer.
Scope Properties	Window Sampling parameter	R2013a Spectrum Analyzer does not have a Periodic option. All window sampling is now symmetric in the R2013a Spectrum Analyzer.	n/a
Display Properties	Persistence check box — this setting would execute the equivalent of the MATLAB hold on command, adding another line for each spectrum computation on the display.	This option is not available in the R2013a Spectrum Analyzer, which has replaced this feature with the trace options, Normal Trace , Max Hold Trace , and Min Hold Trace .	Spectrum Scope blocks in R2012b or earlier models with persistence enabled will have their Max Hold Trace check box selected in the R2013a Spectrum Analyzer.
Display Properties	Compact Display check box	There is no equivalent capability in the R2013a Spectrum Analyzer.	n/a
Axis Properties	Inherit Sample time from input check box	R2013a Spectrum Analyzer always uses the sample time of the input signal.	n/a

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Axis Properties	Frequency display limits parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If this parameter was set to: <ul style="list-style-type: none"> • Auto — R2013a Spectrum Analyzer selects the Full Span check box on the Spectrum Settings panel, Main options pane. • User-defined — R2013a Spectrum Analyzer clears the Full Span check box on the Spectrum Settings panel Main options pane.
Axis Properties	Minimum frequency (Hz) parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If the User-defined parameter was chosen, then this parameter maps to the R2013a Spectrum Analyzer FStart (Hz) parameter.
Axis Properties	Maximum frequency (Hz) parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If the User-defined parameter was chosen, then this parameter maps to the R2013a Spectrum Analyzer FStop (Hz) parameter.
Line Properties	Line visibilities, Line styles, Line markers, and Line colors parameters	There are no equivalent capabilities in the R2013a Spectrum Analyzer.	Once the simulation has started, you can modify the line styles, markers, and colors using the Style dialog box.

- The R2012b Spectrum Scope allowed you to retain the axes limits over multiple simulations by selecting **Axes > Save Axes Settings**. There is no equivalent capability in the R2013a Spectrum Analyzer. However, you can automatically scale the axes to a specified range using the Tools—Plot Navigation Properties dialog box.

Time Scope block with triggering and peak finder features

As of R2013a, DSP System Toolbox provides the following enhancements to the Time Scope block and the `dsp.TimeScope` System object:

- “Triggers Panel” on page 19-8
- “Peak Finder Features” on page 19-8

- “Panning Capability” on page 19-8
- “Programmatic Access” on page 19-8
- “Scale Axes Limits After 10 Updates” on page 19-8

For more information on these enhancements, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Triggers Panel

The Time Scope contains a new **Triggers** panel that allows you to pause the display only when certain events occur. You can use the Triggers panel when you want to align or search for events.


- In the **Mode** pane, you choose how often the display should update.
- In the **Source / Type** pane, you choose the type of events on which to stop. Events include edges (rising, falling, or both), pulse width, transition, runt, window, or timeout.
- In the **Levels / Timing** pane, you can set the trigger level and hysteresis value.
- In the **Delay / Holdoff** pane, you can offset the trigger position by a fixed delay or set the minimum possible time between trigger events.

To access the **Triggers** panel, in the Time Scope toolbar, click the Triggers button (). Alternatively, in the Time Scope menu, select **Tools > Triggers**.

Peak Finder Features

Effective in R2013a, the peak finder now has the option of displaying more than 10 maxima at once. You can also use the Peaks panel to toggle the labels for each local maximum. This capability allows you to choose whether all the labels show time, value, or both time and value. R2013a also provides the capability to sort in either ascending or descending order by value or by time.

Panning Capability

Effective in R2013a, you can pan in all directions on the Time Scope display. In the Time Scope toolbar, click the Pan button (). Alternatively, in the Time Scope menu, select **Tools > Pan**. Then, click and drag the mouse to the left or right to view a different range of data on the *time*-axis. Click and drag the mouse up or down to see a different Amplitude range on the *y*-axis.

Programmatic Access

Effective in R2013a, you can change the properties of the Time Scope block using MATLAB commands. To do so, you can first use the Simulink `get_param` function to get an instance of the `Simulink.scopes.TimeScopeConfiguration` class. Then, you can use dot notation or the `get` and `set` commands to modify properties of the Time Scope block.

For more information on these enhancements, see the `Simulink.scopes.TimeScopeConfiguration` class reference topic.

Scale Axes Limits After 10 Updates

Effective in R2013a, you can scale the axes limits of the Time Scope displays soon after the simulation starts. To do so, in the Time Scope menu, select **Tools > Scale Axes Limits After 10 Updates**.

Change of the default for audio hardware API on Linux

In this release, the default for audio hardware API on Linux has changed from OSS to ALSA.

Change of the default for audio file formats in multimedia blocks and audio file reader and writer System objects

In the To Multimedia File block and the `dsp.AudioFileWriter` System object, the file format default is now WAV. In the From Multimedia File block and the `dsp.AudioFileReader` System object, the file format default is now MP3.

Change of property default in the audio file reader System object

In the `dsp.AudioFileReader` System object, the property `OutputDataType` returns doubles as default.

Removal of the `signalblks` package

The `signalblks` package has been removed. Instead, for System object classes, and properties, use the `dsp` package.

Version History

To automatically update the existing code, where the `signalblks` package is used, run `sysobjupdate`. This function updates System object code to work in the current release. The application recursively searches the specified folder and subfolders for MATLAB files that contain renamed System object packages, classes, and properties.

Scope Snapshot display of additional scopes in Simulink Report Generator

Using Simulink Report Generator™ software, you can include snapshots of the display produced by a Scope block in a generated report. The Scope Snapshot component, which inserts images of the Simulink Scope block and XY Graph block, now supports the Time Scope block and Spectrum Analyzer block in DSP System Toolbox software.

Note This feature requires that you have a license for the Simulink Report Generator product.

For more information, see the Simulink Report Generator product documentation.

Unoriented vector treated as column vector in the Biquad Filter

Starting this release, the unoriented vector is treated as a column vector in numerator and denominator coefficient ports.

NCO HDL Optimized block

The NCO HDL Optimized block provides hardware friendly control signals, optional reset port, and an optional external dither input port. It also provides a reset function that resets the phase to its initial value during the sinusoid output generation. In addition, it includes an option to output the internal phase and hardware-friendly control signals including valid in and valid out.

HDLNCO System object

The `dsp.HDLNCO` System object, like the `dsp.NCO` System object, generates real or complex sinusoidal signals. In addition, the NCO HDL-optimized System object provides hardware friendly control signals, optional reset signal, and an optional external dither input signal.

HDL code generation for NCO HDL Optimized block and System object

Release R2013a provides HDL code generation support for the new NCO HDL Optimized block and `dsp.HDLNCO` System object. To generate HDL code, you must have an HDL Coder license.

Support for nonpersistent System objects

You can now generate code for local variables that contain references to System objects. In previous releases, you could not generate code for these objects unless they were assigned to persistent variables.

New method for action when System object input size changes

The new `processInputSizeChangeImpl` method allows you to specify actions to take when an input to a System object you defined changes size. If an input changes size after the first call to `step`, the actions defined in `processInputSizeChangeImpl` occur when `step` is next called on that object.

Scaled double data type support for System objects

System objects now support scaled double data types.

R2012b





Version: 8.3

New Features

Version History

SpectrumAnalyzer System object

As of R2012b, DSP System Toolbox software provides a new Spectrum Analyzer System object that enables you to view the power spectrum or power spectral density of signals. To create a Spectrum Analyzer System object variable called *h*, at the MATLAB command prompt, type `h=dsp.SpectrumAnalyzer`. The Spectrum Analyzer has several panels and dialog boxes that allow you to perform the following operations:

- **Spectrum Settings panel** — The **Spectrum Settings** panel enables you to modify settings to control the manner in which the spectrum is calculated. You can modify such parameters as frequency span, resolution bandwidth, number of spectral averages, and number of FFT points. You can also choose between a one-sided or two-sided spectrum and toggle normal, maximum hold, and minimum hold trace views. To hide or display the **Spectrum Settings** panel, in the Spectrum Analyzer toolbar, select the Spectrum Settings button (). Alternatively, in the Spectrum Analyzer menu, select **View > Spectrum Settings**.
- **Peak Finder panel** — The **Peak Finder** panel displays maxima and the frequencies at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion. In the Spectrum Analyzer toolbar, click the Peak Finder button (). Alternatively, in the Spectrum Analyzer menu, select **Tools > Measurements > Peak Finder**.
- **Properties dialog box** — The Spectrum Analyzer provides a dialog box that allows you to control the most common properties of a display, including the grid lines, titles, y-axis labels and y-axis limits. To change options for a display, in the Spectrum Analyzer toolbar, click the Properties button (). Alternately, in the Spectrum Analyzer menu, select **View > Properties**. You can also right-click the display, and select **Properties**.
- **Style dialog box** — The Spectrum Analyzer allows you to customize the style of displays using a Style dialog box. You can change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display. To view or modify the line style of the active signal, in the Spectrum Analyzer menu, select **View > Style**. You can also right-click the display and select **Style**.
- **Axes Scaling Options** — The Spectrum Analyzer enables you with the ability to automatically scale the axes to a specified range. In the Spectrum Analyzer menu, select **Tools > Axes Scaling Options** to modify these settings. To manually scale the axes to the limits you specified, click the Scale Axes Limits button ().

For more information, see the `dsp.SpectrumAnalyzer` System object reference topic.

Cross-platform support for reading and writing WAV, FLAC, OGG, MP3 (read only), MP4 (read only), and M4a (read only)

The `dsp.AudioFileReader` System object, and the From Multimedia File block, now support the following audio file formats on all platforms:

- MP3
- MP4
- M4a
- WAV

-
- FLAC
 - OGG

The `dsp.AudioFileWriter` System object, and the To Multimedia File block, now support the following audio file formats on all platforms:

- WAV
- FLAC
- OGG

Support for code generation for CICDecimator and CICInterpolator System objects

The following System objects now support code generation in MATLAB via the `codegen` command:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`

To use the `codegen` function, you must have a MATLAB Coder license. See *Use System Objects in MATLAB Code Generation* for more information.

Support for HDL code generation for multichannel Discrete FIR Filter block

Discrete FIR Filter block accepts vector input and supports multichannel implementation for better resource utilization.

- With vector input and channel sharing option `on`, the block supports multichannel fully parallel FIR, including direct form FIR, sym/antisym FIR, and FIRT. Support for all implementation parameters, for example: multiplier pipeline, add pipeline registers.
- With vector input and channel sharing option `off`, the block instantiates one filter implementation for each channel. If the input vector size is N , N identical filters are instantiated.

