

Communications 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

Communications Toolbox™ Release Notes

© COPYRIGHT 2011–2022 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.

OFDM functionality updates	1-2
Sample rate match (SRM) block and SRM delay function	1-2
Sample rate offset block	1-2
Train DQN agent for beam selection example	1-2
CSI feedback with autoencoders example	1-2
5G LDPC block error rate simulation using the cloud or a cluster example	1-2
UWB signal recovery and channel modeling examples	1-2
Support for ZigBee and UWB waveforms in Wireless Waveform Generator	1-3
Support for radar waveforms in Wireless Waveform Generator	1-3
Waveform transmission at full radio device rates in Wireless Waveform Generator	1-3
Specify load resistance when using the Amplifier block	1-3
Ray tracing with SBR method finds paths with exact geometric accuracy	1-3
Ray tracing functions consider multipath interference	1-5
Customize spacing of launched rays for ray tracing with SBR method ..	1-5
MATLAB authored comm.ErrorRate System object	1-6
Updates to channel visualization display	1-6
Write Data to TDMS-Files	1-6
Functionality being removed or changed	1-6
Viterbi decoders clip unquantized input values to +/- 1012	1-6
comm.LTEMIMOChannel System object has been removed	1-6
hilbir function has been removed	1-7

Waveform transmission using SDR hardware and Wireless Waveform Generator app	2-2
Neural network for digital predistortion design deep learning example	2-2
Neural network for beam selection deep learning example	2-2
Frequency offset function	2-2
SNR converter function	2-2
Read all baseband file samples to one output frame when using the baseband file reader	2-2
Enhancements to PSK modulation and demodulation functions	2-2
CCDF measurement mode incorporated in powermeter System object ..	2-3
Updated display for constellation diagram block	2-3
Output noise variance when using awgn function	2-3
Specify load resistance when using receiver thermal noise System object and block	2-3
Read Data from TDMS-Files	2-3
Improved performance for SBR ray tracing point-to-area coverage and point-to-point analysis cases	2-4
Improved speed performance for certain Communications Toolbox features	2-5
Support for C code generation using biterr and symerr functions	2-5
Functionality being removed or changed	2-5
Communications Toolbox Library for the Bluetooth Protocol add-on has been removed for R2022a	2-5
Default symbol order for pskmod and pskdemod functions is now Gray-coded ordering	2-6
Bit to Integer Converter and Integer to Bit Converter blocks added to Simulink Logic and Bit Operations library	2-6
propagationModel('raytracing-image-method') syntax will be removed in a future release	2-6
NumReflections name-value argument of the raytrace function will be removed	2-6
comm.EyeDiagram System object has been removed	2-6
comm.PSKCoarseFrequencyEstimator System object has been removed	2-6

comm.QAMCoarseFrequencyEstimator System object has been removed	2-7
commtest.ErrorRate and testconsole.Results objects have been removed	2-7
equalizer and adaptalg packages and functions have been removed	2-7
Certain equalization blocks have been removed	2-8

R2021b

Multiband signal combiner System object and block	3-2
Sample rate offset System object	3-2
Low-density parity check (LDPC) encoding and decoding functions	3-2
Spectrum sensing with deep learning example	3-2
Ultra wideband (UWB) IEEE 802.15.4z wireless communications examples	3-2
Wireless Waveform Generator app export-to-Simulink capability	3-3
Ray tracing analysis with SBR method	3-3
Ray tracing channel System object support for control of start time and frame continuity	3-3
Computation of channel impulse response without applying channel filtering	3-3
Support for importing and viewing RF propagation paths within indoor scenes	3-3
Model and visualize RF propagation using MATLAB Online	3-4
Updated display for constellation diagram System object	3-4
Bit Error Rate Analysis app acceleration enhancements	3-4
Phase noise System object and block support for multichannel input ...	3-4
Functions for converting between integer and binary data include column-wise and 3D input-output support	3-4
Improved speed performance for certain Communications Toolbox features	3-5
Communications Toolbox Support Package for USRP Radio: Improved performance for SDRu System objects	3-5
Enhanced support for valid data in SDRu receiver System object	3-5

Improved performance for SDRu receiver System object with data reception and number of overruns	3-5
Improved performance for SDRu transmitter System object with number of underruns	3-6
Improved performance for SDRu System objects with burst mode	3-7
Functionality being removed or changed	3-8
Communications Toolbox Library for the ZigBee Protocol has moved to Communications Toolbox Library for ZigBee and UWB	3-8
Default value of Method name-value argument for propagationModel function is now shooting and bouncing rays method	3-8
raytrace, coverage, link, sigstrength, and sinr use shooting and bouncing rays method	3-8
ReflectionLocations and NumReflections properties of the comm.Ray object have been removed	3-9
NumReflections property of the raytrace function will be removed	3-9
bi2de function is not recommended	3-9
de2bi function is not recommended	3-9
comm.LDPCDecoder and comm.LDPCDecoder System objects will be removed	3-10
bin2gray and gray2bin functions will be removed	3-10
comm.src.pn object will be removed	3-11
hank2sys function will be removed	3-13
hilbiir function will be removed	3-13
rcosfir, rcosiir, rcosflt, and rcosine functions will be removed	3-13
comm.BitToInteger System object has been removed	3-15
comm.IntegerToBit System object has been removed	3-15
Align Signals block has been removed	3-16
Certain interleaver and deinterleaver System objects have been removed	3-16

R2021a

Ray tracing analysis using SBR method	4-2
RF propagation object and functions deployment using MATLAB Compiler	4-2
Autoencoders for wireless communications example	4-2
Memoryless nonlinearity System object mode and visualization updates	4-2
Turbo encoding and decoding System objects and blocks enable custom bit ordering and puncturing	4-3
Bit Error Rate Analysis app updates	4-3
Wireless Waveform Generator app support for 5G NR uplink and downlink carrier waveforms	4-3

Power measurement of voltage signal using MATLAB and Simulink	4-3
Bluetooth LE and BR/EDR features	4-3
Data over cable service interface specification (DOCSIS) example	4-3
Updated HDL-optimized QPSK example	4-4
Visual appearance updates to plots generated with eyediagram and scatterplot functions	4-4
Support for C code generation using certain RF propagation functions	4-4
SIMD code generation using Intel AVX2 for certain Communications Toolbox features	4-4
Functionality being removed or changed	4-5
Semianalytic tab in Bit Error Rate Analysis app has been removed	4-5
Default settings change for phase noise block and System object	4-5
Synchronization component blocks will be removed	4-5
Insert Zero block will be removed	4-6
comm.EyeDiagram System object will be removed	4-6
randseed function has been removed	4-6
lteZadoffChuSeq has been renamed to zadoffChuSeq	4-6
Noise generator blocks have been removed	4-7
propagationModel('raytracing-image-method') returns a RayTracing model	4-7

R2020b

RF Propagation and Ray-Tracing Enhancements	5-2
Ray-Tracing Channel Modeling and Visualization	5-2
Apply Multipath Gains with Channel Filter	5-2
Bluetooth LE and BR/EDR Features and Examples	5-2
Training and Testing a Neural Network for LLR Estimation in MATLAB	5-3
RF Component Modeling	5-4
MATLAB Coder support added to comm.Ray	5-4
Functionality being removed or changed	5-4
comm.EyeDiagram System object will be removed	5-4
hank2sys function will be removed	5-4
hilbir function will be removed	5-4

crc.generator object will be removed	5-5
crc.detector object will be removed	5-6
comm.BitToInteger System object will be removed	5-7
comm.IntegerToBit System object will be removed	5-8
commsrc.pn object will be removed	5-11
comm.RectangularQAMModulator and comm.RectangularQAMDemodulator System objects will be removed	5-9
stdchan(ts,fd,chanType) syntax has been removed	5-9
comm.BinarySymmetricChannel System object has been removed	5-10
rayleighchan function has been removed	5-11
ricianchan function has been removed	5-13
legacychannelsim function has been removed	5-14
doppler.jakes object has been removed	5-15
doppler.flat object has been removed	5-15
doppler.bell object has been removed	5-15
doppler.rounded object has been removed	5-15
doppler.rjakes object has been removed	5-15
doppler.ajakes object has been removed	5-15
doppler.gaussian object has been removed	5-16
doppler.biggaussian object has been removed	5-16

R2020a

Wireless Waveform Generator app adds 5G Fixed Reference Channel waveform generation support	6-2
RF fingerprinting with deep learning examples	6-2
Bluetooth LE examples and BR/EDR PHY features and examples	6-2
Site Viewer, RF propagation and ray tracing enhancements	6-3
Amplifier block to model a memoryless nonlinear amplifier using one of four methods	6-3
Interference Modeling in Simulink Example	6-3
Multi-Band Signal Generation Example	6-3
Top Down Design of RF Receiver Example	6-3
Multuser Block Diagonalization Beamforming	6-4
Channel modeling coordinate system transformations	6-4
Channel delay computation	6-4
MATLAB Compiler support added to eye diagram	6-4
Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Detect NR cell signals off the air	6-4

Functionality being removed or changed	6-4
bin2gray and gray2bin functions will be removed	6-4
vec2mat function is not recommended	6-5
comm.PSKCoarseFrequencyEstimator System object will be removed	6-6
comm.QAMCoarseFrequencyEstimator System object will be removed	6-6
commscope.eyediagram object and EyeScope function have been removed	6-6
comm.CPMCarrierPhaseSynchronizer System object has been removed	6-6
CPM Phase Recovery block has been removed	6-7
M-PSK Phase Recovery block has been removed	6-7
Insert Zero block will be removed	6-7
Gaussian Filter block has been removed	6-7
Bernoulli Binary Generator and Random Integer Generator block updates supported in Upgrade Advisor	6-8
Certain random source generator blocks automatically update	6-8
Equalizer and adaptalg packages and functions will be removed	6-8
Certain equalization blocks will be removed	6-9

R2019b

Wireless waveform generator app updates	7-2
RF propagation tools for ray tracing and visualization	7-2
GSM Waveform generation functions	7-4
Bluetooth protocol updates and examples	7-4
LDPC decoder uses multicore processing	7-4
Class definition now available for CPM demodulator System object	7-4
Playback control behavior changed for scopes in referenced models	7-4
Functionality being removed or changed	7-5
Code generation for models using EVM Measurement or MER Measurement block has changed	7-5
Certain interleaver and deinterleaver System objects will be removed	7-16
randseed function will be removed	7-6
Align Signals block will be removed	7-6
commtest.ErrorRate and testconsole.Results objects will be removed	7-6
comm.BinarySymmetricChannel System object will be removed	7-6
comm.BitToInteger and comm.IntegerToBit System objects will be removed	7-7
oqpskmod and oqpskdemod functions have been removed	7-7

Introducing Communications Toolbox Library for the Bluetooth Protocol	8-2
Wireless Waveform Generator app updates	8-2
Digital predistortion estimator and compensator System objects and blocks	8-2
Modulation classification with deep learning example	8-3
Decision feedback and linear equalizer System objects and blocks	8-3
Link budget analysis example	8-3
Turbo product code (TPC) decoder function and block support for early termination	8-3
Memoryless nonlinearity System object support for custom AM/AM and AM/PM data	8-3
Constellation diagram System object and block support for multiple input	8-3
Support for C code generation using certain interleaver and deinterleaver functions	8-3
Support for C code generation using certain modulation functions	8-4
Class definition now available for CPM modulator System object	8-4
Phase Noise block reimplemented as a System block	8-4
Track ships using automatic identification system example	8-4
Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Support for Xilinx ZCU102 and Analog Devices FMCOMMS2/3/4	8-4
Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Support for Xilinx ZC706 and Analog Devices FMCOMMS5	8-5
Functionality being removed or changed	8-5
lteZadoffChuSeq is being renamed to zadoffChuSeq	8-6
randseed function will be removed	8-5
Align Signals block will be removed	8-5
Equalizer and adaptalg packages and functions will be removed	8-5
Certain equalization blocks will be removed	8-7
comm.PSKCoarseFrequencyEstimator System object will be removed	8-7
comm.QAMCoarseFrequencyEstimator System object will be removed	8-7
CRC-N Generator block has been removed	8-7
CRC-N Syndrome Detector block has been removed	8-8
commmeasure package has been removed	8-8

modem package has been removed	8-8
gfhelp has been removed	8-9

R2018b

Wireless Waveform Generator App: Create, impair, visualize, and export modulated waveforms	9-2
APSK Modulator and Demodulator Blocks: Simulate amplitude phase shift keying (APSK) modulation and demodulation for satellite applications	9-2
Turbo Product Code Encoder and Decoder Blocks: Simulate block product codes for defense and satellite communications	9-2
Multichannel Constellation Diagram Plot Support: Plot multichannel signals in constellation diagram displays	9-2
ADS-B example for HDL code generation	9-2
Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio supports burst mode reception	9-3
Communications Toolbox Support Package for Xilinx Zynq-Based Radio transitions to libiio-based infrastructure	9-3
Updated WINNER2 Channel Filtering: Filter provides flatter bandedge response in fading channel System object	9-3
Functionality being removed or changed	9-3

R2018a

APSK Modulator and Demodulator: Simulate amplitude phase shift keying modulation for satellite applications	10-2
Turbo Product Code Encoder and Decoder: Simulate block product codes for defense and satellite communications	10-2
Massive MIMO Example: Simulate an end-to-end MIMO link using hybrid beamforming	10-2
OFDM Modulator and Demodulator: Simulate orthogonal frequency division modulation	10-2

Enhanced Fidelity of Phase Noise: Model close-in phase noise more precisely	10-2
Frequency Correction Support: Adjust ADALM-Pluto Radio frequency to correct frequency offset (Introduced November 2017)	10-2
Custom Filter Wizard: Design custom filtering for ADALM-Pluto Radio (Introduced November 2017)	10-3
Low IF Receiver Architecture Example: Simulate a low IF receiver architecture RF model	10-3
ALOHA and CSMA/CA Packetized Wireless Network Example: Simulate an ALOHA or CSMA/CA medium access controller	10-3
Examples added to library for ZigBee Protocol	10-3
Examples added to library for NFC Protocol	10-3
Updated Fading Channel Filtering: Filter provides flatter bandedge response in fading channel System objects and blocks	10-3
Functionality being removed or changed	10-4

R2017b

Library for ZigBee Protocol: Simulate ZigBee low-rate wireless personal area network (LRWPAN) technologies	11-2
Library for NFC Protocol: Simulate Near Field Communication (NFC) wireless technologies	11-2
OQPSK Simulation: Simulate practical OQPSK links with modulation, frequency and timing synchronization, and demodulation functions	11-2
MIMO Channel Enhancements: Specify an arbitrary number of antennas and include antenna polarization when you simulate MIMO fading channels	11-2
Path Loss Functions: Account for path loss due to free space, fog, gas, and rain	11-2
Communications System Toolbox Support Package for Analog Devices ADALM-Pluto Radio: Lower minimum baseband sample rate (Introduced in May 2017)	11-3
Communications System Toolbox Support Package for Xilinx Zynq-Based Radio: PicoZed SDR renamed to ADI RF SOM	11-3

USRP E312 Support: Prototype and test USRP E312 software-defined radio (SDR) systems	11-4
Key Features	11-4
Communications System Toolbox Support Package for Xilinx FPGA-Based Radio: Support removed	11-5
Functionality being removed or changed	11-5

R2017a

WINNER II Channel Model: Model and simulate spatially defined MIMO channels	12-2
USRP E310 Support: Prototype and test USRP E310 software-defined radio (SDR) systems	12-2
Key Features	12-2
Required MathWorks Products	12-3
Required Third-Party Tools	12-3
Hardware Support	12-3
Bundled USRP Radios: Synchronize multiple USRP radios in frequency and time for transmission and reception	12-3
Example of Proposed 5G Modulations: Simulate end-to-end links by using universal filtered multicarrier (UFMC) and filtered OFDM (F-OFDM)	12-4
Packetized Modem Example: Model PHY and MAC layer operations in transmission of packetized waveforms	12-4
Communications System Toolbox Support Package for Analog Devices ADALM-Pluto Radio: Prototype and test ADALM-Pluto radio systems	12-4
Radio Broadcast Data System Transmitter: Generate RBDS-compliant waveforms	12-4
RBDS Modulation and Demodulation: Modulate and demodulate RBDS waveforms using the FM broadcast modulator and demodulator	12-4
MIMO-OFDM Precoding with Phased Arrays Example	12-5
Tab Completion: Complete parameter names and options in select MATLAB function calls	12-5
Remove antenna limitations on comm.OFDMModulator and comm.OFDMDemodulator	12-5
Parameter names changed for comm.FMBroadcastModulator and comm.FMBroadcastDemodulator	12-5

Functionality being removed	12-6
Communications System Toolbox Support Package for Xilinx FPGA-Based Radio: Support package being removed	12-7
Communications System Toolbox Support Package for Xilinx Zynq-Based Radio: FMCOMMS1 support removed	12-7

R2016b

Eye Diagram Object and Block: Measure signal quality, and visualize histograms and bathtub curves	13-2
Baseband File Reader and Writer: Record captured baseband signals as files and use the files to test wireless designs	13-2
UFMC and FBMC Examples: Characterize the performance of 5G modulation techniques	13-2
Preamble Detector: Determine the location of the preamble in a packet	13-2
I/Q Imbalance: Apply amplitude and phase imbalance to baseband signal	13-2
MATLAB Compiler support added to constellation diagram	13-2
Printer support added to constellation and eye diagrams	13-2
Simpler way to call System objects	13-2
Functionality being removed	13-3

R2016a

EVM Measurements: Measure error vector magnitude with added flexibility	14-2
Eye Diagram Block: Plot histograms of time-varying signals	14-2
Quadrature Amplitude Modulation: Modulate binary data and demodulate binary and soft-decision outputs	14-2
Communications System Toolbox Support Package for USRP Radio: Transmit and receive RF signals using B200mini	14-2

Class definition now available for comm.LDPCDecoder System object	14-2
Function definitions now available for qammod, qamdemod, and vitdec	14-3
Functions now supporting code generation	14-3
Inherited option removed from filters	14-3
Functionality being removed	14-3
Communications System Toolbox Support Package for Xilinx Zynq-Based Radio updates (Introduced in April 2016)	14-4
Hardware/Software Co-Design	14-4

R2015b

cdma2000 and 1xEV-DO Waveform Generators: Model the physical layer of North American CDMA standards and provide reference channels and waveforms	15-2
Coarse Frequency Compensator: Correct carrier frequency offsets in PAM, PSK, and QAM signals	15-2
Polynomial Strings: Specify Galois field polynomials as strings for error correction coding and sequence generation	15-2
AGC object and block have simplified interfaces, better dynamic range, and faster convergence times	15-2
Improved interface on the constellation diagram object and block facilitates easier application of reference constellations and EVM/MER measurements	15-3
Source blocks output frames of contiguous time samples but do not use the frame attribute	15-3
Functionality being changed or removed	15-4
mimoChan will be removed	15-4
Noise Generator blocks will be removed	15-4
Communications System Toolbox Support Package for RTL-SDR Radio Updates (v 15.2.0)	15-4
Deploy SDR models for standalone applications	15-4
Code deployment to external host hardware	15-5
Communications System Toolbox Support Package for USRP Radio Updates (v 15.2.0)	15-5
2x2 MIMO Support for USRP Radio B210 and X Series Boards (Introduced April 2015)	15-5

Communications System Toolbox Support Package for Xilinx Zynq-Based Radio Updates (v 15.2.0)	15-5
PicoZed SDR Support	15-5
Simultaneous transmission and reception on a single board	15-6
Third-Party Software Support Update	15-6

R2015a

Symbol Timing Synchronizer: Correct for symbol timing clock skew between a transmitter and receiver	16-2
Carrier Synchronizer: Synchronize phase and frequency on a received waveform	16-2
Baseband and Broadcast FM: Modulate and demodulate baseband and broadcast FM signals	16-2
Interactive QAM Example: Simulate an end-to-end QAM link with RF impairments and corrections	16-2
Communications System Toolbox Simulink Model Template: Automatically configure the Simulink environment for communications modeling	16-2
Library for HDL-supported Communications System Toolbox blocks ...	16-3
Frame-based processing	16-3
Input processing parameter set to Inherited	16-3
Rate options parameter set to Inherit from input	16-4
Class definitions now available for MATLAB-authored System objects	16-5
Functionality being changed or removed	16-6
Support Package for USRP Radio B Series and X Series Board Support (Introduced November 2014)	16-7
Support Package for Xilinx Zynq-Based Radio (Introduced November 2014)	16-8
Supported Hardware and Software	16-8

R2014b

I/Q Imbalance Compensator System object and block that remove I/Q amplitude and phase imbalance	17-2
--------------------------------------------------------------------------------------------------------------	-------------

Eye Diagram block that plots eye diagrams faster than its predecessor	17-2
Channel visualization for plotting impulse response, frequency response, and Doppler spectrum added to the Rayleigh, Rician, and MIMO Channel System objects	17-2
Sum-of-sinusoids modeling technique added to the Rayleigh, Rician, and MIMO Channel System objects	17-2
Trajectory diagram visualization added to the Constellation Diagram block and System object	17-2
Support Package for RTL-SDR Radio Update	17-3
Linux Support	17-3
Mac Support	17-3

R2014a

OFDM modulator and demodulator System objects and blocks	18-2
DC blocker System object and block	18-2
Direct and nondirect modes for HDL-optimized CRC generator and detector	18-2
Additional featured examples such as 802.11 OFDM synchronization and HDL Optimized QAM Transmitter and Receiver	18-2
APP Decoder System object parameter change	18-2
GPU System Object Support in System Block	18-3
System object templates	18-3
System objects infer number of inputs and outputs from stepImpl method	18-3
System objects setupImpl method enhancement	18-3
System objects base class renamed to matlab.System	18-3
System objects Propagates mixin methods	18-4
System objects infoImpl method allows variable inputs	18-4
Support Package for RTL-SDR Radio (v 14.1.0)	18-4
Key Features	18-4
Blocks and System Objects	18-4
RTL-SDR Examples	18-5

Hardware and Software Requirements	18-5
Support Package for Xilinx FPGA-Based Radio updates (v 14.1.0)	18-5
Intermediate frequency tuning	18-5
DC blocking filter	18-6
QPSK targeting examples	18-6

R2013b

Simulink blocks for MIMO channel, sphere decoder, and constellation diagram	19-2
Code generation for all MIMO channel Doppler spectra	19-2
Open-loop PSK and QAM carrier synchronizers in MATLAB	19-2
HDL-Optimized QPSK Receiver with Captured Data	19-2
Raised cosine transmit and receive filter System objects	19-3
Rayleigh and Rician fading channel System objects	19-3
System objects matlab.system.System warnings	19-3
Restrictions on modifying properties in System object Impl methods	19-3
Block parameter prompt changes for raised cosine filter blocks	19-4
NumTransmitAntennas and NumReceiveAntennas properties added back to MIMOChannel System object.	19-5
Functionality Being Changed or Removed	19-5
Migrate Code from firrcos and rcosfir to rcosdesign	19-6
Support Package for Xilinx FPGA-Based Radio	19-8

R2013a

Sphere Decoder System object for MIMO receiver processing	20-2
Constellation Diagram System object with measurements	20-2
LTE space-frequency block coding and LTE GPU-accelerated turbo coding examples	20-2

HDL code generation for CRC Generator, CRC Detector, RS Encoder, and RS Decoder System objects	20-2
Variable-size support for AWGN, MIMO, and LTE MIMO Channel System objects	20-2
IEEE 802.11 WLAN - HDL optimized beacon frame receiver example with captured data	20-3
Automatic gain controller block and System object	20-3
Additional CRC algorithm implementation	20-3
ATSC digital television example	20-3
Disable second output port on APP Decoder	20-3
Behavior change of locked System objects for loading, saving, and cloning	20-4
Dynamic memory allocation based on size	20-4
Naming convention change for LTE examples	20-5
APP Decoder System Object parameter change	20-5
Functions to remain in the product	20-5
Communications System Toolbox Functionality Being Changed or Removed	20-6
Update Legacy Code to use System object	20-6

R2012b

Support for C code generation for all System objects in Communications System Toolbox	21-2
Support for HDL code generation for Reed-Solomon encoder, decoder, and CRC detector blocks	21-2
Support for HDL code generation for Rectangular QAM and PSK Demodulator System objects	21-2
LTE Zadoff-Chu sequence generator function	21-3
LTE downlink shared channel example	21-3
Phase Noise block and System object, specifying phase noise spectrum with a vector of frequencies	21-3

IEEE 802.11 beacon with captured data example	21-3
P25 spectrum sensing example	21-3
MATLAB-based QPSK transceiver example	21-3
Design Iteration Workflow	21-3
Constellation method for modulator and demodulator System objects	21-4
Specify initial states of Gold Sequence Generator and PN Sequence Generator System objects	21-4
System object tunable parameter support in code generation	21-4
save and load for System objects	21-5
Save and restore SimState not supported for System objects	21-5
Functionality Being Changed or Removed	21-5
Update Legacy Code to use System objects	21-6
Frame-Based Processing	21-15

R2012a

MIMO Multipath Fading Channel System Objects	22-2
Multi-H Support for CPM Modulator and Demodulator Simulink Blocks and MATLAB System Objects	22-2
GPU System Objects	22-2
MATLAB Compiler Support for GPU System Objects	22-2
Code Generation Support	22-2
HDL Code Generation from MATLAB code	22-3
HDL Support For HDL CRC Generator Block	22-3
Enhancements for System Objects Defined by Users	22-3
Code Generation for System Objects	22-3
New System Object Option on File Menu	22-3
Variable-Size Input Support for System Objects	22-3
Data Type Support for System Objects	22-3
New Property Attribute to Define States	22-3
New Methods to Validate Properties and Get States from System Objects	22-4

matlab.system.System changed to matlab.System	22-4
New and Enhanced Demos	22-4
Functionality Being Changed or Removed	22-4
Frame-Based Processing	22-7
Inherited Option of the Input Processing Parameter Now Warns	22-7
Inherited Option of the Rate Options Parameter Now Warns	22-8

R2011b

New Demos	23-2
Turbo Codes	23-2
USRP2 Migration	23-2
GPU System Objects	23-2
Custom System Objects	23-2
Variable-Size Support	23-3
System Object Code Generation Support	23-3
Delayed Reset for Viterbi Decoder	23-4
System Objects FullPrecisionOverride Property Added	23-4
APP Decoder System Object Parameter Change	23-5
System Object DataType and CustomDataType Properties Changes	23-5
Conversion of System Object Error and Warning Message Identifiers	23-5
Frame-Based Processing	23-6

R2011a

Product Restructuring	24-2
LDPC Encoder and Decoder System Objects	24-2
LDPC GPU Decoder System Object	24-2

Variable-Size Support	24-2
Algorithm Improvements for CRC Blocks	24-3
MATLAB Compiler Support for System Objects	24-3
'Internal rule' System Object Property Values Changed to 'Full precision'	24-3
System Object Code Generation Support	24-4
LDPC Decoder Block Warnings	24-4
Phase/Frequency Offset Block and System Object Change	24-4
Derepeat Block Changes	24-4
Version 2, 2.5, and 3.0 Obsolete Blocks Removed	24-4
System Objects Input and Property Warnings Changed to Errors	24-5
Frame-Based Processing	24-5
General Product-Wide Changes	24-6
Blocks with a New Input Processing Parameter	24-7
AWGN Channel Block Changes	24-7
Multirate Processing Parameter Changes	24-8
Sample-Based Row Vector Processing Changes	24-9
CMA Equalizer Changes	24-10
Differential Encoder Changes	24-10
Find Delay and Align Signal Block Changes	24-10
New Demos	24-10

R2022b

Version: 7.8

New Features

Bug Fixes

Compatibility Considerations

OFDM functionality updates

- Use the `ofdmEqualize` function in MATLAB® or the OFDM Equalizer block in Simulink® to perform minimum mean square error or zero forcing equalization of a received OFDM signal. The OFDM equalization requires a frequency domain estimate of the channel.
- The `ofdmmod` and `ofdmmodem` functions add an optional oversampling factor name-value pair to support applications that require oversampling of the OFDM signal. Addition of oversampling to the FFT processing in the OFDM modulation workflow avoids an extra upsample operation that may add signal reflections, amplitude distortion, and phase distortion to the signal output.
- The “OFDM Transmitter and Receiver” example runs a complete end-to-end OFDM transmission system for a single-input, single-output (SISO) system.

Sample rate match (SRM) block and SRM delay function

The Sample-Rate Match block enables you to upsample the input signals with different sample rates to a common output sample rate.

The `srmdelay` function computes the delay that the sample-rate match operation introduces.

Sample rate offset block

The Sample Rate Offset block applies the sample rate offset to an input signal.

Train DQN agent for beam selection example

The “Train DQN Agent for Beam Selection” example trains a deep Q-network (DQN) reinforcement learning agent to accomplish the beam selection task in a 5G New Radio (NR) communications system. Instead of an exhaustive beam search over all the beam pairs, the trained agent increases beam selection accuracy by selecting the beam with highest signal strength while reducing beam transition cost.

CSI feedback with autoencoders example

The “CSI Feedback with Autoencoders” example compresses downlink channel state information (CSI) for 5G systems by using an autoencoder neural network.

5G LDPC block error rate simulation using the cloud or a cluster example

The “5G LDPC Block Error Rate Simulation Using the Cloud or a Cluster” example highlights the use of the cloud or a cluster for block error rate (BLER) simulation of low-density parity-check (LDPC) coding for the 5G NR downlink shared transport channel (DL-SCH).

UWB signal recovery and channel modeling examples

Use features available in the Communications Toolbox Library for ZigBee® and UWB add-on to run these examples.

- The “Recovery of IEEE 802.15.4z UWB Signals” example implements a practical IEEE® 802.15.4z™ PHY receiver that recovers and decodes UWB waveforms captured over the air.

-
- The “UWB Channel Models” example implements UWB channel models for a variety of propagation environments. The implementations are based on recommendations from the channel modeling subgroup of IEEE 802.15.4a.

Support for ZigBee and UWB waveforms in Wireless Waveform Generator

You can now use the **Wireless Waveform Generator** app to parameterize and generate ZigBee and UWB (IEEE 802.15.4a/z) waveforms. To use this feature, download the Communications Toolbox Library for ZigBee and UWB add-on.

Support for radar waveforms in Wireless Waveform Generator

You can now use the **Wireless Waveform Generator** app to parameterize and generate radar waveforms. This feature requires “Phased Array System Toolbox”.

Waveform transmission at full radio device rates in Wireless Waveform Generator

You can now use the **Wireless Waveform Generator** app to transmit your waveform over the air at full radio device rate using your radio and Wireless Testbench™ software. For a list of radios that support full device rates, see “Supported Radio Devices” (Wireless Testbench). This feature requires “Wireless Testbench”. For an example, see “Transmit App-Generated Wireless Waveform Using Radio Transmitters”.

Specify load resistance when using the Amplifier block

Starting in R2022b, when using the Amplifier block, you can model actual load resistance in ohms by setting the **Reference load** parameter value. The block uses this value to convert the signal and noise levels between a voltage level and a power level based on actual load resistance. Previously, the load resistance was 1 ohm.

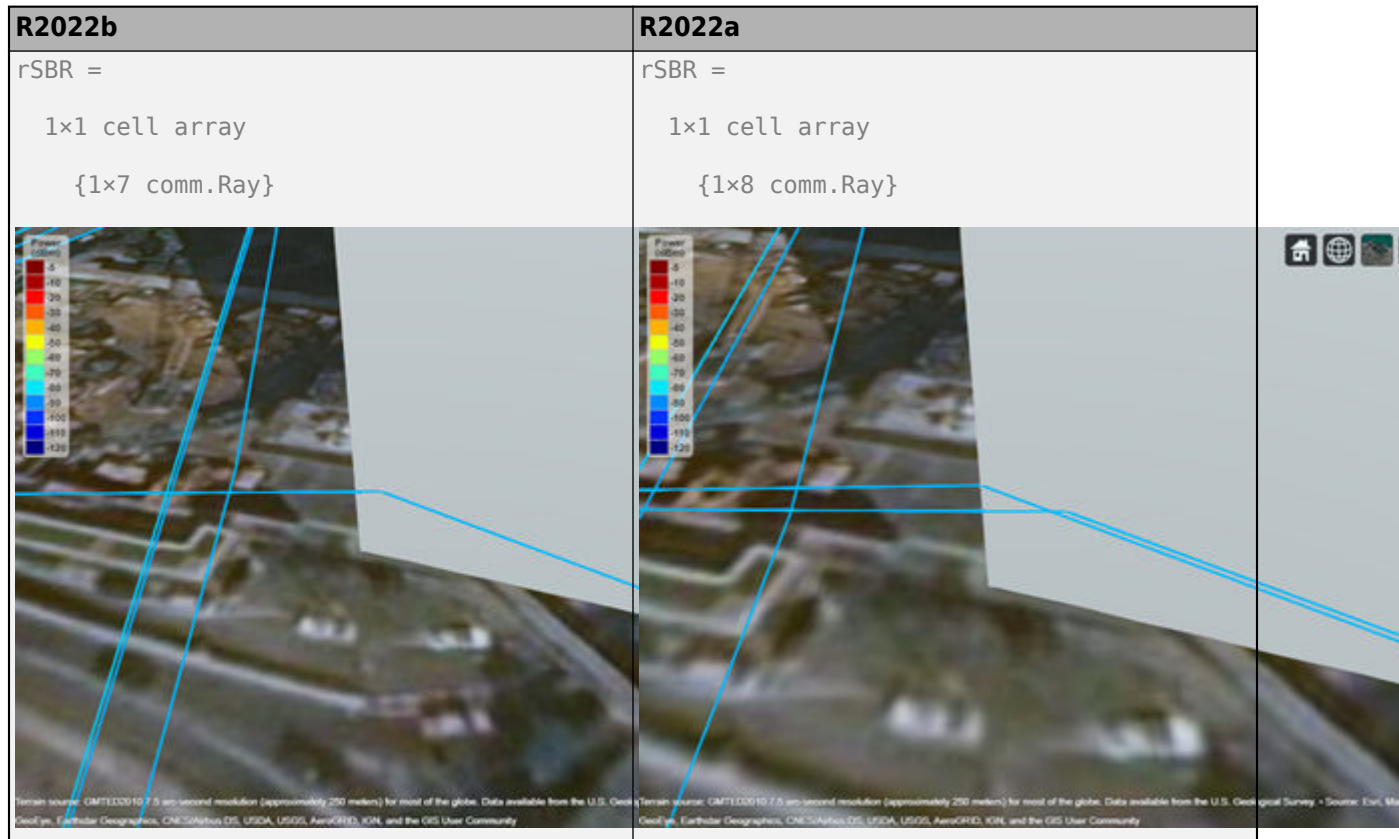
Ray tracing with SBR method finds paths with exact geometric accuracy

When you find propagation paths by using the `raytrace` function and the shooting and bouncing rays (SBR) method, MATLAB corrects the results so that the geometric accuracy of each path is exact, using single-precision floating-point computations. In previous releases, the paths have approximate geometric accuracy.

For example, this code finds propagation paths between a transmitter and receiver by using the default SBR method and returns the paths as `comm.Ray` objects. In R2022b, the `raytrace` function finds seven propagation paths. In earlier releases, the function approximates eight propagation paths, one of which is a duplicate path.

```
viewer = siteviewer(Buildings="hongkong.osm");  
  
tx = txsite(Latitude=22.2789,Longitude=114.1625,AntennaHeight=10, ...  
    TransmitterPower=5,TransmitterFrequency=28e9);  
rx = rxsite(Latitude=22.2799,Longitude=114.1617,AntennaHeight=1);
```

```
rSBR = raytrace(tx,rx)
raytrace(tx,rx)
```



Paths calculated using the SBR method in R2022b more closely align with paths calculated using the image method. The image method finds all possible paths with exact geometric accuracy. For example, this code uses the image method to find propagation paths between the same transmitter and receiver.

```
viewer = siteviewer(Buildings="hongkong.osm");

tx = txsite(Latitude=22.2789,Longitude=114.1625, ...
    AntennaHeight=10,TransmitterPower=5, ...
    TransmitterFrequency=28e9);
rx = rxsite(Latitude=22.2799,Longitude=114.1617, ...
    AntennaHeight=1);

pm = propagationModel("raytracing",Method="image",MaxNumReflections=2);

rImage = raytrace(tx,rx,pm)

rImage =
1x1 cell array
{1x7 comm.Ray}
```

In this case, the SBR method finds the same number of propagation paths as the image method. In general, the SBR method finds a subset of the paths found by the image method. When both the image and SBR methods find the same path, the points along the path are the same within a tolerance of machine precision for single-precision floating-point values.

This code compares the path losses, within a tolerance of 0.0001, calculated by the SBR and image methods.

```
abs([rSBR{1}.PathLoss]-[rImage{1}.PathLoss]) < 0.0001
```

```
ans =
```

```
1x7 logical array
```

```
1 1 1 1 1 1 1
```

The path losses are the same within the specified tolerance.

Compatibility Considerations

The `raytrace` function can return different results in R2022b compared to previous releases.

- The function can return a different number of `comm.Ray` objects because it discards invalid or duplicate paths.
- The function can return different `comm.Ray` objects because it calculates exact paths rather than approximate paths.

These differences can also affect the path losses calculated by the `raypl` and `pathloss` functions.

Ray tracing functions consider multipath interference

When calculating received power using ray tracing models, the `sigstrength`, `coverage`, `sinr`, and `link` functions now incorporate multipath interference by using a phasor sum. In previous releases, the functions used a power sum. As a result, the calculations in R2022b are more accurate than in previous releases.

Compatibility Considerations

The `sigstrength`, `coverage`, `sinr`, and `link` functions can return different results in R2022b compared to previous releases.