For fully parallel architecture option for FIR filters only.


Time Scope enhancements, including new cursors, embedded simulation controls, and External and Rapid Accelerator modes

As of R2012b, DSP System Toolbox provides the following enhancements to the Time Scope block and the `dsp.TimeScope` System object:

- “Cursor measurements panel” on page 20-4
- “Additional embedded simulation controls” on page 20-4
- “Support for external mode and rapid accelerator mode” on page 20-4
- “Properties dialog box” on page 20-5
- “Axes Maximization” on page 20-5
- “Automatic calculation of Time Span” on page 20-5

- “ReduceUpdates property” on page 20-6
- “Support for conditional subsystems” on page 20-6


Cursor measurements panel

The Time Scope contains a new **Cursor Measurements** panel that shows cursors on all the Time Scope displays. In the **Settings** pane, you may choose either waveform cursors, which are always attached to the signal data, or screen cursors, which may be placed anywhere on the axes. The **Measurements** pane shows the time, amplitude, and other calculated values at the locations of the cursors. In the Time Scope toolbar, click the Cursor Measurements button (). Alternatively, in the Time Scope menu, select **Tools > Measurements > Cursor Measurements**.

For more information, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Additional embedded simulation controls

Effective in R2012b, additional embedded simulation controls are available through the Simulation Toolbar and the Simulation Stepping Options dialog box. In previous releases, the Time Scope block featured a Simulation Toolbar. With this toolbar, you could control the progression of increasing simulation time from the Time Scope GUI by clicking the Run, Pause, Stop, and Next Step buttons. In R2012b, the Simulation Stepping Options dialog box provides you with the ability to further control the simulation behavior. This dialog box allows you to enable the button on the Simulation Toolbar to take a Previous Step. Additionally, you can pause the simulation at a specified time, specify previous stepping options, and modify the number of steps for forward and backward movement. To access these controls, from the Time Scope menu, select **Simulation > Stepping Options**. Alternatively, if previous stepping is disabled, in the Time Scope toolbar, click the Previous Step button. The Simulation Stepping Options dialog box appears.

You can enable Time Scope to show a Previous Step button () , which allows you to move the simulation time backward by one time step. In the Simulation Stepping Options dialog box, in the **Previous stepping options** group, select the **Enable Previous Stepping** check box. To test this feature, first run the simulation, and then pause the simulation. When the simulation is paused, you can now click the Previous Step button to regress the simulation back by one time step.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information, see the Time Scope block reference topic.

Support for external mode and rapid accelerator mode

As of R2012b, the Time Scope block supports two additional simulation modes in Simulink, External mode and Rapid Accelerator mode. You can use External mode to tune block parameters in real time and view block outputs in many types of blocks and subsystems. External mode establishes communication between a host system, where the Simulink environment resides, and a target system, where the executable runs after it is generated by the code generation and build process. For more information about External mode, see Host/Target Communication in the Simulink Coder product documentation.


You can use Rapid Accelerator mode as a method to increase the execution speed of your Simulink model. Rapid Accelerator mode creates an executable that includes the solver and model methods.

This executable resides outside of MATLAB and Simulink. Rapid Accelerator mode uses External mode to communicate with Simulink. For more information about Rapid Accelerator mode, see Acceleration in the Simulink product documentation.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information about Time Scope, see the Time Scope block reference topic.

Properties dialog box

As of R2012b, the Time Scope block provides a centralized location where you can modify the most important properties of a display. The Properties dialog box contains the most frequently modified Time Scope settings, including all the parameters from the Tools:Plot Navigation Options dialog box and the Visuals:Time Domain Options dialog box. It also includes **Open at Start of Simulation** and **Number of Input Ports** from the **File** menu. To open this dialog box, in the Time Scope toolbar, click the Properties button (). Alternately, in the Time Scope menu, select **View > Properties**. You can also right-click on the display and select **Properties**.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object. In the `dsp.TimeScope` System object GUI, when you select **View > Properties**, the Visuals:Time Domain Options dialog box appears, as in R2012a. This same dialog box also appears when you right-click on the display and select **Properties**.

For more information about Time Scope, see the Time Scope block reference topic.

Axes Maximization

In R2012b, you can specify whether to display the Time Scope block or System object in maximized axes mode. In this mode, the axes are expanded to fill the entire display. In each display, there is no space to show titles or axis labels. The values at the axis tick marks appear on top of the axes. You can select one of the following options:

- **Auto** — In this mode, the axes appear maximized in all displays only if the **Title** and **Y-Axis label** parameters are empty for every display. If you enter any value in any display for either of these parameters, the axes are not maximized.
- **On** — In this mode, the axes appear maximized in all displays. Any values entered into the **Title** and **Y-Axis label** parameters are hidden.
- **Off** — In this mode, none of the axes appear maximized.

The default setting is **Auto**. In the Time Scope GUI, you can set this property in the **Main** pane of the Properties dialog box. To change options using the `dsp.TimeScope` System object, set the `MaximizeAxes` property to the intended option. For more information, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Automatic calculation of Time Span

As of R2012b, the Time Scope block can automatically calculate the **Time Span** parameter using the simulation **Start Time** and **Stop Time** parameters. By default, the Time Scope block has the **Time**

Span parameter set to **Auto calculate**. To modify the **Time Span** parameter to use a different value, open the Properties dialog box and click the **Time** tab.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information about Time Scope, see the Time Scope block reference topic.

ReduceUpdates property

As of R2012b, the Time Scope System object has an additional property called `ReduceUpdates`. By default, this property is set to `true`. When this property is `true`, the Time Scope updates the displays at a rate not exceeding 20 hertz. When you set this property to `false`, the Time Scope updates every time the `step` method is called. The simulation speed is faster when this property is set to `true`. Using this property is equivalent to selecting the **Reduce Updates to Improve Performance** check box in the **Simulation** menu of the Time Scope GUI.

For more information about this property, see the `dsp.TimeScope` System object reference topic.

Support for conditional subsystems

In previous releases, the Time Scope block could be used within an enabled subsystem. As of R2012b, the Time Scope block can also be placed in a triggered subsystem, an enabled and triggered subsystem, and a function-call subsystem. For more information about these types of subsystems, see Conditional Subsystems in the Simulink documentation.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information about Time Scope, see the Time Scope block reference topic.

Source and sink blocks being replaced

The following Windows platform blocks now map to other existing blocks that work on all platforms:

Deprecated blocks	Blocks mapped to
From Wave Device	From Audio Device
To Wave Device	To Audio Device
From Wave File	From Multimedia File
To Wave File	To Multimedia File

This mapping is transparent; no `supdate` is needed. When you open an existing model that contains the original blocks, the replacement blocks are automatically substituted. If you save the model, the replacement blocks are saved instead of the original blocks.

Version History

Because mapped blocks do not have identical functionality, incompatibilities can be introduced in certain cases. The From Wave File can have an optional `start-of-file` indicator port, whereas the

From Multimedia File cannot. Therefore, if you load an old model where From Wave File has the `start-of-file` port, a broken link results. In this case, a warning message appears, providing a link to `start_of_file_example`, which shows you how to correct the problem.

Discrete IIRFilter and AllpoleFilter System objects

This release introduces a discrete IIR Filter System object `dsp.IIRFilter` and a discrete Allpole Filter System object `dsp.AllpoleFilter`.

The IIR Filter System object implements the algorithm, inputs, and outputs described on the Discrete Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. This System object supports code generation.

The Allpole Filter System object implements the algorithm, inputs, and outputs described on the Allpole Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. This System object supports code generation.

Support for MATLAB Compiler for CICDecimator and CICInterpolator System objects

The following System objects are now supported by MATLAB Compiler™:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`

For more information, see [Using System Objects with MATLAB Compiler](#).

Code generation support for SignalSource System object

`dsp.SignalSource` now support code generation in MATLAB via the `codegen` command: To use the `codegen` function, you must have a MATLAB Coder license. See [Use System Objects in MATLAB Code Generation](#) for more information.

Behavior change of locked System objects for loading, saving, and cloning

In the previous release, saving, loading, and cloning a locked System object would result in an unlocked System object. This System object had the same property values as the one from which it was cloned, but not the same internal state.

In this release, it does not matter whether you save a locked System object into a MAT file and load it later or clone a locked System object using the `clone` method. In either case, the result is a locked System object with the same property values and the same internal states.

There are, however, a few exceptions. For the following System objects, if you call the `clone` method, the resulting System object is not locked, but if you save or load the System object into and from a MAT file, the result is a locked System object.

- `dsp.MatFileWriter`
- `dsp.AudioRecorder`
- `dsp.AudioPlayer`

Thus, for the above System objects, if you call the `clone` method, you get an unlocked System object with the same property values.

Another exception involves the following System objects:

- `dsp.AudioFileWriter`
- `dsp.AudioFileReader`

For these System objects, if you save a locked System object to a MAT file and load it later, you get an unlocked System object with the same property values. However you do not get the same internal states. This behavior is the same as in the previous release.

Behavior change of statistics blocks for variable-size inputs

When the inputs are of variable size, the running mode behavior of the following blocks has changed:

- Mean
- RMS
- Variance
- Standard Deviation
- Minimum
- Maximum

If the **Input processing** parameter is set to `Elements as channels (sample based)`, the block state is reset when any input dimension changes.

If the **Input processing** parameter is set to `Columns as channels (frame based)`, then the behavior depends on two options:

- When the number of input channels (i.e., number of columns) changes, the block state is reset to its initial condition.
- When the number of input channels remains the same, there is no reset, even if the channel length (i.e., number of rows) changes.

Simulation state save and restore for additional blocks

In this release, there are additional blocks that support simulation state save and restore. These are:

- Cumulative Sum
- Cumulative Product
- Mean
- Variance
- RMS
- Standard Deviation

-
- Queue
 - Stack

Note The Queue and Stack blocks do not support SimState save and restore in dynamic memory allocation mode.

For more information on simulation state save and restore, see Save and Restore Simulation State as SimState.

For Each subsystem support for additional blocks

In this release, most DSP blocks have been updated to support the For Each subsystem. For details about the For Each subsystem, see For Each. For a list of supported blocks, see the *For Each Subsystem Support* column in the table titled *Simulink Block Data Type Support for DSP System Toolbox*. This table can be accessed by typing `showsignalblockdatatypetable` at the command line.

Multi-instance model referencing support for additional blocks

In this release, most DSP blocks have been updated to support multi-instance normal mode model referencing. For details about model referencing, see Model Reference. The blocks that support model referencing are the same blocks that support the For Each subsystem. Therefore for a list of supported blocks, see the *For Each Subsystem Support* column in the table titled *Simulink Block Data Type Support for DSP System Toolbox*. This table can be accessed by typing `showsignalblockdatatypetable` at the command line.

Expanded analysis support for filter System objects

In the previous release, the filter analysis methods for `dfilt` and `mfilt` objects were extended to filter System objects. This release expands the number of supporting analysis methods to include the following:

- `noisepsd`
- `noisepsdopts`
- `freqrespest`
- `freqrespopts`

For a comprehensive list of supported analysis methods, see Analysis Methods for Filter System Objects.

Removal of the `signalblks` package

In this release, the `signalblks` package is being removed and any instantiation of a `signalblks` object causes an error message. In future releases, the functionality will be removed entirely, so you should now use the corresponding object in the DSP System Toolbox.

Discrete filter block visible in DSP library

In addition to accessing the Discrete Filter block from the Simulink library, you can now also access it from the DSP System Toolbox library.

System object tunable parameter support in code generation

You can change tunable properties in user-defined System objects at any time, regardless of whether the object is locked. For System objects predefined in the software, the object must be locked. In previous releases, you could tune System object properties only for a limited number of predefined System objects in generated code.

save and load methods for System objects

You can use the `save` method to save System objects to a MAT file. If the object is locked, its state information is saved, also. You can recall and use those saved objects with the `load` method.

You can also create your own save and load methods for a System object you create. To do so, use the `saveObjectImpl` and `loadObjectImpl`, respectively, in your class definition file.

Save and restore SimState not supported for System objects

The Save and Restore Simulation State as SimState option is no longer supported for any System object in a MATLAB Function block. This option was removed because it prevented parameter tunability for System objects, which is important in code generation.

Version History

If you need to save and restore simulation states, you may be able to use a corresponding Simulink block, instead of a System object.

Map integer delay to RAM on Delay block

`UseRAM` is a block-level parameter on the IntegerDelay block that you access with the HDL Block properties GUI. You assign it an `On` or `Off` value:

- `On`: Map the integer delay to a RAM. `On` is not a guarantee that a RAM is inferred: if all conditions are met (including the threshold criteria), only then is the RAM inferred.
- `Off`: The integer delay is always mapped to registers.

HDL support for System objects

HDL support for the following System objects has been added with release R2012b:

- `dsp.BiquadFilter`

HDL resource sharing for Biquad Filter block

HDL support for second-order section, direct-form I and second-order section, direct-form II filter structures has been added with release R2012b.

Supported architectures:

- Fully Parallel ('default' interface)

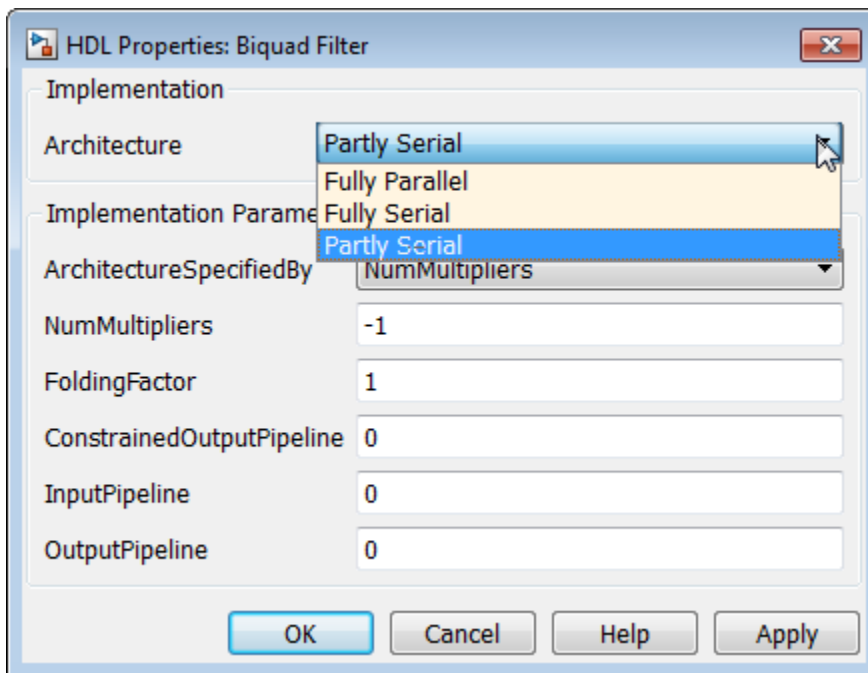
AddPipelineRegisters, ConstrainedOutputPipeline, CoeffMultipliers, InputPipeline, OutputPipeline

- Fully Serial

ConstrainedOutputPipeline, InputPipeline, OutputPipeline

- Partly Serial

ConstrainedOutputPipeline, InputPipeline, OutputPipeline, ArchitectureSpecifiedBy, FoldingFactor, NumMultipliers



R2012a

Version: 8.2

New Features

Bug Fixes

Version History

Frame-Based Processing

Beginning in R2010b, MathWorks has been significantly changing the handling of frame-based processing. For more information, see “Frame-Based Processing” on page 22-2 in the R2011b Release Notes.

The following sections provide more detailed information about the specific R2012a DSP System Toolbox software changes that are helping to enable the transition to the new paradigm for frame-based processing:

- “Inherited Option of the Input Processing Parameter Now Warns” on page 21-2
- “Logging Frame-Based Signals in Simulink” on page 21-3
- “Model Reference and Using slupdate” on page 21-3
- “Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay” on page 21-4
- “Audio Output Sampling Mode Added to the From Multimedia File Block” on page 21-4

Inherited Option of the Input Processing Parameter Now Warns

Some DSP System Toolbox blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior.

To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending upon the type of processing you want. The third choice, `Inherited (this choice will be removed - see release notes)`, is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release your model will warn when the following conditions are all met for any block in your model:

- The **Input processing** parameter is set to `Inherited (this choice will be removed - see release notes)`.
- The input signal is sample based.
- The input signal is a vector, matrix, or N-dimensional array.

Version History

To eliminate this warning, you must upgrade your existing models using the `slupdate` function. The function detects all blocks that have `Inherited (this choice will be removed - see release notes)` selected for the **Input processing** parameter. It then asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels (frame based)`. If the bit is 0 (samples), the function sets the parameter to `Elements as channels (sample based)`.

In a future release, the frame bit and the `Inherited (this choice will be removed - see release notes)` option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame`

based) or `Elements` as channels (sample based). The option set will depend on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `supdate` function. Therefore, you should upgrade your existing modes using `supdate` as soon as possible.

Logging Frame-Based Signals in Simulink

In this release, a new warning message appears when a Simulink model is logging frame-based signals and the **Signal logging format** is set to `ModelDataLogs`. In `ModelDataLogs` mode, signals are logged differently depending on the status of the frame bit, as shown in the following table:

Status of Frame Bit	Today	When Frame Bit Is Removed
Sample-based	3-D array with samples in time in the third dimension	3-D array with samples in time in the third dimension
Frame-based	2-D array with frames in time concatenated in the first dimension	3-D array with samples in time in the third dimension

This warning advises you to switch your **Signal logging format** to `Dataset`. The `Dataset` logging mode logs all 2-D signals as 3-D arrays, so its behavior is not dependent on the status of the frame bit.

When you get the warning message, to continue logging signals as a 2-D array:

- 1 Select **Simulation > Model Configuration Parameters > Data Import/Export**, and change **Signal logging format** to `Dataset`. To do so for multiple models, click on the link provided in the warning message.
- 2 Simulate the model.
- 3 Use the `dsp.util.getLogArray` function to extract the logged signal as a 2-D array.

Model Reference and Using `supdate`

In this release, the `Model` block has been updated so that its operation does not depend on the frame status of its input signals.

Version History

In a future release, signals will not have a `frameness` attribute, therefore models that use the `Model` block must be updated to retain their behavior. If you are using a model with a `Model` block in it, follow the steps below to update your model:

- 1 For both the child and the parent models:
 - In the **Model Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
 - Change the **Block behavior depends on input frame status** parameter to `warning`.
- 2 For both the child and the parent models, run `supdate`.
- 3 For the child model only:
 - In the **Model Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.

- Change the **Block behavior depends on input frame status** parameter to error.

Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay

This release phases out support for buses with mixed sample- and frame-based elements on Simulink's Unit Delay and Delay blocks. Support is also removed from the DSP System Toolbox's Delay block. When the frame bit is removed in a future release, any Delay block that has a bus input of mixed frameness will produce different results.

Version History

This incompatibility is phased over multiple releases. In 2012a, the blocks will produce a warning message. In a future release, when the frame bit is removed, the blocks will produce an error message.

Audio Output Sampling Mode Added to the From Multimedia File Block

The From Multimedia File block now provides a new parameter. This parameter allows you to select frame- or sample-based audio output processing.

System Object Enhancements

- "Code Generation for System Objects" on page 21-4
- "New MAT-File Reader and Writer System Objects" on page 21-4
- "New System Object Option on File Menu" on page 21-4
- "Variable-Size Input Support for System Objects" on page 21-4
- "Data Type Support for System Objects" on page 21-5
- "New Property Attribute to Define States" on page 21-5
- "New Methods to Validate Properties and Get States from System Objects" on page 21-5
- "matlab.system.System changed to matlab.System" on page 21-5

Code Generation for System Objects

System objects defined by users now support C code generation. To generate code, you must have the MATLAB Coder product.

New MAT-File Reader and Writer System Objects

R2012a adds two new System objects, `dsp.MatFileReader` and `dsp.MatFileWriter`. These System objects stream data into and out of MAT-files.

New System Object Option on File Menu

The File menu on the MATLAB desktop now includes a **New > System object** menu item. This option opens a System object class template, which you can use to define a System object class.

Variable-Size Input Support for System Objects

System objects that you define now support inputs that change size at run time.

Data Type Support for System Objects

System objects that you define now support all MATLAB data types as inputs and outputs.

New Property Attribute to Define States

R2012a adds the new `DiscreteState` attribute for properties in your System object class definition file. Discrete states are values calculated during one step of an object's algorithm that are needed during future steps.

New Methods to Validate Properties and Get States from System Objects

The following methods have been added:

- `validateProperties` - Checks that the System object is in a valid configuration. This applies only to objects that have a defined `validatePropertiesImpl` method.
- `getDiscreteState` - Returns a struct containing System object properties that have the `DiscreteState` attribute.

matlab.system.System changed to matlab.System

The base System object class name has changed from `matlab.system.System` to `matlab.System`.

Version History

The previous `matlab.system.System` class will remain valid for existing System objects. When you define new System objects, your class file should inherit from the `matlab.System` class.