Customize spacing of launched rays for ray tracing with SBR method

When performing ray tracing using the shooting and bouncing rays (SBR) method, you can customize the spacing of launched rays by specifying the `AngularSeparation` property of the `RayTracing` object as a numeric value in degrees. In previous releases, the `AngularSeparation` property supported only the options "high", "medium", and "low".

To improve the accuracy of the number of paths found by the SBR method, decrease the value of `AngularSeparation`. Decreasing the value of `AngularSeparation` can increase the amount of time MATLAB requires to perform the analysis.

MATLAB authored comm.ErrorRate System object

The `comm.ErrorRate` System object™ is now a MATLAB authored System object that enables you to view the underlying source code.

Updates to channel visualization display

The channel visualization feature for the `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects, and the MIMO Fading Channel and SISO Fading Channel blocks now:

- Displays configuration settings in the bottom toolbar on the plot window.
- Plots side-by-side in one window when you select the Impulse and frequency responses channel visualization option.

In prior releases:

- Two separate plot windows opened when Impulse and frequency responses was chosen for the channel visualization option.
- Configuration settings were displayed in the plot area.
- The interpolated path gains were traced with a dotted line.

Write Data to TDMS-Files

Communications Toolbox now allows you to write data and property values to individual NI™ TDMS format files. The data can be in the form of tables or timetables. You can write multiple tables of data to separate channel groups in the file with a single function call.

To write to an individual TDMS-file from MATLAB, use the following functions:

- `tdmswrite` — Write data to TDMS-file
- `tdmswriteprop` — Write properties to TDMS-file

Functionality being removed or changed

Viterbi decoders clip unquantized input values to +/- 10¹²

Behavior change

Starting in R2022b, the `vitdec` function, `comm.ViterbiDecoder` System object, and Viterbi Decoder block clip unquantized input values outside of the range $[-10^{12}, 10^{12}]$ to -10^{12} and 10^{12} , respectively.

Compatibility Considerations

In prior releases, +/- Inf unquantized input values could result in inaccurate results.

comm.LTEMIMOChannel System object has been removed

Errors

The `comm.LTEMIMOChannel` System object has been removed. To filter input signal through a MIMO multipath fading channel, use the `comm.MIMOChannel` System object instead.

hilbiir function has been removed

Errors

The `hilbiir` function has been removed. To design a Hilbert transform filter, use the `fdesign.hilbert` object.

R2022a

Version: 7.7

New Features

Bug Fixes

Compatibility Considerations

Waveform transmission using SDR hardware and Wireless Waveform Generator app

Transmit waveforms over the air by using a software-defined radio (SDR) and the **Wireless Waveform Generator** app. To transmit a waveform, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer, have the associated add-on installed, and configure connected SDR in the **Transmit** tab of the **Wireless Waveform Generator** app. For more information, see Use Wireless Waveform Generator App and Supported Hardware - Software-Defined Radio.

Neural network for digital predistortion design deep learning example

The Neural Network for Digital Predistortion Design - Offline Training example shows how to use a neural network (NN) to apply digital predistortion (DPD) to offset the effects of nonlinearities in a power amplifier (PA). The example focuses on offline training of the NN-DPD by using PA measurements, and then uses the NN-DPD to apply the DPD.

Neural network for beam selection deep learning example

The Neural Network for Beam Selection example shows how to use a neural network to reduce the overhead in the beam selection task, using only the location of the receiver rather than the knowledge of the communication channels.

Frequency offset function

The `frequencyOffset` function applies a frequency offset to an input signal with the specified sample rate.

SNR converter function

The `convertSNR` function converts signal-to-noise ratio (SNR) values between E_s/N_0 , E_b/N_0 , and SNR.

Read all baseband file samples to one output frame when using the baseband file reader

Starting in R2022a, the `comm.BasebandFileReader` System object or Baseband File Reader block reads all of the baseband file samples to one output frame when you set the number of samples per frame to `Inf`.

Enhancements to PSK modulation and demodulation functions

Starting in R2022a, the `pskmod` and `pskdemod` functions support:

- Binary input/output
- Custom symbol mapping
- Soft-decision demodulation by using the bit-wise log-likelihood or approximate log-likelihood algorithm
- All built-in numeric datatypes

-
- Reference constellation visualization

CCDF measurement mode incorporated in powermeter System object

Starting in R2022a, the `powermeter` System object can be used to compute and output the CCDF measurement.

Updated display for constellation diagram block

The display for Constellation Diagram block has an updated interface with separate toolstrip tabs to configure constellation diagram plots and measurements.

Output noise variance when using `awgn` function

Starting in R2022a, you can return the total noise variance used to generate random noise samples as an output argument from the `awgn` function.

Specify load resistance when using receiver thermal noise System object and block

Starting in R2022a, you can specify the load resistance in ohms to convert the noise power to a voltage. Previously the load resistance was 1 ohm, so the voltage equalled the value of the noise power.

- When using the `comm.ThermalNoise` System object, specify the resistance with the `ReferenceLoad` property value.
- When using the Receiver Thermal Noise block, specify the resistance with the **Reference load** parameter value.

Read Data from TDMS-Files

Communications Toolbox now allows you to read data from NI TDMS format files.

To read from an individual TDMS-file into MATLAB, use the following functions:

- `tdmsread` — Read columnar data from TDMS-file
- `tdmsreadprop` — Read properties as single row table from TDMS-file
- `tdmsinfo` — Get information about TDMS-file

To read from a collection of TDMS-files, you can access them from a datastore object in MATLAB. Use the following functions to create a `TDMSDatastore` object, and read from its files:

- `tdmsDatastore` — Create a `TDMSDatastore` object based on a collection of TDMS-files
- `preview` — Read 8 rows from start of datastore
- `read` — Read subset of data from datastore
- `readall` — Read all data from datastore
- `hasdata` — Query if more data to be read in datastore
- `reset` — Reset datastore to start of data

Note TDMS functions are supported on Windows® platforms only.

Improved performance for SBR ray tracing point-to-area coverage and point-to-point analysis cases

The shooting and bouncing ray tracing method shows improved performance for point-to-area coverage and point-to-point analysis cases with low angular separation.

As the angular separation decreases (and number of launched rays increases) performance improvements are greater for the point-to-area coverage cases. For example, this code runs up to five times faster than the previous release for the point-to-area coverage case with low angular separation:

```
viewer = siteviewer("Buildings","chicago.osm");
tx = txsite("Latitude",41.8800,"Longitude",-87.6295);
pm = propagationModel("raytracing", ...
    "Method","sbr", ...
    "AngularSeparation","low", ...
    "MaxNumReflections",3);

timeit(@() coverage(tx,pm,"MaxRange",250))
```

The approximate execution times are:

- **R2021b**: 182 s
- **R2022a**: 29 s

On the same test platform, this code runs two times faster than previous release for the point-to-point analysis case with low angular separation:

```
sv = siteviewer("SceneModel","conferenceroom.stl");
tx = txsite("cartesian", ...
    "AntennaPosition":[-1.45; -1.4; 1.5]);
rx = rxsite("cartesian", ...
    "AntennaPosition",[.3; .2; .5]);
pm = propagationModel("raytracing", ...
    "CoordinateSystem","cartesian", ...
    "Method","sbr", ...
    "AngularSeparation","low", ...
    "MaxNumReflections",10);

timeit(@() raytrace(tx,rx,pm))
```

The approximate execution times are:

- **R2021b**: 6.3 s
- **R2022a**: 3.0 s

These code segments were timed on a Windows 10, Intel® Xeon® CPU E5-1650 v4 @ 3.60 GHz test system.

Improved speed performance for certain Communications Toolbox features

Starting in R2022a, these features have an improved speed performance under specified conditions.

Feature	Conditions	Speedup mode
comm.FMBroadcastModulator	Input signal has a data type of single or double	<p>You can notice a speedup when you:</p> <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
comm.RaisedCosineReceiveFilter		
Raised Cosine Receive Filter	<p>Input processing parameter is set to Columns as channels (frame based)</p> <p>Input signal has a data type of single or double</p>	<p>You can notice a speedup when the block:</p> <ul style="list-style-type: none"> • 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 <p>For more details on these modes, see Choosing a Simulation Mode (Simulink) and Choose Simulation Modes for Model Hierarchies (Simulink).</p>

Support for C code generation using biterr and symerr functions

The biterr and symerr functions now support C code generation.

Functionality being removed or changed

Communications Toolbox Library for the Bluetooth Protocol add-on has been removed for R2022a

Behavior change

The features and examples comprising Communications Toolbox Library for the Bluetooth® Protocol add-on are available in Bluetooth Toolbox software, introduced in R2022a.

Default symbol order for pskmod and pskdemod functions is now Gray-coded ordering*Behavior change*

Starting in R2022a, Gray-coded ordering is used when you modulate and demodulate using the default symbol order setting in the pskmod and pskdemod functions. To modulate and demodulate using natural binary-coded ordering, set the optional fourth input argument to 'bin'.

Compatibility Considerations

Since adjacent symbols always differ by only one bit in Gray-coded ordering, you can expect to see slightly improved bit error rate performance when using Gray-coded ordering instead of natural binary-coded ordering.

Bit to Integer Converter and Integer to Bit Converter blocks added to Simulink Logic and Bit Operations library*Behavior change*

The Bit to Integer Converter (Simulink) and Integer to Bit Converter (Simulink) blocks have been added to the **Simulink > Logic and Bit Operations** library and remain available in the **Communications Toolbox > Utility Blocks** library. All existing models continue to work.

propagationModel('raytracing-image-method') syntax will be removed in a future release*Warns*

The propagationModel('raytracing-image-method') syntax will be removed in a future release.

To use the image ray tracing method, use the propagationModel('raytracing',Method='image') syntax. To use the SBR ray tracing method, use the propagationModel('raytracing') syntax. For more information, see the propagationModel function reference page.

NumReflections name-value argument of the raytrace function will be removed*Warns*

The NumReflections name-value argument of the raytrace function will be removed in a future release. The NumReflections name-value argument now applies only for the image ray tracing method. Instead, create a propagation model by using the propagationModel function with its MaxNumReflections name-value argument. Then, use the raytrace function with the propagation model as an input. This example shows the recommended workflow.

```
pm = propagationModel('raytracing', ...  
    Method='image',MaxNumReflections=2);  
rays = raytrace(tx,rx,pm);
```

comm.EyeDiagram System object has been removed*Errors*

The comm.EyeDiagram System object has been removed. To display the eye diagram of a signal, use the eyediagram function instead.

comm.PSKCoarseFrequencyEstimator System object has been removed*Errors*

`comm.PSKCoarseFrequencyEstimator` has been removed. To estimate and compensate for frequency offsets in PSK signals, use the `comm.CoarseFrequencyCompensator System` object instead.

comm.QAMCoarseFrequencyEstimator System object has been removed

Errors

`comm.QAMCoarseFrequencyEstimator` has been removed. To estimate and compensate for frequency offsets in QAM signals, use the `comm.CoarseFrequencyCompensator System` object instead.

commtest.ErrorRate and testconsole.Results objects have been removed

Errors

The `commtest.ErrorRate` and `testconsole.Results` objects have been removed. To compute error rates, use the `comm.ErrorRate System` object or the **Bit Error Rate Analysis** app instead.

Compatibility Considerations

For alternate swept BER simulation workflows, see Bit Error Rate Analysis Techniques.

For more information, see the **Compatibility Considerations** section on the `commtest.ErrorRate` and `testconsole.Results` object reference pages.

equalizer and adaptalg packages and functions have been removed

Errors

The `equalizer` and `adaptalg` packages and functions have been removed. Use `comm.LinearEqualizer` and `comm.DecisionFeedbackEqualizer` instead.

To update your code, change instances of the package and function names to the recommended replacement in this table.

Removed	Recommended
<code>equalize</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>lineareq</code> and <code>equalizer.lineareq</code>	<code>comm.LinearEqualizer</code>
<code>dfe</code> and <code>equalizer.dfe</code>	<code>comm.DecisionFeedbackEqualizer</code>
<code>lms</code> and <code>adaptalg.lms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>rls</code> and <code>adaptalg.rls</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>cma</code> and <code>adaptalg.cma</code>	<code>comm.LinearEqualizer</code>
<code>signlms</code> and <code>adaptalg.signlms</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> with the adaptive algorithm set to LMS.
<code>normlms</code> and <code>adaptalg.normlms</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> with the adaptive algorithm set to LMS.

Removed	Recommended
<code>varlms</code> and <code>adaptalg.varlms</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> with the adaptive algorithm set to LMS.

Certain equalization blocks have been removed

Errors

The equalization blocks listed in the **Removed** column of this table have been removed. Use the Decision Feedback Equalizer or Linear Equalizer block instead.

To update your model, change instances of the blocks to the recommended replacement in this table.

Removed	Recommended
CMA Equalizer	Linear Equalizer
LMS Linear Equalizer	
RLS Linear Equalizer	
Normalized LMS Linear Equalizer	Consider using Linear Equalizer with the adaptive algorithm set to LMS.
Sign LMS Linear Equalizer	
Variable Step LMS Linear Equalizer	
LMS Decision Feedback Equalizer	Decision Feedback Equalizer
RLS Decision Feedback Equalizer	
Normalized LMS Decision Feedback Equalizer	Consider using Decision Feedback Equalizer with the adaptive algorithm set to LMS.
Sign LMS Decision Feedback Equalizer	
Variable Step LMS Decision Feedback Equalizer	

For more information, see the Adaptive Equalizers topic.

R2021b

Version: 7.6

New Features

Bug Fixes

Compatibility Considerations

Multiband signal combiner System object and block

The `comm.MultibandCombiner` System object and Multiband Combiner block shift input signals to the specified frequency bands and combine them into a single signal. Use these features for carrier aggregation of baseband signals, interference modeling, and coexistence modeling. The Multiband Signal Generation example illustrates use of the `comm.MultibandCombiner` System object in MATLAB. The Interference Modeling example illustrates use of the Multiband Combiner block in Simulink.

Sample rate offset System object

The `comm.SampleRateOffset` System object creates a sample rate offset object that you can use to apply sample rate offset to an input signal.

Low-density parity check (LDPC) encoding and decoding functions

The `ldpcEncode` and `ldpcDecode` functions encode and decode multicodeword inputs by using an LDPC code. To configure inputs for the LDPC encoder and decoder functions, use these functions.

- To specify an LDPC encoder configuration object, use the `ldpcEncoderConfig` function.
- To specify an LDPC decoder configuration object, use the `ldpcDecoderConfig` function. You can choose belief propagation, layered belief propagation, normalized min-sum, or offset min-sum LDPC decoding by using the `Algorithm` property.
- To compute the parity-check matrix of a quasi-cyclic LDPC code, use the `ldpcQuasiCyclicMatrix` function.

Spectrum sensing with deep learning example

The Spectrum Sensing with Deep Learning to Identify 5G and LTE Signals example shows how to train a semantic segmentation network using deep learning for spectrum monitoring. One of the uses of spectrum monitoring is to characterize spectrum occupancy. The neural network in this example is trained to identify 5G and LTE signals in a wideband of spectrum.

Ultra wideband (UWB) IEEE 802.15.4z wireless communications examples

The Communications Toolbox Library for ZigBee and UWB add-on includes these new examples to simulate UWB communications:

- The HRP UWB IEEE 802.15.4a/z Waveform Generation example shows how to generate standard-compliant high rate pulse repetition frequency (HRP) ultra wideband (UWB) waveforms of the IEEE.802.15.4a/z standard.
- The End-to-End Simulation of HRP UWB IEEE 802.15.4a/z PHY example performs end-to-end simulation over an additive white Gaussian noise (AWGN) channel for the high rate pulse repetition frequency (HRP) ultra wideband (UWB) PHY of the IEEE.802.15.4a/z standard.
- The UWB Ranging Using IEEE 802.15.4z example shows how to estimate distance between two devices as per the IEEE 802.15.4z standard.

-
- The UWB Localization Using IEEE 802.15.4z example shows how to estimate the location of a single device as per the IEEE 802.15.4z standard.

Wireless Waveform Generator app export-to-Simulink capability

The **Wireless Waveform Generator** app now enables you to export to Simulink. You can use the exported source block to generate waveforms in Simulink. For more information about the block that is created, see the [Waveform From Wireless Waveform Generator App](#) block reference page.

Ray tracing analysis with SBR method

- The default ray tracing method is now 'sbr' with up to two reflections (instead of 'image' with up to one reflection).
- Updates in the ray tracing algorithm result in minor improvements to ray precision and the number of rays computed.

For more information, see the [Ray Tracing for Wireless Communications and Choose a Propagation Model](#) topics and the `propagationModel` function.

Ray tracing channel System object support for control of start time and frame continuity

The `comm.RayTracingChannel` System object now enables you to:

- Add a start time delay prior to the first received impulse to account for the propagation delay between a transmitter and receiver.
- Support discontinuous frame transmissions by specifying the start time of each input frame when calling the ray tracing channel object.
- Model 5G NR antennas and arrays by using the `phased.NRAntennaElement` (Phased Array System Toolbox) and `phased.NRRectangularPanelArray` (Phased Array System Toolbox) System objects. To use these features, you must have the Phased Array System Toolbox product.
- Generate the channel impulse response matrix without applying channel filtering by using the added `ChannelFiltering`, `NumSamples`, and `OutputDataType` properties.

Computation of channel impulse response without applying channel filtering

Generate the channel impulse response matrix without applying channel filtering by using the `ChannelFiltering`, `NumSamples`, and `OutputDataType` properties that have been added to the `comm.RayTracingChannel`, `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects.

Support for importing and viewing RF propagation paths within indoor scenes

In Site Viewer, you can now import and view standard tessellation language (STL) models or triangulation representations that have Cartesian coordinates by using the `siteviewer` object. The object has these added properties to support Cartesian coordinates.

Property	Description
SceneModel	STL model file or triangulation object
Transparency	Transparency of model
ShowOrigin	Option to show origin
ShowEdges	Option to show edges of model

These functions now support `txsite` and `rxsite` objects that have Cartesian coordinates.

- Display RF propagation rays by using the `raytrace` function.
- Display the line-of-sight (LOS) visibility status between sites by using the `los` function.
- Display the communication link status between sites by using the `link` function.
- Display the radiation patterns for a site by using the `pattern` function.
- Control the visibility of sites by using the `show` and `hide` functions.

When you import a scene using Site Viewer, analysis functions such as `raytrace` and `los` use the scene as the 3-D environment.

Model and visualize RF propagation using MATLAB Online

Starting in R2021b, you can model and visualize RF propagation using MATLAB Online™. For more information about RF propagation modeling, see Propagation and Channel Models.

Updated display for constellation diagram System object

The display for `comm.ConstellationDiagram` System object has an updated interface with separate toolstrip tabs to configure constellation diagram plots and measurements.

Bit Error Rate Analysis app acceleration enhancements

The **Bit Error Rate Analysis** app now supports:

- Simulation acceleration by using the Parallel Computing Toolbox product for MATLAB code and Simulink models
- Running simulations using MEX File Functions. For more information, see Code Generation and Acceleration Support.

Phase noise System object and block support for multichannel input

The `comm.PhaseNoise` System object and Phase Noise block now support multichannel inputs.

Functions for converting between integer and binary data include column-wise and 3D input-output support

The `int2bit` and `bit2int` functions enable you to convert between integer and binary data for wireless communication signals. The functions support 3-D inputs and outputs, use a column-wise orientation of signal samples, support orienting the most-significant bit (MSB), apply integer-only processing for integer data type inputs and outputs, and support code generation. These functions

can help you avoid the need to reshape and manipulate data to orient the data for multichannel signal processing.

Improved speed performance for certain Communications Toolbox features

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

Feature	Conditions	Speedup mode
comm.FMBroadcastDemodulator	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
comm.RaisedCosineTransmitFilter		
comm.OQPSKModulator		
comm.RBDSWaveformGenerator		
Windowed Integrator	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 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
Raised Cosine Transmit Filter		

Communications Toolbox Support Package for USRP Radio: Improved performance for SDRu System objects

Enhanced support for valid data in SDRu receiver System object

The `comm.SDRuReceiver` (Communications Toolbox Support Package for USRP Radio) System object now receives valid data each time you call the System object.

Improved performance for SDRu receiver System object with data reception and number of overruns

The `comm.SDRuReceiver` (Communications Toolbox Support Package for USRP Radio) System object shows improved performance with data reception time and number of overruns. For example, compared to the previous release, this code is about 3.3x faster and reports fewer overruns with a sample rate of 20 MHz:

```

rx = comm.SDRuReceiver('Platform','B210','SerialNum','30F597A', ...
    'CenterFrequency',2.45e9,'MasterClockRate',40e6,'DecimationFactor',2, ...
    'TransportDataType','int16','OutputDataType','double', ...
    'SamplesPerFrame',1e4);

overrunCount = 0;
data = rx();
tic
for i = 1:2e4
    [data,dataLength,overrun] = rx();
    if overrun
        overrunCount = overrunCount + 1;
    end
end
receiveTime = toc;
disp(['Overrun count = ',num2str(overrunCount)]);
release(rx);

```

The approximate execution times are:

R2021a: 32.99 s (with 2214 overruns)

R2021b: 10.00 s (with 1 overrun)

The code was timed on a Windows 10, Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz test system by running the above script.

Performance improves when you set the `SamplesPerFrame` property value greater than or equal to 10000 for sample rates greater than 10 MHz.

Improved performance for SDRu transmitter System object with number of underruns

The `comm.SDRuTransmitter` (Communications Toolbox Support Package for USRP Radio) System object shows improved performance with number of underruns. For example, compared to the previous release, this code is about 1.1x faster and reports fewer underruns with a sample rate of 20 MHz:

```

tx = comm.SDRuTransmitter('Platform','B210','SerialNum','30F59A1', ...
    'CenterFrequency',2.45e9, ...
    'MasterClockRate',20e6,'InterpolationFactor',1);

SamplesPerFrame = 1e4;
sinGen = dsp.SineWave('Frequency',100e3,'SampleRate',20e6, ...
    'SamplesPerFrame',SamplesPerFrame,'ComplexOutput',true);

data = sinGen();
underrunCount = 0;
tx(data);
tic
for i = 1:2e4
    underrun = tx(data);
    if underrun
        underrunCount = underrunCount + 1;
    end
end
transmitTime = toc;
disp(['underrun count = ',num2str(underrunCount)]);
release(tx);

```

The approximate execution times are:

R2021a: 11.17 s (with 1080 underruns)

R2021b: 10.01 s (with 21 underruns)

The code was timed on a Windows 10, Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz test system by running the above script.

Performance improves when the input data length is greater than or equal to 10000 for sample rates greater than 10 MHz.

Improved performance for SDRu System objects with burst mode

The SDRu System objects shows improved performance with burst mode buffering.

Receiver burst mode processing

The `comm.SDRuReceiver` System object shows improved performance with burst mode. For example, compared to the previous release, this code is about 10x faster to capture 0.2 seconds of data at sample rate of 20 MHz:

```
rx = comm.SDRuReceiver('Platform','B200','SerialNum','30EA06C', ...
    'CenterFrequency',2.45e9,'MasterClockRate',40e6,'DecimationFactor',2, ...
    'TransportDataType','int16','OutputDataType','double', ...
    'EnableBurstMode',true,'NumFramesInBurst',4000,'SamplesPerFrame',1e3);

data = rx();
tic
for i = 1:3999
    data = step(rx);
end
retrievalTime = toc;
disp(['Burst retrieval time = ', num2str(retrievalTime)]);
release(rx);
```

The approximate burst retrieval times are:

R2021a: 4.39 s

R2021b: 0.46 s

The code was timed on a Windows 10, Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz test system by running the above script.

Transmitter burst mode processing

The `comm.SDRuTransmitter` System object shows improved performance with burst mode. For example, compared to the previous release, this code is about 4.5x faster to send 0.2 seconds of data at sample rate of 20 MHz:

```
tx = comm.SDRuTransmitter('Platform','B200','SerialNum','30EA06C', ...
    'CenterFrequency',2.45e9, ...
    'MasterClockRate',40e6,'InterpolationFactor',2, ...
    'EnableBurstMode',true,'NumFramesInBurst',4000);

SamplesPerFrame = 1e3;
sinGen = dsp.SineWave('Frequency',100e3,'SampleRate',20e6, ...
    'SamplesPerFrame',SamplesPerFrame,'ComplexOutput',true);

data = sinGen();
tx(data);
tic
```

```
for i = 1:3999
    tx(data);
end
transmitTime = toc;
disp(['Burst transmit time = ', num2str(transmitTime)]);
release(tx);
```

The approximate burst transmission times are:

R2021a: 3.17 s

R2021b: 0.71 s

The code was timed on a Windows 10, Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz test system by running the above script.

Functionality being removed or changed

Communications Toolbox Library for the ZigBee Protocol has moved to Communications Toolbox Library for ZigBee and UWB

Behavior change

Starting in R2021b, the Communications Toolbox Library for the ZigBee Protocol add-on has been renamed to Communications Toolbox Library for ZigBee and UWB.

Default value of Method name-value argument for propagationModel function is now shooting and bouncing rays method

Behavior change

Starting in R2021b, when you create a propagation model using the syntax `propagationModel('raytracing')`, MATLAB returns a `RayTracing` model with the `Method` name-value argument value set to `'sbr'` and two reflections (instead of `'image'` and one reflection as in previous releases). To create ray tracing propagation models that use the image method, use the syntax `propagationModel('raytracing','Method','image')`. For more information, see the [Choose a Propagation Model](#) topic and the `propagationModel` function.

raytrace, coverage, link, sigstrength, and sinr use shooting and bouncing rays method

Behavior change

Starting in R2021b, when you use the `raytrace`, `coverage`, `link`, `sigstrength`, or `sinr` function and specify the `propmodel` argument or `PropagationModel` name-value argument as `'raytracing'`, the function uses the shooting and bouncing rays (SBR) method and calculates up to two reflections. In previous releases, the functions use the image method and calculate up to one reflection.

If you prefer to use the image method instead of the SBR method, create a propagation model by using the `propagationModel` function. Then, use the `raytrace`, `coverage`, `link`, `sigstrength`, or `sinr` function with the propagation model as input. This example shows how to update your code.

```
pm = propagationModel('raytracing','Method','image');
coverage(txs,pm)
```

For information about the SBR and image methods, see [Choose a Propagation Model](#).

ReflectionLocations and NumReflections properties of the comm.Ray object have been removed

Errors

The ReflectionLocations and NumReflections properties have been removed from the comm.Ray object. To accommodate reflections, use the Interactions property to replace the ReflectionLocations property and use the NumInteractions property to replace the NumReflections property.

NumReflections property of the raytrace function will be removed

Still runs

The NumReflections name-value pair argument of the raytrace function will be removed in a future release. The NumReflections name-value pair argument now only applies for the image ray-tracing method. Instead, create a propagation model by using the propagationModel function with its MaxNumReflections name-value argument. Then, use the raytrace function with the propagation model as an input. This example shows the recommended workflow.

```
pm = propagationModel('raytracing', ...
    'Method','image','MaxNumReflections',2);
rays = raytrace(tx,rx,pm);
```

bi2de function is not recommended

Still runs

The bi2de function is not recommended. Use the bit2int function instead. If converting numbers from a nonbase-2 representation to decimal, use base2dec.

The code in this table shows binary-to-decimal conversion for various inputs using the recommended function.

Discouraged Feature	Recommended Replacement
<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,1],n*bpi,1); y = bi2de(reshape(x,bpi,[]),'left-msb');</pre>	<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,1],n*bpi,1); y = bit2int(x,bpi);</pre>
<pre>% Default row vector (or matrix) input x = [0 1 1]; bi2de(x)</pre>	<pre>% Default row vector (or matrix) input x = [0 1 1]; bit2int(x,length(x),0)'</pre>
<pre>% Right MSB, logical input n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = logical(randi([0,1],n*bpi,1)); y = bi2de(reshape(x,bpi,[]),'right-msb');</pre>	<pre>% Right MSB, logical input n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = logical(randi([0,1],n*bpi,1)); y = bit2int(x,bpi,false);</pre>
<pre>% Right MSB, signed input, single input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer x = randi([0,1],n*bpi,1,'single'); y = bi2de(reshape(x,bpi,[]),'right-msb'); N = 2^bpi; y = y - (y>=N/2)*N;</pre>	<pre>% Right MSB, signed input, single input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer x = randi([0,1],n*bpi,1,'single'); y = bit2int(x,bpi,false); N = 2^bpi; y = y - (y>=N/2)*N;</pre>

de2bi function is not recommended

Still runs

The de2bi function is not recommended. Use the int2bit function instead. If converting the representation of numbers from decimal to a base other than 2, use dec2base.

The code in this table shows decimal-to-binary conversion for various inputs using the recommended function.

Discouraged Feature	Recommended Replacement
<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,2^bpi-1],n,1); y = reshape(de2bi(x,bpi,'left-msb'),[],1)</pre>	<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,2^bpi-1],n,1); y = int2bit(x,bpi)</pre>
<pre>% Default vector (or scalar) input x = [4 5 9]; y = de2bi(x)</pre>	<pre>% Default vector (or scalar) input x = [4 5 9]; y = int2bit(x,ceil(log2(max(x) + 1)), 0)'</pre>
<pre>% Right MSB n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = randi([0,2^bpi-1],n,1); y = reshape(de2bi(x,bpi,'right-msb'),[],1)</pre>	<pre>% Right MSB n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = randi([0,2^bpi-1],n,1); y = int2bit(x,bpi,false)</pre>
<pre>% Right MSB, signed input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer N = 2^bpi; x = randi([-N/2,N/2-1],n,1); y = reshape(de2bi(x+(x<0)*N,bpi,'right-msb'),[],1)</pre>	<pre>% Right MSB, signed input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer N = 2^bpi; x = randi([-N/2,N/2-1],n,1); y = int2bit(x+(x<0)*N,bpi,false)</pre>

comm.LDPCDecoder and comm.LDPCDecoder System objects will be removed

Still runs

The comm.LDPCDecoder and comm.LDPCDecoder System objects will be removed in a future release. Instead, use the ldpcEncode and ldpcDecode and ldpcDecode functions. To specify the LDPC code applied by the ldpcEncode function, use the configuration object returned by the ldpcEncoderConfig object. To specify the LDPC code applied by the ldpcDecode function, use the configuration object returned by the ldpcDecoderConfig object.

The code in this table shows how to LDPC encode and decode inputs using the recommended function and configuration object.

Discouraged Feature	Recommended Replacement
<pre>% Encode using parity-check matrix (pcmatrix) enc = comm.LDPCDecoder(pcmatrix); codeword = enc(infoBits);</pre>	<pre>% Encode using parity-check matrix (pcmatrix) cfg = ldpcEncoderConfig(pcmatrix); codeword = ldpcEncode(infoBits,cfg);</pre>
<pre>% Decode using parity-check matrix (pcmatrix) dec = comm.LDPCDecoder(pcmatrix); dec.OutputValue = 'Whole codeword'; dec.DecisionMethod = 'Soft decision'; dec.MaximumIterationCount = 10; dec.IterationTerminationCondition = 'Parity check satisfied'; output = dec(LLR);</pre>	<pre>% Decode using parity-check matrix (pcmatrix) cfg = ldpcDecoderConfig(pcmatrix); output = ldpcDecode(LLR,cfg,10, ... 'OutputFormat','whole', ... 'DecisionType','soft', ... 'Termination','early');</pre>

bin2gray and gray2bin functions will be removed

Warns

The bin2gray and gray2bin functions will be removed in a future release. Instead, use the appropriate modulation object or function to remap constellation points. This table shows the remapping based on modulation type.

When you use the workflow that is discouraged, the bin2gray and gray2bin functions convert a binary representation to a natural binary or Gray encoding. After the conversion, you must specify 'bin' for the symbol order when you call the modulation and demodulation functions.

When you use the workflow that is recommended, for any given of modulation scheme, you provide decimal values when you call the modulation and demodulation functions. When you call the modulation and demodulation functions, specify the symbol order as 'bin' for natural binary encoding or 'gray' for Gray encoding.

If your workflow uses bin2gray or gray2bin with any of the modulation schemes in this table, follow the appropriate example.

Modulation	Discouraged Feature	Recommended Replacement
QAM (qammod and qamdemod)	<pre>x = randi([0 63],1,100); y = bin2gray(x,'qam',64); z = qammod(y,64,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = qamdemod(x,64,'bin'); z = gray2bin(y,'qam',64);</pre>	<pre>x = randi([0 63],1,100); z = qammod(x,64,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = qamdemod(x,64,'gray')</pre>
PAM (pammod and pamdemod)	<pre>x = randi([0 63],1,100); y = gray2bin(x,'pam',64); z = pammod(y,64,pi/4,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = pamdemod(x,64,pi/4,'bin'); z = bin2gray(y,'pam',64);</pre>	<pre>x = randi([0 63],1,100); z = pammod(x,64,pi/4,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = pamdemod(x,64,pi/4,'gray')</pre>
FSK (fskmod and fskdemod)	<pre>x = randi([0 63],1,100); y = gray2bin(x,'fsk',64); z = fskmod(y,64,1,256,256,'cont','bin'); x = 2*(randn(512,1)+1j*randn(512,1)); y = fskdemod(x,64,1,256,256,'bin'); z = bin2gray(y,'fsk',64)</pre>	<pre>x = randi([0 63],1,100); z = fskmod(x,64,1,256,256,'cont','gray'); x = 2*(randn(512,1)+1j*randn(512,1)); z = fskdemod(x,64,1,256,256,'gray');</pre>
DPSK (dpskmod and dpskdemod)	<pre>x = randi([0 63],1,100); y = gray2bin(x,'dpsk',64); z = dpskmod(y,64,pi/4,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = dpskdemod(x,64,pi/4,'bin'); z = bin2gray(y,'dpsk',64);</pre>	<pre>x = randi([0 63],1,100); z = dpskmod(x,64,pi/4,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = dpskdemod(x,64,pi/4,'gray');</pre>
PSK (pskmod and pskdemod)	<pre>x = randi([0 63],1,100); y = gray2bin(x,'psk',64); z = pskmod(y,64,0,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = pskdemod(x,64,0,'bin'); z = bin2gray(y,'psk',64);</pre>	<pre>x = randi([0 63],1,100); z = pskmod(x,64,0,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = pskdemod(x,64,0,'gray');</pre>

commsrc.pn object will be removed

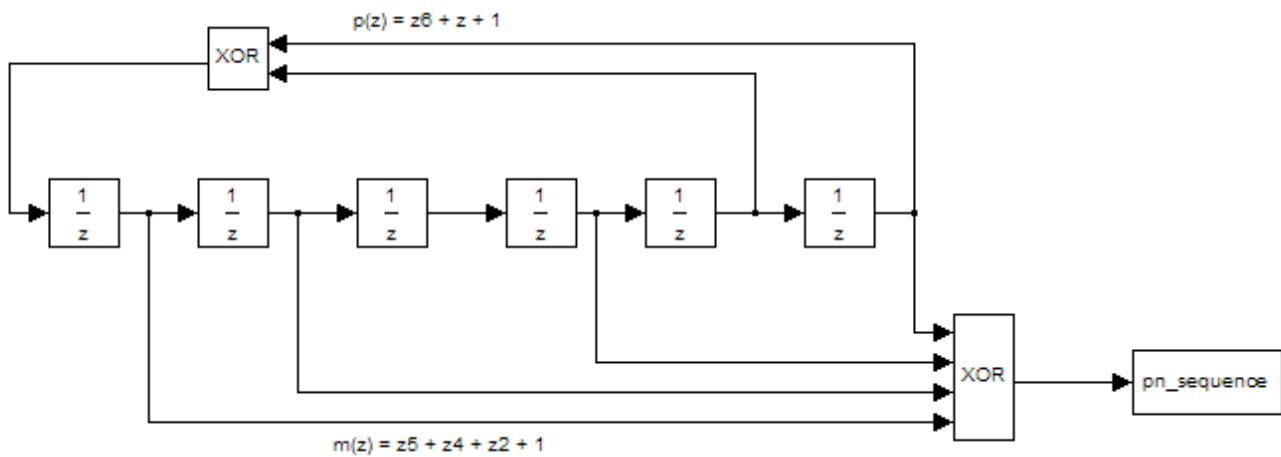
Warns

The `commsrc.pn` object will be removed in a future release. To generate a pseudo-noise (PN) sequence, use the `comm.PNSequence` System object.

Replace instances of the `commsrc.pn` object with a `comm.PNSequence` System object. This table shows the mapping between `commsrc.pn` properties and `comm.PNSequence` properties.

commsrc.pn Object Properties	comm.PNSequence System Object Properties
GenPoly	Polynomial
InitialStates	InitialConditions
Mask	Mask
NumBitsOut	SamplesPerFrame
CurrentStates	Not applicable

For example, consider this PN sequence generator with a generator polynomial $p(z) = z^6 + z + 1$.



This table shows some typical usages of `commsrc.pn` and how to update your code to use `comm.PNSequence` instead.