Time Scope Enhancements

- "Time Domain Measurements in Time Scope" on page 21-5
- "Multiple Display Support in Time Scope" on page 21-6
- "Style Dialog Box in Time Scope" on page 21-6
- "Sampled Data as Stairs in Time Scope" on page 21-6
- "Complex Data Support in Time Scope" on page 21-7
- "Additional Time Scope Enhancements" on page 21-7

Time Domain Measurements in Time Scope

As of R2012a, the Time Scope block and System object support the time domain signal measurements **Signal statistics**, **Bilevel measurements**, and **Peak finder**. The **Signal Statistics** panel displays the maximum, minimum, peak-to-peak difference, mean, median, and RMS values of a selected signal. It also displays the times at which the maximum and minimum values occur. The **Bilevel Measurements** panel displays information about a selected signal's transitions, overshoots or undershoots, and cycles. The **Peak Finder** panel displays maxima and the times at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

To use the new time domain measurements features in the Time Scope block, click one of the three corresponding buttons in the Time Scope toolbar. You can also access these panels by selecting **Measurements** from the **Tools** menu.

See the Time Scope reference topic for more information.

Multiple Display Support in Time Scope

R2012a allows you to choose to have multiple displays in the Time Scope, using both the block and the System object. This feature allows you to tile your screen into a number of separate displays, up to a grid of 4 rows and 4 columns. You may find multiple displays useful when the Time Scope takes multiple input signals.

To set the number of displays on the Time Scope, click the layout button in the Time Scope toolbar. You can also select **Layout** from the **View** menu. To set the number of displays using the `dsp.TimeScope` System object, set the `LayoutDimensions` property.

To change options for a display, select **Properties** from the **View** menu. Select the **Display** tab, and use the menu to select the display you want to update. You can also right-click on the axes, and select **Properties**. To change options using the `dsp.TimeScope` System object, set the `ActiveDisplay` property to the intended display number.

See the Time Scope reference topic for more information.

Style Dialog Box in Time Scope

R2012a enhances the Time Scope block and System object by allowing you to customize the style of displays using a Style dialog box. You are able to change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display.

The Style dialog box replaces the **Line Properties** menu item that was used in previous releases for customizing line properties. To open the Style dialog box, select **Style** from the **View** menu.

Note This release changes the Time Scope default axes and line colors. The Time Scope initially displays the axes as black instead of white, as shown in previous releases. For a real, single-channel signal, Time Scope now displays a yellow line instead of a blue line. Models containing Time Scope blocks that were created using older versions of DSP System Toolbox will not be affected by this change.

See the Time Scope reference topic for more information.

Sampled Data as Stairs in Time Scope

In previous releases, the Time Scope plotted a sampled signal as lines connecting each of the sampled values. This approach is similar to the functionality of the MATLAB `line` or `plot` function. In R2012a, the Time Scope block and System object can also plot a sampled signal as horizontal lines. These lines represent a sample value for a discrete sample period connected by vertical lines to represent a change in values occurring at each new sample. This type of plot is commonly called a Stairstep graph and has functionality similar to that of the MATLAB `stairs` function. Stairstep graphs are useful for drawing time history graphs of digitally sampled data.

To display a sampled signal as a Stairstep graph, first select **Style** from the **View** menu. The Style dialog box opens, allowing you to set the **Plot type** drop down box to **Stairs**. The three options available are **Line**, **Stairs**, and **Auto**. If using the `dsp.TimeScope` System object, set the `PlotType` property to the string `'stairs'`.

See the Time Scope reference topic for more information.

Complex Data Support in Time Scope

Beginning in this release, the Time Scope block and System object will support complex data input. The complex data is displayed by default in real and imaginary form as differently colored lines on the same axes. Alternately, you can display the magnitude and phase of the signal on separate axes in the same display.

To change the complex data options in the Time Scope display, select **Properties** from the **View** menu. Then, select or deselect the **Plot signals as magnitude and phase** check box. You can also right-click on the axes and select **Properties**. To change these properties using the `dsp.TimeScope` System object, set the `MagnitudePhase` property to either `true` or `false`.

See the Time Scope reference topic for more information.

Additional Time Scope Enhancements

In addition to the modifications mentioned above, R2012a also includes the following enhancements to the Time Scope:

Ability to Change the Time Units of the Display

In previous releases Time Scope always displayed time in metric units. In R2012a, the Time Scope block and System object allow you to label the X-axis in two additional ways. First, you can ensure that the X-axis is always labeled as `Time (seconds)` and that the appropriate power of 10 appears in the bottom-right corner of the Time Scope display. Second, you can remove the units in the X-axis label entirely.

To change the manner in which the time units are displayed, select **Properties** from the **View** menu. Then, set the **Time Units** parameter to either `Seconds` or `None`. The default option is `Metric (based on Time Span)`. To change these properties using the `dsp.TimeScope` System object, set the `TimeUnits` property to either `Seconds` or `None`. The default property value is `Metric`.

See the Time Scope reference topic for more information.

Simulink Enumerations Supported in Time Scope Block

In previous releases, the Time Scope block supported input signals of floating-point and fixed-point data types. R2012a adds support for Simulink enumerated data types.

Note This feature is available only for the Time Scope block but not for the Time Scope System object. Also, support is provided only for Simulink enumerations, but not for generic MATLAB enumeration classes.

See the Time Scope reference topic for more information.

ASIO Support in To/From Audio Device Blocks and Objects

The To and From Audio Device blocks and the `dsp.AudioPlayer` and `dsp.AudioRecorder` system objects all now support ASIO as an API. ASIO is used to communicate with the audio hardware. To set ASIO as the Audio Hardware API, select **Preferences** from the MATLAB Toolstrip. Then select DSP System Toolbox from the tree menu. If the ASIO selection is disabled, it is due to the ASIO device not being connected.

Video Processing Enabled for the DSP System Toolbox Multimedia File Blocks

The To Multimedia File and From Multimedia File blocks no longer require a Computer Vision System Toolbox license for video processing. You can use the DSP From Multimedia File block to read video and the To Multimedia File block to write video files.

System Objects Integrated into Filter Design Workflow

- “Integration of System Objects into Filter Design via `fdesign`, `FDATool`, and `Filterbuilder`” on page 21-8
- “Convert `dfilt` and `mfilt` Filter Objects to System Objects” on page 21-8
- “Filter Analysis and Conversion Methods for System Object Filters” on page 21-8

Integration of System Objects into Filter Design via `fdesign`, `FDATool`, and `Filterbuilder`

In R2012a, you can use the `fdesign` workflow and interactive tools, `fdatool` and `filterbuilder`, to create IIR, FIR, and multirate System object filters.

Using the interactive tools, you can generate MATLAB code to construct a System object filter. You can also generate MATLAB code to filter data with your System object that is compatible with C/C++ code generation. C/C++ code generation requires the MATLAB Coder software.

Convert `dfilt` and `mfilt` Filter Objects to System Objects

In R2012a, you can convert `dfilt` and `mfilt` objects to System objects. Use the `sysobj` method to convert an existing `dfilt` or `mfilt` object to a System object. Refer to the command-line help for `dfilt.sysobj` and `mfilt.sysobj` for details on supported single-rate and multirate filter structures.

Filter Analysis and Conversion Methods for System Object Filters

The R2012a release extends filter analysis and conversion methods for `dfilt` and `mfilt` objects to System object filters. For System object filters, you can examine the filter magnitude, impulse, step, zero phase, group delay, and phase responses. You can also view a pole-zero plot of your filter’s z -transform.

Additionally, you can obtain detailed measurements and implementation costs for your System object filter. For System object filters, you can determine if the phase response is linear. For FIR linear-phase filters, you can determine the type of linear phase. You can also assess the stability of your filter design and whether your design represents a minimum-phase or maximum-phase system.

To obtain a comprehensive list of supported methods and links to the command-line help, enter

```
dsp.SystemObjectFilter.helpFilterAnalysis
```

at the command line, where *SystemObjectFilter* is a specific System object filter class name. For example:

```
dsp.BiquadFilter.helpFilterAnalysis
```

New Measurement Workflow

- “Measurements for Bilevel Pulse Waveforms” on page 21-9
- “System Objects for Peak-to-RMS and Peak-to-Peak Measurements” on page 21-9

Measurements for Bilevel Pulse Waveforms

The R2012a release introduces System objects that perform a number of basic measurements on bilevel pulse waveforms. These measurements include:

- State-level estimation for bilevel pulse waveforms using the histogram method. See the help for `dsp.StateLevels` for details.
- Transition metrics for bilevel pulse waveforms. The System object, `dsp.TransitionMetrics`, determines low-, middle-, and high-reference level crossings and also duration and slew rate. You can also use `dsp.TransitionMetrics` to measure the behavior of bilevel waveforms in pretransition and posttransition regions such as overshoot, undershoot, and settling time.

Using `dsp.PulseMetrics`, you can measure transition rise and fall times. `dsp.PulseMetrics` contains a superset of the capabilities found in `dsp.TransitionMetrics`.

- Cycle metrics for bilevel pulse waveforms. You can use `dsp.PulseMetrics` to measure pulse width, pulse separation, pulse period, and duty cycle.

System Objects for Peak-to-RMS and Peak-to-Peak Measurements

The R2012a release introduces System objects to measure the root-mean-square (RMS) value of a waveform. These System objects also measure the peak-to-RMS and peak-to-peak values. For details, see the reference pages for `dsp.RMS`, `dsp.PeakToRMS`, and `dsp.PeakToPeak`.

Discrete FIR Filter System Object

This release introduces a discrete FIR Filter System object, which filters each channel of the input using static or time-varying FIR filter implementations. This System object implements the algorithm, inputs, and outputs described on the Discrete FIR Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. The object uses the `FrameBasedProcessing` property. The block uses the **Input processing** parameter. This System object supports code generation.

Inverse Dirichlet Sinc-Shaped Passband Design Added to Constrained FIR Equiripple Filter

The R2012a release adds the ability to design a constrained FIR equiripple filter with an inverse-Dirichlet-sinc-shaped passband using `firceqrip`. An inverse-Dirichlet-sinc-shaped response is often used to compensate for the Dirichlet-sinc-shaped response of a cascade integrator comb (CIC) filter. An inverse-sinc-shaped response is a valid approximation to the response of a CIC filter only when the sampling rate change is sufficiently high. The Dirichlet sinc provides a more exact match to the response of a CIC filter.

Code Generation Support Added to FIR Decimator System Object

In this release, FIR Decimator has been added to the list of objects that can now support code generation in MATLAB via the `codegen` function.

Filter Block Enhancements

- “IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function” on page 21-10
- “Reset Port for Resetting Filter State in Filter Blocks” on page 21-10

IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function

The Simulink Discrete Filter and Discrete Transfer Function blocks now have the capability of specifying the numerator and denominator coefficients via either input parameter ports or block dialog boxes. Also, the initial filter states can be specified via an input parameter port or a block dialog box.

Reset Port for Resetting Filter State in Filter Blocks

The Simulink Discrete Filter and Discrete Transfer Function blocks now allow the filter states to be reset via a reset parameter port called External reset.

Discrete FIR Filter Block Coefficient Port Changes

In this release, if you feed a column vector input into the coefficient port of the Discrete FIR Filter block, the block issues a command-line warning. This warning will state that column vector inputs to the coefficient port are not supported and that you will see an error message in future releases.

Version History

You are advised to run `s_lupdate` to insert a reshape block and transpose the input from a column vector to a row vector.

Statistics Blocks and Objects Warning for Region of Interest Processing

ROI processing will be removed in a future release. Currently, ROI processing is available only if you have a Computer Vision System Toolbox license. If you do not have a license for that product, you can still use ROI processing, but you are limited to the use of ROI type rectangles.

Version History

When you use region of interest (ROI) processing, MATLAB will issue a warning. ROI processing will be removed in a future release.

New and Updated Demos

R2012a adds the following new demo:

Using System Objects with MATLAB Coder — Shows you how to use the MATLAB Coder product to generate code for a MATLAB file that uses System objects.

Additionally, this release updates the Arbitrary Magnitude Filter Design demo to include examples that showcase the new capabilities of the arbitrary magnitude filter design.

R2011b

Version: 8.1

New Features

Bug Fixes

Version History

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. DSP System Toolbox documentation refers to the former as frame-based processing and the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal by means of a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame based and displays it as a double line, rather than the single line sample-based signal.

General Product-Wide Changes

Beginning in R2010b, MathWorks® started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, many blocks have received new parameters. You can view an example of how to use these parameters to control sample- and frame-based processing in R2011b and future releases. To open the model, type `ex_inputprocessing` at the MATLAB command line. This model demonstrates how a block can process a signal as sample based or frame based, depending on the setting of that block's **Input processing** parameter.

Notice that when the Discrete FIR Filter and Time Scope blocks are configured to perform frame-based processing, they interpret columns as channels and treat the 2-by-2 input signal as two independent channels. Conversely, when the blocks are configured to perform sample-based processing, they interpret elements as channels and treat the 2-by-2 input signal as four independent channels. For further information about sample- and frame-based processing, see [Sample- and Frame-Based Concepts](#).

The following sections provide more detailed information about the specific R2011b DSP System Toolbox software changes that are helping to enable the transition to the new way of frame-based processing:

- “Logging Signals in Simulink” on page 22-3
- “Triggered to Workspace” on page 22-3
- “Digital Filter Design Block” on page 22-4
- “Filterbuilder, FDATool and the Filter Realization Wizard Block” on page 22-5
- “Changes to Row Vector Processing for `dsp.Convolver`, `dsp.CrossCorrelator`, and `dsp.Interpolator` System Objects” on page 22-5

Version History

During this transition to the new way of handling frame-based processing, both the old way (frame status as an attribute of a signal) and the new way (each block controls whether to treat inputs as samples or as frames) will coexist for a few releases. For now, the frame bit will still flow throughout a model, and you will still see double signal lines in your existing models that perform frame-based processing.

-
- **Backward Compatibility** — By default, when you load an existing model in R2011b any new parameters related to the frame-based processing change will be set to their backward-compatible option. For example, if any blocks in your existing models received a new **Input processing** parameter this release, that parameter will be set to **Inherited** (this choice will be removed - see release notes) when you load your model in R2011b. This setting enables your existing models to continue working as expected until you upgrade them. Because the inherited option will be removed in a future release, you should upgrade your existing models as soon as possible.
 - **slupdate Function** — To upgrade your existing models to the new way of handling frame-based processing, you can use the `slupdate` function. Your model must be compilable in order to run the `slupdate` function. The function detects all blocks in your model that are in need of updating, and asks you whether you would like to upgrade each block. If you select yes, the `slupdate` function updates your blocks accordingly.
 - **Timely Update to Avoid Unexpected Results** — It is important to update your existing models as soon as possible because the frame bit will be removed in a future release. At that time, any blocks that have not yet been upgraded to work with the new paradigm of frame-based processing will automatically transition to perform their library default behavior. The library default behavior of the block might not produce the results you expected, thus causing undesired results in your models. Once the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Logging Signals in Simulink

R2011b adds new capabilities to the DSP System Toolbox product for logging signals in Simulink. When you log signals using the **Dataset** logging mode, you can now use DSP System Toolbox utility functions to help you access that logged data in either a 2-D or 3-D format. For more information about selecting a signal logging format, see *Specifying the Signal Logging Data Format* in the Simulink documentation.

After you log a signal using the **Dataset** logging mode, you can choose to extract that logged signal in either a 2-D or 3-D format. To fully support this new workflow, the following utility functions and class have been added to the DSP System Toolbox product:

- `dsp.util.getLogArray` — Formats and returns a 2-D or 3-D MATLAB array from a logged signal in a **Dataset** object.
- `dsp.util.getSignalPath` — Returns all paths to signals with a specified name in the **Dataset** object.
- `dsp.util.SignalPath` — Contains path information for signals in `Simulink.SimulationData.Dataset` objects.

Triggered to Workspace

R2011b adds a new **Save 2-D signals as** parameter to the **Triggered to Workspace** block. This parameter allows you to specify whether the block saves 2-D signals as 2-D arrays or as 3-D arrays. To provide for backward compatibility, the **Save 2-D signals as** parameter also has an option **Inherit from input** (this choice will be removed – see release notes). When you select this option, the block saves sample-based data as a 3-D array and frame-based data as a 2-D array.

Version History

In a future release, the following option will be removed: **Inherit from input** (this choice will be removed – see release notes). From this time forward, you must specify whether the block saves 2-D signals as 2-D or 3-D arrays. The block will no longer make that choice based on the status of the frame bit.

You can use the `slupdate` function to upgrade your existing models that contain a Triggered To Workspace block. The function detects whether your models contain any Triggered To Workspace blocks with the **Save 2-D signals as** parameter set to **Inherit from input** (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit and sets the **Save 2-D signals as** parameter accordingly.

- If the input signal is frame based, the function sets the **Save 2-D signals as** parameter to 2-D array (concatenate along first dimension).
- If the input signal is sample based, the function sets the **Save 2-D signals as** parameter to 3-D array (concatenate along third dimension).

Digital Filter Design Block

R2011b adds a new **Input processing** parameter to the Digital Filter Design block. This parameter allows you to choose whether you want the block to perform sample- or frame-based processing on the input. You can set this parameter to either **Elements as channels** (sample based) or **Columns as channels** (frame based). The third choice, **Inherited** (this choice will be removed - see release notes), is a temporary selection. This additional option will help you as you move control of frame-based processing from the signals to the blocks themselves.

Version History

When you load an existing model R2011b, all Digital Filter Design blocks in your model will have the new **Input processing** parameter. By default, it will be set to **Inherited** (this choice will be removed - see release notes). This setting enables your existing models to continue to work as expected until you upgrade them. Although your old models will still work when you open and run them in R2011b, you should upgrade them as soon as possible.

You can upgrade your existing models using the `slupdate` function. The function detects all blocks that have **Inherited** (this choice will be removed - see release notes) selected for the **Input processing** parameter. It then asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to **Columns as channels** (frame based). If the bit is 0 (samples), the function sets the parameter to **Elements as channels** (sample based).

In a future release, the frame bit and the **Inherited** (this choice will be removed - see release notes) option will be removed. At that time, the **Input processing** parameter on blocks in models that have not been upgraded will automatically be set to the block's library default setting. In the case of the Digital Filter Design block, the library default setting is **Columns as channels** (frame based). If the library default setting does not match the parameter setting in your model, your model will produce unexpected results.

Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Filterbuilder, FDATool and the Filter Realization Wizard Block

R2011b adds new **Input processing** and **Rate options** parameters to `filterbuilder`, `FDATool` and the Filter Realization Wizard block. For `filterbuilder`, these new parameters are available when you click the **Generate Model** button on the Code Generation pane of the dialog box. For `FDATool` and the Filter Realization Wizard block, the new parameters are available on the Realize Model pane of the dialog box. When you use the **Realize Model** button to create a block, you can use the **Input processing** parameter to specify whether the block will perform sample- or frame-based processing on its input.

If you are creating a multirate filter block, the **Rate options** parameter will also be available. This parameter allows you to specify whether the filter block you create will **Enforce single-rate processing** or **Allow multirate processing**.

Changes to Row Vector Processing for `dsp.Convolver`, `dsp.CrossCorrelator`, and `dsp.Interpolator System Objects`

In previous releases, the `dsp.Convolver`, `dsp.CrossCorrelator`, and `dsp.Interpolator System` objects processed row vector inputs as a column vector. As of R2011b, these objects now process row vector inputs as a row vector (multiple channels).

Version History

Starting in R2011b, you must update your code to transpose the row vector data to a column vector before providing it as an input to the `dsp.Convolver`, `dsp.CrossCorrelator`, or `dsp.Interpolator System` objects.

Custom System Objects

You can now create custom System objects in MATLAB. This capability allows you to define your own System objects for time-based and data-driven algorithms, I/O, and visualizations. The System object API provides a set of implementation and service methods that you incorporate into your code to implement your algorithm. See [Define New System Objects](#) for more information.

New Allpole Filter Block

R2011b adds a new Allpole Filter block to the Filtering/Filter Implementations library. This block provides direct form, direct form transposed, and Lattice AR allpole filter structures.

New Audio Weighting Filter Functionality

R2011b adds new audio weighting filter functionality to MATLAB and Simulink. In MATLAB, you can now design audio weighting filters in the `filterbuilder` GUI or by using the preexisting `fdesign.audioweighting` object. In Simulink, you can use the new Audio Weighting Filter block from the Filtering/Filter Designs library.

Time Scope Enhancements

R2011b includes the following enhancements to the Time Scope:

- **Improvements to default signal names in the scope legend** — In previous releases, the default names for signals displayed by the Time Scope block were Channel 1, Channel 2, Channel 3, etc. In R2011b, the default naming convention has been improved to also identify the source of the signal. For example, if the input to the Time Scope block is two separate two channel signals named SignalA and SignalB, the default legend names would appear as:

SignalA:1, SignalA:2, SignalB:1, SignalB:2

See the Time Scope reference topic for more information.