Discouraged Usage	Recommended Replacement
<p>Define the PN sequence generator.</p> <pre>h1 = commsrc.pn('GenPoly',[1 0 0 0 0 1 1], 'Mask',[1 1 0 1 0 1]); h2 = commsrc.pn('GenPoly',[1 0 0 0 0 1 1], 'Shift',22); mask2shift([1 0 0 0 0 1 1],[1 1 0 1 0 1]) ans = 22</pre> <p>Alternatively, input GenPoly as the exponents of z for the nonzero terms of the polynomial in descending order of powers.</p> <pre>h = commsrc.pn('GenPoly',[6 1 0], 'Mask',[1 1 0 1 0 1]) h = GenPoly: [1 0 0 0 0 1 1] InitialStates: [0 0 0 0 0 1] CurrentStates: [0 0 0 0 0 1] Mask: [1 1 0 1 0 1] NumBitsOut: 1</pre>	<p>Define the PN sequence generator.</p> <pre>h1 = comm.PNSequence('Polynomial',[1 0 0 0 0 1 1], ... 'InitialConditions', [1 1 0 1 0 1]); h2 = comm.PNSequence('Polynomial',[1 0 0 0 0 1 1], ... 'Mask',22); mask2shift([1 0 0 0 0 1 1],[1 1 0 1 0 1]) ans = 22</pre> <p>Alternatively, input the polynomial exponents of z for the nonzero terms of the polynomial in descending order of powers.</p> <pre>h = comm.PNSequence('Polynomial',[6 1 0],... 'InitialConditions',[1 1 0 1 0 1]) h = comm.PNSequence with properties: Polynomial: [6 1 0] InitialConditionsSource: 'Property' InitialConditions: [1 1 0 1 0 1] MaskSource: 'Property' Mask: 0 VariableSizeOutput: false SamplesPerFrame: 1 ResetInputPort: false BitPackedOutput: false OutputDataType: 'double'</pre>

hank2sys function will be removed

Warns

The hank2sys function will be removed in a future release.

hilbiir function will be removed

Warns

The hilbiir function will be removed in a future release. To design a Hilbert transform filter, use the fdesign.hilbert object.

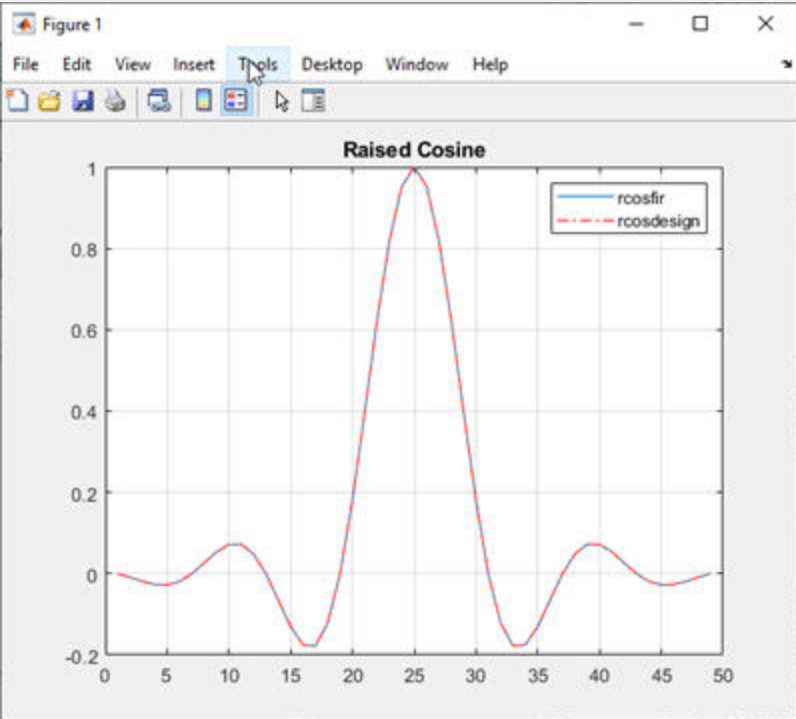
rcosfir, rcosiir, rcosflt, and rcosine functions will be removed

Warns

The rcosfir, rcosiir, rcosine, and rcosflt functions will be removed in a future release. To design a raised cosine pulse-shaping filter, use the rcosdesign function.

- Replace all instances of rcosfir, rcosiir, and rcosine with the rcosdesign function.
- Replace all instances of rcosflt with a comm.RaisedCosineTransmitFilter or comm.RaisedCosineReceiveFilter System object.

This example shows typical usage of `rcosfir` and how to use `rcosdesign` instead. You can apply a similar workflow to update `rcosfir` and `rcosine` to `rcosdesign` instead.

Discouraged Usage	Recommended Replacement
<p>Design a raised cosine filter by using the <code>rcosfir</code> function.</p> <pre data-bbox="238 472 857 619">% Filter length is 2*extent*rate + 1 rolloff = 0.3; % Rolloff factor extent = 4; % Filter extent rate = 6; % Oversampling rate T = 1; % Input sample rate b3 = rcosfir(rolloff,extent,rate,T,'normal');</pre>	<p>Design an identical raised cosine filter by using the <code>rcosdesign</code> function.</p> <pre data-bbox="857 472 1479 619">% Filter length is 2*extent*rate + 1 rolloff = 0.3; % Rolloff factor extent = 4; % Filter extent rate = 6; % Oversampling rate T = 1; % Input sample rate b3n = rcosdesign(rolloff,2*extent,rate,'normal'); b3n = b3n/max(b3n); % Normalize coefficients</pre>
<p>To show that the two filters are identical by plotting the filter coefficient values.</p> <pre data-bbox="238 766 857 997">figure plot(b3) hold on plot(b3n,'r-.') grid on title('Raised Cosine') legend('rcosfir','rcosdesign') max(abs(b3-b3n))</pre>	
	

Discouraged Usage	Recommended Replacement
Design a square-root raised cosine filter by using <code>rcosfir</code> . <pre>b4 = rcosfir(rolloff,extent,rate,1,'sqrt')</pre>	Design identical square-root raised cosine filter by using <code>rcosdesign</code> . <pre>b4n = rcosdesign(rolloff,2*extnet,rate,'sqrt'); % Normalize coefficients b4n = b4n/max(b4n)* ... ((-1./(pi.*rate).* ... (pi.*(rolloff-1)-4.*rolloff))*sqrt(rate);</pre>

comm.BitToInteger System object has been removed

Errors

The `comm.BitToInteger` System object has been removed. Use the `bit2int` function instead.

Input types that are supported by `comm.BitToInteger` are not inherently supported by functions. The code in this table shows binary-to-decimal conversion for various inputs using the recommended function.

Removed Feature	Recommended Replacement
<pre>% Default (left MSB) n = randi([1 100]); % Number of integers h1 = comm.BitToInteger; bpi = h1.BitsPerInteger; x = randi([0,1],n*bpi,1); y = h1(x);</pre>	<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,1],n*bpi,1); y = bit2int(x,bpi);</pre>
<pre>% Right MSB, logical input n = randi([1 100]); % Number of integers h2 = comm.BitToInteger(... 'BitsPerInteger',5, ... 'MSBFirst',false); bpi = h2.BitsPerInteger; x = logical(randi([0,1],n*bpi,1)); y = h2(x);</pre>	<pre>% Right MSB, logical input n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = logical(randi([0,1],n*bpi,1)); y = bit2int(x,bpi,false);</pre>
<pre>% Right MSB, signed input, single input n = randi([1 100]); % Number of integers h3 = comm.BitToInteger(... 'BitsPerInteger',8, ... 'MSBFirst',false, ... 'SignedIntegerOutput',true); bpi = h3.BitsPerInteger; x = randi([0,1],n*bpi,1,'single'); y = h3(x);</pre>	<pre>% Right MSB, signed input, single input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer x = randi([0,1],n*bpi,1,'single'); y = bit2int(x,bpi,false); N = 2^bpi; y = y - (y>=N/2)*N;</pre>

comm.IntegerToBit System object has been removed

Errors

The `comm.IntegerToBit` System object has been removed. Use the `int2bit` function instead.

Input types that are supported by `comm.IntegerToBit` are not inherently supported by functions. The code in this table shows decimal-to-binary conversion for various inputs using the recommended function.

Removed Feature	Recommended Replacement
<pre>% Default (left MSB) n = randi([1 100]); % Number of integers h1 = comm.IntegerToBit; bpi = h1.BitsPerInteger; x = randi([0,2^bpi-1],n,1); y = h1(x);</pre>	<pre>% Default (left MSB) n = randi([1 100]); % Number of integers bpi = 3; % Bits per integer x = randi([0,2^bpi-1],n,1); y = int2bit(x,bpi);</pre>

Removed Feature	Recommended Replacement
<pre>% Right MSB n = randi([1 100]); % Number of integers h2 = comm.IntegerToBit(... 'BitsPerInteger',5, ... 'MSBFirst',false); bpi = h2.BitsPerInteger; x = randi([0,2^bpi-1],n,1); y = h2(x);</pre>	<pre>% Right MSB n = randi([1 100]); % Number of integers bpi = 5; % Bits per integer x = randi([0,2^bpi-1],n,1); y = int2bit(x,bpi,false);</pre>
<pre>% Right MSB, signed input n = randi([1 100]); % Number of integers h3 = comm.IntegerToBit(... 'BitsPerInteger',8, ... 'MSBFirst',false, ... 'SignedIntegerInput',true); bpi = h3.BitsPerInteger; N = 2^bpi; x = randi([-N/2,N/2-1],n,1); y = h3(x);</pre>	<pre>% Right MSB, signed input n = randi([1 100]); % Number of integers bpi = 8; % Bits per integer N = 2^bpi; x = randi([-N/2,N/2-1],n,1); y = int2bit(x+(x<0)*N,bpi,false);</pre>

Align Signals block has been removed

Errors

The Align Signals block has been removed. Use the Find Delay block to find delays and the Delay (Simulink) block to apply delays instead.

For more information, see the Compatibility Considerations section on the Align Signals block reference page.

Certain interleaver and deinterleaver System objects have been removed

Errors

Starting in R2021b, use of the System objects in this table results in an error message. These objects have been removed. This table shows the recommended replacement function for each System object.

Removed Feature	Recommended Replacement
comm.AlgebraicInterleaver	algintrlv
comm.AlgebraicDeinterleaver	algdeintrlv
comm.BlockInterleaver	intrlv
comm.BlockDeinterleaver	deintrlv
comm.MatrixInterleaver	matintrlv
comm.MatrixDeinterleaver	matdeintrlv
comm.MatrixHelicalScanInterleaver	helscanintrlv
comm.MatrixHelicalScanDeinterleaver	helscandeintrlv

R2021a

Version: 7.5

New Features

Compatibility Considerations

Ray tracing analysis using SBR method

You can now perform ray tracing analysis using the shooting and bouncing rays (SBR) method. To display propagation paths or coverage maps using the SBR method, create a ray tracing propagation model using the `propagationModel` function, and then specify the model as an input when you call the `raytrace` or `coverage` function.

For example, this code shows how to display the coverage map for the transmitter site, `tx`, based on the ray tracing propagation model, `pm`, that uses the SBR method.

```
tx = txsite;  
pm = propagationModel('raytracing','Method','sbr');  
coverage(tx,pm)
```

Propagation models that use the SBR method support up to 10 path reflections and are generally faster than propagation models that use the image method. Ray tracing propagation models that use the SBR method calculate approximate propagation paths, and ray tracing propagation models that use the image method calculate exact propagation paths. For more information about the SBR and image methods, see [Choose a Propagation Model](#).

RF propagation object and functions deployment using MATLAB Compiler

You can now use the MATLAB Compiler™ software to deploy RF propagation objects and functions, such as the `siteviewer` object and the `propagationModel`, `coverage`, `raytrace`, and `addCustomTerrain` functions.

For a full list of RF propagation objects and functions, see [Propagation and Channel Models](#).

Autoencoders for wireless communications example

The Autoencoders for Wireless Communications example models an end-to-end communications system with an autoencoder to reliably transmit information bits over a wireless channel. This example uses features from the Deep Learning Toolbox™ and Statistics and Machine Learning Toolbox™ software.

Memoryless nonlinearity System object mode and visualization updates

The `comm.MemorylessNonlinearitySystem` object now supports these features:

- Visualization of amplitude-to-amplitude modulation (AM/AM) and amplitude-to-phase modulation (AM/PM) characteristics using the `plot` object function to plot gain versus P_{in} , P_{out} versus P_{in} , and phase change versus P_{in}
- Modified Rapp model that includes AM/PM characteristics using properties to specify phase gain, phase saturation, and phase smoothness
- Cubic polynomial model that adds third order intercept (TOI) modes to specify OIP_3 , IP_{1dB} , OP_{1dB} , IP_{sat} , and OP_{sat} cubic nonlinearity modes
- Specification of the reference impedance of the signal

Turbo encoding and decoding System objects and blocks enable custom bit ordering and puncturing

The `comm.TurboEncoder` and `comm.TurboDecoder` System objects and the Turbo Encoder and Turbo Decoder blocks now support fixed-rate and adaptive-rate custom bit ordering and puncturing.

Bit Error Rate Analysis app updates

The **Bit Error Rate Analysis** app now includes these updates:

- The **Monte Carlo** tab now includes the **Simulation environment** parameter to distinguish between MATLAB and Simulink user simulations.
- Sessions are now saved as binary files. Sessions saved from previous app versions are still loadable.

For more information, see [Use Bit Error Rate Analysis App](#).

Wireless Waveform Generator app support for 5G NR uplink and downlink carrier waveforms

You can now use the **Wireless Waveform Generator** app to parameterize and generate NR uplink and downlink carrier waveforms. Use of this 5G feature requires the 5G Toolbox software.

Power measurement of voltage signal using MATLAB and Simulink

Compute the average power, peak power, and peak-to-average power ratio of a voltage signal by using the MATLAB powermeter System object or the Simulink Power Meter block. While calculating the power, the object and the block account for the reference load. Use of this feature requires the DSP System Toolbox™ software.

Bluetooth LE and BR/EDR features

Communications Toolbox Library for the Bluetooth Protocol now supports these Bluetooth low energy (BLE) functionalities.

- The `bleChannelSelection` System object now supports low energy (LE) isochronous events defined in Bluetooth Core Specification 5.2.
- The `bleWaveformGenerator` and `bleIdealReceiver` functions now support configuring the data-whiten status.

To download the Bluetooth add-on, see [Communications Toolbox Library for the Bluetooth Protocol](#). For more information on Communications Toolbox Library for the Bluetooth Protocol features and examples, see [Bluetooth](#).

Data over cable service interface specification (DOCSIS) example

The DOCSIS Upstream TDMA Link Simulation example shows how to implement the physical layer of the DOCSIS in the upstream TDMA mode using single carrier quadrature amplitude modulation (SC-QAM). The transmitted signal includes a DOCSIS-compliant medium access control layer (MAC)

header plus a payload of random data. The example includes a comparison of error-rate performance between the simulation and theoretical results. The propagation channel models a practical cable channel described in the DOCSIS Best Practices and Guidelines (CM-GL-PNMP-V03-160725).

Updated HDL-optimized QPSK example

The HDL QPSK Transmitter and Receiver example, replaces the former "HDL Optimized QPSK Transmitter" and "HDL Optimized QPSK Receiver with Captured Data" examples. The new example improves performance and shows best practices for implementing designs for FPGA and ASIC hardware.

Visual appearance updates to plots generated with eyediagram and scatterplot functions

The `eyediagram` and `scatterplot` functions now provide black plot backgrounds by default.

Support for C code generation using certain RF propagation functions

The `raypl`, `buildingMaterialPermittivity`, and `earthSurfacePermittivity` functions now support C code generation.

SIMD code generation using Intel AVX2 for certain Communications Toolbox features

SIMD is a computing paradigm in which a single instruction processes multiple data units. Many modern processors have SIMD instructions that, for example, perform several additions or multiplications at once. The Intel AVX2 SIMD intrinsics significantly improve the performance of the code generated using the computationally intensive algorithms listed here on Intel platforms, in most cases meeting or exceeding the simulation performance.

To generate SIMD code from MATLAB System objects, you need the MATLAB Coder™ and Embedded Coder® software. This table lists objects that support SIMD code generation using Intel AVX2 technology under the specified conditions.

MATLAB System Objects	Conditions
<code>comm.RaisedCosineTransmitFilter</code>	<ul style="list-style-type: none"> Input signal is real or complex value Input signal must be of data type <code>single</code> or <code>double</code>
<code>comm.RaisedCosineReceiveFilter</code>	

To generate SIMD code from Simulink blocks, you need Simulink Coder and Embedded Coder software. This table lists blocks that support SIMD code generation using Intel AVX2 technology under the specified conditions.

Simulink Blocks	Conditions
Raised Cosine Transmit Filter	<ul style="list-style-type: none"> Input signal must be a complex value Input signal must be of data type <code>single</code> or <code>double</code>

Simulink Blocks	Conditions
Raised Cosine Receive Filter	<ul style="list-style-type: none"> • Input processing parameter must be set to Columns as channels (frame based) • Rate options parameter must be set to Enforce single-rate processing

Functionality being removed or changed

Semianalytic tab in Bit Error Rate Analysis app has been removed

Behavior change

The **Semianalytic** tab and its functionality in the **Bit Error Rate Analysis** app have been removed. To generate semianalytic BER results, use the `semianalytic` function.

For example, this code shows how to use the `semianalytic` function to programmatically generate semianalytic BER results for a BPSK-modulated signal.

```
data = [0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0].';
txSig = rectpulse(step(comm.BPSKModulator,data),16);
rxSig = rectpulse(step(comm.BPSKModulator,data),16); % Before receive filter
modType = 'psk';
modOrder = 2;
sps = 16; % Samples per symbol
num = ones(16,1)/16; % Filter numerator
den = 1; % Filter denominator
EbNo = 0:18; % dB
BER = semianalytic(txSig,rxSig,modType,modOrder,sps,num,den,EbNo);
semilogy(EbNo,BER);
```

Default settings change for phase noise block and System object

Behavior change

The `comm.PhaseNoise` System object has new default values as shown in this table.

Property	New Default	Previous Default
Level	[-80 -100]	[-60 -80]
FrequencyOffset	[2000 20000]	[20 200]
SampleRate	1e6	1024
RandomStream	'Global stream'	'Global stream'

The Phase Noise block has new default values as shown in this table.

Parameter	New Default	Previous Default
Phase noise level (dBc/Hz)	[-80 -100]	[-60 -80]
Frequency offset (Hz)	[2000 20000]	[20 200]
Initial seed	1e6	1024
Simulate using	Interpreted execution	Interpreted execution

When executing code created in a previous release, confirm the phase noise settings. The new default settings will change your results. To produce results using the previous default phase noise settings, you can manually update the settings of your System object.

Synchronization component blocks will be removed

Warns

These blocks will be removed in a future release:

- Discrete-Time VCO
- Phase-Locked Loop
- Baseband PLL
- Linearized Baseband PLL
- Continuous-Time VCO
- Charge Pump PLL

To design voltage-controlled oscillators (VCOs) and phase-locked loops (PLLs), use the appropriate block listed in Phase-Locked Loops (Mixed-Signal Blockset).

Insert Zero block will be removed

Warns

The Insert Zero block will be removed in a future release. To insert zeros into a random number stream, use MATLAB code in a MATLAB Function (Simulink) block instead.

For example, to insert zeros into a data stream, you can use this code in a MATLAB Function block.

```
function y = fcn(u,insertZeroVector)
    numSeg = length(u)/sum(insertZeroVector);
    c = zeros(length(insertZeroVector),numSeg,'like',u);
    c(logical(insertZeroVector),:) = reshape(u,[],numSeg);
    y = c(:);
end
```

For the Insert Zero block, the input length must be an integer multiple of the number of ones in the **Insert zero vector** parameter. This same restriction applies for the input to the MATLAB code in a MATLAB Function block.

For an example using this code, see Insert Zeros into Random Number Stream.

comm.EyeDiagram System object will be removed

Warns

The `comm.EyeDiagram` System object will be removed in a future release. To display the eye diagram of a signal, use the `eyediagram` function instead.

randseed function has been removed

Errors

The `randseed` function has been removed. Use either of these syntaxes for the `rng` function instead: `rng(seed)` or `rng('shuffle')`.

The functionality provided by `randseed` is no longer necessary for controlling random number generation. Instead, use `rng(seed)` to seed the random number generator with the input nonnegative integer `seed` or `rng('shuffle')` to seed the random number generator based on the current time.

lteZadoffChuSeq has been renamed to zadoffChuSeq

Errors

The `lteZadoffChuSeq` function has been renamed to `zadoffChuSeq`. The functionality remains the same. Replace all instances of `lteZadoffChuSeq` with `zadoffChuSeq`. Beginning with this release, `lteZadoffChuSeq` errors.

Noise generator blocks have been removed

Errors

This table shows the noise generator blocks that have been removed and their recommended replacement mapping.

Removed Block	Recommended Replacement
Gaussian Noise Generator	MATLAB Function (Simulink) block and <code>randn</code> function
Rayleigh Noise Generator	MATLAB Function (Simulink) block and <code>randn</code> function
Rician Noise Generator	MATLAB Function (Simulink) block and <code>randn</code> function
Uniform Noise Generator	MATLAB Function (Simulink) block and <code>rand</code> function

For more information, see [Random Noise Generators](#).

`propagationModel('raytracing-image-method')` returns a `RayTracing` model

Behavior change

Starting in R2021a, when you create a propagation model using the syntax `propagationModel('raytracing-image-method')`, instead of a `RayTracingImageMethod` model, MATLAB returns a `RayTracing` model with a `Method` property value of `'image'`. All functions that accept `RayTracingImageMethod` propagation models also accept `RayTracing` propagation models, so this change does not affect your existing code.

To create ray tracing propagation models that use the image method, instead of the syntax `propagationModel('raytracing-image-method')`, use `propagationModel('raytracing','Method','image')`. For more information about creating propagation models, see the `propagationModel` function.

R2020b

Version: 7.4

New Features

Compatibility Considerations

RF Propagation and Ray-Tracing Enhancements

Enhancements to the RF propagation and ray-tracing capabilities enable you to:

- Specify the `Antenna` property of `txsite` and `rxsite` objects as an `arrayConfig` object. This feature enables you to model a uniform rectangular array (URA) with isotropic antenna elements, a uniform linear array (ULA) with isotropic antenna elements, or a single isotropic antenna element.
- Specify the `CoordinateSystem` and `AntennaPosition` properties of `txsite` and `rxsite` objects to specify an antenna site location using Cartesian coordinates.
- Use functions like `sigstrength`, `sinr`, and `pathloss` for point-to-point RF propagation analysis when using Cartesian coordinates for antenna sites.
- Use `raytrace` to perform ray tracing analysis in any 3-D map environment by specifying the `Map` parameter to an STL file or a object.
- To enable accurate path loss calculations, define surface materials for a 3-D map using new ray tracing propagation model properties such as `CoordinateSystem`, `SurfaceMaterial`, `SurfaceMaterialPermittivity`, and `SurfaceMaterialConductivity`.

The Indoor MIMO-OFDM Communication Link using Ray Tracing example analyzes and visualizes ray-tracing in Cartesian coordinate system by using enhancements to the RF propagation and ray-tracing capabilities.

Ray-Tracing Channel Modeling and Visualization

Enhancements to the Ray-tracing channel modeling capabilities enable you to:

- Filter a signal through a multipath fading channel that is defined by propagation rays from ray tracing using the `comm.RayTracingChannel` System object and visualize the channel profile using the `showProfile` object function.
- Model a URA with isotropic antenna elements, a ULA with isotropic antenna elements, or a single isotropic antenna element at the transmitter and receiver in the ray tracing channel model, using the `arrayConfig` configuration object.
- Specify the transmit and receive antenna arrays in the ray-tracing channel model using Phased Array System Toolbox antenna array or element objects.

The Indoor MIMO-OFDM Communication Link using Ray Tracing example analyzes and visualizes a ray-tracing multipath fading channel model by the using added features that include the `comm.RayTracingChannel` System object and `arrayConfig` configuration object.

Apply Multipath Gains with Channel Filter

To filter a signal using multipath gains at specified path delays, use the `comm.ChannelFilter` System object.

Bluetooth LE and BR/EDR Features and Examples

These are the newly added Bluetooth basic rate/enhanced data rate (BR/EDR) and Bluetooth low energy (BLE) features and functionalities.

- The `bluetoothFrequencyHop` object uses adaptive frequency hopping (AFH) to select the channel index for the Bluetooth BR/EDR physical layer (PHY) transmission mode. To generate a hopping sequence for inquiry, paging, and connection procedures, use `bluetoothFrequencyHop` object and the associated `nextHop` object function.
- Estimate the angle of arrival (AoA) or angle of departure (AoD) by using the `bleAngleEstimate` function. To parameterize this function, use the `bleAngleEstimateConfig` configuration object and the associated `getNumElements` and `getElementPosition` object functions. For more information about AoA and AoD direction finding capabilities, see Bluetooth Location and Direction Finding.
- The `bleWaveformGenerator` and `bleIdealReceiver` functions now support AoA and AoD capabilities.
- Write generated or recovered BLE link layer (LL) packets to a file with the `.pcap` or `.pcapng` extension by using the `pcapWriter` or `pcapngWriter` object, respectively. To write the BLE LL packets, use these objects with a wrapper `blePCAPWriter` object.

These added examples use BR/EDR features.

- Bluetooth Full Duplex Data and Voice Transmission in MATLAB - This example is a MATLAB implementation of the Bluetooth Full Duplex Voice and Data Transmission example with additional functionalities. You can now simulate a Bluetooth piconet and show a full-duplex communication between Bluetooth BR devices over asynchronous connection-oriented (ACL) or synchronous connection-oriented (SCO) links. The example uses the `bluetoothFrequencyHop` object to implement AFH.
- End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping - This example demonstrates an end-to-end simulation to study the performance of AFH in the Bluetooth BR/EDR PHY communication with WLAN signal interference.
- Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift - This example shows how to perform Bluetooth BR RF-PHY transmitter tests specific to the modulation characteristics, carrier frequency offset and drift of the Bluetooth BR waveform. The test measurement values are compared with the limits specified in the Bluetooth RF-PHY Test Specifications.
- Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability - This example shows how to perform Bluetooth EDR RF-PHY transmitter tests specific to the carrier frequency stability and modulation accuracy of the Bluetooth EDR waveform. The test measurement values are compared with the limits specified in the Bluetooth RF-PHY Test Specifications.

This added example uses BLE features.

- End-to-End BLE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN - This example demonstrates an end-to-end BLE simulation for different PHY transmission modes in the presence of path loss models, RF impairments, and additive white Gaussian noise (AWGN).

To download the Bluetooth add-on, see Communications Toolbox Library for the Bluetooth Protocol. For more information on features and examples in Communications Toolbox Library for the Bluetooth Protocol, see Bluetooth.

Training and Testing a Neural Network for LLR Estimation in MATLAB

The Training and Testing a Neural Network for LLR Estimation example shows how to generate signals and channel impairments to train a neural network, called LLRNet, to estimate exact log

likelihood ratio (LLR). LLRNet is a neural network to estimate the exact LLR values given the baseband complex received symbol for a given signal to noise ratio (SNR) value. A shallow network with a small number of hidden layers has the potential to estimate the exact LLR values at a complexity similar to the approximate LLR algorithm.

RF Component Modeling

New RF component modeling examples show:

- **Power Amplifier Characterization**

The Power Amplifier Characterization example shows how to characterize a power amplifier (PA) using measured input and output signals of an NXP Airfast PA. Optionally, you can use a hardware test setup including an NI PXI chassis with a vector signal transceiver (VST) to measure the signals at run time.

- **Simulation and Verification of Power Amplifier Backoff**

The Simulate and Verify Power Amplifier Backoff example shows how to use backoff to scale a signal prior to inputting it to a table-based power amplifier. It also shows how to examine the power distribution of the signal input to the amplifier, and to verify that the actual behavior of the amplifier matches the specification.

- **RF Transceiver Impairment Modeling in Simulink**

The Impact of RF Effects on Communication System Performance example shows how to model thermal noise, phase noise, and nonlinearity impairments of an RF transceiver. The model measures the effects of the impairments on the bit error rate (BER) of a communications system.

- **Effect of a High-Power Interferer on ADC Performance**

The Effect of a High-Power Interferer on ADC Performance example shows the effect of a high-power in-band or out-of-band interferer on the performance of a communications system with an analog-to-digital converter (ADC).

MATLAB Coder support added to comm.Ray

The `comm.Ray` object now supports MATLAB Coder.

Functionality being removed or changed

comm.EyeDiagram System object will be removed

Still runs

The `comm.EyeDiagram System` object will be removed in a future release. To display the eye diagram of a signal, use the `eyediagram` function instead.

hank2sys function will be removed

Still runs

The `hank2sys` function will be removed in a future release.

hilbiir function will be removed

Still runs

The `hilbiir` function will be removed in a future release. To design a Hilbert transform filter, use the `fdesign.hilbert` object.

crc.generator object will be removed

Warns

The `crc.generator` object will be removed in a future release. To generate cyclic redundancy check (CRC) code bits and append them to input data, use the `comm.CRCGenerator System` object.

Replace instances of `crc.generator` objects with a `comm.CRCGenerator System` object. This table shows the mapping between `crc.generator` properties and `comm.CRCGenerator` properties.

crc.generator Object Properties	comm.CRCGenerator System Object Properties
Polynomial	Polynomial
InitialState	InitialConditions
ReflectInput	ReflectInputBytes
ReflectRemainder	ReflectChecksums
FinalXOR	FinalXOR

This table shows some typical usages of `crc.generator` and how to update your code to use `comm.CRCGenerator` instead.

Not Recommended	Recommended
<pre>data = randi([0 1],100,1); old0 = crc.generator encOld = generate(old0,data); old0 = Type: CRC Generator Polynomial: 0x1021 InitialState: 0xFFFF ReflectInput: false ReflectRemainder: false FinalXOR: 0x0000</pre>	<pre>data = randi([0 1],100,1); sys0 = comm.CRCGenerator('InitialConditions',1) encNew = sys0(data); sys0 = comm.CRCGenerator with properties: Polynomial: 'z^16 + z^12 + z^5 + 1' InitialConditions: 1 DirectMethod: false ReflectInputBytes: false ReflectChecksums: false FinalXOR: 0 ChecksumsPerFrame: 1</pre>

<pre>data = randi([0 1],96,1); old0 = crc.generator('Polynomial','0xF','InitialState',0,'ReflectInput',true,'FinalXOR','0x0'); encOld = generate(old0,data); Type: CRC Generator Polynomial: 0xF InitialState: 0xF ReflectInput: true ReflectRemainder: false FinalXOR: 0x0</pre>	<pre>data = randi([0 1],96,1); sys0 = comm.CRCGenerator('Polynomial',... '0x1F','InitialConditions',[1 1 1 1],... 'ReflectInputBytes',true,'FinalXOR',0); encNew = sys0(data); sys0 = comm.CRCGenerator with properties: Polynomial: '0x1F' InitialConditions: [1 1 1 1] DirectMethod: false ReflectInputBytes: true ReflectChecksums: false FinalXOR: 0 ChecksumsPerFrame: 1</pre>
<pre>data = randi([0 1],100,5); old0 = crc.generator([1 1 1 1 1]) encOld = generate(old0,data); old0 = Type: CRC Generator Polynomial: 0xF InitialState: 0x0 ReflectInput: false ReflectRemainder: false FinalXOR: 0x0</pre>	<pre>data = randi([0 1],100,5); sys0 = comm.CRCGenerator('Polynomial',[1 1 1 1 1],... 'ChecksumsPerFrame',5); encNew = sys0(data(:)); sys0 = comm.CRCGenerator with properties: Polynomial: [1 1 1 1 1] InitialConditions: 0 DirectMethod: false ReflectInputBytes: false ReflectChecksums: false FinalXOR: 0 ChecksumsPerFrame: 5</pre>

crc.detector object will be removed

Warns

The `crc.detector` object will be removed in a future release. To detect errors using the cyclic redundancy check (CRC) code bits in the input data, use the `comm.CRCDetector` System object instead.

Replace instances of `crc.detector` objects with a `comm.CRCDetector` System object. This table shows the mapping between `crc.detector` properties and `comm.CRCDetector` properties.

crc.detector Object Properties	comm.CRCDetector System Object Properties
Polynomial	Polynomial
InitialState	InitialConditions
ReflectInput	ReflectInputBytes
ReflectRemainder	ReflectChecksums

crc.detector Object Properties	comm.CRCDetector System Object Properties
FinalXOR	FinalXOR

This table shows some typical usages of `crc.detector` and how to update your code to use `comm.CRCDetector` instead.

Not Recommended	Recommended
<pre>old0 = crc.generator; old0det = crc.detector data = randi([0 1],100,1); enc_old = generate(old0,data); [dec_old, err_old] = detect(old0det,enc_old); old0det = Type: CRC Detector Polynomial: 0x1021 InitialState: 0xFFFF ReflectInput: false ReflectRemainder: false FinalXOR: 0x0000</pre>	<pre>sys0 = comm.CRCDetector('InitialConditions',1); sys0det = comm.CRCDetector('InitialConditions',1) data = randi([0 1],100,1); enc_new = sys0(data); [dec_new, err] = sys0det(enc_new); sys0det = comm.CRCDetector with properties: Polynomial: 'z^16 + z^12 + z^5 + 1' InitialConditions: 1 DirectMethod: false ReflectInputBytes: false ReflectChecksums: false FinalXOR: 0 ChecksumsPerFrame: 1</pre>
<pre>old0 = crc.generator([1 1 1 1 1]); old0det = crc.detector([1 1 1 1 1]) data = randi([0 1],100,1); enc_old = generate(old0,data); [dec_old, err_old] = detect(old0det,enc_old); old0det = Type: CRC Detector Polynomial: 0xF InitialState: 0x0 ReflectInput: false ReflectRemainder: false FinalXOR: 0x0</pre>	<pre>sys0 = comm.CRCDetector('Polynomial',[1 1 1 1 1]); sys0det = comm.CRCDetector('Polynomial',[1 1 1 1 1]) data = randi([0 1],100,1); enc_new = sys0(data); [dec_new, err] = sys0det(enc_new); sys0det = comm.CRCDetector with properties: Polynomial: [1 1 1 1 1] InitialConditions: 0 DirectMethod: false ReflectInputBytes: false ReflectChecksums: false FinalXOR: 0 ChecksumsPerFrame: 1</pre>

comm.BitToInteger System object will be removed

Warns

The `comm.BitToInteger` System object will be removed in a future release. Instead, use the `bi2de` function.

Data types supported by the `comm.BitToInteger` System object are not inherently supported by the `bi2de` function. For code samples that show how to convert various data types when using these functions, see the Compatibility Considerations section of the `comm.BitToInteger` System object reference page.

comm.IntegerToBit System object will be removed

Warns

The `comm.IntegerToBit` System object will be removed in a future release. Use the `de2bi` function instead.

Data types supported by the `comm.IntegerToBit` System object are not inherently supported by the `de2bi` function. For code samples that show how to convert various data types when using these functions, see the Compatibility Considerations section of the `comm.IntegerToBit` System object reference page.

commsrc.pn object will be removed

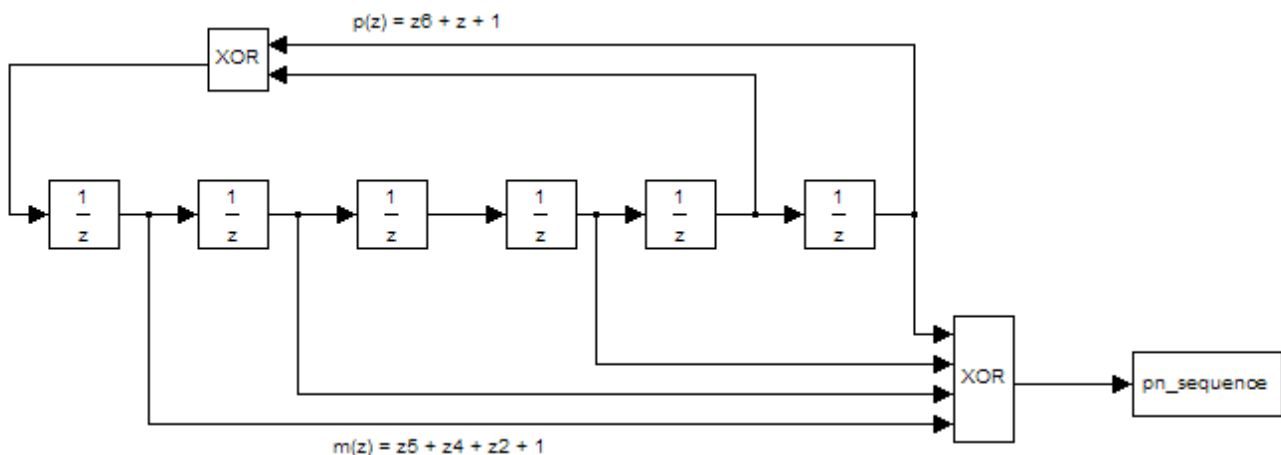
Still runs

The `commsrc.pn` object will be removed in a future release. To generate a pseudo-noise (PN) sequence, use the `comm.PNSequence` System object.

Replace instances of `commsrc.pn` object with a `comm.PNSequence` System object. This table shows the mapping between `commsrc.pn` properties and `comm.PNSequence` properties.

commsrc.pn Object Properties	comm.PNSequence System Object Properties
GenPoly	Polynomial
InitialStates	InitialConditions
Mask	Mask
NumBitsOut	SamplesPerFrame
CurrentStates	Not Applicable

For example, consider this PN sequence generator with a generator polynomial $p(z) = z^6 + z + 1$.



This table shows some typical usages of `commsrc.pn` and how to update your code to use `comm.PNSequence` instead.