- **New scrolling display mode simplifies debugging process** — R2011b adds a new option to the Time Scope block and System object. This option allows you to specify how the scope displays new data beyond the visible time span. In previous releases, the scope always displayed new data up until it reached the maximum X-axis limit. When the data reached the maximum X-axis limit of the scope window, the scope cleared the display and updated the time offset value. It then displayed subsequent data points starting from the minimum X-axis limit. In the new scrolling display mode, the scope scrolls old data to the left to make room for new data on the right side of the scope display. This mode is graphically intensive and can affect run-time performance, but it is beneficial for debugging and for monitoring time-varying signals.

To use the new scrolling display mode in the Time Scope block, set the **Time span overrun mode** parameter to `Scroll` on the **Visuals:TimeDomainOptions** dialog box. To use the new scrolling display mode in the `dsp.TimeScope` System object, set the `TimeSpanOverrunMode` property to `Scroll`. By default, both the block and the System object display data using the previously supported `Wrap` mode.

New Arbitrary Group Delay Design Support

This release adds a new `fdesign.arbgrpdelay` filter specification object. Arbitrary group delay filters are allpass filters useful for correcting phase distortion introduced by other filters. Systems with nonlinear phase responses result in nonconstant group delay, which causes dispersion of the frequency components of the signal. This type of phase distortion can be undesirable even if the magnitude distortion introduced by the filter produces the desired effect. In these cases, you can compensate for the phase distortion by cascading the frequency-selective filter with an allpass filter that compensates for the group delay.

Arbitrary Magnitude Responses Now Support Minimum Order and Minimum/Maximum Phase Equiripple Design Options

R2011b adds new minimum order, minimum phase equiripple, and maximum phase equiripple design options. These design options are now available on the Arbitrary Response design panel in `filterbuilder` and through the `fdesign.arbmag` filter specification object.

Support for Constrained Band Equiripple Designs in MATLAB and Simulink

R2011b adds support for constrained band equiripple designs to the following filter response types:

fdesign Filter Specification Object	filterbuilder Response String	dspfdesign Library Block
<code>fdesign.arbmag</code>	<code>arbmag</code>	Arbitrary Response Filter
<code>fdesign.bandpass</code>	<code>bandpass</code>	Bandpass Filter
<code>fdesign.bandstop</code>	<code>bandstop</code>	Bandstop Filter
<code>fdesign.differentiator</code>	<code>diff</code>	Differentiator Filter

New Sinc Frequency Factor and Sinc Power Design Options for Inverse Sinc Filters

R2011b adds two new design options for designing inverse sinc filters in MATLAB and Simulink. The new design options allow you to control the sinc frequency factor and sinc power for inverse sinc filters designed with `fdesign.isinclp`, the new `fdesign.isinchp`, the `isinc` filterbuilder response type, or the Inverse Sinc Filter block in the `dspfdesign` library.

These new design options allow you to design inverse sinc lowpass filters with a passband magnitude response equal to $H(\omega) = \text{sinc}(C\omega)^{-P}$. C is the sinc frequency factor, and P is the sinc power. Similarly, you can design inverse sinc highpass filters with a passband magnitude response equal to $H(\omega) = \text{sinc}(C(1-\omega))^{-P}$. For both the highpass and lowpass filters, the default values of C and P are set to 0.5 and 1, respectively. For more information about the sinc frequency factor and sinc power design options, see the corresponding reference topics.

New Inverse Sinc Highpass Filter Designs

This release adds support for designing highpass inverse sinc filters in MATLAB and Simulink. This capability is available through a new `fdesign.isinchp` filter specification object as well as a new `isinchp` filterbuilder response type.

Filterbuilder and dspfdesign Library Blocks Now Support Different Numerator and Denominator Orders for IIR Filters

As of R2011b, you can now specify different numerator and denominator orders for IIR filters designed using certain filter responses. This capability is available for the bandpass, bandstop, highpass, and lowpass filter responses in the `filterbuilder` GUI and the corresponding `dspfdesign` library blocks.

New Stopband Shape and Stopband Decay Design Options for Equiripple Highpass Filter Designs

This release adds new stopband shape and stopband decay design options in MATLAB and Simulink. These options are available through the `fdesign.highpass` filter specification object, the `highpass` filterbuilder response type, and the Highpass Filter block in the `dspfdesign` library.

FFTW Library Support for Non-Power-of-Two Transform Length

The FFT, IFFT blocks, and the `dsp.IFFT`, `dsp.FFT` System objects include the use of the FFTW library. The blocks and objects now support non-power-of-two transform lengths.

MATLAB Compiler Support for `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter`

R2011b adds MATLAB Compiler support for the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. With this capability, you can use the MATLAB Compiler to take MATLAB files, which can include System objects, as input and generate standalone applications.

Complex Input Support for `dsp.DigitalDownConverter`

The `dsp.DigitalDownConverter` System object now supports complex inputs.

`getFilters` Method of `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` Now Return Actual Fixed-Point Settings

You can now access the actual fixed-point settings of the filter being used by the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. To do so, you must first provide a fixed-point input to the object using the `step` method. Then, after the object is locked, call the `getFilters` method to access the actual fixed-point properties of the filter being implemented by the System object. Calling `getFilters` on an unlocked `dsp.DigitalDownConverter` or `dsp.DigitalUpConverter` System object returns the same results as previous releases.

`dsp.SineWave` and `dsp.BiquadFilter` Properties Not Tunable

The following `dsp.SineWave` properties are now nontunable:

- Frequency
- PhaseOffset

The following `dsp.BiquadFilter` properties are now nontunable:

- SOSMatrix
- ScaleValues

When objects are locked (i.e., after calling the `step` method), you cannot change any nontunable property values.

Version History

Review any code that changes any `dsp.SineWave` or `dsp.BiquadFilter` property value after calling the `step` method. You should update the code to use property values that do not change.

System Object `DataType` and `CustomDataType` Properties Changes

When you set a System object, fixed-point `<xxx>DataType` property to 'Custom', it activates a dependent `Custom<xxx>DataType` property. If you set that dependent `Custom<xxx>DataType` property before setting its `<xxx>DataType` property, a warning message displays. `<xxx>` differs for each object.

Version History

Previously, setting the dependent Custom<xxx>DataType property would automatically change its <xxx>DataType property to 'Custom'. If you have code that sets the dependent property first, avoid warnings by updating your code. Set the <xxx>DataType property to 'Custom' before setting its Custom<xxx>DataType property.

Note If you have a Custom<xxx>DataType in your code, but do not explicitly update your code to change <xxx>DataType to 'Custom', you may see different numerical output.

System Objects Variable-Size Input Dimensions

System objects that process variable-size input now also accept inputs where the number of input dimensions change.

Conversion of Error and Warning Message Identifiers

R2011b changes some error and warning message identifiers in DSP System Toolbox software. For System objects, both error and warning message identifiers have changed. On the Simulink side, the Time Scope block has one warning message with a new identifier.

Version History

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages. You can also use them in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, for System objects, the MATLAB:system:System:inputSpecsChangedWarning identifier has changed to MATLAB:system:inputSpecsChangedWarning. If your code checks for MATLAB:system:System:inputSpecsChangedWarning, you must update it to check for MATLAB:system:inputSpecsChangedWarning instead.

For the Time Scope block, the Simulink:Engine:Simulink:Engine:UnableToUpdateDisplayInRapidAccelMode identifier has changed to Simulink:Engine:UINotUpdatedDuringRapidAccelSim. If your code checks for Simulink:Engine:Simulink:Engine:UnableToUpdateDisplayInRapidAccelMode, you must update it to check for Simulink:Engine:UINotUpdatedDuringRapidAccelSim instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable *MSGID*.

To determine the identifier for an error that appears at the MATLAB prompt, run the following command just after you see the error:

```
exception = MException.last;  
MSGID = exception.identifier;
```

Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs without warnings.

New and Updated Demos

R2011b adds the following new demos:

- **3-Band Parametric Audio Equalizer Using UDP Packets and Code Generation** — Provides a three-band parametric equalizer algorithm based in MATLAB. The demo allows you to dynamically adjust the coefficients of the filters and shows you how to use the MATLAB Coder product to build a standalone executable file that you can run outside of MATLAB.
- **Creating New Kinds of System Objects for File Input and Output** — Provides an example of creating custom System objects in MATLAB for file input and output.

Additionally, this release updates the IIR Filter Design Given a Prescribed Group Delay demo to use the new `fdesign.arbgrpdelay` object.

Blocks Being Removed in a Future Release

The following blocks will be removed from the DSP System Toolbox product in a future release.

Block Being Removed (library)	Replacement Block
Digital FIR Filter Design (<code>dspddes3</code>)	Discrete FIR Filter
Remez FIR Filter Design (<code>dspddes3</code>)	Discrete FIR Filter
Least Squares FIR Filter Design (<code>dspddes3</code>)	Discrete FIR Filter
Digital FIR Raised Cosine Filter Design (<code>dspddes3</code>)	Discrete FIR Filter
Digital IIR Filter Design (<code>dspddes3</code>)	Discrete Filter
Yule-Walker IIR Filter Design (<code>dspddes3</code>)	Discrete Filter
Integer Delay (<code>dspobslib</code>)	Delay
Dyadic Analysis Filter Bank (<code>dspobslib</code>)	Dyadic Analysis Filter Bank (<code>dspmlti4</code>)
Dyadic Synthesis Filter Bank (<code>dspobslib</code>)	Dyadic Synthesis Filter Bank (<code>dspmlti4</code>)
Wavelet Analysis (<code>dspobslib</code>)	DWT
Wavelet Synthesis (<code>dspobslib</code>)	IDWT
Direct-Form II Transpose Filter (<code>dsparch3</code>)	Digital Filter
Time-Varying Direct-Form II Transpose Filter (<code>dsparch3</code>)	Digital Filter, Discrete FIR Filter, or Allpole Filter
Time-Varying Lattice Filter (<code>dsparch3</code>)	Digital Filter, Discrete FIR Filter, or Allpole Filter

Version History

Beginning in R2011b, Simulink will generate a warning when you load a model that contains one or more of the blocks listed in the preceding table. To ensure that your models continue to work as expected when these blocks are removed from the product in a future release, it is strongly

recommended that you replace these unsupported blocks as soon as possible. You can automatically update the blocks in your model by using the `s!update` function.

R2011a

Version: 8.0

New Features

Bug Fixes

Version History

Product Restructuring

The DSP System Toolbox product replaces the Signal Processing Blockset™ and Filter Design Toolbox™ products in R2011a.

You can access archived documentation for the Signal Processing Blockset and Filter Design Toolbox products on the MathWorks Web site.