Not Recommended	Recommended
------------------------	--------------------

<p>Define the PN sequence generator.</p> <pre>h1 = commsrc.pn('GenPoly', ... [1 0 0 0 0 1 1], 'Mask', [1 1 0 1 0 1]); h2 = commsrc.pn('GenPoly', ... [1 0 0 0 0 1 1], 'Shift', 22); mask2shift ([1 0 0 0 0 1 1], [1 1 0 1 0 1])</pre> <p>Alternatively, input GenPoly as the exponents of z for the nonzero terms of the polynomial in descending order of powers.</p> <pre>h = commsrc.pn('GenPoly', [6 1 0], 'Mask', ... [1 1 0 1 0 1])</pre> <pre>h = GenPoly: [1 0 0 0 0 1 1] InitialStates: [0 0 0 0 0 1] CurrentStates: [0 0 0 0 0 1] Mask: [1 1 0 1 0 1] NumBitsOut: 1</pre>	<p>Define the PN sequence generator.</p> <pre>h1 = comm.PNSequence('Polynomial', ... [1 0 0 0 0 1 1], 'InitialConditions', ... [1 1 0 1 0 1]); h2 = comm.PNSequence('Polynomial', ... [1 0 0 0 0 1 1], 'Mask', 22); mask2shift ([1 0 0 0 0 1 1], [1 1 0 1 0 1])</pre> <pre>ans = 22</pre> <p>Alternatively, input the polynomial exponents of z for the nonzero terms of the polynomial in descending order of powers.</p> <pre>h = comm.PNSequence('Polynomial', [6 1 0], ... 'InitialConditions', [1 1 0 1 0 1])</pre> <pre>h = comm.PNSequence with properties: Polynomial: [6 1 0] InitialConditionsSource: 'Property' InitialConditions: [1 1 0 1 0 1] MaskSource: 'Property' Mask: 0 VariableSizeOutput: false SamplesPerFrame: 1 ResetInputPort: false BitPackedOutput: false OutputDataType: 'double'</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

comm.RectangularQAMModulator and comm.RectangularQAMDemodulator System objects will be removed

Warns

The `comm.RectangularQAMModulator` and `comm.RectangularQAMDemodulator` System Objects will be removed in a future release. Instead, use the `qammod` and `qamdemod` functions.

The `comm.RectangularQAMModulator` and `comm.RectangularQAMDemodulator` System objects support constellation normalization by `PeakPower` and nonunity `AveragePower`. The `qammod` and `qamdemod` functions do not inherently support constellation normalization. For code samples that achieve such constellation normalization when using the `qammod` and `qamdemod` functions, see the [Compatibility Considerations](#) section of the `comm.RectangularQAMModulator` and the [Compatibility Considerations](#) section of the `comm.RectangularQAMDemodulator` reference pages respectively.

stdchan(ts,fd,chanType) syntax has been removed

Errors

The `stdchan(ts,fd,chanType)` syntax has been removed. To create a channel System object from a set of standardized channel models, use the `stdchan(chanType,rs,fd)` syntax.

Compatibility considerations for the `stdchan` function includes addition of a function syntax, removal of a function syntax, and removal of configuration support for several channel models.

- The syntax `chan = stdchan(ts, fd, chantype)` has been removed. A System object is returned using the new syntax. See this new syntax in `stdchan`.
- `stdchan` has removed support for configuration of several channel models by supported standards and associated syntax, compatibility considerations are indicated here:

Standard	Removed Syntax	New Syntax to Return System Object	Notes
3GPP, CDMA	<code>stdchan(ts, fd, '3gppXXX')</code>	<code>stdchan('cdmaXXX', rs, fd)</code>	The prefix of the input argument changed from '3gpp' to 'cdma'. <code>ts</code> and <code>rs</code> are reciprocal values.
GSM	<code>stdchan(ts, fd, 'gsmXXX')</code>	<code>stdchan('gsmXXX', rs, fd)</code>	<code>ts</code> and <code>rs</code> are reciprocal values.
ITU-R HF	<code>stdchan(ts, fd, 'iturHFXXX')</code>	<code>stdchan('iturHFXXX', rs, fd)</code>	<code>ts</code> and <code>rs</code> are reciprocal values.
COST207	<code>stdchan(ts, fd, 'cost207XXX')</code>	Not applicable	<code>stdchan</code> no longer configures these channels. Use the <code>comm.RayleighChannel</code> or <code>comm.RicianChannel</code> System object to configure the channel models for COST207, ITU-R 3G, JTC, HIPERLAN/2, and 802.11a/b/g standards. For guidance on mapping parameters, see “Rayleigh Channel Compatibility Considerations” on page 10-12 and “Rician Channel Compatibility Considerations” on page 10-14.
ITU-R 3G	<code>stdchan(ts, fd, 'itur3GXXX')</code>	Not applicable	
JTC	<code>stdchan(ts, fd, 'jtcXXX')</code>	Not applicable	
HIPERLAN/2	<code>stdchan(ts, fd, 'hiperlan2XXX')</code>	Not applicable	
802.11a/b/g	<code>stdchan(ts, fd, '802.11X')</code>	Not applicable	

comm.BinarySymmetricChannel System object has been removed

Errors

The `comm.BinarySymmetricChannel` System object has been removed. Instead, use the `bsc` function.

This table shows some typical usages of `comm.BinarySymmetricChannel` and how to update your code to use `bsc` instead.

Not Recommended	Recommended
<p>Using a <code>comm.BinarySymmetricChannel</code> System object.</p> <pre data-bbox="238 384 776 531">% generate binary data data = randi([0 1],1000,2); % Default probability is 0.05 bscObj = comm.BinarySymmetricChannel; outOld = bscObj(data);</pre>	<p>Using the <code>bsc</code> function.</p> <pre data-bbox="857 352 1252 436">% generate binary data data = randi([0 1],1000,2); outNew = bsc(data,0.05);</pre>
<p>Simulate a BSC and set the error probability to 0.3.</p> <pre data-bbox="238 625 833 804">% generate binary data data = randi([0 1],1000,2); p = 0.3; % probability bscObj = comm.BinarySymmetricChannel(... 'ErrorProbability',p); outOld = bscObj(data);</pre>	<p>Simulate a BSC and set the error probability to 0.3.</p> <pre data-bbox="857 625 1252 741">% generate binary data data = randi([0 1],1000,2); p = 0.3; % probability outNew = bsc(data,p);</pre>
<p>Simulate a BSC and set the <code>ErrorVectorOutputPort</code> property to <code>true</code> to output the error vector.</p> <pre data-bbox="238 930 833 1077">% generate binary data data = randi([0 1],1000,2); bscObj = comm.BinarySymmetricChannel(... 'ErrorVectorOutputPort',true); [outOld,errOld] = bscObj(data);</pre>	<p>Simulate a BSC with an object and output the error vector.</p> <pre data-bbox="857 898 1338 982">% generate binary data data = randi([0 1],1000,2); [outNew,errNew] = bsc(data,0.05);</pre>
<p>Simulate BSC and set the <code>OutputDataType</code> property to <code>logical</code>.</p> <pre data-bbox="238 1171 833 1287">data = randi([0 1],1000,2); bscObj = comm.BinarySymmetricChannel(... 'OutputDataType','logical'); outOld = bscObj(data);</pre>	<p>Simulate BSC and assign the output to <code>logical</code> data type.</p> <pre data-bbox="857 1171 1338 1255">% generate binary data data = randi([0 1],1000,2); outNew = logical(bsc(data,0.05));</pre>
<p>Simulate a BSC using an input that is <code>logical</code> data.</p> <pre data-bbox="238 1381 776 1497">% generate binary data data = logical(randi([0 1],1000,2)); bscObj = comm.BinarySymmetricChannel; outOld = bscObj(data);</pre>	<p>Simulate a BSC using an input that is <code>logical</code> data.</p> <pre data-bbox="857 1381 1382 1476">% generate binary data data = logical(randi([0 1],1000,2)); outNew = bsc(double(data),0.05);</pre>
<p>Simulate BSC using an input that is integer, 'int8' data. Integer data type options are <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>.</p> <pre data-bbox="238 1633 776 1776">% generate binary data data = randi([0 1],1000,2,'int8'); % Default probability is 0.05 bscObj = comm.BinarySymmetricChannel; outOld = bscObj(int8(data));</pre>	<p>Simulate BSC using an input that is integer, 'int8' data. Integer data type options are <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>.</p> <pre data-bbox="857 1633 1354 1717">% generate binary data data = randi([0 1],1000,2,'int8'); outNew = bsc(double(data),0.05);</pre>

rayleighchan function has been removed

Errors

The `rayleighchan` function has been removed. Instead use the `comm.RayleighChannel` System object.

Replace instances of `rayleighchan` with a `comm.RayleighChannel` System object. This table shows the mapping between function parameters and object properties.

rayleighchan		comm.RayleighChannel		Notes
Properties	Associated Functions	Properties	Object Functions	
ChannelType	Not applicable	Not applicable	Not applicable	Indicated by System object name
InputSamplePeriod	Not applicable	SampleRate	Not applicable	Value is reciprocal
DopplerSpectrum	Not applicable	DopplerSpectrum	Not applicable	Not applicable
MaxDopplerShift	Not applicable	MaximumDopplerShift	Not applicable	Not applicable
PathDelays	Not applicable	PathDelays	Not applicable	Not applicable
AvgPathGaindB	Not applicable	AveragePathGains	Not applicable	Not applicable
NormalizePathGains	Not applicable	NormalizePathGains	Not applicable	Not applicable
StoreHistory	Not applicable	Not applicable	Not applicable	Adjacent with the plot function of <code>rayleighchan</code>
PathGains	Not applicable	Not applicable	Returned when object is called (<code>step</code>)	Second output returned by a call of the object (<code>step</code>)
ChannelFilterDelay	Not applicable	Not applicable	Returned by the object function <code>info</code>	Returned by a call to <code>info</code>
ResetBeforeFiltering	Not applicable	Not applicable	Use the object function <code>reset</code>	Call <code>reset</code> before each object function call (<code>step</code>)
NumSamplesProcessed	Not applicable	Not applicable	Returned by the object function <code>info</code>	Returned by a call to <code>info</code>
StorePathGains	Not applicable	PathGainsOutputPort	Not applicable	Not applicable
Not applicable	<code>filter</code>	Not applicable	Returned when object is called (<code>step</code>)	First output returned by a call of the object (<code>step</code>)

Not applicable	reset	Not applicable	Use the object function reset	Use the object function reset
Not applicable	plot	Visualization SamplesToDisplay PathsForDopplerDisplay	object call (step)	The Visualization property setting controls display of graphics during an object call (step).

ricianchan function has been removed

Errors

The ricianchan function has been removed. Instead, use the comm.RicianChannel System object.

Replace instances of ricianchan with a comm.RicianChannel System object. This table shows the mapping between function parameters and object properties.

ricianchan		comm.RicianChannel		Notes
Properties	Associated Functions	Properties	Object Functions	
ChannelType	Not applicable	Not applicable	Not applicable	Indicated by System object name
InputSamplePeriod	Not applicable	SampleRate	Not applicable	Value is reciprocal
DopplerSpectrum	Not applicable	DopplerSpectrum	Not applicable	Not applicable
MaxDopplerShift	Not applicable	MaximumDopplerShift	Not applicable	Not applicable
PathDelays	Not applicable	PathDelays	Not applicable	Not applicable
KFactor	Not applicable	KFactor	Not applicable	Not applicable
DirectPathDopplerShift	Not applicable	DirectPathDopplerShift	Not applicable	Not applicable
DirectPathInitPhase	Not applicable	DirectPathInitialPhase	Not applicable	Not applicable
AvgPathGaindB	Not applicable	AveragePathGains	Not applicable	Not applicable
NormalizePathGains	Not applicable	NormalizePathGains	Not applicable	Not applicable
StoreHistory	Not applicable	Not applicable	Not applicable	Adjacent with the plot function of ricianchan
StorePathGains	Not applicable	PathGainsOutputPort	Not applicable	Not applicable

PathGains	Not applicable	Not applicable	Returned when object is called (step)	Second output returned by a call of the object (step)
ChannelFilterDelay	Not applicable	Not applicable	info	Returned by a call to info
ResetBeforeFiltering	Not applicable	Not applicable	Use the object function reset	Call reset before each object function call (step)
NumSamplesProcessed	Not applicable	Not applicable	Returned by the object function info	Returned by a call to info
Not applicable	filter	Not applicable	Returned when object is called (step)	First output returned by a call of the object (step)
Not applicable	reset	Not applicable	Use the object function reset	Use the object function reset
Not applicable	plot	Visualization SamplesToDisplay PathsForDopplerDisplay	Returned when object is called (step)	The Visualization property setting controls display of graphics during an object call (step).

legacychannelsim function has been removed

Errors

The legacychannelsim function has been removed. Instead use the comm.RayleighChannel or comm.RicianChannel System object.

- For legacychannelsim(true) and a Rayleigh channel, you can achieve equivalent functionality to rayleighchan using a comm.RayleighChannel System object, configured as shown in this code sample:

```
% Set the RNG seed value to 10 and generate a vector
% of random complex numbers.
seed = 10;
x = complex(rand(100,1),rand(100,1));
%%
% Create a Rayleigh channel System object. Set RandomStream to
% 'Global stream', and set the random number generator to 'v5normal'.
chan = comm.RayleighChannel('RandomStream','Global stream');
rng(seed,'v5normal');
chan(x);
```

Similarly for a Rician channel, equivalent functionality to ricianchan can be achieved with the above code using a comm.RicianChannel System object.

- For `legacychannelsim(false)` and a Rayleigh channel, you can achieve equivalent functionality to `rayleighchan` using a `comm.RayleighChannel` System object, configured as shown in this code sample:

```
% Set the RNG seed value to 10 and generate a vector
% of random complex numbers.
seed = 10;
x = complex(rand(100,1),rand(100,1));
%%
% Option 1 - Create a Rayleigh channel System object, set
% RandomStream to 'Global stream', and set the random
% number generator to 'twister'.
chan = comm.RayleighChannel('RandomStream','Global stream');
rng(seed, 'twister');
chan(x);
%%
% Option 2 - Create a Rayleigh channel System object setting
% RandomStream to 'mt19937ar with seed'.
chan = comm.RayleighChannel('RandomStream', ...
    'mt19937ar with seed', 'Seed', seed);
chan(x);
```

Similarly for a Rician channel, equivalent functionality to `ricianchan` can be achieved with the above code using a `comm.RicianChannel` System object.

doppler.jakes object has been removed

Errors

The `doppler.jakes` object has been removed. To create a Doppler spectrum structure with Jakes profile, use the `doppler('Jakes')` function syntax.

doppler.flat object has been removed

Errors

The `doppler.flat` object has been removed. To create a Doppler spectrum structure with a flat profile, use the `doppler('Flat')` function syntax.

doppler.bell object has been removed

Errors

The `doppler.bell` object has been removed. To create a Doppler spectrum structure with a bell profile, use the `doppler('Bell')` function syntax.

doppler.rounded object has been removed

Errors

The `doppler.rounded` object has been removed. To create a Doppler spectrum structure with a rounded profile, use the `doppler('Rounded')` function syntax.

doppler.rjakes object has been removed

Errors

The `doppler.rjakes` object has been removed. To create a Doppler spectrum structure with a restricted Jakes profile, use the `doppler('Restricted Jakes')` function syntax.

doppler.ajakes object has been removed

Errors

The `doppler.ajakes` object has been removed. To create a Doppler spectrum structure with an asymmetric Jakes profile, use the `doppler('Asymmetric Jakes')` function syntax.

doppler.gaussian object has been removed*Errors*

The `doppler.gaussian` object has been removed. To create a Doppler spectrum structure with a Gaussian profile, use the `doppler('Gaussian')` function syntax.

doppler.bigaussian object has been removed*Errors*

The `doppler.bigaussian` object has been removed. To create a Doppler spectrum structure with a biGaussian profile, use the `doppler('BiGaussian')` function syntax.

R2020a

Version: 7.3

New Features

Compatibility Considerations

Wireless Waveform Generator app adds 5G Fixed Reference Channel waveform generation support

Using the **Wireless Waveform Generator** app, you can now:

- Create waveforms compliant with specific 5G uplink and downlink fixed reference channel (FRC) configurations. Use of the 5G feature in the **Wireless Waveform Generator** app requires 5G Toolbox.

RF fingerprinting with deep learning examples

The `Design a Deep Neural Network with Simulated Data to Detect WLAN Router Impersonation` example designs a radio frequency (RF) fingerprinting convolutional neural network (CNN) with simulated data by using Communications Toolbox and Deep Learning Toolbox features. Deep Learning Toolbox features are used to construct, train, and use a CNN. Simulated WLAN beacon frames from known and unknown routers train the CNN and are used by the CNN for RF fingerprinting.

The `Test a Deep Neural Network with Captured Data to Detect WLAN Router Impersonation` example trains a radio frequency (RF) fingerprinting convolutional neural network (CNN) with captured data by using Communications Toolbox and Deep Learning Toolbox features. Deep Learning Toolbox features are used to construct, train, and use a CNN. Signals captured using ADALM-Pluto SDRs and Communications Toolbox Support Package for Analog Devices® ADALM-Pluto Radio train the CNN and are used by the CNN for RF fingerprinting.

Bluetooth LE examples and BR/EDR PHY features and examples

Generate and decode Bluetooth basic rate (BR) and extended data rate (EDR) PHY waveforms by using the `bluetoothWaveformGenerator` and `bluetoothIdealReceiver` functions, respectively. Use `bluetoothPhyConfig` and `bluetoothWaveformConfig` configuration objects to parameterize these functions.

These added examples use BR/EDR features.

- End-to-End Bluetooth BR/EDR PHY Simulations with RF Impairments and Corrections — Generate, demodulate, and decode Bluetooth waveforms using the practical receiver and compute BER and PER.
- Bluetooth BR/EDR Waveform Generation and Transmission using SDR — Generate Bluetooth waveforms and transmit them over the air using the ADALM-PLUTO radio or write to a baseband file (*.bb).
- Bluetooth BR/EDR Waveform Reception by Using SDR — Capture and decode Bluetooth waveforms using the ADALM-PLUTO radio or load them from a baseband file (*.bb).

These added examples use BLE features.

- Estimate Packet Delivery Ratio in Bluetooth Mesh Network — Demonstrate multinode Bluetooth mesh network modeling using discrete event simulation (DES) and model the complete Bluetooth mesh stack over the advertising bearer.

To download the add-on, see Communications Toolbox Library for the Bluetooth Protocol. For more information on features and examples in Communications Toolbox Library for the Bluetooth Protocol, see Bluetooth.

Site Viewer, RF propagation and ray tracing enhancements

These features enable you to:

- Use the `siteviewer` object to import and visualize measurements data in a Site Viewer map.
- Configure a propagation model object for ray tracing by using the updated `propagationModel` and `pathloss` functions.
- Predict the total received power and generate coverage maps with ray tracing by using the updated `raytrace`, `coverage`, `sinr`, `sigstrength`, and `link` functions.
- Programmatically access ray tracing analysis information generated by using the `raytrace` function and `comm.Ray` object.
- Include reflection materials and antenna polarization in path loss calculations by using the `raytrace` and `raypl` functions.
- Calculate permittivity and conductivity of building materials and earth surfaces based on ITU recommendations by using the `buildingMaterialPermittivity` and `earthSurfacePermittivity` functions, respectively.

Amplifier block to model a memoryless nonlinear amplifier using one of four methods

Use the Amplifier block from the Idealized Baseband library to add amplifier noise and to model an amplifier using the Cubic polynomial, AM/AM-AM/PM, Modified Rapp, or Saleh model. To visualize the AM/AM and AM/PM nonlinear characteristics of the amplifier, plot the power characteristics.

Interference Modeling in Simulink Example

The Interference Modeling example generates, combines, and visualizes a QPSK modulated signal of interest and a GMSK modulated interference signal.

Multi-Band Signal Generation Example

The Multiband Signal Generation example generates and combines QPSK and GMSK modulated signals to form a multiband signal. Spectrum visualization is included to show the signals before and after combining.

Top Down Design of RF Receiver Example

The Top-Down Design of an RF Receiver example designs an RF receiver for a ZigBee-like application using a top-down methodology. It verifies the BER of an impairment-free design, and analyzes BER performance after the addition of impairment models. The example uses the **RF Budget Analyzer** App to rank the elements contributing to the noise and nonlinearity budget.

Multiuser Block Diagonalization Beamforming

The new `blkdiagbfweights` function extends spatial multiplexing from single-user to multi-user MIMO communication systems. `blkdiagbfweights` computes the precoding and combining weights for multi-user MIMO beamforming using a block diagonalization algorithm.

The Massive MIMO Hybrid Beamforming example uses the `blkdiagbfweights` function to compute precoding weights when applying the joint spatial division multiplexing (JSDM) technique for a multi-user system.

Channel modeling coordinate system transformations

These features enable you to:

- Convert between local and global coordinate systems by using the `global2localcoord` and `local2globalcoord` functions.
- Convert between Cartesian and spherical coordinate systems by using the `cart2sphvec` and `sph2cartvec` functions.

Channel delay computation

The `channelDelay` function computes the channel delay and impulse response magnitude at receive antennas. To find the peak of the channel impulse response, the function reconstructs the impulse response from the channel path gains array and path filter impulse response matrix. The function returns the estimated timing delay in samples and the channel impulse response magnitude.

MATLAB Compiler support added to eye diagram

The `comm.EyeDiagram` System object now supports MATLAB Compiler.

Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Detect NR cell signals off the air

Using a 5G NR cell search implementation partitioned across the ARM[®] and the FPGA fabric, you can detect NR cell signals off the air. For more information, see [5G NR Cell Search Using Analog Devices AD9361/AD9364](#).

Functionality being removed or changed

bin2gray and gray2bin functions will be removed

Warns

`bin2gray` and `gray2bin` will be removed in a future release. Use the appropriate modulation object or function to remap constellation points instead.

Using the workflow that is not recommended, the `bin2gray` and `gray2bin` functions are used to convert binary representation to natural binary or Gray encoding. After the conversion, you must specify 'bin' for the symbol order when calling the modulation and demodulation functions. This workflow led to confusion.

Using the recommended workflow, for any given of modulation scheme, you provide decimal values when calling the modulation and demodulation functions. When calling the modulation and demodulation functions, specify symbol order as 'bin' for natural binary encoding or 'gray' for Gray encoding.

This table shows usages of bin2gray and gray2bin by modulation type and how to update your code.

Modulation Scheme	Not Recommended	Recommended
QAM	<pre>x = randi([0 63],1,100); y = bin2gray(x,'qam',64); z = qammod(y,64,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = qamdemod(x,64,'bin'); z = gray2bin(y,'qam',64);</pre>	<pre>x = randi([0 63],1,100); z = qammod(x,64,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = qamdemod(x,64,'gray')</pre>
PAM	<pre>x = randi([0 63],1,100); y = gray2bin(x,'pam',64); z = pammod(y,64,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = pamdemod(x,64,pi/4,'bin'); z = bin2gray(y,'pam',64);</pre>	<pre>x = randi([0 63],1,100); z = pammod(x,64,pi/4,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = pamdemod(x,64,pi/4,'gray')</pre>
FSK	<pre>x = randi([0 63],1,100); y = gray2bin(x,'fsk',64); z = fskmod(y,64,1,256,256,'cont','bin'); x = 2*(randn(512,1)+1j*randn(512,1)); y = fskdemod(x,64,1,256,256,'bin'); z = bin2gray(y,'fsk',64)</pre>	<pre>x = randi([0 63],1,100); z = fskmod(x,64,1,256,256,'cont','gray'); x = 2*(randn(512,1)+1j*randn(512,1)); z = fskdemod(x,64,1,256,256,'gray')</pre>
DPSK	<pre>x = randi([0 63],1,100); y = gray2bin(x,'dpsk',64); z = dpskmod(y,64,pi/4,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = dpskdemod(x,64,pi/4,'bin'); z = bin2gray(y,'dpsk',64);</pre>	<pre>x = randi([0 63],1,100); z = dpskmod(x,64,pi/4,'gray'); x=2*(randn(100,1)+1j*randn(100,1)); z = dpskdemod(x,64,pi/4,'gray');</pre>
PSK	<pre>x=randi([0 63],1,100); y=gray2bin(x,'psk',64); z=pskmod(y,64,0,'bin'); x = 2*(randn(100,1)+1j*randn(100,1)); y = pskdemod(x,64,0,'bin'); z = bin2gray(y,'psk',64);</pre>	<pre>x=randi([0 63],1,100); z=pskmod(x,64,0,'gray'); x = 2*(randn(100,1)+1j*randn(100,1)); z = pskdemod(x,64,0,'gray');</pre>

vec2mat function is not recommended

Still runs

vec2mat is not recommended. Use reshape instead.

To convert a vector to a matrix, use the reshape function. This example shows how to add padding, as needed, when converting a vector to matrix.

Not Recommended	Recommended
<p>Create one vector to convert to a matrix and one vector of padding values.</p> <pre>vec = [10;20;30;40;50]; padding = [1,2;3,4;5,6]; n = 4;</pre> <p>Use <code>vec2mat</code> to convert the vector to a matrix. The function automatically adds padding as needed.</p> <pre>[mat4,numPadded4] = vec2mat(vec,n,padding)</pre> <pre>mat4 = 10 20 30 40 50 1 3 5 numPadded4 = 3</pre>	<p>Create one vector to convert to a matrix and one vector of padding values.</p> <pre>vec = [10;20;30;40;50]; padding = [1,2;3,4;5,6]; n = 4;</pre> <p>Compute and append the needed padding to the vector, and then convert the vector to a matrix using <code>reshape</code>.</p> <pre>numPadded = mod(numel(vec),n); if numPadded > 0 numPadded = n - numPadded mat = reshape([vec.' padding(1:numPadded)], n, []).'</pre> <pre>else numPadded % No padding required mat = reshape(vec.', n, []).'</pre> <pre>end</pre> <pre>numPadded = 3 mat = 10 20 30 40 50 1 3 5</pre>

comm.PSKCoarseFrequencyEstimator System object will be removed

Warns

`comm.PSKCoarseFrequencyEstimator` will be removed in a future release. Use `comm.CoarseFrequencyCompensator` instead.

To estimate and compensate for frequency offsets in PSK signals, use the `comm.CoarseFrequencyCompensator` System object.

comm.QAMCoarseFrequencyEstimator System object will be removed

Warns

`comm.QAMCoarseFrequencyEstimator` will be removed in a future release. Use `comm.CoarseFrequencyCompensator` instead.

To estimate and compensate for frequency offsets in QAM signals, use the `comm.CoarseFrequencyCompensator` System object.

commscope.eyediagram object and EyeScope function have been removed

Errors

`commscope.eyediagram` and `EyeScope` have been removed. Use `comm.EyeDiagram` instead. To update your code, replace all instances of `commscope.eyediagram` and `eyescope` with `comm.EyeDiagram`.

For more details, see Eye Diagram Measurements.

comm.CPMCarrierPhaseSynchronizer System object has been removed

Errors

`comm.CPMCarrierPhaseSynchronizer` has been removed. Use `comm.CarrierSynchronizer` instead.

To synchronize MSK signals, use the `comm.CarrierSynchronizer` System object.

CPM Phase Recovery block has been removed

CPM Phase Recovery has been removed. Use Carrier Synchronizer instead.

To synchronize MSK signals, use the Carrier Synchronizer block.

M-PSK Phase Recovery block has been removed

M-PSK Phase Recovery has been removed. Use Carrier Synchronizer instead.

To synchronize M-PSK signals, use the Carrier Synchronizer block.

Insert Zero block will be removed

Still runs

Insert Zero will be removed in a future release. Use MATLAB code in a MATLAB Function block instead.

To insert zeros into a random number stream, use MATLAB code in a MATLAB Function block.

For example, to insert zeros into a data stream, you can use this code in a MATLAB Function block.

```
function y = fcn(u,insertZeroVector)
    numSeg = length(u)/sum(insertZeroVector);
    c = zeros(length(insertZeroVector), numSeg, 'like', u);
    c(logical(insertZeroVector), :) = reshape(u, [], numSeg);
    y = c(:);
end
```

For the Insert Zero, the input length must be an integer multiple of the number of ones in the **Insert zero vector** parameter. This same restriction applies for the input to the MATLAB code in a MATLAB Function block.

For an example using this code, see Insert Zeros into a Random Number Stream.

Gaussian Filter block has been removed

Gaussian Filter has been removed. Use Discrete FIR Filter instead.

Existing models automatically update instances of the Gaussian Filter block to a Discrete FIR Filter block. For more information on block forwarding, see Forwarding Tables (Simulink).

To model a Gaussian filter, use a Discrete FIR Filter block instead. Use the `gaussdesign` function to generate filter coefficients for the Discrete FIR Filter block.

You can specify coefficients for Gaussian filter designs that apply various normalization methods. Assign the desired normalization method by using a Discrete FIR Filter block. The Model Gaussian Filter in Simulink example shows how to use the `gaussdesign` function along with the Discrete FIR Filter block to model a Gaussian filter.

Bernoulli Binary Generator and Random Integer Generator block updates supported in Upgrade Advisor

Behavior change

Starting in R2020a, the Bernoulli Binary Generator and Random Integer Generator blocks now enable you to use the Upgrade Advisor. You can update to the block version introduced in R2015b or keep the block version available in releases before to R2015b.

Use the Upgrade Advisor to update existing models that include the Bernoulli Binary Generator or Random Integer Generator block.

The behavior of the random number generator for the Bernoulli Binary Generator block has changed. The statistics have been improved. For more information, see the “Source blocks output frames of contiguous time samples but do not use frame attribute” topic.

Certain random source generator blocks automatically update

Behavior change

Starting in R2020a, Simulink no longer enables you to use versions of the Poisson Integer Generator, Barker Code Generator, Gold Sequence Generator, Hadamard Code Generator, Kasami Sequence Generator, OVSF Code Generator, PN Sequence Generator, or Walsh Code Generator blocks available in releases before to R2015b.

Existing models automatically update to load the block version introduced in R2015b. For more information on block forwarding, see Forwarding Tables (Simulink).

The behavior of the random number generator for the Poisson Integer Generator block has changed. The statistics have been improved. For more information, see the “Source blocks output frames of contiguous time samples but do not use frame attribute” topic.

Equalizer and adaptalg packages and functions will be removed

Warns

The equalizer and adaptalg packages and functions will be removed in a future release. Use `comm.LinearEqualizer` and `comm.DecisionFeedbackEqualizer` instead.

To update your code, change instances of the package and function names to the corresponding recommended replacement as listed in this table.

Not Recommended	Recommended
<code>equalize</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>lineareq</code> and <code>equalizer.lineareq</code>	<code>comm.LinearEqualizer</code>
<code>dfe</code> and <code>equalizer.dfe</code>	<code>comm.DecisionFeedbackEqualizer</code>
<code>lms</code> and <code>adaptalg.lms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>rls</code> and <code>adaptalg.rls</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>
<code>cma</code> and <code>adaptalg.cma</code>	<code>comm.LinearEqualizer</code>

Not Recommended	Recommended
signlms and adaptalg.signlms	Consider using comm.LinearEqualizer or comm.DecisionFeedbackEqualizer with the adaptive algorithm set to LMS.
normlms and adaptalg.normlms	Consider using comm.LinearEqualizer or comm.DecisionFeedbackEqualizer with the adaptive algorithm set to LMS.
varlms and adaptalg.varlms	Consider using comm.LinearEqualizer or comm.DecisionFeedbackEqualizer with the adaptive algorithm set to LMS.

For more details, see the "Compatibility Considerations" sections on the System object reference page for the applicable recommended replacement.

Certain equalization blocks will be removed

Warns

The equalization blocks listed in the **Not Recommended** column of this table will be removed in a future release. Use the Decision Feedback Equalizer or Linear Equalizer block instead.

To update your model, change instances of the blocks to the corresponding recommended replacement as listed in this table.

Not Recommended	Recommended
CMA Equalizer	Linear Equalizer
LMS Linear Equalizer	
RLS Linear Equalizer	
Normalized LMS Linear Equalizer	Consider using Linear Equalizer with the adaptive algorithm set to LMS.
Sign LMS Linear Equalizer	
Variable Step LMS Linear Equalizer	
LMS Decision Feedback Equalizer	Decision Feedback Equalizer
RLS Decision Feedback Equalizer	
Normalized LMS Decision Feedback Equalizer	Consider using Decision Feedback Equalizer with the adaptive algorithm set to LMS.
Sign LMS Decision Feedback Equalizer	
Variable Step LMS Decision Feedback Equalizer	

For more details, see the "Compatibility Considerations" sections on the block reference page for the applicable recommended replacement.

R2019b

Version: 7.2

New Features

Compatibility Considerations

Wireless waveform generator app updates

Using the **Wireless Waveform Generator** app, you can now:

- Generate Bluetooth Low Energy modulated waveforms. This feature requires Communications Toolbox Library for the Bluetooth Protocol. For information about downloading Add-Ons, see [Get and Manage Add-Ons \(MATLAB\)](#).
- Export a waveform as a MATLAB script. You can run the exported MATLAB script to recreate your waveform outside of the **Wireless Waveform Generator** app.

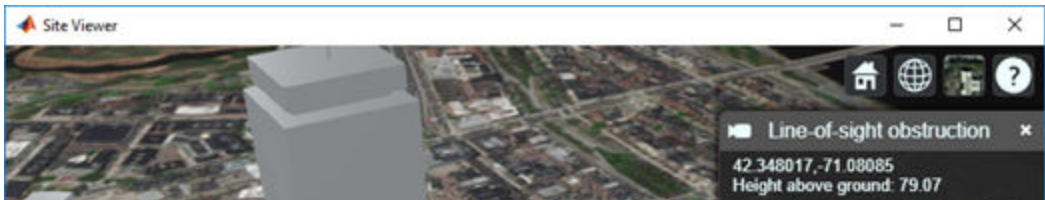
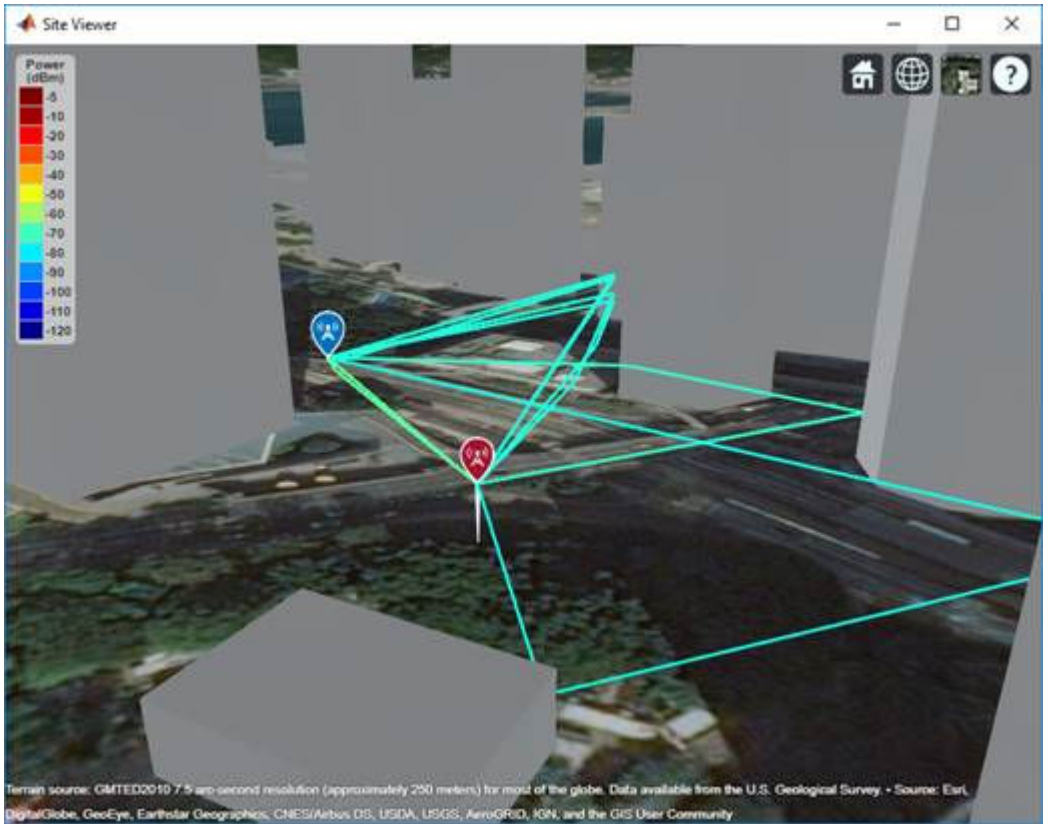
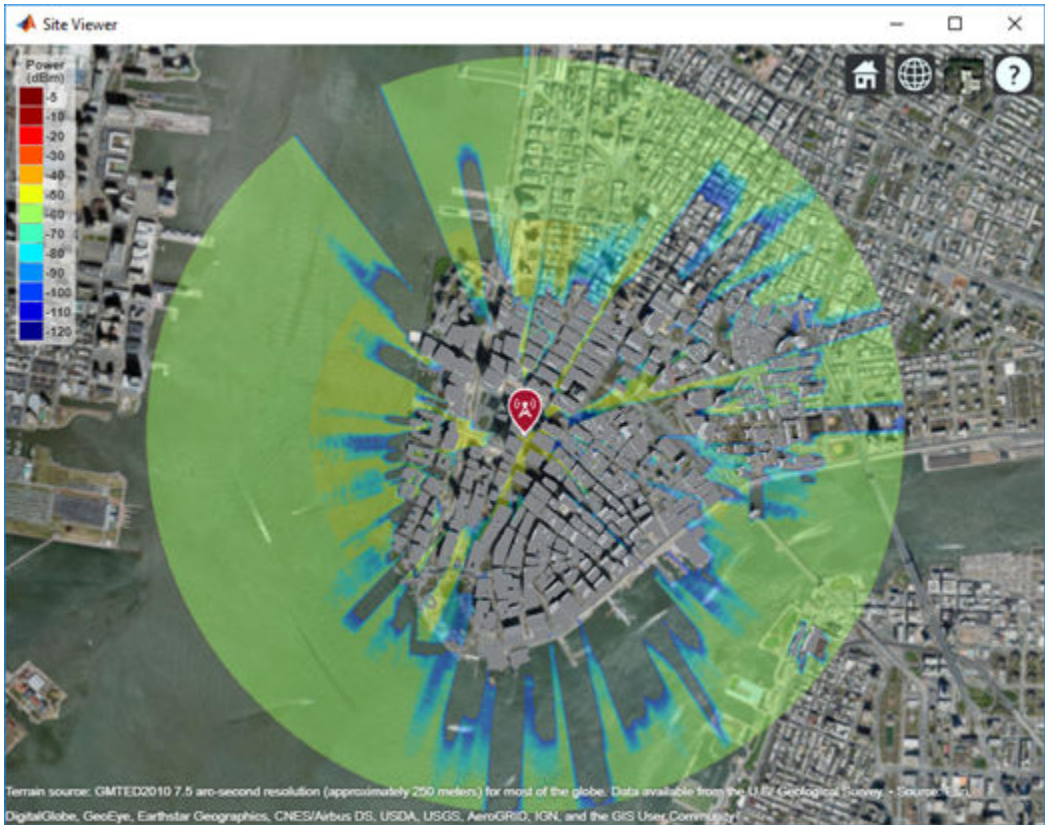
RF propagation tools for ray tracing and visualization

Using the RF propagation tools, you can visualize transmitter sites, receiver sites, buildings, links, ray tracing results and coverage maps on a virtual globe using free-space, terrain, and weather-effects propagation models.

To view the list of new features, see [RF Propagation](#). These features include support for:

- High-quality map visualizations including 3-D virtual globe with terrain
- 3-D building overlays from OpenStreetMap (OSM) files
- 3-D terrain overlays from digital terrain elevation data (DTED) files
- Transmitter and receiver site positioning and visualizations
- Ray tracing computation and visualizations
- Receiver signal strength computation and visualizations
- Coverage map computation and visualizations
- Propagation models including losses due to free-space and weather-effects, and considering urban and terrain settings

The Urban Channel Link Analysis and Visualization using Ray Tracing example uses ray tracing to analyze a communication link in an urban environment. The example uses the `siteviewer` object to import and visualize buildings in Site Viewer and the `raytrace` function to visualize point-to-point ray tracing analysis.



GSM Waveform generation functions

Define GSM-compliant uplink and downlink waveforms by using the `gsmUplinkConfig` and `gsmDownlinkConfig` objects, respectively, to configure the TDMA frame characteristics and the `gsmFrame` function to create a waveform. The `gsmInfo` and `gsmCheckTimeMask` functions provide information about the specified configuration object.

Bluetooth protocol updates and examples

For more information on features and examples in Communications Toolbox Library for the Bluetooth Protocol, see Bluetooth. The examples added in this release are:

- BLE Coexistence Model with WLAN Signal Interference — Simulates Bluetooth low energy coexistence with a WLAN signal interference.
- BLE Blocking, Intermodulation and Carrier to Interference Performance Tests — Models Bluetooth low energy RF-PHY blocking, intermodulation, and carrier to interference (C/I) performance receiver tests according to the Bluetooth RF-PHY Test Specifications.
- BLE Modulation Characteristics, Carrier Frequency Offset and Drift Test Measurements — BLE RF-PHY transmitter tests specific to modulation characteristics, carrier frequency offset, and drift according to the Bluetooth RF-PHY Test Specifications.
- BLE Output Power and In-Band Emissions Test Measurements — BLE transmitter test measurements specific to output power and in-band emissions according to the Bluetooth RF-PHY Test Specifications.
- Bluetooth Mesh Flooding in Wireless Sensor Networks — Demonstrates Bluetooth mesh network flooding. Models only the network layer of the BLE mesh stack.
- Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks — Performs energy profiling of Low Power, battery operated Bluetooth mesh nodes that implement an energy saving feature called *Friendship*. Models BLE mesh stack and Link Layer.

To download the Add-On, see Communications Toolbox Library for the Bluetooth Protocol.

LDPC decoder uses multicore processing


The `comm.LDPCDecoder` System object and LDPC Decoder block now use multicore processing to reduce execution time in simulation mode.

Class definition now available for CPM demodulator System object

You can now view the MATLAB code for the `comm.CPMDemodulator` System object. This visibility into the code enables you to customize its algorithms and functionality.

Playback control behavior changed for scopes in referenced models

When you use a scope block in a referenced model, the playback controls in the scope now match the playback controls of the last model with which you interacted 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 top model, the run button runs the referenced model in isolation.

This change affects the Constellation Diagram, Eye Diagram Scope, and some toolbox-specific scopes. For toolbox-specific scopes, refer to the release notes for that toolbox.

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

Functionality being removed or changed

Code generation for models using EVM Measurement or MER Measurement block has changed

Behavior change

To generate code in a model using the EVM Measurement or MER Measurement block, you must enable **Dynamic Memory Allocation in MATLAB Functions**. For more information, see [Dynamic memory allocation in MATLAB functions \(Simulink\)](#).

Certain interleaver and deinterleaver System objects will be removed

Warns

Starting in R2019b, use of the System objects in this table results in a warning message. These objects will be removed in a future release. This table shows identifies the recommended replacement function for each System object.

Discouraged Usage	Recommended Replacement	Compatibility Considerations
comm.AlgebraicInterleaver	algintrlv	Replace all instances of comm.AlgebraicInterleaver with algintrlv.
comm.AlgebraicDeinterleaver	algdeintrlv	Replace all instances of comm.AlgebraicDeinterleaver with algdeintrlv.
comm.BlockInterleaver	intrlv	Replace all instances of comm.BlockInterleaver with intrlv.
comm.BlockDeinterleaver	deintrlv	Replace all instances of comm.BlockDeinterleaver with deintrlv.
comm.MatrixInterleaver	matintrlv	Replace all instances of comm.MatrixInterleaver with matintrlv.
comm.MatrixDeinterleaver	matdeintrlv	Replace all instances of comm.MatrixDeinterleaver with matdeintrlv.
comm.MatrixHelicalScanInterleaver	helscanintrlv	Replace all instances of comm.MatrixHelicalScanInterleaver with helscanintrlv.
comm.MatrixHelicalScanDeinterleaver	helscandeintrlv	Replace all instances of comm.MatrixHelicalScanDeinterleaver with helscandeintrlv.

For more details, see the **Compatibility Considerations** section on the reference page of the discouraged System object of interest.

randseed function will be removed

Warns

The `randseed` function will be removed in a future release. Use either of these syntaxes for the `rng` function instead: `rng(seed)` or `rng('shuffle')`.

Compatibility Considerations

The functionality provided by `randseed` is no longer necessary for controlling random number generation. Instead, use `rng(seed)`, where `seed` specifies a nonnegative integer seed for the random number generator, or use `rng('shuffle')` to seed the random number generator based on the current time.

Align Signals block will be removed

Warns

The Align Signals block will be removed in a future release. Instead use the Find Delay block to find delays and the Delay block to apply delays.

Compatibility Considerations

For examples, see Align Signals Replacement for Single Rate Signals in Simulink and Align Signals Replacement for Multirate Signals in Simulink.

commtest.ErrorRate and testconsole.Results objects will be removed

Warns

The `commtest.ErrorRate` and `testconsole.Results` objects will be removed in a future release. Instead, use the `comm.ErrorRate` System object or the `bertool` function instead. `bertool` opens the Bit Error Rate Analysis Tool app.

Compatibility Considerations

For alternate swept BER simulation workflows, see Bit Error Rate (BER).

For more information, see the **Compatibility Considerations** section on the `commtest.ErrorRate` and `testconsole.Results` object reference pages.

comm.BinarySymmetricChannel System object will be removed

Warns

The `comm.BinarySymmetricChannel` System object will be removed in a future release. Instead, use the `bsc` function.

Compatibility Considerations

For more information, see the **Compatibility Considerations** section on the `comm.BinarySymmetricChannel` reference page.

comm.BitToInteger and comm.IntegerToBit System objects will be removed

Still runs

The `comm.BitToInteger` and `comm.IntegerToBit` System objects will be removed in a future release. Instead, use the `bi2de` and `de2bi` functions, respectively.

Compatibility Considerations

Data types supported by `comm.BitToInteger` and `comm.IntegerToBit` are not inherently supported by the `bi2de` and `de2bi` functions. For code samples that show scenarios to convert various data types when using these functions, see the **Compatibility Considerations** section of the `comm.BitToInteger` and `comm.IntegerToBit` System object reference pages.

oqpskmod and oqpskdemod functions have been removed

Errors

`oqpskmod` and `oqpskdemod` functions have been removed. Instead, use the `comm.OQPSKModulator` or `comm.OQPSKDemodulator` System objects, respectively.

R2019a

Version: 7.1

New Features

Bug Fixes

Compatibility Considerations

Introducing Communications Toolbox Library for the Bluetooth Protocol

Communications Toolbox Library for the Bluetooth Protocol provides standard-compliant functions and reference examples for the design, modeling, simulation, and testing of Bluetooth communications systems. The library supports protocol layer modeling, network modeling, link-level simulation, golden reference verification and conformance testing, and test waveform generation.

With the library you can configure, simulate, measure, and analyze end-to-end communications links. You can modify or customize the library functions and use them as reference models for implementing Bluetooth systems and devices.

The library provides reference examples to help you model the behavior of Bluetooth devices or a network of Bluetooth devices. Adapt the reference designs provided to explore communication in multi-node networks and study the system performance of your designs. Using the library, you can model and analyze interference from other networks and co-existence with other networks.

All Communications Toolbox Library for the Bluetooth Protocol functions and System objects support ANSI®/ISO® compliant C/C++ code generation.

You can get the Communications Toolbox Library for the Bluetooth Protocol download from Add-On Explorer. For more information, see [Get Add-Ons \(MATLAB\)](#).

Wireless Waveform Generator app updates

Using the **Wireless Waveform Generator** app, you can now:

- Create waveforms compliant with specific LTE uplink and downlink reference measurement channel (RMC) and E-UTRA test model (E-TM) configurations. Use of the LTE feature in the **Wireless Waveform Generator** app requires LTE Toolbox.
- Create WLAN waveforms that are compliant with 802.11ax™ modulation schemes. Use of the WLAN feature in the **Wireless Waveform Generator** app requires WLAN Toolbox .
- Generate a waveform that you can transmit using a connected lab test instrument. The **Wireless Waveform Generator** app can transmit using instruments supported by the `rfsiggen` function. Use of the transmit feature in the **Wireless Waveform Generator** app requires Instrument Control Toolbox.

Digital predistortion estimator and compensator System objects and blocks

To estimate and compensate for distortions introduced by power amplifier, add digital predistortion coefficient estimation and digital predistortion to your simulations.

- In MATLAB, use the `comm.DPDCoefficientEstimator` System object to generate an estimate that you can then use as an input to the `comm.DPD` System object.
- In Simulink, use the DPD Coefficient Estimator block to generate an estimate that you can then use as an input to the DPD block.

These new features are demonstrated in the Power Amplifier Characterization with DPD for Reduced Signal Distortion example.

Modulation classification with deep learning example

The Modulation Classification with Deep Learning example generates a channel impaired waveform by using Communications Toolbox and uses Deep Learning Toolbox to construct, train, and use a convolutional neural network for modulation classification.

Decision feedback and linear equalizer System objects and blocks

The addition of these System objects and blocks streamline your choices for received signal equalization are streamlined. To equalize received signals:

- In MATLAB, use the `comm.DecisionFeedbackEqualizer` or `comm.LinearEqualizer` System object.
- In Simulink, use the Decision Feedback Equalizer or Linear Equalizer block.

Link budget analysis example

The Link Budget Analysis example demonstrates use of an app that calculates uplink and downlink link budgets based on parameters that specify configuration of the link, transmitter, receiver, and propagation path.

Turbo product code (TPC) decoder function and block support for early termination

The `tpcdec` function and the TPC Decoder block now enable you to specify early termination of TPC decoding, based on the calculated syndrome or parity-check of the component code.

Memoryless nonlinearity System object support for custom AM/AM and AM/PM data

The `comm.MemorylessNonlinearity` System object now supports inputs enabling you to import power amplifier performance data in the form of a look up table (LUT). The LUT defines AM/AM and AM/PM characteristics in a three-column matrix that specifies the input power, output power, and phase shift of a power amplifier signal.

Constellation diagram System object and block support for multiple input

You can now specify multiple inputs to the `comm.ConstellationDiagram` System object and Constellation Diagram block. This feature enables you to input and plot multiple signals on a single constellation diagram. You can also include multiple reference constellations on a single axis.

Support for C code generation using certain interleaver and deinterleaver functions

These interleaver and deinterleaver functions now support C code generation: `algintrlv`, `algdeintrlv`, `intrlv`, `deintrlv`, `matintrlv`, `matdeintrlv`, `helscanintrlv`, and `helscandeintrlv`.

Support for C code generation using certain modulation functions

These modulation functions now support C code generation: `genqammod`, `pammod`, and `pamdmod`.

Class definition now available for CPM modulator System object

You can now view the MATLAB code the for `comm.CPMModulator` System object. This visibility into the code enables you to customize its algorithms and functionality.

Phase Noise block reimplemented as a System block

You can now view the MATLAB code the for Phase Noise block. This visibility into the code enables you to customize its algorithms and functionality.

Track ships using automatic identification system example

The Ship Tracking Using AIS Signals example shows how to receive automatic identification system (AIS) signals from ships and track their location. Using this example, you can play back previously captured AIS signals or use an RTL-SDR radio and Communications System Toolbox™ Support Package for RTL-SDR Radio to receive live AIS signals.

Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Support for Xilinx ZCU102 and Analog Devices FMCOMMS2/3/4

Support for Xilinx ZCU102 and Analog Devices FMCOMMS2/3/4 radio hardware is available through updated objects and blocks.

Product Name	Xilinx Part Number	RF Boards	MATLAB Support	Simulink Support	I/O Peripheral Support	Hardware-Software Co-Design Support
ZCU102	EK-U1-ZCU102-G	Analog Devices FMCOMMS2	<code>comm.SDRDevAD936x</code> radio object	AD936x Receiver block	Yes	Yes
		Analog Devices FMCOMMS3	<code>comm.SDRRxAD936x</code> receiver System object	AD936x Transmitter block		
		Analog Devices FMCOMMS4	<code>comm.SDRTxAD936x</code> transmitter System object			

Communications Toolbox Support Package for Xilinx Zynq-Based Radio: Support for Xilinx ZC706 and Analog Devices FMCOMMS5

Support for Xilinx ZC706 and Analog Devices FMCOMMS5 radio hardware is available through added objects and blocks.

Product Name	Xilinx Part Number	RF Boards	MATLAB Support	Simulink Support	I/O Peripheral Support	Hardware-Software Co-Design Support
ZC706	EK-Z7-ZC706-G	Analog Devices FMCOMMS5	comm.SDRDe vFMCOMMS5 radio object comm.SDRRx FMCOMMS5 receiver System object comm.SDRTx FMCOMMS5 transmitter System object	FMCOMMS5 Receiver block FMCOMMS5 Transmitter block	Yes	Yes

Functionality being removed or changed

lteZadoffChuSeq is being renamed to zadoffChuSeq

Warns

The `lteZadoffChuSeq` function will be removed. Use the `zadoffChuSeq` function instead.

Compatibility Considerations

The `lteZadoffChuSeq` function is being renamed to `zadoffChuSeq` and retains the same functionality.

randseed function will be removed

Still runs

The `randseed` function will be removed in a future release. Use either of these `rng` function syntaxes instead: `rng(seed)` or `rng('shuffle')`.

Align Signals block will be removed

Still runs

The `Align Signals` block will be removed in a future release. Use the `Find Delay` and `Delay` blocks instead.

Equalizer and adaptalg packages and functions will be removed

Still runs

The `equalizer` and `adaptalg` packages and functions will be removed in a future release. Use `comm.LinearEqualizer` and `comm.DecisionFeedbackEqualizer` instead. This table shows the recommended replacement mapping.

Old Functionality	Use This Instead	Compatibility Considerations
<code>equalize</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Replace all instances of <code>equalize</code> with <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .
<code>lineareq</code> and <code>equalizer.lineareq</code>	<code>comm.LinearEqualizer</code>	Replace all instances of <code>lineareq</code> and <code>equalizer.lineareq</code> with <code>comm.LinearEqualizer</code> .
<code>dfe</code> and <code>equalizer.dfe</code>	<code>comm.DecisionFeedbackEqualizer</code>	Replace all instances of <code>dfe</code> and <code>equalizer.dfe</code> with <code>comm.DecisionFeedbackEqualizer</code> .
<code>lms</code> and <code>adaptalg.lms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Replace all instances of <code>lms</code> and <code>adaptalg.lms</code> with <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .
<code>rls</code> and <code>adaptalg.rls</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Replace all instances of <code>rls</code> and <code>adaptalg.rls</code> with <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .
<code>cma</code> and <code>adaptalg.cma</code>	<code>comm.LinearEqualizer</code>	Replace all instances of <code>cma</code> and <code>adaptalg.cma</code> with <code>comm.LinearEqualizer</code> .
<code>signlms</code> and <code>adaptalg.signlms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .
<code>normlms</code> and <code>adaptalg.normlms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .
<code>varlms</code> and <code>adaptalg.varlms</code>	<code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code>	Consider using <code>comm.LinearEqualizer</code> or <code>comm.DecisionFeedbackEqualizer</code> .

Compatibility Considerations

For more details, see the "Compatibility Considerations" sections on the System object reference page for the applicable recommended replacement.

Certain equalization blocks will be removed

Still runs

The equalization blocks listed in this **Old Functionality** column will be removed in a future release. Use the Decision Feedback Equalizer or Linear Equalizer block instead. This table shows the recommended replacement mapping.

Old Functionality	Use This Instead	Compatibility Considerations
LMS Linear Equalizer	Linear Equalizer	Replace all instances of these old blocks with Linear Equalizer.
Normalized LMS Linear Equalizer		
Sign LMS Linear Equalizer		
Variable Step LMS Linear Equalizer		
RLS Linear Equalizer		
CMA Equalizer		
LMS Decision Feedback Equalizer	Decision Feedback Equalizer	Replace all instances of these old blocks with Decision Feedback Equalizer.
Normalized LMS Decision Feedback Equalizer		
Sign LMS Decision Feedback Equalizer		
Variable Step LMS Decision Feedback Equalizer		
RLS Decision Feedback Equalizer		

Compatibility Considerations

For more details, see the "Compatibility Considerations" sections on the block reference page for the applicable recommended replacement.

comm.PSKCoarseFrequencyEstimator System object will be removed

Still runs

The comm.PSKCoarseFrequencyEstimator System object will be removed in a future release. Use the comm.CoarseFrequencyCompensator System object instead.

comm.QAMCoarseFrequencyEstimator System object will be removed

Still runs

The comm.QAMCoarseFrequencyEstimator System object will be removed in a future release. Use the comm.CoarseFrequencyCompensator System object instead.

CRC-N Generator block has been removed

Still runs

The CRC-N Generator block has been removed. Use the General CRC Generator block instead. Existing models automatically replace instances of CRC-N Generator with General CRC Generator, retaining parameter settings for the legacy models.

CRC-N Syndrome Detector block has been removed

Still runs

The CRC-N Syndrome Detector block has been removed. Use the General CRC Syndrome Detector block instead. Existing models automatically replace instances of CRC-N Syndrome Detector with General CRC Syndrome Detector, retaining parameter settings for the legacy models.

commmeasure package has been removed

Errors

The commmeasure package functionality has been removed from the product. This table shows the recommended replacement mapping.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
commmeasure.ACPR	Error	comm.ACPR	Replace all instances of commmeasure.ACPR with comm.ACPR.
commmeasure.EVM	Error	comm.EVM	Replace all instances of commmeasure.EVM with comm.EVM.
commmeasure.MER	Error	comm.MER	Replace all instances of commmeasure.MER with comm.MER.

modem package has been removed

Errors

The modem package functionality has been removed from the product. This table shows the recommended replacement mapping.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
modem.dpskmod	Errors	comm.DPSKModulator	See “DPSK Modulation-Demodulation Compatibility Considerations” on page 10-7.
modem.dpskdemod	Errors	comm.DPSKDemodulator	
modem.mskmod	Errors	comm.MSKModulator or mskmod	See “MSK Modulation-Demodulation Compatibility Considerations” on page 10-8.
modem.mskdemod	Errors	comm.MSKDemodulator or mskdemod	
modem.oqpskmod	Errors	comm.OQPSKModulator	See “OQPSK Modulation-Demodulation

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
modem.oqpskdemod	Errors	comm.OQPSKDemodulator	Compatibility Considerations" on page 10-8.
modem.genqammod	Errors	genqammod or comm.GeneralQAMModulator	See "General QAM Modulation-Demodulation Compatibility Considerations" on page 10-9.
modem.genqamdemod	Errors	genqamdemod or comm.GeneralQAMDemodulator	Replace all instances of modem.qammod with qammod.
modem.qammod	Errors	qammod	Replace all instances of modem.qamdemod with qamdemod.
modem.qamdemod	Errors	qamdemod	See "PAM Modulation-Demodulation Compatibility Considerations" on page 10-11.
modem.pammod	Errors	pammod	See "PSK Modulation-Demodulation Compatibility Considerations" on page 10-11.
modem.pamdemod	Errors	pamdemod	
modem.pskmod	Errors	pskmod or comm.PSKModulator	Compatibility Considerations" on page 10-11.
modem.pskdemod	Errors	pskdemodpskdemod or comm.PSKDemodulator	

gfhelp has been removed

Errors

For the content previously presented by gfhelp, see the gf function.

R2018b

Version: 7.0

New Features

Bug Fixes

Compatibility Considerations

Wireless Waveform Generator App: Create, impair, visualize, and export modulated waveforms

Using the **Wireless Waveform Generator** app you can:

- Create OFDM, QAM, or PSK modulated waveforms or a sine wave test waveform.
- Create WLAN waveforms that are compliant with various 802.11™ modulation schemes (a/b/g/j/p/n/ac/ad/ah). Use of the WLAN feature in the Waveform Generator app requires WLAN Toolbox.
- Add RF impairments to your waveforms.
- Plot waveforms in constellation diagram, spectrum analyzer, and time scopes.
- Export waveforms from the app to your MATLAB workspace or a file.

APSK Modulator and Demodulator Blocks: Simulate amplitude phase shift keying (APSK) modulation and demodulation for satellite applications

Use these blocks to perform APSK modulation and demodulation:

- M-APSK Modulator Baseband and M-APSK Demodulator Baseband
- DVBS-APSK Modulator Baseband and DVBS-APSK Demodulator Baseband, as specified in the Digital Video Broadcasting (DVB) Satellite Communications standards defining DVB-S2, DVB-S2X, and DVB-SH
- MIL188-QAM Modulator Baseband and MIL188-QAM Demodulator Baseband, as specified in the Military HF Modem standard MIL-STD-188-110 B & C and STANAG-4539

Turbo Product Code Encoder and Decoder Blocks: Simulate block product codes for defense and satellite communications

The TPC Encoder and TPC Decoder blocks perform 2-D turbo product code encoding and decoding, as used for defense and satellite communications.

Multichannel Constellation Diagram Plot Support: Plot multichannel signals in constellation diagram displays

You can now use the Constellation Diagram block and `comm.ConstellationDiagram` System object to plot multichannel modulated single-carrier signals on a single set of axes.

ADS-B example for HDL code generation

This Airplane Tracking with ADS-B Captured Data example shows how to implement the Automatic Dependent Surveillance - Broadcast (ADS-B) receiver for HDL code generation and hardware implementation. The algorithm decodes ADS-B extended squitter messages from captured over-the-air data. These messages contain position information, which can be used to track airplanes.

Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio supports burst mode reception

If the Pluto Receiver block or `comm.SDRRxPluto` System object cannot keep up with the ADALM-Pluto radio hardware, then your model or code is not processing data in real time. Burst mode enables you to buffer a set of contiguous samples without losing samples by setting the number of frames in a burst.

Communications Toolbox Support Package for Xilinx Zynq-Based Radio transitions to libiio-based infrastructure

Compatibility Considerations

The support package adopted Analog Devices industrial I/O (IIO) drivers and the corresponding libiio library (version 0.14) for interfacing with Linux[®] IIO devices. This transition gives access to advanced device properties through new objects and blocks. For more information, see [Compatibility with Previous Releases \(Communications Toolbox Support Package for Xilinx Zynq-Based Radio\)](#).

Updated WINNER2 Channel Filtering: Filter provides flatter bandedge response in fading channel System object

The `comm.WINNER2Channel` System object uses a new filter. This filter provides flatter bandedge response than the previous filter.

Compatibility Considerations

Regression tests comparing numerics of the `comm.WINNER2Channel` System object do not match those from previous releases.

Functionality being removed or changed

Compatibility Considerations

The following functionality has changed.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
qammod and qamdemod	Still runs for syntaxes that don't use <code>iniPhase</code> .	To model QAM with an initial phase offset, see the Compatibility Considerations section on <code>qammod</code> and <code>qamdemod</code> . You can also use <code>genqammod</code> and <code>genqamdemod</code> .	The behavior of <code>qammod</code> and <code>qamdemod</code> changed in R2018b. Removed the <code>iniPhase</code> input argument.
Binary Symmetric Channel	Still runs	<ul style="list-style-type: none"> Specify Initial Seed as a scalar. 	The behavior changed in R2018b. Removed support for specifying Initial Seed as a vector.

The following functionality may be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>comm.PAMModulator</code>	Still runs	<code>pammod</code>	<code>comm.PAMModulator</code> is not recommended, use <code>pammod</code> instead.
<code>comm.PAMDemodulator</code>	Still runs	<code>pamdemod</code>	<code>comm.PAMDemodulator</code> is not recommended, use <code>pamdemod</code> instead.

The following functionality will be removed in a future

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>comm.BinarySymmetricChannel</code>	Still runs	<code>bsc</code>	Replace all instances of <code>comm.BinarySymmetricChannel</code> with <code>bsc</code> .
<code>comm.RectangularQAMModulator</code>	Still runs	<code>qammod</code>	Replace all instances of <code>comm.RectangularQAMModulator</code> with <code>qammod</code> .
<code>comm.RectangularQAMDemodulator</code>	Still runs	<code>qamdemod</code>	Replace all instances of <code>comm.RectangularQAMDemodulator</code> with <code>qamdemod</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Gaussian Filter	Warns	A model that includes both <code>gaussdesign</code> and Discrete FIR Filter	See “Gaussian Filter Compatibility Considerations” on page 9-5.
<code>comm.LTEMIMOChannel</code>	Warns	<code>comm.MIMOChannel</code>	Replace all instances of <code>comm.LTEMIMOChannel</code> with <code>comm.MIMOChannel</code> .

The following functionality has been removed.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>modem.qammod</code>	Errors	<code>qammod</code>	Replace all instances of <code>modem.qammod</code> with <code>qammod</code> .
<code>modem.qamdemod</code>	Errors	<code>qamdemod</code>	Replace all instances of <code>modem.qamdemod</code> with <code>qamdemod</code> .

Gaussian Filter Compatibility Considerations

Replace all instances of the Gaussian Filter block with a modeling solution that includes the `gaussdesign` function and the Discrete FIR Filter block. Model Gaussian Filter in Simulink demonstrates using the `gaussdesign` function along with the Discrete FIR Filter block to model a Gaussian filter. In Simulink, you can use the `PreLoadFcn` callback to specify the filter coefficients as workspace variables. You can specify coefficients for Gaussian filter designs that apply various normalization methods, that you assign in a Discrete FIR Filter block.

Rectangular QAM Modulation-Demodulation Compatibility Considerations

Replace all instances of `comm.RectangularQAMModulator` and `comm.RectangularQAMDemodulator` with `qammod` and `qamdemod`, respectively.

Constellation normalization by `PeakPower` and `AveragePower` (other than unit average power), as supported by the System objects, is not inherently provided by the `qammod` and `qamdemod` functions. For code samples that achieve such constellation normalization when using these functions, see the **Compatibility Considerations** section on `comm.RectangularQAMModulator` and `comm.RectangularQAMDemodulator`.

R2018a

Version: 6.6

New Features

Compatibility Considerations

APSK Modulator and Demodulator: Simulate amplitude phase shift keying modulation for satellite applications

The following functions perform APSK modulation and demodulation:

- Generic APSK (`apskmod` and `apskdemod`)
- Digital Video Broadcasting standard-specific APSK (`dvbsapskmod` and `dvbsapskdemod`), as specified in the Digital Video Broadcasting (DVB) Satellite Communications standards defining DVB-S2, DVB-S2X, and DVB-SH
- MIL-STD APSK (`mil188qammod` and `mil188qamdemod`), as specified in the Military HF Modem standard MIL-STD-188-110 B & C and STANAG-4539

Turbo Product Code Encoder and Decoder: Simulate block product codes for defense and satellite communications

The `tpcenc` and `tpcdec` functions perform 2-D turbo product code encoding and decoding, as used for defense and satellite communications.

Massive MIMO Example: Simulate an end-to-end MIMO link using hybrid beamforming

The Massive MIMO Hybrid Beamforming example demonstrates hybrid beamforming employed at the transmit end of a massive MIMO communications system.

The example uses full channel sounding for determining the channel state information at the transmitter. Using different techniques for multi-user and single-user systems, the required precoding is partitioned into digital baseband and analog RF components. Simplified all-digital receivers recover the multiple transmitted data streams to present the resulting EVM and BER.

OFDM Modulator and Demodulator: Simulate orthogonal frequency division modulation

The `ofdmmod` and `ofdmmodemod` functions perform OFDM modulation and demodulation.

Enhanced Fidelity of Phase Noise: Model close-in phase noise more precisely

The fidelity of close-in phase noise modeled by the `comm.PhaseNoise` System object and Phase Noise block has been enhanced.

Frequency Correction Support: Adjust ADALM-Pluto Radio frequency to correct frequency offset (Introduced November 2017)

The correction of frequency offset error and A/D and D/A clock error is available when using the ADALM-Pluto Radio blocks (Pluto Receiver, Pluto Transmitter) and System objects (`comm.SDRRxPluto`, `comm.SDRTxPluto`). The Frequency Correction for ADALM-PLUTO Radio example demonstrates this feature in MATLAB.

Custom Filter Wizard: Design custom filtering for ADALM-Pluto Radio (Introduced November 2017)

Use the Analog Devices filter wizard to design custom pulse shape filtering for the ADALM-Pluto Radio. For more information, see `designCustomFilter`.

Low IF Receiver Architecture Example: Simulate a low IF receiver architecture RF model

The Low IF Receiver example models a receiver design using components from RF Blockset™. The example evaluates BER performance with and without phase noise.

ALOHA and CSMA/CA Packetized Wireless Network Example: Simulate an ALOHA or CSMA/CA medium access controller

The ALOHA and CSMA/CA Packetized Wireless Networks example simulates a PHY with an ALOHA or CSMA/CA MAC using Simulink, Stateflow®, and the Communications System Toolbox.

Examples added to library for ZigBee Protocol

The Communications Toolbox Library for the ZigBee Protocol now includes these examples:

- ZigBee Light Link Frame Generation and Decoding — Generate and decode ZigBee Light Link application profile frames.
- ZigBee Frame Generation and Decoding for General Commands — Generate and decode ZigBee specification General Command frames.
- ZigBee Smart Energy Frame Generation and Decoding — Generate and decode ZigBee Smart Energy frames.

Examples added to library for NFC Protocol

The Communications Toolbox Library for the NFC Protocol now includes these examples:

- Introduction to Communications System Toolbox Library for the NFC Protocol Video — Get started using the library.
- NFC Application Layer — Exchange data between application layers of two Near Field Communication (NFC) devices.

Updated Fading Channel Filtering: Filter provides flatter bandedge response in fading channel System objects and blocks

The `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects and the MIMO Fading Channel block use a new filter.

This filter provides flatter bandedge response than the previous filter.

Compatibility Considerations

Regression tests comparing numerics of the `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects and the MIMO Fading Channel block do not match those from previous releases.

Functionality being removed or changed

Compatibility Considerations

The following functionality has changed.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
<code>commsrc.combinedjitter</code>	Still runs	Added <code>SamplingFrequency</code> and <code>PeriodicFrequencyHz</code> properties.	The behavior changed in R2018a. Use <code>PeriodicFrequencyHz</code> instead of <code>PeriodicFrequency</code> .

The following functionality will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>huffmandict</code>	Still runs	Not applicable	The behavior changed in R2018a. When duplicate dictionary symbols are input, <code>huffmandict</code> merges duplicated symbols by summing their probabilities.
<code>modem.dpskmod</code>	Warns	<code>comm.DPSKModulator</code>	See “DPSK Modulation-Demodulation Compatibility Considerations” on page 10-7.
<code>modem.dpskdemod</code>	Warns	<code>comm.DPSKDemodulator</code>	
<code>modem.mskmod</code>	Warns	<code>comm.MSKModulator</code> or <code>mskmod</code>	See “MSK Modulation-Demodulation Compatibility Considerations” on page 10-8.
<code>modem.mskdemod</code>	Warns	<code>comm.MSKDemodulator</code> or <code>mskdemod</code>	
<code>modem.oqpskmod</code>	Warns	<code>comm.OQPSKModulator</code>	See “OQPSK Modulation-Demodulation Compatibility Considerations” on page 10-8.
<code>modem.oqpskdemod</code>	Warns	<code>comm.OQPSKDemodulator</code>	

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
modem.genqammod	Warns	genqammod or comm.GeneralQAMModulator	See “General QAM Modulation-Demodulation Compatibility Considerations” on page 10-9.
modem.genqamdemod	Warns	genqamdemod or comm.GeneralQAMDemodulator	
modem.pammod	Warns	pammod	See “PAM Modulation-Demodulation Compatibility Considerations” on page 10-11.
modem.pamdemod	Warns	pamdemod	
modem.pskmod	Warns	pskmod or comm.PSKModulator	See “PSK Modulation-Demodulation Compatibility Considerations” on page 10-11.
modem.pskdemod	Warns	pskdemod or comm.PSKDemodulator	
rayleighchan	Warns	comm.RayleighChannel	See “Rayleigh Channel Compatibility Considerations” on page 10-12.
ricianchan	Warns	comm.RicianChannel	See and “Rician Channel Compatibility Considerations” on page 10-14.
doppler.jakes	Warns	doppler	To generate a Jakes Doppler spectrum use <code>doppler('Jakes')</code> instead.
doppler.flat	Warns	doppler	To generate a flat Doppler spectrum use <code>doppler('Flat')</code> instead.
doppler.bell	Warns	doppler	To generate a bell Doppler spectrum use <code>doppler('Bell')</code> instead.
doppler.rounded	Warns	doppler	To generate a rounded Doppler spectrum use <code>doppler('Rounded')</code> instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>doppler.rjakes</code>	Warns	<code>doppler</code>	To generate a restricted Jakes Doppler spectrum use <code>doppler('Restricted Jakes')</code> instead.
<code>doppler.ajakes</code>	Warns	<code>doppler</code>	To generate an asymmetrical Jakes Doppler spectrum use <code>doppler('Asymmetric Jakes')</code> instead.
<code>doppler.gaussian</code>	Warns	<code>doppler</code>	To generate a Gaussian Doppler spectrum use <code>doppler('Gaussian')</code> instead.
<code>doppler.bigaussian</code>	Warns	<code>doppler</code>	To generate a bi-Gaussian Doppler spectrum use <code>doppler('BiGaussian')</code> instead.
<code>stdchan(ts,fd,chanType)</code>	Warns	<code>stdchan(chanType,rs,fd)</code>	See “stdchan Compatibility Considerations” on page 10-15.
<code>legacychannelsim</code>	Warns	<code>comm.RayleighChannel</code> or <code>comm.RicianChannel</code>	See “Legacy Channel Simulation Compatibility Considerations” on page 10-16.
<code>oqpskmod</code>	Warns	<code>comm.OQPSKModulator</code>	Replace all instances of <code>oqpskmod</code> with <code>comm.OQPSKModulator</code> .
<code>oqpskdemod</code>	Warns	<code>comm.OQPSKDemodulator</code>	Replace all instances of <code>oqpskdemod</code> with <code>comm.OQPSKDemodulator</code> .
<code>comm.CPMCarrierPhaseSynchronizer</code>	Warns	<code>comm.CarrierSynchronizer</code>	Replace all instances of <code>comm.CPMCarrierPhaseSynchronizer</code> with <code>comm.CarrierSynchronizer</code> .
<code>commscope.ScatterPlot</code>	Errors	<code>comm.ConstellationDiagram</code>	Replace all instances of <code>commscope.ScatterPlot</code> with <code>comm.ConstellationDiagram</code> .

DPSK Modulation-Demodulation Compatibility Considerations

Replace all instances of `modem.dpskmod` and `modem.dpskdemod` with `comm.DPSKModulator` and `comm.DPSKDemodulator`.

- User-defined symbol mapping is not supported.
- The `comm.DPSKModulator` System object does not support initial phase specification. However, this code sample shows you how to specify initial phase for DPSK-modulated signals:

```
%% Initial Phase
x = randi([0,7],10,1);
phRot = pi/8;
initPh = pi/4;

%% Using modem.dpskmod modulate the signal
m = modem.dpskmod('M',8,'PhaseRotation',phRot,'InitialPhase',initPh);
ym = modulate(m, x);

%% Using comm.DPSKModulator modulate the signal
s = comm.DPSKModulator('PhaseRotation',phRot,'SymbolMapping','Binary');
ys = s(x);
ys1 = ys .* exp(1i * initPh);

%% Compare results
err = max(abs(ys1 - ym))
```

```
err =

    2.6390e-15
```

- This code sample shows you how to simulate multichannel input/output for DPSK-modulated signals:

```
%% Multichannel input/output
x = randi([0,7],10,3);
phRot = pi/8;

%% Using modem.dpskmod modulate the signal
m = modem.dpskmod('M',8,'PhaseRotation',phRot);
ym1 = m.modulate(x);
% Process one more frame to verify interframe state handling
ym2 = m.modulate(x);

%% Using comm.DPSKModulator modulate the signal
% Create as many System objects as number of channels
s1 = comm.DPSKModulator('PhaseRotation',phRot,'SymbolMapping','Binary');
s2 = comm.DPSKModulator('PhaseRotation',phRot,'SymbolMapping','Binary');
s3 = comm.DPSKModulator('PhaseRotation',phRot,'SymbolMapping','Binary');

ys1 = complex(zeros(size(x),'like',x));
ys2 = ys1;

% Process first frame
% Process each channel by its specific System object
ys1(:,1) = s1(x(:,1));
ys1(:,2) = s2(x(:,2));
ys1(:,3) = s3(x(:,3));

% Process second frame
ys2(:,1) = s1(x(:,1));
ys2(:,2) = s2(x(:,2));
ys2(:,3) = s3(x(:,3));

%% Compare results
err1 = (max(abs(ys1(:) - ym1(:))))
err2 = (max(abs(ys2(:) - ym2(:))))
```

```
err1 =

    2.0411e-15
```

```
err2 =
    2.4431e-15
```

MSK Modulation-Demodulation Compatibility Considerations

Replace instances of `modem.mskmod` and `modem.mskdemod` with `comm.MSKModulator` and `comm.MSKDemodulator`. For precoded MSK modulation or demodulation, use `mskmod` and `mskdemod`.