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. DSP System Toolbox documentation refers to the former as frame-based processing and the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal by means of a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame based and displays it as a double line, rather than the single line sample-based signal.

General Product-Wide Changes

Beginning in R2010b, MathWorks started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, many blocks have received new parameters. You can view an example of how to use these parameters to control sample- and frame-based processing in R2011a and future releases. To open the model, type `ex_inputprocessing` at the MATLAB command line. This model demonstrates how a block can process a signal as sample based or frame based, depending on the setting of that block's **Input processing** parameter.

Notice that when the Discrete FIR Filter and Time Scope blocks are configured to perform frame-based processing, they interpret columns as channels and treat the 2-by-2 input signal as two independent channels. Conversely, when the blocks are configured to perform sample-based processing, they interpret elements as channels and treat the 2-by-2 input signal as four independent channels. For further information about sample- and frame-based processing, see [Sample- and Frame-Based Concepts](#).

The following sections provide more detailed information about the specific R2011a DSP System Toolbox software changes that are helping to enable the transition to the new way of frame-based processing:

- “Blocks with a New Input Processing Parameter” on page 23-3
- “Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks” on page 23-5
- “Difference Block Changes” on page 23-5
- “Signal To Workspace Block Changes” on page 23-6

-
- “Spectrum Scope Block Changes” on page 23-6
 - “Sample-Based Row Vector Processing Changes” on page 23-6

Version History

During this transition to the new way of handling frame-based processing, both the old way (frame status as an attribute of a signal) and the new way (each block controls whether to treat inputs as samples or as frames) will coexist for a few releases. For now, the frame bit will still flow throughout a model, and you will still see double signal lines in your existing models that perform frame-based processing.

- **Backward Compatibility** — By default, when you load an existing model in R2011a any new parameters related to the frame-based processing change will be set to their backward-compatible option. For example, if any blocks in your existing models received a new **Input processing** parameter this release, that parameter will be set to **Inherited (this choice will be removed - see release notes)** when you load your model in R2011a. This setting enables your existing models to continue working as expected until you upgrade them. Because the inherited option will be removed in a future release, you should upgrade your existing models as soon as possible.
- **slupdate Function** — To upgrade your existing models to the new way of handling frame-based processing, you can use the `slupdate` function. Your model must be compilable in order to run the `slupdate` function. The function detects all blocks in your model that are in need of updating, and asks you whether you would like to upgrade each block. If you select yes, the `slupdate` function updates your blocks accordingly.
- **Timely Update to Avoid Unexpected Results** — It is important to update your existing models as soon as possible because the frame bit will be removed in a future release. At that time, any blocks that have not yet been upgraded to work with the new paradigm of frame-based processing will automatically transition to perform their library default behavior. The library default behavior of the block might not produce the results you expected, thus causing undesired results in your models. Once the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

For more detailed information about the specific compatibility considerations related to the R2011a frame-based processing changes, see the following Compatibility Considerations sections.

Blocks with a New Input Processing Parameter

Some DSP System Toolbox blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will require a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks are receiving a new **Input processing** parameter. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending upon the type of processing you want. The third choice, **Inherited (this choice will be removed - see release notes)**, is a temporary selection. This additional option will help you to migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

For a list of blocks that received a new **Input processing** parameter in R2011a, expand the following list.

Blocks with the New Input Processing Parameter

- Arbitrary Response Filter
- Bandpass Filter
- Bandstop Filter
- CIC Compensator
- CIC Filter
- Comb Filter
- Differentiator Filter
- Halfband Filter
- Highpass Filter
- Hilbert Filter
- Inverse Sinc Filter
- Lowpass Filter
- Nyquist Filter
- Octave Filter
- Parametric Equalizer
- Peak-Notch Filter
- Pulse Shaping Filter
- Unwrap

For a list of blocks that received an **Input processing** parameter in R2010b, see the R2010b Signal Processing Blockset Release Notes.

Version History

When you load an existing model R2011a, any block with the new **Input processing** parameter will show a setting of `Inherited` (this choice will be removed - see release notes). This setting enables your existing models to continue to work as expected until you upgrade them. Although your old models will still work when you open and run them in R2011a, you should upgrade them as soon as possible.

You can upgrade your existing models, using the `sIupdate` function. The function detects all blocks that have `Inherited` (this choice will be removed - see release notes) selected for the **Input processing** parameter, and asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels (frame based)`. If the bit is 0 (samples), the function sets the parameter to `Elements as channels (sample based)`.

In a future release, the frame bit and the `Inherited` (this choice will be removed - see release notes) option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer

be able to upgrade your models using the `s_lupdate` function. Therefore, you should upgrade your existing models using `s_lupdate` as soon as possible.

Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks

R2011a updates the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks to the use new way of frame-based processing. In previous releases, the frame status of the input signal determined how these blocks processed the input. In R2011a, the default behavior of these blocks is to always perform frame-based processing.

Unless you specify otherwise, these blocks now treat each column of the input signal as an individual channel, regardless of its frame status. You can now enable the behavior change in these blocks while still allowing for backward compatibility. This release adds a **Treat Mx1 and unoriented sample-based signals as** parameter for this purpose. This parameter will be removed in a future release, at which point the blocks will always perform frame-based processing.

Version History

The **Treat Mx1 and unoriented sample-based signals as** parameter will be removed in a future release. From that point, the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks will always perform frame-based processing.

You can use the `s_lupdate` function to upgrade your existing models that contain one of these blocks. The function detects all Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks in your model. Then, if you allow it to, `s_lupdate` performs the following actions:

- If the input to the block is an M -by-1 or unoriented sample-based signal, the `s_lupdate` function:
 - Places a Transpose block in front of the affected block in your model. This block transposes the M -by-1 or unoriented sample-based input into a 1-by- M row vector. By converting the input to a row vector, the block continues to produce the same results as in previous releases (an M_o -by- M_i output).
 - Sets the **Treat Mx1 and unoriented sample-based signals as** parameter to `One channel`. This setting ensures that your model will continue to produce the same results when the **Treat Mx1 and unoriented sample-based signals as** parameter is removed in a future release.
- If the input to the block is *not* an M -by-1 or unoriented sample-based signal, the `s_lupdate` function sets the **Treat Mx1 and unoriented sample-based signals as** parameter to `One channel`. This setting does not affect the behavior of your current model. However, the change does ensure that your model will continue to produce the same results when the **Treat Mx1 and unoriented sample-based signals as** parameter is removed in a future release.

Difference Block Changes

R2011a adds a new **Running difference** parameter to the Difference block.

Version History

In a future release, the following option for the **Running difference** parameter will be removed: `Inherit from input` (this choice will be removed – see release notes). From this time forward, you must specify whether or not the block computes a running difference; the block will no longer make that choice based on the status of the frame bit.

You can use the `sIupdate` function to upgrade your existing models that contain a Difference block. The function detects whether your models contain any Difference blocks with the **Running difference** parameter set to *Inherit from input* (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit, and sets the **Running difference** parameter accordingly.

Signal To Workspace Block Changes

R2011a updates the Signal To Workspace block. The block now allows you to choose an output format using the **Save format** parameter. You can choose to save your data as an Array, Structure, or Structure with time.

Additionally, the old **Frames** parameter has been replaced by a new **Save 2-D signals as** parameter. This parameter allows you to specify whether the block saves 2-D signals as a 2-D array, or as a 3-D array. To provide for backward compatibility, the **Save 2-D signals as** parameter also has an option *Inherit from input* (this choice will be removed – see release notes). When you select this option, the block saves sample-based data as a 3-D array and frame-based data as a 2-D array.

Version History

In a future release, the following option will be removed: *Inherit from input* (this choice will be removed – see release notes). From this time forward, you must specify whether the block saves signals as a 2-D or 3-D array. The block will no longer make that choice based on the status of the frame bit.

You can use the `sIupdate` function to upgrade your existing models that contain a Signal To Workspace block. The function detects whether your models contain any Signal To Workspace blocks with the **Save 2-D signals as** parameter set to *Inherit from input* (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit and sets the **Save 2-D signals as** parameter accordingly.

- If the input signal is frame based, the function sets the **Save 2-D signals as** parameter to 2-D array (concatenate along first dimension).
- If the input signal is sample based, the function sets the **Save 2-D signals as** parameter to 3-D array (concatenate along third dimension).

Spectrum Scope Block Changes

R2011a updates the Spectrum Scope block to use the new way of frame-based processing. To enable this change, the block received a new **Treat Mx1 and unoriented sample-based signals as** parameter. This new parameter is available only when you select the **Buffer input** check box. By default, the new parameter is set to *One channel*. In this mode, the block treats *M*-by-1 and unoriented sample-based input as a single column vector and buffers the input along that column.

Sample-Based Row Vector Processing Changes

In previous releases, some DSP System Toolbox blocks handled sample-based row vector inputs in a special way. Of the blocks that can treat sample-based row vector inputs differently, there are two categories:

- Some blocks have a **Treat sample-based row input as a column** check box which allows you to explicitly specify how the block should treat sample-based row vector inputs. Expand the following section for a full list of these blocks.

Blocks with a Check Box

- Maximum
 - Mean
 - Median
 - Minimum
 - Normalization
 - RMS
 - Standard Deviation
 - Variance
- Other blocks automatically treat a sample-based row vector input as a single channel (column vector). Expand the following section for a full list of these blocks.

Blocks That Implicitly Treat Sample-Based Row Vectors as a Single Channel

- Autocorrelation
- Autocorrelation LPC
- Burg AR Estimator
- Burg Method
- Complex Cepstrum
- Convolution
- Correlation
- Covariance AR Estimator
- Covariance Method
- DCT
- FFT
- IDCT
- IFFT
- Interpolation
- Levinson-Durbin
- LPC to LSF/LSP Conversion
- LPC to/from Cepstral Coefficients
- LPC to/from RC
- LPC/RC to Autocorrelation
- LSF/LSP to LPC Conversion
- Modified Covariance AR Estimator
- Modified Covariance Method
- Peak Finder
- Polynomial Stability Test
- Real Cepstrum
- Sort

- Window Function
- Yule-Walker AR Estimator
- Yule-Walker Method

The special treatment of sample-based row vector inputs will be removed in a future release. See the compatibility considerations for more information about how this change will affect your models.

Version History

The blocks listed will continue to work as expected in R2011a. However, in a future release these blocks will produce a warning when you provide them with a sample-based row vector input, and eventually, their behavior will change.