- This code sample shows you how to simulate multichannel input/output for MSK-modulated signals:

```
%% Multichannel input/output
x = randi([0,1],4,3);

%% Modulation
% Using modem.mskmod modulate the multiple channel signal
m_msk = modem.mskmod;
ym1 = m_msk.modulate(x);
% Process one more frame to verify interframe state handling
ym2 = m_msk.modulate(x);

% Using mskmod modulate the multiple channel signal
[yf1, phOut] = mskmod(x,8);
yf2 = mskmod(x,8,[],phOut);

% Compare the modulation results
err1 = max(abs(ym1(:) - yf1(:)))
err2 = max(abs(ym2(:) - yf2(:)))

err1 = 5.4672e-16
err2 = 5.4672e-16

%% Demodulation
% Using modem.mskdemod demodulate the multiple channel signal
dm_msk = modem.mskdemod;
ym1_dm = dm_msk.demodulate(ym1)
% Process one more frame to verify interframe state handling
ym2_dm = dm_msk.demodulate(ym2)

% Using mskdemod demodulate the multiple channel signal
[yf1_dm, phOut] = mskdemod(yf1,8)
yf2_dm = mskdemod(yf2,8,[],phOut)

% Compare the demodulation results to transmission signal
%isequal(x,yf1_dm)
%isequal(x,yf2_dm)

% For the modem object must account for two bit output delay
%isequal(x(1:end-2,:),ym1_dm(3:end,:))
%isequal(x(1:end-2,:),ym2_dm(3:end,:))
```

OQPSK Modulation-Demodulation Compatibility Considerations

Replace all instances of `modem.oqpskmod` and `modem.oqpskdemod` with `comm.OQPSKModulator` and `comm.OQPSKDemodulator`.

- This code sample shows you how to simulate multichannel input/output for OQPSK-modulated signals:

```

%% Multichannel input/output OQPSK modulation
m = 4;
numChan = 3;
x = randi([0,m-1],10,numChan);
sps = 2; % Samples per symbol

%% Using modem.oqpskmod modulate the signal
m_oqpsk = modem.oqpskmod();
ym1 = m_oqpsk.modulate(x);
% Process one more frame to verify interframe state handling
ym2 = m_oqpsk.modulate(x);

%% Using comm.OQPSKModulator modulate the signal
% Create as many System objects as number of channels
s1_oqpsk = comm.OQPSKModulator('SymbolMapping','Binary','SamplesPerSymbol',sps);
s2_oqpsk = comm.OQPSKModulator('SymbolMapping','Binary','SamplesPerSymbol',sps);
s3_oqpsk = comm.OQPSKModulator('SymbolMapping','Binary','SamplesPerSymbol',sps);

% Preallocate output matrix
ys1 = complex(zeros(sps*length(x),numChan,'like',x));
ys2 = ys1;

% Process first frame
% Process each channel by its specific System object
ys1(:,1) = s1_oqpsk(x(:,1));
ys1(:,2) = s2_oqpsk(x(:,2));
ys1(:,3) = s3_oqpsk(x(:,3));

% Process second frame
ys2(:,1) = s1_oqpsk(x(:,1));
ys2(:,2) = s2_oqpsk(x(:,2));
ys2(:,3) = s3_oqpsk(x(:,3));

% The modulated signals are not compared because the
% comm.OQPSKModulator performs joint filtering and
% modulation of the signal, while the modem.oqpskmod
% only modulates the signal.

```

- The `comm.OQPSKDemodulator` System object does not support soft decision output for OQPSK-modulated signals. However, this code sample shows you how to simulate soft decision output for OQPSK-modulated signals:`qamdemod`

```

%% Soft decision OQPSK demodulation
% Create an OQPSK signal and add noise to the signal
sps = 4;
msg = randi([0 1],1000,1);
oqpskMod = comm.OQPSKModulator('SamplesPerSymbol',sps,'BitInput',true);
oqpskSig = oqpskMod(msg);

impairedSig = awgn(oqpskSig,15);

% Soft-demodulate using the qamdemod function

% Align I and Q (create QPSK equivalent signal)
impairedQPSK = complex(real(impairedSig(1+sps/2:end-sps/2)), imag(impairedSig(sps+1:end)));

% Apply matched filtering to the received OQPSK signal
halfSinePulse = sin(0:pi/sps:(sps)*pi/sps);
matchedFilter = dsp.FIRDecimator(sps, halfSinePulse, 'DecimationOffset',sps/2);
filteredQPSK = matchedFilter(impairedQPSK);

% To perform soft demodulation of the filtered OQPSK signal use the
% qamdemod function. Align symbol mapping of qamdemod with the symbol
% mapping used by the comm.OQPSKModulator, then demodulate the signal.
oqpskModSymbolMapping = [1 3 0 2];
demodulated = qamdemod(filteredQPSK,4,oqpskModSymbolMapping,'OutputType','llr');

```

General QAM Modulation-Demodulation Compatibility Considerations

Replace instances of `modem.genqammod` and `modem.genqamdemod` with `genqammod` and `genqamdemod`. To generate soft decision output, use `comm.GeneralQAMModulator` and `comm.GeneralQAMDemodulator`.

- This code sample shows you bit input and multichannel input support for general QAM modulated signals:

```
%% Bit Input
% Multichannel input
x = randi([0,1],16,3);

M = 16;
nBits = log2(M);
inSize = size(x);
outSize = inSize;
% nBits bits make one symbol
outSize(1) = outSize(1)/nBits;

% M-QAM constellation
const = qammod((0:M-1)',M);

% Modulate
m1 = modem.genqammod('InputType','Bit','Constellation',const);
yml = m1.modulate(x);

% Convert bit input to integer
% Each column is a bit representation of an integer
xTmp = reshape(x,nBits,numel(x)/nBits);
% each row is a bit representation of an integer, left-MSB
xTmp = xTmp';
xIntCol = bi2de(xTmp,'left-msb');
xIntMat = reshape(xIntCol,outSize);

% Compare modulator function output
yf1 = genqammod(xIntMat,const);
err = max(abs(yml(:) - yf1(:)))

err =
    0
```

- This code sample shows you how to simulate multichannel input/output and soft decision output for general QAM-modulated signals:

```
%% Soft decision output
% Multichannel input
x = randi([0,1],16,3);

M = 16;
nBits = log2(M);
inSize = size(x);
outSize = inSize;
% nBits bits make one symbol
outSize(1) = outSize(1)/nBits;

% 16-QAM constellation
const = qammod((0:M-1)',M);

% Modulate
m1 = modem.genqammod('InputType','Bit','Constellation',const);
yml = m1.modulate(x);

% Convert bit input to integer
% Each column is a bit representation of an integer
xTmp = reshape(x,nBits,numel(x)/nBits);
% Each row is a bit representation of an integer, left-MSB
xTmp = xTmp';
xIntCol = bi2de(xTmp,'left-msb');

% Compare System object output
s1 = comm.GeneralQAMModulator('Constellation',const);
```

```

ysTmp = s1(xIntCol);
ys1 = reshape(ysTmp,outSize);

err1 = max(abs(ym1(:) - ys1(:)))

% Demodulate (LLR)
m2 = modem.genqamdemod('Constellation',const,'OutputType','Bit', ...
'DecisionType','LLR','NoiseVariance',0.2);
ym2 = m2.demodulate(ym1);

s2 = comm.GeneralQAMDemodulator('Constellation',const,'BitOutput',true, ...
'DecisionMethod','Log-likelihood ratio','Variance',0.2);
% Columnize ys1 as System object supports only column vector input
ys2 = s2(ys1(:));
ys2 = reshape(ys2,inSize);
err2 = max(abs(ym2(:) - ys2(:)))

err1 =

    0

err2 =

    0

```

PAM Modulation-Demodulation Compatibility Considerations

Replace all instances of `modem.pammod` and `modem.pamdemod` with `pammod` and `pamdemod`.

- To model PAM-modulated signals, follow guidance in “General QAM Modulation-Demodulation Compatibility Considerations” on page 10-9 for bit input support, multichannel input-output support, and soft decision output.
- This code sample shows how to apply user-defined symbol mapping for PAM-modulated signals:

```

%% User-defined symbol mapping
M = 8;
% Custom symbol mapping
symMap = [3,7,4,0,2,6,1,5];
m = modem.pammod('M',M,'SymbolOrder','User-defined', ...
'SymbolMapping',symMap);
c = m.Constellation;
[~,lb] = ismember((0:M-1),symMap);

% Specify the constellation in General QAM System object
% in the right order
s = comm.GeneralQAMModulator('Constellation',complex(c(lb)));

x = randi([0,M-1],100,1);
ym = m.modulate(x);
ys = s(x);

err = max(abs(ym - ys))

err =

    0

```

PSK Modulation-Demodulation Compatibility Considerations

Replace instances of `modem.pskmod` and `modem.pskdemod` with `pskmod` and `pskdemod`. If you require bit input, user-defined symbol mapping, or soft decision output, use `comm.PSKModulator` and `comm.PSKDemodulator` System objects.

- This code sample shows you how to simulate multichannel input/output for PSK-modulated signals:

```

%% Multi-channel input-output
M = 16;
numChan = 1;
x = randi([0,M-1],16,3);
phOff = pi/8;

%% Using modem object modulate and demodulate
% Modulate using modem.pskmod modulate the signal
mpsk = modem.pskmod('M',M,'PhaseOffset',phOff);
ym1mod = mpsk.modulate(x);

% LLR demodulate using modem.pskdemod
demodpsk = modem.pskdemod('OutputType','Bit','PhaseOffset',phOff, ...
    'DecisionType','LLR','NoiseVariance',0.2);
ym1demod = demodpsk.demodulate(ym1mod);

%% Using System objects modulate and demodulate
% Modulate using comm.PSKModulator
% Create as many System objects as number of channels
s1mod = comm.PSKModulator('ModulationOrder',M, ...
    'SymbolMapping','Binary');
s2mod = clone(s1mod);
s3mod = clone(s1mod);

ys1mod = complex(zeros(size(x),'like',x));

% Process each channel by it's specific System object
ys1mod(:,1) = s1mod(x(:,1));
ys1mod(:,2) = s2mod(x(:,2));
ys1mod(:,3) = s3mod(x(:,3));

% LLR Demodulate using comm.PSKDemodulate
% Create comm.PSKDemodulator objects for each channel
% Create as many System objects as number of channels
s1demod = comm.PSKDemodulator('ModulationOrder',M, ...
    'DecisionMethod','Log-likelihood ratio', ...
    'Variance',0.2,'SymbolMapping','Binary');
s2demod = clone(s1demod);
s3demod = clone(s1demod);

% Using comm.PSKDemodulator demodulate the signal, process each
% channel by it's specific System object
ysl1demod = complex(zeros(length(x),numChan,'like',x));
ysl1demod(:,1) = s1demod(ys1mod(:,1));
ysl1demod(:,2) = s2demod(ys1mod(:,2));
ysl1demod(:,3) = s3demod(ys1mod(:,3));

%% Compare modulation and demodulation for System object and modem object
% Residual modulation error is small
err1 = (max(abs(ysl1demod(:) - ym1mod(:))))
% Check to see that demodulated data signal equals the original data
t_f = isequal(ysl1demod,x)

err1 =

    2.4493e-16

t_f =

    logical

     1

```

Rayleigh Channel Compatibility Considerations

Replace instances of `rayleighchan` with a `comm.RayleighChannel` System object. Note the mapping between function parameters and object properties:

Comparison Between rayleighchan and comm.RayleighChannel				
rayleighchan		comm.RayleighChannel		Notes
Properties	Associated functions	Properties	Object functions	
ChannelType				Indicated by System object name
InputSamplePeriod		SampleRate		Value is reciprocal
DopplerSpectrum		DopplerSpectrum		
MaxDopplerShift		MaximumDopplerShift		
PathDelays		PathDelays		
AvgPathGaindB		AveragePathGains		
NormalizePathGains		NormalizePathGains		
StoreHistory				Adjacent with the plot function of rayleighchan
PathGains			object call (step)	Second output returned by a call of the object (step)
ChannelFilterDelay			info	Returned by a call to info
ResetBeforeFiltering			reset	Call reset before each object function call (step)
NumSamplesProcessed			info	Returned by a call to info
	filter		object call (step)	First output returned by a call of the object (step)
	reset		reset	
	plot	Visualization SamplesToDisplay PathsForDopplerDisplay	object call (step)	The Visualization property setting controls display of graphics during an object call (step).

StorePathGains		PathGainsOutputPort		
----------------	--	---------------------	--	--

Rician Channel Compatibility Considerations

Replace instances of `ricianchan` with a `comm.RicianChannel` System object. Note the mapping between function parameters and object properties:

Comparison Between <code>ricianchan</code> and <code>comm.RicianChannel</code>				
ricianchan		comm.RicianChannel		Notes
Properties	Associated functions	Properties	Object functions	
ChannelType				Indicated by System object name
InputSamplePeriod		SampleRate		Value is reciprocal
DopplerSpectrum		DopplerSpectrum		
MaxDopplerShift		MaximumDopplerShift		
PathDelays		PathDelays		
KFactor		KFactor		
DirectPathDopplerShift		DirectPathDopplerShift		
DirectPathInitialPhase		DirectPathInitialPhase		
AvgPathGaindB		AveragePathGains		
NormalizePathGains		NormalizePathGains		
StoreHistory				Adjacent with the plot function of <code>ricianchan</code>
StorePathGains		PathGainsOutputPort		
PathGains			object call (step)	Second output returned by a call of the object (step)
ChannelFilterDelay			info	Returned by a call to info

ResetBeforeFiltering			reset	Call reset before each object function call (step)
NumSamplesProcessed			info	Returned by a call to info
	filter		object call (step)	First output returned by a call of the object (step)
	reset		reset	
	plot	Visualization SamplesToDisplay PathsForDopplerDisplay	object call (step)	The Visualization property setting controls display of graphics during an object call (step).

stdchan Compatibility Considerations

Compatibility considerations for the stdchan function includes addition of a function syntax, and announcing removal of one syntax and removal of configuration support for several channel models.

- The syntax `chan = stdchan(ts, fd, chantype)` will be removed in a future release. A System object is returned using the new syntax. See this new syntax in stdchan.
- In the future stdchan will remove support for configuration of several channel models by supported standards and associated syntax, compatibility considerations are indicated here:

Standard	Previous Syntax	New Syntax to Return System Object	Notes
3GPP, CDMA	<code>stdchan(ts, fd, '3gppXXX')</code>	<code>stdchan('cdmaXXX', rs, fd)</code>	Prefix changed from '3gpp' to 'cdma'. ts and rs are reciprocal values.
GSM	<code>stdchan(ts, fd, 'gsmXXX')</code>	<code>stdchan('gsmXXX', rs, fd)</code>	ts and rs are reciprocal values.
ITU-R HF	<code>stdchan(ts, fd, 'iturHFXXX')</code>	<code>stdchan('iturHFXXX', rs, fd)</code>	ts and rs are reciprocal values.
COST207	<code>stdchan(ts, fd, 'cost207XXX')</code>	N/A	In the future stdchan will not configure these channels. Use <code>comm.RayleighChannel</code> or <code>comm.RicianChannel</code> to configure the channel models for
ITU-R 3G	<code>stdchan(ts, fd, 'itur3GXXX')</code>	N/A	
JTC	<code>stdchan(ts, fd, 'jtcXXX')</code>	N/A	

HIPERLAN/2	stdchan(ts,fd,'hiperlan2XXX')	N/A	COST207, ITU-R 3G, JTC, HIPERLAN/2, and 802.11a/b/g standards. For guidance mapping parameters, see “Rayleigh Channel Compatibility Considerations” on page 10-12 and “Rician Channel Compatibility Considerations” on page 10-14.
802.11a/b/g	stdchan(ts,fd,'802.11X')	N/A	

Legacy Channel Simulation Compatibility Considerations

- For `legacychannelsim(true)` and a Rayleigh channel, you can achieve equivalent functionality to `rayleighchan` using a `comm.RayleighChannel` System object, configured as shown in this code sample:

```
% Set the RNG seed value to 10 and generate a vector
% of random complex numbers.
seed = 10;
x = complex(rand(100,1),rand(100,1));
%%
% Create a Rayleigh channel System object. Set RandomStream to
% 'Global stream', and set the random number generator to 'v5normal'.
chan = comm.RayleighChannel('RandomStream','Global stream');
rng(seed,'v5normal');
chan(x);
```

Similarly for a Rician channel, equivalent functionality to `ricianchan` can be achieved with the above code using a `comm.RicianChannel` System object.

- For `legacychannelsim(false)` and a Rayleigh channel, you can achieve equivalent functionality to `rayleighchan` using a `comm.RayleighChannel` System object, configured as shown in this code sample:

```
% Set the RNG seed value to 10 and generate a vector
% of random complex numbers.
seed = 10;
x = complex(rand(100,1),rand(100,1));
%%
% Option 1 - Create a Rayleigh channel System object, set
% RandomStream to 'Global stream', and set the random
% number generator to 'twister'.
chan = comm.RayleighChannel('RandomStream','Global stream');
rng(seed,'twister');
chan(x);
%%
% Option 2 - Create a Rayleigh channel System object setting
% RandomStream to 'mt19937ar with seed'.
chan = comm.RayleighChannel('RandomStream', ...
    'mt19937ar with seed','Seed',seed);
chan(x);
```

Similarly for a Rician channel, equivalent functionality to `ricianchan` can be achieved with the above code using a `comm.RicianChannel` System object.

R2017b

Version: 6.5

New Features

Bug Fixes

Compatibility Considerations

Library for ZigBee Protocol: Simulate ZigBee low-rate wireless personal area network (LRWPAN) technologies

The Communications Toolbox Library for the ZigBee Protocol uses MATLAB to design and verify practical ZigBee systems. Using the examples in this library, you can generate and decode waveforms and frames formatted for transmission in these layers of the protocol stack:

- IEEE® 802.15.4 physical (PHY) and medium-access control (MAC) layers
- ZigBee network (NWK) layer and application (APL) layer

You can download the Communications Toolbox Library for the ZigBee Protocol from Add-On Explorer. For more information, see [Get Add-Ons](#) and [Manage Your Add-Ons](#).

Library for NFC Protocol: Simulate Near Field Communication (NFC) wireless technologies

The Communications Toolbox Library for the NFC Protocol uses MATLAB to design and verify practical NFC systems. Using the examples in this library, you can generate and decode waveforms and frames formatted for transmission in the NFC standard.

You can download the Communications Toolbox Library for the NFC Protocol from Add-On Explorer. For more information, see [Get Add-Ons](#) and [Manage Your Add-Ons](#).

OQPSK Simulation: Simulate practical OQPSK links with modulation, frequency and timing synchronization, and demodulation functions

The following System objects now enable easier simulation of practical OQPSK links: `comm.OQPSKModulator`, `comm.OQPSKDemodulator`, `comm.CoarseFrequencyCompensator`, `comm.CarrierSynchronizer`, and `comm.SymbolSynchronizer`.

MIMO Channel Enhancements: Specify an arbitrary number of antennas and include antenna polarization when you simulate MIMO fading channels

The `comm.MIMOChannel` System object and MIMO Fading Channel block now allow you to specify antenna polarization and an arbitrary number of antennas in your system.

In addition, you can now use the MIMO Fading Channel block to specify Rayleigh or Rician fading for MIMO channels.

Path Loss Functions: Account for path loss due to free space, fog, gas, and rain

Functions are added to model path loss due to free space (`fspl`), fog and clouds (`fogpl`), atmosphere (`gaspl`), and rain (`rainpl`).

Communications System Toolbox Support Package for Analog Devices ADALM-Pluto Radio: Lower minimum baseband sample rate (Introduced in May 2017)

You can specify baseband sample rates as low as 65.104 kHz. Previously, the minimum sample rate permitted was 520.841 kHz.

Communications System Toolbox Support Package for Xilinx Zynq-Based Radio: PicoZed SDR renamed to ADI RF SOM

The PicoZed™ SDR product has been renamed to ADI RF SOM. Support for the ADI RF SOM product is available through new MATLAB objects and Simulink blocks.

New Product Name	Analog Devices Part Number	Description	MATLAB Support	Simulink Support	I/O Peripheral Support	HW/SW Co-Design Support
ADI RF SOM	ADRV9361	XC7Z035 / AD9361 SOM	comm.SDRDe vADIRFSOM radio object comm.SDRRx ADIRFSOM receiver System object comm.SDRTx ADIRFSOM transmitter System object	ADI RF SOM Receiver block ADI RF SOM Transmitter block	Yes	Yes

For more information, see ADI RF SOM (Communications System Toolbox Support Package for Xilinx Zynq®-Based Radio).

Compatibility Considerations

Starting in R2018a, the name 'PicoZed SDR' will no longer work in function calls, and the PicoZed SDR library will be removed from Simulink. As of R2017b, use the name 'ADI RF SOM' in function calls, and use the new ADI RF SOM features for all PicoZed SDR products supported in previous releases. See PicoZed SDR (Before R2017b) Communications System Toolbox Support Package for Xilinx Zynq-Based Radio.

Feature Removed	Result	Use Instead	Compatibility Considerations
<code>comm.SDRDevPicoZedSDR</code> radio object	Object creation warns and <code>comm.SDRDevADIRFSOM</code> radio object is created instead.	<code>comm.SDRDevADIRFSOM</code> radio object	Replace 'PicoZed SDR' with 'ADI RF SOM' in all calls to <code>sdrdev</code> .
<code>comm.SDRRxPicoZedSDR</code> receiver System object	System object creation warns and <code>comm.SDRRxADIRFSOM</code> receiver System object is created instead.	<code>comm.SDRRxADIRFSOM</code> receiver System object	Replace 'PicoZed SDR' with 'ADI RF SOM' in all calls to <code>sdr rx</code> .
<code>comm.SDRTxPicoZedSDR</code> transmitter System object	System object creation warns and <code>comm.SDRTxADIRFSOM</code> transmitter System object is created instead.	<code>comm.SDRTxADIRFSOM</code> transmitter System object	Replace 'PicoZed SDR' with 'ADI RF SOM' in all calls to <code>sdr tx</code> .
PicoZed SDR Receiver block	Warns and forwards library link to ADI RF SOM Receiver block.	ADI RF SOM Receiver block	Replace all PicoZed SDR Receiver blocks with ADI RF SOM Receiver blocks.
PicoZed SDR Transmitter block	Warns and forwards library link to ADI RF SOM Transmitter block.	ADI RF SOM Transmitter block	Replace all PicoZed SDR Transmitter blocks with ADI RF SOM Transmitter blocks.

USRP E312 Support: Prototype and test USRP E312 software-defined radio (SDR) systems

Support for USRP ¹E312 embedded series radio is now available with the Communications System Toolbox Support Package for USRP Embedded Series Radio. The support is seamlessly integrated through the features and functionality already available for USRP E310.

Product Name	MATLAB Support	Simulink Support	I/O Peripheral Support	HW/SW Co-Design Support
USRP E312	<code>comm.SDRDevE310</code> radio object <code>comm.SDRRxE310</code> receiver System object <code>comm.SDRTxE310</code> transmitter System object	E310 Receiver block E310 Transmitter block	Yes	Yes

Key Features

- Use USRP E312 as an I/O peripheral to transmit and receive real-time arbitrary waveforms using MATLAB System objects and Simulink blocks.

¹ USRP, USRP2, UHD, and Ettus Research are trademarks of National Instruments Corp.

- Transmit and receive RF signals out of the box, enabling the testing of SDR designs quickly, under real-world conditions.
- Transmit and receive data on one or two channels.
- Configure RF radio settings easily.
- Acquire high-bandwidth signals by using burst mode.
- Customize and prototype SDR algorithms in Simulink, and deploy partitioned hardware-software co-design implementations across the ARM processor and the FPGA fabric of the device (Windows operating system only).
- Run all USRP E310-based application examples for the USRP E312.

Communications System Toolbox Support Package for Xilinx FPGA-Based Radio: Support removed

Communications System Toolbox Support Package for Xilinx FPGA-Based Radio has been discontinued for R2017b.

Compatibility Considerations

The support package is available for use with previous releases.

Functionality being removed or changed

The following functionality will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Multipath Rayleigh Fading Channel	SISO Fading Channel is loaded, retaining parameter settings for legacy model	SISO Fading Channel	Existing models automatically replace instances of Multipath Rayleigh Fading Channel with SISO Fading Channel.
Multipath Rician Fading Channel	SISO Fading Channel is loaded, retaining parameter settings for legacy model	SISO Fading Channel	Existing models automatically replace instances of Multipath Rayleigh Fading Channel with SISO Fading Channel.
oqpskmod	Still runs	comm.OQPSKModulator	Replace all instances of oqpskmod with comm.OQPSKModulator.
oqpskdemod	Still runs	comm.OQPSKDemodulator	Replace all instances of oqpskdemod with comm.OQPSKDemodulator.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>commsrc.pn</code>	Still runs	<code>comm.PNSequence</code>	Replace all instances of <code>commsrc.pn</code> with <code>comm.PNSequence</code> .
<code>commscope.eyediagram</code>	Warns	<code>comm.EyeDiagram</code>	Replace all instances of <code>commscope.eyediagram</code> with <code>comm.EyeDiagram</code> .
<code>eyescope</code>	Warns	<code>comm.EyeDiagram</code>	Replace all instances of <code>eyescope</code> with <code>comm.EyeDiagram</code> . For more information, see Eye Diagram Measurements.
<code>commscope.ScatterPlot</code>	Warns	<code>comm.ConstellationDiagram</code>	Replace all instances of <code>commscope.ScatterPlot</code> with <code>comm.ConstellationDiagram</code> .
<code>comm.CPMCarrierPhaseSynchronizer</code>	Warns	<code>comm.CarrierSynchronizer</code>	Replace all instances of <code>comm.CPMCarrierPhaseSynchronizer</code> with <code>comm.CarrierSynchronizer</code> .
<code>modem.qamdemod</code>	Warns	<code>qamdemod</code>	Replace all instances of <code>modem.qamdemod</code> with <code>qamdemod</code> .
<code>modem.qammod</code>	Warns	<code>qammod</code>	Replace all instances of <code>modem.qammod</code> with <code>qammod</code> .

The following functionality has been removed.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>mimochan</code>	Errors	<code>comm.MIMOChannel</code>	Replace all instances of <code>mimochan</code> with <code>comm.MIMOChannel</code> .
<code>comm.PSKCarrierPhaseSynchronizer</code>	Errors	<code>comm.CarrierSynchronizer</code>	Replace all instances of <code>comm.PSKCarrierPhaseSynchronizer</code> with <code>comm.CarrierSynchronizer</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
comm.MuellerMullerTimingSynchronizer	Errors	comm.SymbolSynchronizer	Replace all instances of comm.MuellerMullerTimingSynchronizer with comm.SymbolSynchronizer.
comm.GardnerTimingSynchronizer	Errors	comm.SymbolSynchronizer	Replace all instances of comm.GardnerTimingSynchronizer with comm.SymbolSynchronizer.
comm.EarlyLateGateTimingSynchronizer	Errors	comm.SymbolSynchronizer	Replace all instances of comm.EarlyLateGateTimingSynchronizer with comm.SymbolSynchronizer.
Squaring Timing Recovery	Errors	Symbol Synchronizer	Replace all instances of Squaring Timing Recovery with Symbol Synchronizer.
Early-Late Gate Timing Recovery	Errors	Symbol Synchronizer	Replace all instances of Early-Late Gate Timing Recovery with Symbol Synchronizer.
Gardner Timing Recovery	Errors	Symbol Synchronizer	Replace all instances of Gardner Timing Recovery with Symbol Synchronizer.
Mueller-Muller Timing Recovery	Errors	Symbol Synchronizer	Replace all instances of Mueller-Muller Timing Recovery with Symbol Synchronizer.

R2017a

Version: 6.4

New Features

Compatibility Considerations

WINNER II Channel Model: Model and simulate spatially defined MIMO channels

Releases R2017a and R2016b (since October 2016) add the WINNER II Channel Model for Communications System Toolbox. Using WINNER II channel models, you can model and simulate spatially defined channels for multi-user MIMO wireless systems. You can specify an arbitrary number of base stations (BS) and mobile stations (MS) together with their geometry and location information.

The channel model enables you to simulate both line-of-sight (LOS) and non-LOS propagation conditions. It also enables you to apply multiple indoor and outdoor propagation scenarios. You can perform channel filtering in a streaming fashion with WINNER-generated channel coefficients.

The channel model supports:

- RF frequencies up to 6 GHz and signal bandwidths up to 100 MHz
- LOS and non-LOS propagation
- 12 indoor and outdoor propagation scenarios
- Arbitrarily large antenna arrays (for massive MIMO applications)
- Isotropic, dipole, and custom-defined antenna element patterns
- Linear, circular, and custom-defined antenna arrays

You can get the WINNER II Channel Model for Communications System Toolbox download from Add-On Explorer. For more information, see [Get Add-Ons and Manage Your Add-Ons](#).

USRP E310 Support: Prototype and test USRP E310 software-defined radio (SDR) systems

The Communications System Toolbox Support Package for USRP Embedded Series Radio is available for releases R2017a and R2016b (since October 2016). With this support package, you can design, prototype, and verify practical wireless communications systems on USRP embedded series radio hardware.

Key Features

- Use the USRP embedded series radio as an I/O peripheral to transmit and receive real-time arbitrary waveforms using MATLAB System objects or Simulink blocks.
- Transmit and receive RF signals out of the box, enabling the testing of SDR designs quickly, under real-world conditions.
- Transmit and receive data on one or two channels.
- Easy configuration of the RF radio settings.
- Burst mode option for acquiring high-bandwidth signals.
- In Simulink, customize and prototype SDR algorithms deploying partitioned hardware-software co-design implementations across the ARM processor and the FPGA fabric of the USRP device (Windows operating system only).
- Application examples for getting started.

Required MathWorks Products

To use the I/O mode features of the support package, the following MathWorks® products are required:

- MATLAB
- Communications System Toolbox
- Signal Processing Toolbox™
- DSP System Toolbox

To work with the hardware-software co-design workflow, the following MathWorks products are also required:

- Simulink
- To target the FPGA fabric on the device: HDL Coder™ Support Package for Xilinx Zynq Platform (requires HDL Coder)
- To target the ARM processor on the device: Embedded Coder Support Package for Xilinx Zynq Platform (requires Embedded Coder, Simulink Coder, and Embedded Coder Support Package for ARM Cortex®-A Processors)

Required Third-Party Tools

- Third-party tools required by the I/O mode features are automatically downloaded and installed during the support package installation.
- To work with the hardware-software co-design workflow, you must manually install the following third-party tools:
 - Xilinx Vivado® development tools version 2015.4
 - Xilinx SDK development tools version 2015.4

Hardware Support

Development Board	I/O Peripheral Support	Hardware-Software Co-Design Support
USRP E310	Yes	Yes

For more information, see Communications System Toolbox Support Package for USRP® Embedded Series Radio.

Bundled USRP Radios: Synchronize multiple USRP radios in frequency and time for transmission and reception

Since R2016b (since October 2016), the Communications System Toolbox Support Package for USRP Radio enables you to bundle multiple radios to support Multiple Input Multiple Output (MIMO) on more than two channels. Synchronize frequency and timing of bundled channels by connecting the multiple radios to an external 10 MHz clock source and an external PPS source.

MIMO using multiple radios is available for N-series and X-series USRP radios.

For more information, see Multiple Channel Input and Output Operations (Communications System Toolbox Support Package for USRP Radio).

Example of Proposed 5G Modulations: Simulate end-to-end links by using universal filtered multicarrier (UFMC) and filtered OFDM (F-OFDM)

The 5G Waveforms with LTE example demonstrates the integration of the new F-OFDM and UFMC waveforms into an existing LTE processing chain in place of Cyclic-Prefix OFDM (CP-OFDM).

The F-OFDM vs. OFDM Modulation example compares Orthogonal Frequency Division Multiplexing (OFDM) with filtered OFDM (F-OFDM) and highlights the merits of the new candidate modulation scheme for emergent Fifth Generation (5G) communication systems.

Packetized Modem Example: Model PHY and MAC layer operations in transmission of packetized waveforms

The Packetized Modem with Data Link Layer example demonstrates a PHY and MAC model that transmits packetized QPSK.

Communications System Toolbox Support Package for Analog Devices ADALM-Pluto Radio: Prototype and test ADALM-Pluto radio systems

The Communications System Toolbox Support Package for Analog Devices ADALM-Pluto Radio enables you to design, prototype, and verify practical wireless communications systems on a software-defined radio (SDR). With this support package, you can use a Pluto SDR as a standalone peripheral for live RF data I/O over USB.

Radio Broadcast Data System Transmitter: Generate RBDS-compliant waveforms

Generate configurable standard-compliant baseband RBDS waveforms in MATLAB using the `comm.RBDSWaveformGenerator` System object. Such waveforms can be modulated at the 57 kHz band of the baseband FM radio signals.

RBDS Modulation and Demodulation: Modulate and demodulate RBDS waveforms using the FM broadcast modulator and demodulator

To modulate the baseband RBDS waveform:

- In MATLAB, set the `RBDS` property of the `comm.FMBroadcastModulator` System object to `true`.
- In Simulink, select the **RBDS modulation** parameter of the FM Broadcast Modulator Baseband block.

To demodulate the RBDS waveform:

- In MATLAB, set the `RBDS` property of the `comm.FMBroadcastDemodulator` System object to `true`.

-
- In Simulink, select the **RBDS demodulation** parameter of the FM Broadcast Demodulator Baseband block.

The RBDS waveforms are modulated at the 57 kHz band.

MIMO-OFDM Precoding with Phased Arrays Example

The MIMO-OFDM Precoding with Phased Arrays example demonstrates using phased arrays in a MIMO-OFDM communication system employing beamforming. The Phased Array System Toolbox™ is used to model the transmit and receive antenna arrays. The channel is modeled using the WINNER II Channel Model for Communications System Toolbox.

Tab Completion: Complete parameter names and options in select MATLAB function calls

You can now use tab completion in select MATLAB function calls.

Remove antenna limitations on comm.OFDMModulator and comm.OFDMDemodulator

Maximum number of antenna limitations have been removed for comm.OFDMDemodulator and comm.OFDMModulator System object usage.

Compatibility Considerations

Removal of the maximum number of antennas constraint means you can model MIMO systems larger than 8-by-8. A warning is issued if the output data will exceed 1e6 bytes.

Parameter names changed for comm.FMBroadcastModulator and comm.FMBroadcastDemodulator

This change renames two fields returned by the info method for comm.FMBroadcastModulator and comm.FMBroadcastDemodulator. DecimationFactor and InterpolationFactor have been renamed to AudioDecimationFactor and AudioInterpolationFactor, and two more fields (RBDSInterpolationFactor and RBDSDecimationFactor) have been added for RBDS.

Compatibility Considerations

Up to R2016b, the info method for comm.FMBroadcastModulator and comm.FMBroadcastDemodulator returned a structure containing two fields: DecimationFactor and InterpolationFactor.

In R2017a, the info method for these functions returns a structure containing four fields: AudioDecimationFactor, AudioInterpolationFactor, RBDSInterpolationFactor, and RBDSDecimationFactor.

Functionality	New Behavior (as of R2017a)	Old Behavior (up to R2016b)	Compatibility Considerations
<pre>fmbMod = comm.FMBroadcastModu lator; fmbMod.info</pre>	<pre>Returns: >> fmbMod.info ans = struct with fields: AudioDecimationFacto r: 1 AudioInterpolationFa ctor: 5 RBDSDecimationFactor : 19 RBDSInterpolationFac tor: 384</pre>	<pre>Returned: >> fmbMod.info ans = struct with fields: DecimationFactor: 1 InterpolationFactor: 5</pre>	<p>Existing code that assigns the field names returned by the info method must be updated. Replace all instances of old field names with new field names.</p>
<pre>fmbDemod = comm.FMBroadcastDemo dulator; fmbDemod.info</pre>	<pre>Returns: >> fmbDemod.info ans = struct with fields: AudioDecimationFacto r: 20 AudioInterpolationFa ctor: 19 RBDSDecimationFactor : 20 RBDSInterpolationFac tor: 19</pre>	<pre>Returned: >> fmbDemod.info ans = struct with fields: DecimationFactor: 20 InterpolationFactor: 19</pre>	<p>Existing code that assigns the field names returned by the info method must be updated. Replace all instances of old field names with new field names.</p>

Functionality being removed

These functions will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>commscope.ScatterPlot</code>	still runs	<code>comm.ConstellationDiagram</code>	Replace all instances of <code>commscope.ScatterPlot</code> with <code>comm.ConstellationDiagram</code> .
<code>commscope.eyediagram</code>	still runs	<code>comm.EyeDiagram</code>	Replace all instances of <code>commscope.eyediagram</code> with <code>comm.EyeDiagram</code> .
<code>eyescope</code>	still runs	<code>comm.EyeDiagram</code>	Replace all instances of <code>eyescope</code> with <code>comm.EyeDiagram</code> .

Communications System Toolbox Support Package for Xilinx FPGA-Based Radio: Support package being removed

Communications System Toolbox Support Package for Xilinx FPGA-Based Radio will be removed in a future release.

Communications System Toolbox Support Package for Xilinx Zynq-Based Radio: FMCOMMS1 support removed

Hardware support for the Analog Devices FMCOMMS1 Rev B/C RF card has been removed.

R2016b

Version: 6.3

New Features

Bug Fixes

Compatibility Considerations

Eye Diagram Object and Block: Measure signal quality, and visualize histograms and bathtub curves

The `comm.EyeDiagram` System object and Eye Diagram block enable you to plot eye diagrams and characterize signal quality through a variety of measurements. In addition, you can visualize noise and jitter histograms as well as vertical and horizontal bathtub curves.

Baseband File Reader and Writer: Record captured baseband signals as files and use the files to test wireless designs

The Baseband File Reader and Baseband File Writer blocks `comm.BasebandFileReader` and `comm.BasebandFileWriter` System objects support file operations with a new binary file type.

UFMC and FBMC Examples: Characterize the performance of 5G modulation techniques

The FBMC vs. OFDM Modulation and UFMC vs. OFDM Modulation examples demonstrate the performance of universal filtered multicarrier (UFMC) and filter bank multicarrier (FBMC) modulation schemes.

Preamble Detector: Determine the location of the preamble in a packet

The `comm.PreambleDetector` System object and Preamble Detector block determine the location of the preamble in a communication packet.

I/Q Imbalance: Apply amplitude and phase imbalance to baseband signal

The `iqimbal` function applies an in-phase and quadrature (I/Q) amplitude and phase imbalance to baseband signals.

MATLAB Compiler support added to constellation diagram

The `comm.ConstellationDiagram` System object now supports MATLAB Compiler.

Printer support added to constellation and eye diagrams

The Constellation Diagram and Eye Diagram blocks and `comm.ConstellationDiagram` and `comm.EyeDiagram` System objects now support printing to figures and printers.

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 will continue to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `comm.AGC System` object named `agc`, then you call the `System` object as a function with that name.

```
agc = comm.AGC;  
rxSig = agc(txSig);
```

The equivalent operation using the `step` method is:

```
agc = comm.AGC;  
rxSig = step(agc, txSig);
```

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

Functionality being removed

These functions have been removed.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>randint</code>	Errors	<code>randi</code>	Replace all instances of <code>randint</code> with <code>randi</code> .

R2016a

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

EVM Measurements: Measure error vector magnitude with added flexibility

The EVM System object and block now support:

- Flexible measurement intervals
- Reference constellations that can be used instead of a reference signal
- N-dimensional input signals
- Variable-sized input signals
- Scalar inputs

Eye Diagram Block: Plot histograms of time-varying signals

Histogram plots can now be generated by the Eye Diagram block using one of three oversampling methods.

Quadrature Amplitude Modulation: Modulate binary data and demodulate binary and soft-decision outputs

The `qammod` and `qamdemod` functions support these additional capabilities:

- Soft-decision demodulation (`qamdemod` only)
- Bit inputs and outputs
- Constellation plots
- Code generation (requires MATLAB Coder)

Note The `qammod` and `qamdemod` functions will not support the initial phase input argument in a future release.

Communications System Toolbox Support Package for USRP Radio: Transmit and receive RF signals using B200mini

The Communications System Toolbox Support Package for USRP Radio now supports the B200mini radio by Ettus Research™. This radio is similar to the B200 but has a smaller form factor. It supports 1×1 communication links.

Class definition now available for `comm.LDPCDecoder` System object

You can now view the MATLAB code for `LDPCDecoder.m` in the `<matlabroot>/toolbox/comm/comm/+comm` folder.

Note By default, `comm.LDPCDecoder` does not support code generation. To generate code, specify the `ParityCheckMatrix` property as a non-sparse row-column index matrix.

For example, you can create an LDPC decoder for the DVB-S.2 standard by using this syntax:

```
dec = comm.LDPCDecoder(dvbs2ldpc(1/2, 'indices'));
```

Because `dvbs2ldpc(1/2, 'indices')` returns a non-sparse index matrix, the object, `dec`, can be used in code generation.

Function definitions now available for `qammod`, `qamdemod`, and `vitdec`

You can now view the MATLAB code for the `qammod`, `qamdemod`, and `vitdec` functions in the `<matlabroot>/toolbox/comm/comm` folder.

Functions now supporting code generation

If you have MATLAB Coder, you can generate code for these functions:

- `convenc`
- `vitdec`
- `dpskmod`
- `dpskdemod`

Inherited option removed from filters

The `Inherited` option has been removed from the **Input Processing** parameter for these blocks:

- Raised Cosine Transmit Filter
- Raised Cosine Receive Filter
- Ideal Rectangular Pulse Filter
- Windowed Integrator

Functionality being removed

The following features will be removed in a future release.

Function, Block, or System Object	Use This Instead
<code>modem.qammod</code>	<code>qammod</code>
<code>modem.qamdemod</code>	<code>qamdemod</code>
<code>comm.CPMCarrierPhaseSynchronizer</code>	<code>comm.CarrierSynchronizer</code>
CPM Phase Recovery	Carrier Synchronizer

These functions have been removed.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>fec.bchdec</code>	Errors	<code>comm.BCHDecoder</code>	Replace all instances of <code>fec.bchdec</code> with <code>comm.BCHDecoder</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>fec.bchenc</code>	Errors	<code>comm.BCHEncoder</code>	Replace all instances of <code>fec.bchenc</code> with <code>comm.BCHEncoder</code> .
<code>fec.ldpcdec</code>	Errors	<code>comm.LDPCDecoder</code>	Replace all instances of <code>fec.ldpcdec</code> with <code>comm.LDPCDecoder</code> .
<code>fec.ldpcenc</code>	Errors	<code>comm.LDPCEncoder</code>	Replace all instances of <code>fec.ldpcenc</code> with <code>comm.LDPCEncoder</code> .
<code>fec.rsdec</code>	Errors	<code>comm.RSDecoder</code>	Replace all instances of <code>fec.rsdec</code> with <code>comm.RSDecoder</code> .
<code>fec.rsenc</code>	Errors	<code>comm.RSEncoder</code>	Replace all instances of <code>fec.rsenc</code> with <code>comm.RSEncoder</code> .

Communications System Toolbox Support Package for Xilinx Zynq-Based Radio updates (Introduced in April 2016)

Hardware/Software Co-Design

Design algorithms for hardware and software implementation on SDR hardware. Hardware-software (HW/SW) co-design involves implementing algorithms on the Zynq radio platform that are partitioned across the ARM and the FPGA fabric. When deployed, you can control and tune parameters on the HDL logic. For more information, see Hardware-Software Co-Design.

HW/SW co-design is supported for Simulink on Windows only.

This feature requires the following MathWorks software:

- Communications System Toolbox Support Package for Xilinx Zynq-Based Radio
- HDL Coder Support Package for Xilinx Zynq Platform (requires HDL Coder)
- Embedded Coder Support Package for Xilinx Zynq Platform (requires Embedded Coder and, for use with HW/SW co-design, Simulink Coder and the ARM Cortex-A support package)

For more about HW/SW co-design requirements, see Hardware-Software Co-Design for Zynq SDR Applications.

R2015b

Version: 6.1

New Features

Bug Fixes

Compatibility Considerations

cdma2000 and 1xEV-DO Waveform Generators: Model the physical layer of North American CDMA standards and provide reference channels and waveforms

Define cdma2000 and 1xEV-DO standards compliant reference channels and waveforms.

Coarse Frequency Compensator: Correct carrier frequency offsets in PAM, PSK, and QAM signals

Correct for coarse carrier frequency offsets in PAM, PSK, and QAM signals. Used with the `CarrierSynchronizer` System object, you can accurately and quickly compensate for a wide range of frequency offsets.

Polynomial Strings: Specify Galois field polynomials as strings for error correction coding and sequence generation

In addition to binary and numeric vectors, you can specify polynomials as strings for coding, sequence generation, and scrambling functions. The new syntax allows you to specify polynomials in either ascending or descending order. This improvement allows you to use polynomial specifications as they appear in text books and journal articles.

AGC object and block have simplified interfaces, better dynamic range, and faster convergence times

The AGC System object and block are improved to incorporate a simplified interface, tolerate a significantly larger input signal power range, and converge more quickly.

Compatibility Considerations

The algorithm and some properties have been changed in this release. The properties and behavior of the previous releases can be accessed by setting the hidden `LegacyMode` property to `true`. By default, `LegacyMode` is `false`. The properties associated with the two legacy mode states are summarized.

Property	LegacyMode = true	LegacyMode = false
<code>AdaptationStepSize</code>		X
<code>ReferenceLevel</code>		X
<code>AveragingLength</code>		X
<code>MaximumGain</code>	X	X
<code>DetectorMethod</code>	X	
<code>LoopMethod</code>	X	
<code>UpdatePeriod</code>	X	
<code>StepSize</code>	X	
<code>GainOutputPort</code>	X	

Note For Simulink models in which the output gain port is enabled, the legacy mode is automatically enabled. This is required because the port is not available from the updated AGC block.

Improved interface on the constellation diagram object and block facilitates easier application of reference constellations and EVM/MER measurements

The improvements to the constellation diagram object and block allow you to easily access EVM and MER measurements. In addition, you can now display common reference constellations by selecting them from a drop down menu.

Source blocks output frames of contiguous time samples but do not use the frame attribute

Source blocks output frames of contiguous time samples but do not use the frame attribute. Frame processing is still supported.

For the Bernoulli Binary Generator, Poisson Integer Generator, and the Random Integer Generator blocks, the following changes were made:

- Removed **Frame-based outputs** and **Interpret vector parameters as 1-D** parameters. Blocks always output a sample-based 2-D vector.
- Introduced **Source of initial seed** parameter.

To use the default MATLAB random number generator, leave the **Source of initial seed** parameter set to Auto. To set an initial seed, set **Source of initial seed** to Parameter and then set the **Initial seed** value.

- Behavior of the random number generator is changed. The statistics are improved.

For the Poisson Integer Generator block, the **Lambda** parameter is now **Poisson parameter (lambda)**. For the Random Integer Generator block, the **M-ary number** parameter is now **Set size**.

These source blocks do not support the Upgrade Advisor. To update your model to use the new functionality, replace each block as follows:

- 1 Open the Simulink Library Browser and add the new source block to your model from the Random Data Sources sublibrary of the Comm Sources library.
- 2 Configure the new block to use the same settings as the original source block.
- 3 Replace the old block with the new one.

The **Frame-based outputs** parameter was removed for these blocks:

- Barker Code Generator
- Gold Sequence Generator
- Hadamard Code Generator
- Kasami Sequence Generator
- OVSF Code Generator
- PN Sequence Generator

- Walsh Code Generator

They always output sample-based 2-D vectors. These blocks can be upgraded using the Upgrade Advisor.

Functionality being changed or removed

- “mimochan will be removed” on page 15-4
- “Noise Generator blocks will be removed” on page 15-4

mimochan will be removed

The following functions are being removed:

Function	Use This Instead
mimochan	comm.MIMOChannel

Noise Generator blocks will be removed

Warns

The Gaussian Noise Generator, Rayleigh Noise Generator, and Rician Noise Generator, and Uniform Noise Generator blocks will be removed in a future release. This table shows the recommended replacement mapping.

Block	Use This Instead
Gaussian Noise Generator	MATLAB Function block and randn function
Rayleigh Noise Generator	MATLAB Function block and randn function
Rician Noise Generator	MATLAB Function block and randn function
Uniform Noise Generator	MATLAB Function block and rand function

For more information, see <https://www.mathworks.com/help/releases/R2015b/comm/ug/sources-and-sinks.html#bsxk1g0-1>.

Communications System Toolbox Support Package for RTL-SDR Radio Updates (v 15.2.0)

- “Deploy SDR models for standalone applications” on page 15-4
- “Code deployment to external host hardware” on page 15-5

Deploy SDR models for standalone applications

Create standalone applications to simulate SDR models on target machines that do not have MATLAB installed. Create a standalone executable by using the codegen function.

Key features:

- Generate an executable file for validation and testing.
- Create static or dynamic library files for reusing or sharing algorithms.

MATLAB Coder required. Your executable may require deploying additional MATLAB libraries. All operating systems supported by MATLAB Coder are supported for RTL-SDR code generation. See [Generating Standalone C/C++ Executables from MATLAB Code](#) for details.

Code deployment to external host hardware

Deploy code to external hardware for prototyping and faster processing. With Simulink, Simulink Support Package for Raspberry Pi Hardware , and the Communications System Toolbox Support Package for RTL-SDR Radio , generate and deploy code on Raspberry Pi hardware with a connected RTL-SDR radio.

For more information, see the documentation for Communications System Toolbox Support Package for RTL-SDR Radio.

Communications System Toolbox Support Package for USRP Radio Updates (v 15.2.0)

2x2 MIMO Support for USRP Radio B210 and X Series Boards (Introduced April 2015)

The Communications System Toolbox Support Package for USRP Radio now supports 2x2 MIMO operations for the B210, X300, and X310 boards from Ettus Research.

Communications System Toolbox Support Package for Xilinx Zynq-Based Radio Updates (v 15.2.0)

- “PicoZed SDR Support” on page 15-5
- “Simultaneous transmission and reception on a single board” on page 15-6
- “Third-Party Software Support Update” on page 15-6

PicoZed SDR Support

Support for PicoZed SDR platform is now available with the Communications System Toolbox Support Package for Xilinx Zynq-Based Radio. This feature support for the combination of a carrier board and a system-on-module (SOM) enables you to take your design from the prototype stage all the way up to deployment in the field using the Support Package for Xilinx Zynq-Based Radio and MATLAB or Simulink.

Key features include:

- PicoZed SDR platform as an I/O peripheral for streaming RF signals
- Burst mode option for high bandwidth signal acquisition
- Fast prototyping
- Easily transition from prototype to deployment
- Automated workflow for customizing FPGA fabric using HDL Coder (Xilinx Vivado® Design Suite required. See "Hardware and Software Support" in the Communications System Toolbox Support Package for Xilinx Zynq-Based Radio documentation to determine the version you need.)
- Works with all AD9361 application examples included with this support package

Supported PicoZed SDR platforms:

Product Name	Avnet Part Number	Description
PicoZed SDR 7035/AD9361	AES-PZSDR-Z7035-AD9361-G	Z7035 / AD9361 SOM

Simultaneous transmission and reception on a single board

To transmit and receive signals at the same time on the same board, use the `transmitRepeat` function together with normal receive functionality using the support package blocks and System objects. For an example of this feature, see "Transmit and Receive LTE MIMO Using a Single Analog Devices AD9361/AD9364" (shipped with the Communications System Toolbox Support Package for Xilinx Zynq-Based Radio).

Third-Party Software Support Update

For FPGA Targeting, the following software updates are required:

- Xilinx Vivado development tools version 2014.4.
- Xilinx SDK development tools version 2014.4.

R2015a

Version: 6.0

New Features

Bug Fixes

Compatibility Considerations

Symbol Timing Synchronizer: Correct for symbol timing clock skew between a transmitter and receiver

This release adds a new `comm.SymbolSynchronizer` System object and Symbol Synchronizer block to provide symbol timing synchronization between a single-carrier transmitter and receiver.

Carrier Synchronizer: Synchronize phase and frequency on a received waveform

This release adds a `comm.CarrierSynchronizer` System object and Carrier Synchronizer block to provide carrier phase and frequency synchronization between a single-carrier transmitter and receiver for PSK and QAM modulation schemes.

Baseband and Broadcast FM: Modulate and demodulate baseband and broadcast FM signals

This release adds the following System objects and blocks to provide baseband and broadcast FM modulation and demodulation capability.

- `comm.FMBroadcastModulator`
- `comm.FMBroadcastDemodulator`
- `comm.FMModulator`
- `comm.FMDemodulator`
- FM Broadcast Modulator
- FM Broadcast Demodulator
- FM Modulator
- FM Demodulator

Interactive QAM Example: Simulate an end-to-end QAM link with RF impairments and corrections

This release adds an interactive MATLAB example, featuring a graphical user interface, which shows the effects of RF impairments and corrections on an end-to-end QAM link simulation. See End-to-End QAM Simulation.

Communications System Toolbox Simulink Model Template: Automatically configure the Simulink environment for communications modeling

The Communications System Toolbox Simulink model template enables reuse of settings, including configuration parameters. Create models from the template to encourage best practices and take advantage of previous solutions to common problems. Instead of the default canvas of a new model, use the template model to create a skeletal model using settings recommended for Communications System Toolbox.

You can use the built-in template or create templates from models that you already configured for your environment or application.

For more information, see [Configure the Simulink Environment for Communications Models](#).

Library for HDL-supported Communications System Toolbox blocks

To find Communications System Toolbox blocks that support HDL code generation, from the Simulink library browser, open the 'Communications System Toolbox HDL Support' library. Alternatively, at the MATLAB command prompt, enter `commhdllib`.

The blocks in this library have parameters set for HDL code generation. To generate HDL code, you must have an HDL Coder license.

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 Communications System Toolbox software changes for frame-based processing:

- "Input processing parameter set to Inherited" on page 16-3
- "Rate options parameter set to Inherit from input" on page 16-4

Input processing parameter set to Inherited

Setting the `Input processing` parameter to `Inherited` now causes an error in these blocks:

- AWGN Channel
- Gaussian Filter
- Windowed Integrator
- Derepeat
- Ideal Rectangular Pulse Filter
- Raised Cosine Transmit Filter
- Raised Cosine Receive Filter
- Repeat

Compatibility Considerations

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 signal 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.

For more information, see the DSP System Toolbox release notes.

Rate options parameter set to Inherit from input

Setting the **Rate options** parameter to `Inherit from input` now causes an error for these blocks:

- M-FSK Modulator Baseband
- M-FSK Demodulator Baseband
- OQPSK Modulator Baseband
- OQPSK Demodulator Baseband
- CPM Modulator Baseband
- CPM Demodulator Baseband
- GMSK Modulator Baseband
- GMSK Demodulator Baseband
- MSK Modulator Baseband
- MSK Demodulator Baseband
- CPFSK Modulator Baseband
- CPFSK Demodulator Baseband
- Repeat
- Derepeat
- Raised Cosine Receive Filter
- Raised Cosine Transmit Filter

Compatibility Considerations

To ensure consistent results for models created in older 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`.

Simulink Upgrade Advisor

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

Class definitions now available for MATLAB-authored System objects

You can now view the MATLAB code for the following System object class definitions in the <matlabroot>/toolbox/comm/comm/+comm folder:

- ACPR.m
- AGC.m
- AWGNChannel.m
- BarkerCode.m
- BinarySymmetricChannel.m
- CarrierSynchronizer.m
- CCDF.m
- ConstellationDiagram.m
- DifferentialDecoder.m
- DifferentialEncoder.m
- DiscreteTimeVCO.m
- FMBroadcastDemodulator.m
- FMBroadcastModulator.m
- FMDemodulator.m
- FMModulator.m
- GeneralQAMTCMModulator.m
- GoldSequence.m
- HadamardCode.m
- HelicalDeinterleaver.m
- HelicalInterleaver.m
- IQImbalanceCompensator.m
- LTEMIMOChannel.m
- MemorylessNonlinearity.m
- MIMOChannel.m
- OFDMDemodulator.m
- OFDMModulator.m
- OVSFCode.m
- PhaseNoise.m
- PSKCoarseFrequencyEstimator.m
- PSKTCMModulator.m
- QAMCoarseFrequencyEstimator.m
- RaisedCosineReceiveFilter.m
- RaisedCosineTransmitFilter.m

- RayleighChannel.m
- RectangularQAMTCMModulator.m
- RicianChannel.m
- RSDecoder.m
- RSEncoder.m
- SphereDecoder.m
- SymbolSynchronizer.m
- ThermalNoise.m
- TurboDecoder.m
- TurboEncoder.m
- WalshCode.m

You can also view the MATLAB code for the following System object class definitions in the `<matlabroot>/toolbox/comm/comm/+comm/+internal` folder:

- CDF.m
- CDFEvaluation.m
- CoarseFrequencyEstimatorBase.m
- ConstellationBase.m
- DopplerVisualization.m
- FadingChannel.m
- FMModemBase.m
- GoldSequenceXor.m
- MeasureBase.m
- MultiBandFilter.m
- OFDMBase.m
- PathGainsVisualization.m
- Percentile.m
- PowerMeasurements.m
- Probability.m
- RaisedCosineFilterBase.m

Functionality being changed or removed

The following functions are being removed:

Function	Use This Instead
rsdecof	comm.RSDecoder
rsencof	comm.RSEncoder

The following blocks will be removed in a future release.

Block	Use This Instead
M-PSK Phase Recovery	Carrier Synchronizer
Squaring Timing Recovery	Symbol Synchronizer
Early-Late Gate Timing Recovery	Symbol Synchronizer
Gardner Timing Recovery	Symbol Synchronizer
Mueller-Muller Timing Recovery	Symbol Synchronizer

The following System objects will be removed in a future release.

System object	Use This Instead
<code>comm.PSKCarrierPhaseSynchronizer</code>	<code>comm.CarrierSynchronizer</code>
<code>comm.EarlyLateGateTimingSynchronizer</code>	<code>comm.SymbolSynchronizer</code>
<code>comm.GardnerTimingSynchronizer</code>	<code>comm.SymbolSynchronizer</code>
<code>comm.MuellerMullerTimingSynchronizer</code>	<code>comm.SymbolSynchronizer</code>

Support Package for USRP Radio B Series and X Series Board Support (Introduced November 2014)

The Communications System Toolbox Support Package for USRP Radio now supports the B200, B210, X300, and X310 boards from Ettus Research.

Key Features

- System object and block support for B Series and X Series Radio SDR development:
 - SDRu Receiver
 - SDRu Transmitter
 - `comm.SDRuReceiver`
 - `comm.SDRuTransmitter`
- Updated UHD support version 003.007.003
- Functions `findsdru` and `probesdru` updated to find and report information for Bus Series and X Series radios.
- Function `sdruLoad` updated to provide firmware/FPGA loading support for Bus Series and X Series radios.

These blocks and System objects provide SISO support and enable you to configure the master clock rate.

Note The master clock rate is different for the Bus Series and X Series radios than it is for the N Series radios. As such, the interpolation and decimation factors specified for N Series radios give different results for the Bus Series and X Series radios. Adjust your interpolation and decimation factors for USRP blocks and System objects to achieve the same baseband rates that you used for the N Series radios.

Limitations

- Communications System Toolbox Support Package for USRP Radio is not supported on Windows 8 operating systems.

Support Package for Xilinx Zynq-Based Radio (Introduced November 2014)

Design and prototype SDR applications with Zynq-based radio hardware. Supports the use of Zynq-based radio as an I/O peripheral to send and receive arbitrary waveforms from and to MATLAB and Simulink. Includes MATLAB System objects, Simulink blocks, and a full Zynq-based hardware design to transmit and receive data from the hardware using a Gigabit Ethernet connection. In addition, this support package enables you to customize the FPGA logic for prototyping your SDR applications from Simulink using HDL Coder (FPGA Targeting).

For more information, see Xilinx Zynq-Based Radio Support from Communications System Toolbox.

Supported Hardware and Software

Hardware Support

Development Board	RF Boards	I/O Peripheral Support	FPGA Targeting Support
Xilinx Zynq-7000 All Programmable SoC ZC706 Evaluation Kit	<ul style="list-style-type: none"> • Analog Devices FMCOMMS1 Rev B/C • Analog Devices FMCOMMS2 • Analog Devices FMCOMMS3 Avnet® Zynq-7000 All Programmable SoC / AD9361 Software-Defined Radio Systems Development Kit (includes ZC706 and FMCOMMS3)	Yes	Yes
Avnet ZedBoard™	<ul style="list-style-type: none"> • Analog Devices FMCOMMS1 Rev B/C • Analog Devices FMCOMMS2 • Analog Devices FMCOMMS3 Avnet Zynq-7000 All Programmable SoC / AD9361 Software-Defined Radio Evaluation Kit (includes ZedBoard and FMCOMMS2)	Yes	No

Caution ZedBoard Rev B and earlier revisions must have a heat sink attached. (Rev C and later revisions have the heat sink attached already.) If your board has an FMC radio attached but no heat sink, the Zynq processor overheats and stops working.

To acquire a heat sink you can attach yourself, see the Avnet Express website: CTS BDN09-3CB/A01 Extruded Heat Sink.

For more information, see <http://zedboard.org/content/updated-rev-c-schematicbom>.

Note If you find that the FMCOMMS RF card tends to fall out of the slot on the Zynq development board, especially when using a heavy antenna, use the standoffs that come with the SDR development kit.

Required Third-Party Software

There is no required third-party software to use any of the supported radios in I/O mode.

For FPGA Targeting (ZC706 only), the following software is required:

- Xilinx Vivado development tools version 2013.4.
- Xilinx SDK development tools version 2013.4.

Note You must install Xilinx SDK at the same time that you install Vivado.

Required MathWorks Products

For all Support Package for Xilinx Zynq-based Radio software functionality, the following MathWorks products are required:

- MATLAB
- Signal Processing Toolbox
- DSP System Toolbox
- Communications System Toolbox

For FPGA Targeting, the following products are also required:

- Simulink
- HDL Coder

Recommended products:

- MATLAB Coder

R2014b

Version: 5.7

New Features

Bug Fixes

Compatibility Considerations

I/Q Imbalance Compensator System object and block that remove I/Q amplitude and phase imbalance

This release adds an I/Q imbalance compensator to remove the amplitude and phase imbalance between the in-phase and quadrature components of a modulated signal. In addition to the compensator System object and block, two blocks and two functions were added which convert an imbalance into a compensator coefficient and vice versa.

Eye Diagram block that plots eye diagrams faster than its predecessor

This release adds a new Eye Diagram Simulink block to the Comm Sinks library.

Compatibility Considerations

The Eye Diagram block is a replacement for the Discrete-Time Eye Diagram Scope block. When existing models are loaded for the first time, the new eye diagram will automatically replace the old Discrete-Time Eye Diagram Scope.

Channel visualization for plotting impulse response, frequency response, and Doppler spectrum added to the Rayleigh, Rician, and MIMO Channel System objects

Visualization capabilities for the `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects and for the MIMO Channel block have been added for this release.

Sum-of-sinusoids modeling technique added to the Rayleigh, Rician, and MIMO Channel System objects

This release adds sum-of-sinusoids modeling to the `comm.MIMOChannel`, `comm.RayleighChannel`, and `comm.RicianChannel` System objects. Sum-of-sinusoids is ideally suited to modeling bursty channels and is an addition to the filtered Gaussian noise technique.

Trajectory diagram visualization added to the Constellation Diagram block and System object

This release adds a signal trajectory diagram capability to the Constellation Diagram block and the `comm.ConstellationDiagram` System object.

Compatibility Considerations

The enhanced Constellation Diagram block replaces the Discrete-Time Signal Trajectory Scope block. When existing models are loaded for the first time, the Constellation Diagram block with the **Show Signal Trajectory** option enabled will automatically replace the old Discrete-Time Signal Trajectory Scope block.

Support Package for RTL-SDR Radio Update

Linux Support

With release R2014b, support has been added for using the Support Package for RTL-SDR Radio on Linux operating systems.

Versions of Linux supported by MATLAB can be found in System Requirements.

Mac Support

With release R2014b, support has been added for using the Support Package for RTL-SDR Radio on Mac operating systems.

Versions of Mac supported by MATLAB can be found in System Requirements.

R2014a

Version: 5.6

New Features

Bug Fixes

Compatibility Considerations

OFDM modulator and demodulator System objects and blocks

This release adds OFDM modulation and demodulation capability by the addition of System objects and blocks. For more information, see the `comm.OFDMModulator` and `comm.OFDMDemodulator` System object Help pages.

DC blocker System object and block

The release adds a new DC blocker System object and block. For more information, see the `dsp.DCBlocker` Help page.

Direct and nondirect modes for HDL-optimized CRC generator and detector

This release allows the selection of either the direct or non-direct algorithm for CRC checksum calculations for the HDL-optimized CRC generator and detector System objects and blocks. For more information, see the `comm.HDLCRCDetector` and the `comm.HDLCRCGenerator` System object Help pages.

Additional featured examples such as 802.11 OFDM synchronization and HDL Optimized QAM Transmitter and Receiver

Additional featured examples:

- 802.11 synchronization
- HDL Optimized QAM Transmitter and Receiver

Generate HDL code from hardware-optimized 64-QAM transmitter and receiver. This example addresses real-world communications issues and generates HDL code for FPGA implementation (HDL Coder license required).

- HDL Optimized QPSK Transmitter

This example shows how Simulink blocks that support HDL code generation can be used to implement the baseband processing of a digital communications transmitter (HDL Coder license required).

APP Decoder System object parameter change

Beginning in release R2012a, the `Algorithm` property replaced the `MetricMethod` property for the `comm.APPDecoder` System object. At this time, an error will occur in legacy code that uses the `MetricMethod` property.

Compatibility Considerations

If you have any existing System object code that uses the `MetricMethod` property, you must change the property to `Algorithm`.

GPU System Object Support in System Block

GPU System objects are now supported in the System Block. The following System objects are supported in the current release:

- `comm.gpu.AWGNChannel`
- `comm.gpu.BlockDeinterleaver`
- `comm.gpu.BlockInterleaver`
- `comm.gpu.ConvolutionalDeinterleaver`
- `comm.gpu.ConvolutionalEncoder`
- `comm.gpu.ConvolutionalInterleaver`
- `comm.gpu.PSKDemodulator`
- `comm.gpu.PSKModulator`
- `comm.gpu.TurboDecoder`
- `comm.gpu.ViterbiDecoder`

See System Block Support for GPU System Objects.

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 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.

Compatibility Considerations

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`

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.

Support Package for RTL-SDR Radio (v 14.1.0)

Design and prototype software-defined radio (SDR) systems using MATLAB and Simulink with the Communications System Toolbox Support Package for RTL-SDR Radio.

For full access to features and documentation, download the support package from the Hardware Support page. To get help for the RTL-SDR Radio support package after you install it, enter `help sdr` at the MATLAB command line.

- “Key Features” on page 18-4
- “Blocks and System Objects” on page 18-4
- “RTL-SDR Examples” on page 18-5
- “Hardware and Software Requirements” on page 18-5

Key Features

- RTL-SDR radio as an I/O peripheral to receive streaming RF signals
- Configurable center frequency and sample rate
- NooElec™ NESDR Mini USB Stick (R820T) and NooElec NESDR Nano USB Stick (R820T) SDR devices with frequency range 30MHz - 1.8GHz
- Compatible with other RTL-SDR USB radios (for example, Terratec T-Stick E4000).
- Several application examples for getting started

Blocks and System Objects

- Simulink radio receiver block: RTL-SDR Receiver
- MATLAB radio System object: `comm.SDRRTLReceiver`

RTL-SDR Examples

- Spectrum Analysis with RTL-SDR Radio for MATLAB and Simulink
- Frequency Offset Calibration with RTL-SDR Radio for MATLAB and Simulink
- FM Monophonic Receiver with RTL-SDR Radio for MATLAB and Simulink
- FM Stereo Receiver with RTL-SDR Radio for MATLAB and Simulink
- FRS/GMRS Walkie-Talkie Receiver with RTL-SDR Radio for MATLAB and Simulink

Enter `sdrexamples` at the MATLAB command prompt for a full index of SDR support package examples.

Hardware and Software Requirements

For both MathWorks and third-party software and hardware requirements, see RTL-SDR Support from Communications System Toolbox.

Support Package for Xilinx FPGA-Based Radio updates (v 14.1.0)

- “Intermediate frequency tuning” on page 18-5
- “DC blocking filter” on page 18-6
- “QPSK targeting examples” on page 18-6

Intermediate frequency tuning

This feature supports a second stage tuning for both transmit and receive data paths. The tuner is configurable at run-time (tunable). It has a finer resolution compared to the primary tuner on RF card, and the ability to remove unwanted interference from the pass band of interest.

- Transmitter and Receiver blocks: Set the **Intermediate Frequency** parameter in the block mask.

The intermediate frequency (IF) tuner allows you to account for the error in tuning between target frequency and actual frequency.

- Transmitter and Receiver System objects: Set the `IntermediateFrequency` property for the System object. For example:

```
so = comm.SDRADIFMCOMMSTransmitter;  
so.IntermediateFrequency = txIFValue
```

See the reference pages for `comm.SDRADIFMCOMMSReceiver`, `comm.SDRADIFMCOMMSTransmitter`, or `comm.SDREpiqBitsharkReceiver`.

- With the HDL Coder workflow advisor (for Simulink only): Choose to include or not include the Intermediate Frequency tuner in the FPGA when using the targeting workflow.

At step 4.1, Set SDR Options:

- For transmit, select **Include transmitter intermediate frequency tuner**.
- For receiver, select **Include receiver intermediate frequency tuner**.

DC blocking filter

Choose to bypass the DC bias removal filter. Use this feature when the filter is also blocking some signal and you need to use a different DC bias compensation scheme. By default, this option is not selected, which means to include the automatic DC blocking filter.

- Blocks: Select parameter **Bypass DC blocking filter**.

See the reference pages for the Analog Devices FCOMMS Receiver block, the Analog Devices FCOMMS Transmitter block, or the Epiq Bitshark Receiver block.

- System objects: Set property `BypassDCBlockingFilter` to true.

See the reference pages for `comm.SDRADIFMCOMMSReceiver`, `comm.SDRADIFMCOMMSTransmitter`, or `comm.SDREpiqBitsharkReceiver`.

QPSK targeting examples

- **Targeting HDL Optimized QPSK Receiver with SDR Platform:** Learn how to model an HDL-optimized QPSK receiver and prototype it on the SDR hardware using the HDL Coder workflow advisor.
- **Targeting HDL Optimized QPSK Transmitter with SDR Platform:** Learn how to model an HDL-optimized QPSK transmitter and prototype it on the SDR hardware using the HDL Coder workflow advisor.

R2013b

Version: 5.5

New Features

Bug Fixes

Compatibility Considerations

Simulink blocks for MIMO channel, sphere decoder, and constellation diagram

This release includes a new MIMO Channel, sphere decoder, and constellation diagram blocks.

The MIMO Channel block filters an input signal using a multiple-input multiple-output (MIMO) multipath fading channel. For more information, see MIMO Channel.

The Sphere Decoder block offers MIMO receiver processing for communications systems using spatial multiplexing with high data rates, such as 802.11n, LTE, and WiMAX. This implementation offers maximum likelihood performance with reduced complexity. For more information, see Sphere Decoder.

The Constellation Diagram block plots constellation diagrams and provides the ability to perform EVM and MER measurements. The Constellation Diagram block replaces the Discrete-Time Scatter Plot block. For more information, see Constellation Diagram.

Compatibility Considerations

The Constellation Diagram block display enforces a 1:1 aspect ratio. The Discrete-Time Scatter Plot block, which the Constellation Diagram block replaces, does not enforce a 1:1 aspect ratio. For a non-unity display aspect ratio, you can use the Simulink XY Graph block.

Code generation for all MIMO channel Doppler spectra

The `comm.MIMOChannel` System object now generates C code for the following Doppler spectra:

- Rounded
- Bell
- Asymmetric Jakes
- Restricted Jakes
- Gaussian
- BiGaussian
- Flat
- Jakes

Open-loop PSK and QAM carrier synchronizers in MATLAB

This release provides new open-loop carrier synchronizer System objects, which allow you to estimate and compensate for carrier offset due to transceiver impairments. For more information, see:

- `comm.PSKCoarseFrequencyEstimator`
- `comm.QAMCoarseFrequencyEstimator`

HDL-Optimized QPSK Receiver with Captured Data

The example HDL Optimized QPSK Receiver with Captured Data shows how to optimize an QPSK receiver for HDL code generation and hardware implementation. The HDL-optimized model shows a

QPSK receiver that addresses real-world communications issues like carrier frequency, phase offset, and timing recovery for the hardware implementation.

Raised cosine transmit and receive filter System objects

This release provides new raised cosine transmit and receive filter System objects. For more information, see:

- `comm.RaisedCosineTransmitFilter`
- `comm.RaisedCosineReceiveFilter`

Rayleigh and Rician fading channel System objects

This release provides new fading channel System objects. For more information, see:

- `comm.RayleighChannel`
- `comm.RicianChannel`

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.

Compatibility Considerations

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

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.

Compatibility Considerations

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.

Block parameter prompt changes for raised cosine filter blocks

In this release, several block parameter prompts on the Raised Cosine Transmit Filter block and the Raised Cosine Receive Filter block have changed.

How to map old block property names to new block property names

To map the old Raised Cosine Transmit Filter block parameter prompts to the new block parameter prompts, refer to the following table.

Old Parameter Name	New Parameter Name	Notes
Filter type	Filter shape	N/A
Group delay (number of symbols)	Filter span in symbols	Set the Filter span in symbols as twice the value of the Group delay (number of symbols) parameter.
Upsampling factor (N)	Samples per symbol	N/A
Filter gain	N/A	The block only allows a user-specified gain.

To map the old Raised Cosine Receive Filter block parameter prompts to the new block parameter prompts, refer to the following table.