You can prepare your models for the upcoming change by running the `s_lupdate` function. If the function detects any blocks that have a **Treat sample-based row input as a column** check box, it performs the following actions:

- If the input to the block is a sample-based row vector, and the **Treat sample-based row input as a column** check box is selected, the `s_lupdate` function places a Transpose block in front of the affected block. The Transpose block transposes the sample-based row vector into a column vector, which is then input into the affected block. Transposing the input signal ensures that your model will produce the same results in future releases.
- If the **Treat sample-based row input as a column** check box is not selected, or if the input to the block is not a sample-based row vector, the `s_lupdate` function takes no action. Your model will continue to work as expected in future releases.

If the `s_lupdate` function detects any blocks that automatically treat sample-based row vectors as a column, it performs the following actions:

- If the input to the block is a sample-based row vector, the `s_lupdate` function places a Transpose block in front of the affected block. The Transpose block transposes the sample-based row vector into a column vector, which is then input into the affected block. Transposing the input signal ensures that your model will produce the same results in future releases.
- If the input to the block is not a sample-based row vector, the `s_lupdate` function takes no action. Your model will continue to work as expected in future releases.

New Function for Changing the System Object Package Name from `signalblks` to `dsp`

In R2010b, the package name of Signal Processing Blockset™ System objects changed from `signalblks` to `dsp`. In R2011a, a new function is available to help you update your code. You can use the `sysobjupdate` function to recursively search a folder and its subfolders for MATLAB files that contain System object packages, classes, and properties that have been renamed.

Version History

If you have any existing System object code that uses a package name of `signalblks`, you should use the `sysobjupdate` function to update your code. For more information, type `help sysobjupdate` at the MATLAB command line.

New Discrete FIR Filter Block

R2011a adds a new Discrete FIR Filter block to the DSP System Toolbox Filtering/Filter Implementations library. The block is an implementation of the Simulink Discrete FIR Filter block.

New Printing Capability from the Time Scope Block

You can now print the data you see in the Time Scope block. To send the data to your printer, select **File > Print ...** from the scope menu. To print the data to a MATLAB figure, select **File > Print to Figure**.

Improved Display Updates for the Time Scope Block and System Object

R2011a introduces the capability to improve the performance of the Time Scope block and `dsp.TimeScope` System object by reducing the frequency with which the display updates. You can now choose between this new enhanced performance mode and the old behavior by selecting **Reduce Updates to Improve Performance** from the **Simulation** menu of the block, or the **Playback** menu of the System object. By default, both the block and System object operate in the new enhanced performance mode.

New Implementation Options Added to Blocks in the Filter Designs Library

This release provides filter customization options for blocks in the Filtering/Filter Designs library. You can access these options in the **Filter implementation** section of the block dialog box:

- Implement designed filters as Simulink basic elements or as a digital filter.
- Customize filters built using Simulink basic elements using the **Optimizations** parameters.

Blocks in the Filtering/Filter Designs library also support **Input processing** and **Rate options** parameters in R2011a. For more information, see “Blocks with a New Input Processing Parameter” on page 23-3.

Version History

- Frame-based processing and filters with algebraic loops — For filters that contain sample-by-sample feedback, using a lumped-element implementation instead of Simulink basic elements can eliminate algebraic loops. For supported blocks, use the `s_lupdate` function on older models with designed filters to convert the designed filters into lumped filters. You can enable this feature manually by clearing the **Use basic elements for filter customization** check box.

For filters with algebraic loops that do not have this option, specify sample-based processing by setting the **Input processing** parameter to `Elements as channels (sample based)`.

- **Rate Options** parameter — Filters that allow multirate processing, such as FIR decimators and interpolators, perform single-rate processing by default. For more information, see the block reference pages.

New `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System Objects

This release adds new `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. The digital up converter (DUC) and digital down converter (DDC) System objects provide tools to design interpolation/decimation filters and simplify the steps required to implement the up/down conversion process.

Improved Performance of FFT Implementation with FFTW library

The FFT, IFFT blocks include the use of the FFTW library.

Variable-Size Support for System Objects

The following System objects support inputs that change their size at runtime.

- `dsp.ArrayVectorAdder`
- `dsp.ArrayVectorDivider`
- `dsp.ArrayVectorMultiplier`
- `dsp.ArrayVectorSubtractor`
- `dsp.FFT`
- `dsp.IFFT`
- `dsp.Maximum`
- `dsp.Mean`
- `dsp.Minimum`
- `dsp.Normalizer`
- `dsp.RMS`
- `dsp.StandardDeviation`
- `dsp.UDPReceiver`
- `dsp.UDPSender`
- `dsp.Variance`

Version History

For the `dsp.UDPSender` and `dsp.UDPReceiver` System objects only, you should update your code to stop sending or receiving any data length settings. Support for variable-size data makes the data length settings redundant. For example,

```
% Change these lines to remove explicit lengths:
    step(hudps, dataSent, dataLength);
    [dataReceived len] = step(hudpr);
    bytesReceived = bytesReceived + ...
        length(dataReceived) len;

% Code lines with lengths removed:
    step(hudps,datasent);
    [dataReceived] = step(hudpr);
```

```
bytesReceived = bytesReceived + ...
    length(dataReceived);
```

System Objects FullPrecisionOverride Property Added

A FullPrecisionOverride property has been added to the System objects listed below. This property is a convenient way to control whether the object uses full precision to process fixed-point input.

When you set this property to `true`, which is the default, it eliminates the need to set many fixed-point properties individually. It also hides the display of these properties (such as `RoundingMode`, `OverflowAction`, etc.) because they are no longer applicable individually.

To set individual fixed-point properties, you must first set FullPrecisionOverride to `false`.

Note The `CoefficientDataType` property is not controlled by FullPrecisionOverride

The following System objects are affected:

- `dsp.ArrayVectorAdder`
- `dsp.ArrayVectorSubtractor`
- `dsp.Autocorrelator`
- `dsp.Convolver`
- `dsp.Crosscorrelator`
- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.FIRRateConverter`
- `dsp.SubbandAnalysisFilter`
- `dsp.SubbandSynthesisFilter`
- `dsp.Window`

Version History

All of these System objects have their new FullPrecisionOverride property set to the default, `true`. If you had set any fixed-point properties to non-default values for these objects, those values are ignored. As a result, you may see different numerical answers from those answers in a previous release. To use your nondefault fixed-point settings, you must first change FullPrecisionOverride to `false`.

'Internal rule' System Object Property Value Changed to 'Full precision'

To clarify the value of many `DataType` properties, the 'Internal rule' option has been changed to 'Full precision'.

Version History

The objects allow you to enter either 'Internal rule' or 'Full precision'. If you enter 'Internal rule', that option is stored as 'Full precision'.

MATLAB Compiler Support for System Objects

The DSP System Toolbox supports the MATLAB Compiler for most System objects. With this capability, you can use the MATLAB Compiler to take MATLAB files, which can include System objects, as input and generate standalone applications.

The following System objects are not supported by the MATLAB Compiler software:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`
- `dsp.DigitalDownConverter`
- `dsp.DigitalUpConverter`
- `dsp.TimeScope`

Viewing System Objects in the MATLAB Variable Editor

The MATLAB Variable Editor now displays System objects properties in the same order as they display at the command line. Note that the Variable Editor provides a read-only view for System objects.

System Object Input and Property Warnings Changed to Errors

When a System object is locked (e.g., after the `step` method has been called), the following situations now produce an error. This change prevents the loss of state information.

- Changing the input data type
- Changing the number of input dimensions
- Changing the input complexity from real to complex
- Changing the data type, dimension, or complexity of tunable property
- Changing the value of a nontunable property

Version History

Previously, the object issued a warning for these situations. The object then unlocked, reset its state information, relocked, and continued processing. To update existing code so that it does not produce an error, use the `release` method before changing any of the items listed above.

New and Updated Demos

R2011a adds the following new demos:

- **Digital Up and Down Conversion for Family Radio Service** — Shows you how to use the new `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects to design a Family Radio Service (FRS) transmitter and receiver.

-
- **Design and Analysis of a Digital Down Converter** — Shows you how to use the `dsp.DigitalDownConverter` System object to simplify the steps required to emulate the TI Graychip 4016 digital down converter.
 - **Using System Objects with MATLAB Compiler** — Shows you how to use MATLAB Compiler to create a standalone application from MATLAB System objects.

Additionally, the Simulink-based demo, **GSM Digital Down Converter**, has been enhanced to use the Fixed-Point Toolbox™ `cordicrotate` function. The demo now allows you to compare an NCO-based mixer to a CORDIC-based mixer.

Documentation Examples Renamed

In previous releases, the example models used throughout the Signal Processing Blockset™ documentation were named with a prefix of `doc_`. In R2011a, this prefix has changed to `ex_`. For example, in R2010b, you could launch an example model using the Time Scope block by typing `doc_timescope_tut` at the MATLAB command line. To launch the same model in R2011a, you must type `ex_timescope_tut` at the command line.

Version History

You can no longer launch DSP System Toolbox documentation example models using the `doc_` name. To open these models in R2011a, you must replace the `doc_` prefix in the model name with `ex_`.

Downsample Block No Longer Has Frame-Based Processing Latency for a Frame Size of One

As of R2011a, the Downsample block no longer exhibits frame-based processing latency when the input frame size is one.

Version History

Existing models that use the Downsample block in frame-based processing mode may produce different results in R2011a. Specifically, the Downsample block no longer has one-frame of latency when the input frame size is one. If your model uses a Downsample block in frame-based processing mode and the input frame size is one, you will see different results when you run your model in R2011a. If you need to restore the one-frame latency, you can use a Delay block to delay the output of the Downsample block by one frame.

SignalReader System Object Accepts Column Input Only

The `SignalReader` System object now accepts column inputs only.

Version History

Update any code with row input to the `SignalReader` object to convert the input to column form before passing it to the object. (Note that this change occurred in R2010b.)

FrameBasedProcessing Property Removed from the dsp.DelayLine and dsp.Normalizer System Objects

In R2010b, the `FrameBasedProcessing` property was removed from the `dsp.DelayLine` and `dsp.Normalizer` System objects. Both objects now treat each column of the input as a separate channel (frame-based processing).

Version History

As of R2010b, MATLAB issues a warning when you set the `FrameBasedProcessing` property of the `dsp.DelayLine` or `dsp.Normalizer` System objects.

R2010a MAT Files with System Objects Load Incorrectly

If you saved a System object to a MAT file in R2010a and load that file in R2011a, MATLAB may display a warning that the constructor must preserve the class of the returned object. This occurs because an aspect of the class definition changed for that object in R2011a. The object's saved property settings may not restore correctly.

Version History

MAT files containing a System object saved in R2010a may not load correctly in R2011a. You should recreate the object with the desired property values and save the MAT file.