Old Parameter Name	New Parameter Name	Notes
Filter type	Filter shape	N/A
Group delay (number of symbols)	Filter span in symbols	Set the Filter span in symbols as twice the value of the Group delay (number of symbols) parameter.
Output mode	N/A	By default, the new block acts as if you select Downsampling . If you have saved an old model with None selected, the new block sets the Decimation factor parameter to 1, implying no decimation.
Downsampling factor	Decimation factor	N/A
Sample offset	Decimation offset	N/A
Filter gain	N/A	The block only allows a user-specified gain.

Compatibility Considerations

The updated Raised Cosine Transmit Filter and Raised Cosine Receive Filter blocks design a unit energy filter and then apply the linear amplitude filter gain to the filter coefficients. If you open a model that was saved in a prior version of the software, the software updates the block parameters. The blocks set the **Filter span in symbols** as twice the value of the **Group delay (number of symbols)** parameter. Similarly, the blocks set the linear amplitude filter gain to use the same filter coefficients as the old model. If you define a parameter value using a variable, you should confirm that the variable propagates correctly after you open the model.

Each time you open a model that was created using a prior release, Simulink automatically sets the block parameter values to obtain the same filter coefficients. If you save the model, the updates become permanent. As a best practice, you should confirm that the parameter values of the filter blocks are valid before saving the updated models.

NumTransmitAntennas and NumReceiveAntennas properties added back to MIMOChannel System object.

In the previous release, the NumTransmitAntennas and NumReceiveAntennas properties were removed from the MIMO Channel System object. This release, the properties were added back to the object. For more information, see `comm.MIMOChannel`

Functionality Being Changed or Removed

Effective this release, you should not use the following block or functions when simulating digital communications systems.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Discrete-Time Scatter Plot block	Still runs	Constellation Diagram	Replace all instances of Discrete-Time Scatter Plot block with Constellation Diagram
Gaussian Filter block	Still runs	<code>gaussdesign</code> function and Discrete FIR Filter, FIR Interpolation, or FIR Decimation block	Replace all instances of Gaussian Filter block with Discrete FIR Filter, FIR Interpolation, or FIR Decimation blocks. Use <code>gaussdesign</code> to generate filter coefficients for these blocks.
<code>rcosfir</code>	Still runs	<code>rcosdesign</code>	Replace all instances of <code>rcosfir</code> with <code>rcosdesign</code> .
<code>rcosflt</code>	Still runs	<code>rcosdesign</code> function and either <code>filter</code> or <code>upfirdn</code> functions	Replace all instances of <code>rcosflt</code> with <code>rcosdesign</code> and either <code>filter</code> or <code>upfirdn</code> .
<code>rcosiir</code>	Still runs	<code>rcosdesign</code> function for FIR raised cosine filters	Replace all instances of <code>rcosiir</code> with <code>rcosdesign</code> .
<code>rcosine</code>	Still runs	<code>rcosdesign</code>	Replace all instances of <code>rcosine</code> with <code>rcosdesign</code> .

Migrate Code from `firrcos` and `rcosfir` to `rcosdesign`

This section helps you update your legacy code using `firrcos` and `rcosfir` to use the recommended `rcosdesign`.

`firrcos` to `rcosdesign`

Design an order 16 FIR raised cosine filter with a carrier frequency of 1 kHz, a roll-off factor of 0.25, and a sampling frequency of 8 kHz.

```
N = 16;
Fc = 1000;
R = 0.25;
Fs = 8000;
b1 = firrcos(N,Fc,R,Fs,'rolloff','normal');
```

To obtain the identical filter using the recommended `rcosdesign`, use

```
b1n = rcosdesign(R, N/(Fs/Fc/2), Fs/Fc/2,'normal');
b1n = b1n / max(b1n) / (Fs/Fc/2);
```


The following code constructs the same raised cosine filter as the previous example. This example demonstrates the equivalence between the input arguments for `firrcos` and `rcosdesign`. The plot and comparison of the filter coefficient values show that the two filters are identical.

```
beta = R;
sps = Fs/(2*Fc);
span = N / sps;
b1n = rcosdesign(beta, span, sps,'normal');
b1n = b1n / max(b1n) / sps;figure
plot(b1)
hold on
plot(b1n, 'r-.')
grid on
legend('firrcos', 'rcosdesign');
max(abs(b1n-b1))
```

Design a square-root raised cosine filter using `firrcos` and obtain the identical filter using `rcosdesign`.

```
b2 = firrcos(N,Fc,R,Fs,'rolloff','sqrt');
b2n = rcosdesign(R, N/(Fs/Fc/2), Fs/Fc/2, 'sqrt');
b2n = b2n / max(b2n) * ((-1 ./ (pi.*(Fs/Fc/2)) .* (pi.*(R-1) - 4.*R)));
```

The following code constructs the same square-root raised cosine filter as the previous example. This example demonstrates the equivalence between the input arguments for `firrcos` and `rcosdesign`. The plot and comparison of the filter coefficient values show that the two filters are identical.

```
beta = R;
sps = Fs/(2*Fc);
span = N / sps;
b2n = rcosdesign(R, span, sps, 'sqrt');
b2n = b2n / max(b2n) * ((-1 ./ (pi.*sps) .* (pi.*(R-1) - 4.*R)));
figure
plot(b2)
hold on
plot(b2n, 'r-.')
grid on
legend('firrcos', 'rcosdesign')
max(abs(b2-b2n))
```

rcosfir to rcosdesign

Design a raised cosine filter using `rcosfir` with sampling period of 1 second, an oversampling rate of 6 (6 output samples for every input sample), and a roll-off factor of 0.3.

```
R = 0.3;
N_T = 4;
RATE = 6;
T = 1;
% filter length is 2*N_T*RATE+1
b3 = rcosfir(R, N_T, RATE, T, 'normal');
```

Design the same filter using the recommended `rcosdesign`.

```
b3n = rcosdesign(R, 2*N_T, RATE,'normal');
b3n = b3n / max(b3n);
```

The following code constructs the same raised cosine filter as the previous example. This example demonstrates the equivalence between the input arguments for `rcosfir` and `rcosdesign`. The plot and comparison of the filter coefficient values show that the two filters are identical.

```
beta = R;
sps = RATE;
span = 2*N_T;
b3n = rcosdesign(beta, span, sps, 'normal');
b3n = b3n / max(b3n)
figure
plot(b3)
hold on
plot(b3n, 'r-.')
grid on
legend('rcosfir', 'rcosdesign')
max(abs(b3-b3n))
```

Design a square-root raised cosine filter using `rcosfir` and obtain the identical filter using `rcosdesign`.

```
b4 = rcosfir(R, N_T, RATE, 1, 'sqrt');
b4n = rcosdesign(R, 2*N_T, RATE, 'sqrt');
b4n = b4n / max(b4n) * ((-1 ./ (pi.*RATE) .* (pi.*(R-1) - 4.*R))) * sqrt(RATE);
```

The following code constructs the same square-root raised cosine filter as the previous example. This example demonstrates the equivalence between the input arguments for `rcosfir` and `rcosdesign`. The plot and comparison of the filter coefficient values show that the two filters are identical.

```
beta = R;
sps = RATE;
span = 2*N_T;
b4n = rcosdesign(R, span, sps, 'sqrt');
b4n = b4n / max(b4n) * ((-1 ./ (pi.*sps) .* (pi.*(R-1) - 4.*R))) * sqrt(RATE)
figure
plot(b4)
hold on
plot(b4n, 'r-.')
grid on
legend('rcosfir', 'rcosdesign')
max(abs(b4-b4n))
```

Support Package for Xilinx FPGA-Based Radio

Design SDR applications for use with FPGA-based radio. Supports both fixed bitstream (Support Package for Xilinx FPGA-Based Radio software provides all logic) and custom bitstream (user-provided logic) workflows (SDR Targeting). This support package includes Simulink receiver and transmitter blocks for use with Simulink and receiver and transmitter System objects for use with MATLAB. These blocks and System objects enable communication with an FPGA-based radio, allowing for development work in software-defined radio applications.

Main Features

- Simulink blocks
 - Analog Devices FCOMMS Receiver
 - Analog Devices FCOMMS Transmitter
 - Epiq Bitshark Receiver
- System objects
 - `comm.SDRADIFCOMMSReceiver`

- `comm.SDRADIFMCOMMSTransmitter`
- `comm.SDREpiqBitsharkReceiver`
- SDR Targeting

SDR Targeting allows you to implement your baseband processing algorithm on the FPGA of the Xilinx development board. By moving part of all of your algorithm to the hardware , you will speed up the host processing. See Implement SDR Targeting.

- Examples
 - HDL Optimized QPSK Receiver with Captured Data demonstrates a hardware-friendly solution that performs baseband processing to handle a time-varying frequency offset and a time-varying symbol delay.
 - QPSK Transmitter and Receiver shows a digital communications system using QPSK modulation.
 - IEEE 802.11 WLAN - HDL Optimized Beacon Frame Receiver with Captured Data shows the reception of beacon frames in an 802.11 based wireless local area % network (WLAN).

Supported Hardware and Software

- Hardware support

FPGA Development Board	RF Board	Fixed Bitstream Support	SDR Targeting Support
Virtex-6 ML605	Epiq Bitshark RevB	Yes	Yes
Virtex-6 ML605	Epiq Bitshark RevC	Yes	Yes
Xilinx ML605	ADI FCOMMS1 RevB	Yes	Yes

- Software requirements
 - SDR fixed bitstream and SDR targeting are tested with Xilinx ISE 13.4.
 - For fixed bitstream, Xilinx iMPACT is required.

R2013a

Version: 5.4

New Features

Bug Fixes

Compatibility Considerations

Sphere Decoder System object for MIMO receiver processing

The Sphere Decoder System object offers MIMO receiver processing for communications systems using spatial multiplexing with high data rates, such as 802.11n, LTE, and WiMAX. This implementation offers maximum likelihood performance with reduced complexity.

For more information, see the `comm.SphereDecoder` Help page.

Constellation Diagram System object with measurements

The constellation diagram System object plots constellation diagrams and provides the ability to perform EVM and MER measurements. For more information, see the `comm.ConstellationDiagram` System object Help page.

LTE space-frequency block coding and LTE GPU-accelerated turbo coding examples

This release includes new LTE examples illustrating space-frequency block coding and GPU-accelerated turbo coding.

The LTE Downlink PDSCH with Transmit Diversity example highlights LTE Downlink PDSCH processing with transmit diversity, including two transmit antenna and four transmit antenna configurations.

The LTE Downlink Shared Channel Processing with GPU Acceleration example shows how you can use GPUs to accelerate bit error rate simulations.

In addition, the existing LTE PHY Downlink with Spatial Multiplexing example includes two new MATLAB-based implementations.

HDL code generation for CRC Generator, CRC Detector, RS Encoder, and RS Decoder System objects

Effective this release, the following System objects provide HDL code generation:

- `comm.HDLCRCDetector`
- `comm.HDLCRCGenerator`
- `comm.HDLRSDecoder`
- `comm.HDLRSEncoder`

To generate HDL code, you must have an HDL Coder license.

Variable-size support for AWGN, MIMO, and LTE MIMO Channel System objects

This release includes variable-size support for the AWGN, MIMO Channel, and LTE MIMO Channel System objects. This support enables you to:

- Vary the number of transmit and receive antennas, which is necessary for LTE modeling

-
- Vary the number of samples per channel, which is helpful for LTE and WiMAX modeling

For more information see:

- `comm.AWGNChannel`
- `comm.MIMOChannel`
- `comm.LTEMIMOChannel`

IEEE 802.11 WLAN - HDL optimized beacon frame receiver example with captured data

This example shows a hardware friendly model that receives beacon frames in an 802.11 wireless local area network (WLAN).

Automatic gain controller block and System object

This release includes a new automatic gain controller (AGC) block and System object. The AGC adaptively adjusts its gain to achieve a constant signal level at the output.

For more information see the AGC block and `comm.AGC` System object Help pages.

Additional CRC algorithm implementation

Effective this release, the CRC blocks and System objects support the direct algorithm, input byte reflection, checksum reflection, and final XOR operation. These features enable more straightforward Ethernet CRC generation and detection. For more information see:

- General CRC Generator
- `comm.CRCGenerator`
- General CRC Syndrome Detector
- `comm.CRCDetector`

ATSC digital television example

The ATSC Digital Television example shows the vestigial sideband modulation with 8 discrete amplitude levels (8-VSB) transmission subsystem of the Advanced Television Systems Committee (ATSC) digital television standard.

Disable second output port on APP Decoder

Beginning in this release, you can disable the second output port, containing coded bit log-likelihood ratios, on the APP Decoder System object and block.

For the System object, disable the `CodedBitLLROutputPort` property.

For the block, select the **Disable L(c) output port** check box.

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.

Dynamic memory allocation based on size

By default, dynamic memory allocation is now enabled for variable-size arrays whose size exceeds a configurable threshold. This behavior allows for finer control over stack memory usage. Also, you can generate code automatically for more MATLAB algorithms without modifying the original MATLAB code. The following System objects support dynamic memory allocation for C code generation:

- `comm.BPSKDemodulator`
- `comm.BPSKModulator`
- `comm.PSKDemodulator`
- `comm.PSKModulator`
- `comm.QPSKDemodulator`
- `comm.QPSKModulator`
- `comm.GeneralQAMDemodulator`
- `comm.GeneralQAMModulator`
- `comm.PAMDemodulator`
- `comm.PAMModulator`
- `comm.RectangularQAMDemodulator`
- `comm.RectangularQAMModulator`
- `comm.BitToInteger`
- `comm.IntegerToBit`
- `comm.OSTBCCCombiner`
- `comm.OSTBCEncoder`
- `comm.CRCDetector`
- `comm.CRCGenerator`
- `comm.ConvolutionalEncoder` (Dynamic memory allocation not supported for punctured applications.)
- `comm.ViterbiDecoder` (Dynamic memory allocation not supported for punctured applications.)
- `comm.TurboEncoder`

Compatibility Considerations

If you use scripts to generate code and you do not want to use dynamic memory allocation, you must disable it. For more information, see [Controlling Dynamic Memory Allocation](#).

Naming convention change for LTE examples

Effective this release, there is a new naming convention for LTE examples. See the following table for more information.

Example Title	Old File Name	New File Name
Downlink Transport Channel (DL-SCH) Processing	commlteDLSCH	LTEDLSCHExample
LTE PHY Downlink with Spatial Multiplexing	commlteDownlink	LTEDownlinkExample

Compatibility Considerations

Typing the old file names at the MATLAB command line no longer opens example models. To open the example models, you must type the new file names.

APP Decoder System Object parameter change

Beginning in release R2012a, the `Algorithm` property replaced the `MetricMethod` property for the `comm.APPDecoder` System object. At this time, any legacy code that uses the `MetricMethod` property generates a warning.

Compatibility Considerations

If you have any existing System object code that uses the `MetricMethod` property, you must use the `sysobjupdate` function to update your code. For more information, type `help sysobjupdate` at the MATLAB command line.

Functions to remain in the product

The following functions, which were previously announced for removal, will remain in the product.

- `bchdec`
- `bchenc`
- `dpskdemod`
- `dpskmod`
- `eyediagram`
- `oqpskdemod`
- `oqpskmod`
- `pamdemod`
- `pammod`

- pskdemod
- pskmod
- qamdemod
- qammod
- rsdec
- rsenc

Communications System Toolbox Functionality Being Changed or Removed

The following function will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
mimochan	Warns	comm.MIMOChannel	Replace all instances of mimochan with comm.MIMOChannel.

Update Legacy Code to use System object

For help updating your legacy code so that it uses the comm.MIMOChannel System object, see the following table.

Map mimochan Properties and Methods to comm.MIMOChannel

mimochan Property	comm.MIMOChannel Property	Note
NumTxAntennas	N/A	This information is derived from the TransmitCorrelationMatrix property.
NumRxAntennas	N/A	This information is derived from the ReceiveCorrelationMatrix property.
InputSamplePeriod	SampleRate	Sample rate is the reciprocal of the input sample period.
DopplerSpectrum	DopplerSpectrum	
MaxDopplerShift	MaximumDopplerShift	
PathDelays	PathDelays	
AvgPathGaindB	AveragePathGains	
TxCorrelationMatrix	TransmitCorrelationMatrix	
RxCorrelationMatrix	ReceiveCorrelationMatrix	

mimochan Property	comm.MIMOChannel Property	Note
KFactor	KFactor	
DirectPathDopplerShift	DirectPathDopplerShift	
DirectPathInitPhase	DirectPathInitialPhase	
NormalizePathGains	NormalizePathGains	
ResetBeforeFiltering	N/A	Use the <code>reset</code> method for the System object before calling the <code>step</code> method <pre>h = comm.MIMOChannel; step(h, ones(10,2)); reset(h); step(h, ones(20,2));</pre>
StorePathGains	N/A	Set the <code>PathGainsOutputPort</code> to <code>true</code> so the <code>step</code> method for the object outputs the path gains.
PathGains	N/A	Set the <code>PathGainsOutputPort</code> to <code>true</code> so the <code>step</code> method for the object outputs the path gains.
ChannelFilterDelay	N/A	Use the <code>info</code> method to display this information.
NumSamplesProcessed	N/A	Use the <code>info</code> method to display this information.
ChannelType	N/A	This read-only property was removed.

mimochan Method	comm.MIMOChannel Method
<code>filter</code>	<code>step</code>
<code>reset</code>	<code>reset</code>

Note `mimochan` and `comm.MIMOChannel` have different APIs. Refer to the following syntax examples when updating your legacy code:

mimochan	comm.MIMOChannel
<pre>chan = mimochan(2, 2, 1e-4, 60, [0 2.5e-4 3e-4]); chan.StorePathGains = 1;</pre>	<pre>h = comm.MIMOChannel(... 'SampleRate', 1e4, ... 'PathDelays', [0 2.5e-4 3e-4], ... 'AveragePathGains', [0 -2 -3], ... 'MaximumDopplerShift', 60, ... 'TransmitCorrelationMatrix', eye(2), ... 'ReceiveCorrelationMatrix', eye(2), ... 'PathGainsOutputPort', true);</pre>

mimochan	comm.MIMOChannel
<pre>y = filter(chan, ones(20, 2)); pathGains = chan.PathGains;</pre>	<pre>[y, pathGains] = step(h, ones(20, 2));</pre>

R2012b

Version: 5.3

New Features

Compatibility Considerations

Support for C code generation for all System objects in Communications System Toolbox

Effective this release, the following System objects provide C code generation:

- `comm.ACPR`
- `comm.BCHDecoder`
- `comm.CCDF`
- `comm.CPMCarrierPhaseSynchronizer`
- `comm.GoldSequence`
- `comm.LDPCDecoder`
- `comm.LDPCEncoder`
- `comm.LTEMIMOChannel`
- `comm.MemorylessNonlinearity`
- `comm.MIMOChannel`
- `comm.PhaseNoise`
- `comm.PSKCarrierPhaseSynchronizer`
- `comm.RSDecoder`
- `comm.ThermalNoise`

All CPU-based System objects in the Communications System Toolbox product generate C code. The GPU-based System objects do not generate C code.

Support for HDL code generation for Reed-Solomon encoder, decoder, and CRC detector blocks

Effective this release, the following blocks provide HDL code generation:

- General CRC Syndrome Detector HDL Optimized
- Integer-Input RS Encoder HDL Optimized
- Integer-Output RS Decoder HDL Optimized

To generate HDL code, you must have an HDL Coder license.

Support for HDL code generation for Rectangular QAM and PSK Demodulator System objects

Effective this release, the following System objects provide HDL code generation:

- `comm.BPSKDemodulator`
- `comm.QPSKDemodulator`
- `comm.PSKDemodulator`
- `comm.RectangularQAMDemodulator`

To generate HDL code, you must have an HDL Coder license.

LTE Zadoff-Chu sequence generator function

Communications System Toolbox includes a Zadoff-Chu sequence generator function. This function is useful when modeling 3GPP LTE physical layer characteristics, downlink primary synchronization signals, or the uplink reference signals and random access preamble sequences. For more information, see the `lteZadoffChuSeq` Help page.

LTE downlink shared channel example

The LTE PHY Downlink with Spatial Multiplexing shows the Downlink Shared Channel (eNodeB to UE) processing of the Long Term Evolution (LTE) physical layer (PHY) specifications developed by the Third Generation Partnership Project (3GPP). LTE-Advanced is one of the candidates for fourth generation (4G) communications systems, approved by the International Telecommunication Union (ITU), with expected downlink peak data rates in excess of 1Gbps (for Release 10 and beyond). Using the Release 10 specifications, this example highlights the multi-antenna transmission scheme that enables such high data rates.

Phase Noise block and System object, specifying phase noise spectrum with a vector of frequencies

The Phase Noise block and System object now have more flexibility for specifying spectral noise characteristics. You can specify a vector of phase noise levels, at more than one frequency value. Previously, the software allowed the specification of a single-phase noise level point. The new implementation enables more realistic noise modeling in your communications models, and allows you to visualize the phase noise spectrum that the block or System object generates.

IEEE 802.11 beacon with captured data example

This example shows reception of beacon frames in an 802.11 wireless local area network (WLAN). You can select one of several captured signals and view the data the beacon frame carries.

P25 spectrum sensing example

This example shows how to use cyclostationary feature detection to distinguish signals with different modulation schemes, including P25 signals. It defines four cases of signals: noise only, C4FM, CQPSK, and one arbitrary type. The example applies the detection algorithm to signals with different SNR values and determines when the signals can be classified as one of the four types.

MATLAB-based QPSK transceiver example

The QPSK Transmitter and Receiver example now includes a MATLAB implementation that uses System objects. This example models a digital communications system to simulate the QPSK transmitter - receiver chain. In particular, this example illustrates a method for tackling real-world wireless communication issues, such as: carrier frequency/phase offset, timing recovery, and frame synchronization.

Design Iteration Workflow

This example illustrates a design workflow and the typical iterations involved in designing a wireless communications system with the Communications System Toolbox. Because Communications System

Toolbox supports both MATLAB and Simulink, this examples showcases separate design iterations using MATLAB functions or Simulink models.

The workflow starts with a simple QPSK modulator system that transmits a signal through an AWGN channel and calculates the bit error rate. To make the system more realistic and improve system performance, the example gradually introduces Viterbi decoding, turbo coding, multipath fading channels, OFDM-based transmission and equalization, and multiple-antenna techniques.

Constellation method for modulator and demodulator System objects

Effective this release, modulator and demodulator System object have a `constellation` method. This method calculates or plots the ideal signal constellation, depending on object settings. The following System objects have the `constellation` method:

- `comm.PSKModulator`
- `comm.PSKDemodulator`
- `comm.RectangularQAMModulator`
- `comm.RectangularQAMDemodulator`
- `comm.PAMModulator`
- `comm.PAMDemodulator`
- `comm.QPSKModulator`
- `comm.QPSKDemodulator`
- `comm.BPSKModulator`
- `comm.BPSKDemodulator`
- `comm.OQPSKModulator`
- `comm.OQPSKDemodulator`
- `comm.gpu.PSKModulator`
- `comm.gpu.PSKDemodulator`

Specify initial states of Gold Sequence Generator and PN Sequence Generator System objects

You can specify the initial states for the PN Sequence Generator and Gold Sequence Generator System objects as inputs to the `step` method. You can use these System objects as scrambling sequence generators. For packet-based systems, including WiMAX and LTE, the initial conditions are a function of time. Therefore, for simulation purposes, you must specify the initial states as an input.

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 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.

Compatibility Considerations

If you need to save and restore simulation states, you may be able to use a corresponding Simulink block, instead of a System object.

Functionality Being Changed or Removed

The following function, which was previously announced for removal and warned at run time, has been removed from the product.

- `seqgen.pn`

The following functions will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>commmeasure.ACPR</code>	Warns	<code>comm.ACPR</code>	Replace all instances of <code>commmeasure.ACPR</code> with <code>comm.ACPR</code> .
<code>commmeasure.EVM</code>	Warns	<code>comm.EVM</code>	Replace all instances of <code>commmeasure.EVM</code> with <code>comm.EVM</code> .
<code>commmeasure.MER</code>	Warns	<code>comm.MER</code>	Replace all instances of <code>commmeasure.MER</code> with <code>comm.MER</code> .
<code>fec.bchdec</code>	Warns	<code>comm.BCHDecoder</code>	Replace all instances of <code>fec.bchdec</code> with <code>comm.BCHDecoder</code> .
<code>fec.bchenc</code>	Warns	<code>comm.BCHEncoder</code>	Replace all instances of <code>fec.bchenc</code> with <code>comm.BCHEncoder</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
fec.ldpcdec	Warns	comm.LDPCDecoder	Replace all instances of fec.ldpcdec with comm.LDPCDecoder.
fec.ldpcenc	Warns	comm.LDPCEncoder	Replace all instances of fec.ldpcenc with comm.LDPCEncoder.
fec.rsdec	Warns	comm.RSDecoder	Replace all instances of fec.rsdec with comm.RSDecoder.
fec.rsenc	Warns	comm.RSEncoder	Replace all instances of fec.rsenc with comm.RSEncoder.

Update Legacy Code to use System objects

For help updating your legacy code so that it uses the new System objects, refer to the following sections.

Map commmeasure.ACPR Properties and Methods to comm.ACPR

commmeasure.ACPR property	comm.ACPR property	Note
Fs	SampleRate	
MainChannelMeasBW	MainMeasurementBandwidth	
AdjacentChannelMeasBW	AdjacentMeasurementBandwidth	
MeasurementFilter	MeasurementFilterSource	
SpectralEstimatorOption	SpectralEstimation	
WindowOption	Window	
SidelobeAtten	SidelobeAttenuation	
FrequencyResolutionOption	FrequencyResolution	
FFTLlength	CustomFFTLlength	
	MainChannelPowerOutputPort(new property)	<p>When you set MainChannelPowerOutputPort to true, the main channel power measurement becomes an output.</p> <p>Note Previously, for the commmeasure.ACPR object, this was the second output argument.</p>

commmeasure.ACPR property	comm.ACPR property	Note
	AdjacentChannelPowerOutputPort (new property)	When you set AdjacentChannelPowerOutputPort to true, the adjacent channel power measurement becomes an output. Note Previously, for the commmeasure.ACPR object, this was the third output argument.
Type	N/A	This read-only property was removed.
FrameCount	N/A	This read-only property was removed.

commmeasure.ACPR method	comm.ACPR method
run	step
reset	reset
copy	clone
disp	N/A

Note commmeasure.ACPR and comm.ACPR have a different API. Refer to the following syntax examples when updating your legacy code:

commmeasure.ACPR	comm.ACPR	Note
commmeasure.ACPR	comm.ACPR	The default settings of the following are different: 'NormalizedFrequency' 'MainMeasurementBandwidth' 'AdjacentChannelOffset' 'AdjacentMeasurementBandwidth' 'MeasurementFilterSource'
<pre>h = commmeasure.ACPR(... 'PowerUnits','linear',... 'SpectralEstimatorOption','UseDefault',... 'SegmentLength',100); [act_ACPR, actMainPow, actAdjPow] = step(h,yPulse); fd = h.MeasurementFilter;</pre>	<pre>h = comm.ACPR(... 'PowerUnits','Watts',... 'SpectralEstimation','SpecifyWindowParameters',... 'SegmentLength',100,... 'MainChannelPowerOffset', true,... 'AdjacentChannelPowerOutputPort', true,... 'MeasurementFilterSource', 'property'); [act_ACPR, actMainPow, actAdjPow] = step(h,yPulse); fdnumerator = h.MeasurementFilter;</pre>	MeasurementFilter changes from a structure to a variable. Specify window parameters',...

Map commmeasure.EVM Properties and Methods to comm.EVM

commmeasure.EVM properties	comm.EVM properties	Note
NormalizationOption	Normalization	
AveragePower	AverageConstellationPower	
PeakPower	PeakConstellationPower	
RSMEVM	N/A	RSMEVM is an output.
MaximumEVM	MaximumEVMOutputPort	When you set MaximumEVMOutputPort to true, MaximumEVM becomes an output.
Percentile	XPercentileValue	XPercentileValue appears when you set the XPercentileEVMOutputPort to true.
PercentileEVM	XPercentileEVMOutputPort	When you set XPercentileEVMOutputPort to true, PercentileEVM becomes an output.
NumberOfSymbols	SymbolCountOutputPort	When you set SymbolCountOutputPort to true, NumberOfSymbols becomes an output.
Type	N/A	This read-only property was removed.

commmeasure.EVM methods	comm.EVM methods
update (no outputs)	step (multiple outputs)
reset	reset
copy	clone

Note commmeasure.evm and comm.evm have a different API. Refer to the following syntax examples when updating your legacy code:

commmeasure.EVM	comm.EVM
hEVM = commmeasure.EVM('Percentile', 90)	hEVM = comm.EVM('XPercentileEVMOutputPort', true, 'XPer
update(hEVM, rcv, xmv) rmsevm = hEVM.RMSEVM	rmsevm = step(hEVM, rcv, xmv)
update(hEVM, rcv, xmv) rmsevm = hEVM.RMSEVM maxevm = hEVM.MaximumEVM pevm = hEVM.PercentileEVM numsym = hEVM.NumberOfSymbols	[rmsevm,maxevm,pevm,numsym] = step(hEVM, rcv, xmv)

Map commmeasure.MER Properties and Methods to comm.MER

commmeasure.MER properties	comm.MER properties	Note
MERdb	N/A	MERdb is an output.
MinimumMER	MinimumMEROutputPort	When you set MinimumMEROutputPort to true, MinimumMER becomes an output.
Percentile	XPercentileValue	XPercentileValue appears when you set the XPercentileMEROutputPort to true.
PercentileMER	XPercentileMEROutputPort	When you set XPercentileMEROutputPort to true, PercentileMER becomes an output.
NumberOfSymbols	SymbolCountOutputPort	When you set SymbolCountOutputPort to true, NumberOfSymbols becomes an output.
Type	N/A	This read-only property was removed.

commmeasure.MER methods	comm.MER methods
update (no outputs)	step (multiple outputs)
reset	reset
copy	clone

Note commmeasure.MER and comm.MER have a different API. Refer to the following syntax examples when updating your legacy code:

commmeasure.MER	comm.MER
hMER = commmeasure.MER('Percentile', 90)	hMER = comm.MER('XPercentileMEROutputPort', true, ... 'XPercentileValue', 90)
update(hMER, rcv, xmv) merdb = hMER.MERdb	merdb = hMER(rcv, xmv)
update(hMER, rcv, xmv) merdb = hEVM.MERdb minimummer = hEVM.MinimumMER pmer = hEVM.PercentileMER numsym = hEVM.NumberOfSymbols	[merdb, minimummer, pmer, numsym] = hMER(rcv, xmv)

Map fec.bchenc Properties to comm.BCHEncoder

fec.bchenc property	comm.BCHEncoder property	Note
N	CodewordLength	

fec.bchenc property	comm.BCHEncoder property	Note
K	MessageLength	
T	The ErrorCorrectionCapability element of the Info method	
ShortenedLength	N/A	This information is included in the CodewordLength and MessageLength properties.
ParityPosition	N/A	Always 'end'.
PuncturePattern	PuncturePattern	This property appears when you set PuncturePatternSource to Property.
GenPoly	GeneratorPolynomial	This property appears when you set GeneratorPolynomialSource to Property.
Type	N/A	This read-only property was removed.

Note fec.bchenc and comm.BCHEncoder have a different API. Refer to the following syntax examples when updating your legacy code:

fec.bchenc	comm.BCHEncoder	Note
h=fec.bchenc	h = comm.BCHEncoder('CodewordLength', 7, 'MessageLength', 4, 'PuncturePatternSource', 'DefaultConfiguration', 'PuncturePattern', [1 1 1].', 'GeneratorPolynomialSource', 'property', 'GeneratorPolynomial', [1 0 1 1]);	Use this syntax to create the encoder. ...
enc = fec.bchenc(7,4); msg = [0 1 1 0]'; code = encode(enc,msg);	h = comm.BCHEncoder('CodewordLength', 7, 'MessageLength', 4, 'PuncturePatternSource', 'property', 'PuncturePattern', [1 1 1].', 'GeneratorPolynomialSource', 'property', 'GeneratorPolynomial', [1 0 1 1]); msg = [0 1 1 0]'; code = step(h,msg)	Use the step method to encode. ... • The step method replaces use of the encode function.
encShort = fec.bchenc(7,4); encShort.ShortenedLength = 1; msgShort = [0 1 1]'; codeShort = encode(encShort,msgShort);	h=comm.BCHEncoder(6,3); msg = [0 1 1]'; code = step(h,msg)	The shortened length information is included in the CodewordLength and MessageLength properties.

Map fec.bchdec Properties to comm.BCHDecoder

fec.bchdecproperty	comm.BCHDecoder property	Note
N	CodewordLength	
K	MessageLength	

fec.bchdecproperty	comm.BCHDecoder property	Note
T	The ErrorCorrectionCapability element of the Info method	This information is included in the CodewordLength and MessageLength properties.
ShortenedLength	N/A	
ParityPosition	N/A	
PuncturePattern	PuncturePattern	This property appears when you set PuncturePatternSource to Property.
GenPoly	GeneratorPolynomial	This property appears when you set GeneratorPolynomialSource to Property.
Type	N/A	This read-only property was removed.

Note fec.bchdec and comm.BCHDecoder have a different API. Refer to the following syntax examples when updating your legacy code:

fec.bchdec	fec.BCHDecoder	Note
h=fec.bchdec	h = comm.BCHDecoder('CodewordLength',4, 'PuncturePatternSource', 'PuncturePattern', [1 1 1], 'GeneratorPolynomialSource', 'GeneratorPolynomial', [1 0 1 1])	Use this syntax to generate the default configuration of fec.bchdec. 'property', ...
dec = fec.bchdec(7,4); code = [0 1 1 0 0 0 1].'; msg = decode(dec,code);	h = comm.BCHDecoder('CodewordLength',4, 'PuncturePatternSource', 'PuncturePattern', [1 1 1].', 'GeneratorPolynomialSource', 'GeneratorPolynomial', [1 0 1 1]); code = [0 1 1 0 0 0 1].'; msg = step(h,code)	Use this syntax to generate the default configuration of fec.bchdec. 'property', ... The PuncturePattern must be a row vector. • The step method replaces use of the decode function.
decShort = fec.bchdec(7,4) decShort.ShortenedLength = 1; code = [0 1 1 1 0 1].'; msg = decode(decShort,code);	h = comm.BCHDecoder('CodewordLength',3, 'PuncturePatternSource', 'PuncturePattern', [1 1 1].', 'GeneratorPolynomialSource', 'GeneratorPolynomial', [1 0 1 1]); code = [0 1 1 1 0 1].'; msg = step(h,code)	Use this syntax to generate the default configuration of fec.bchdec. 'property', ... The ShortenedLength property is included in the CodewordLength and MessageLength properties.

Map fec.Ldpcenc Properties to comm.LDPCDecoder

fec.Ldpcenc property	comm.LDPCDecoder property	Note
ParityCheckMatrix	ParityCheckMatrix	
BlockLength	N/A	This read-only property was removed.
NumInfoBits	N/A	This read-only property was removed.

fec.ldpcenc property	comm.LDPCDecoder property	Note
NumParityBits	N/A	This read-only property was removed.
EncodingAlgorithm	N/A	This read-only property was removed.

Note The comm.LDPCDecoder System object does not contain all the read-only properties of the old object. However, you can obtain the information from the ParityCheckMatrix.

fec.ldpcenc and comm.LDPCDecoder have a different API. Refer to the following syntax example when updating your legacy code:

fec.ldpcenc	comm.LDPCDecoder	Note
<pre>h1 = fec.ldpcenc; xin = ones(32400,1); yout1 = encode(h1,xin.')</pre>	<pre>h = comm.LDPCDecoder; xin = ones(32400,1); yout = step(h, xin)</pre>	<ul style="list-style-type: none"> The fec.ldpcenc object accepted a row vector input. The comm.LDPCDecoder System object accepts a column vector input. The step method replaces use of the encode function

Map fec.ldpcdec Properties to comm.LDPCDecoder

fec.ldpcdec property	comm.LDPCDecoder property	Note
ParityCheckMatrix	ParityCheckMatrix	
DecisionType	DecisionMethod	
OutputFormat	OutputValue	
DoParityChecks	IterationTerminationCondition	Select Parity check satisfied.
NumIterations	MaximumIterationCount	
ActualNumIterations	NumIterationsOutputPort	
FinalParityChecks	FinalParityChecksOutputPort	
BlockLength	N/A	This read-only property was removed.
NumInfoBits	N/A	This read-only property was removed.
NumParityBits	N/A	This read-only property was removed.

Note The comm.LDPCDecoder System object does not contain all the read-only properties of the old object. The ActualNumIterations and FinalParityChecks properties become outputs.

fec.ldpcdec and comm.LDPCDecoder have a different API. Refer to the following syntax example when updating your legacy code.

fec.ldpcdec	comm.LDPCDecoder	Note
<pre>h1 = fec.ldpcdec; yin = ones(64800,1); yout1 = decode(h1,yin.')</pre>	<pre>h = comm.LDPCDecoder yin = ones(64800,1); yout = step(h,yin)</pre>	<ul style="list-style-type: none"> The <code>fec.ldpcdec</code> object accepted a row vector input. The <code>comm.LDPCDecoder</code> System object accepts a column vector input. The <code>step</code> method replaces use of the <code>decode</code> function

Map fec.rsenc Properties to comm.RSEncoder

fec.rsenc	comm.RSEncoder	Note
N	CodewordLength	
K	MessageLength	
T	The ErrorCorrectionCapability element of the Info method	
ShortenedLength	N/A	This information is included in the CodewordLength and MessageLength properties.
ParityPosition	N/A	Always 'end'.
GenPoly	GeneratorPolynomial	This property appears when you set GeneratorPolynomialSource to Property.
Type	N/A	This read-only property was removed.

Note `fec.rsenc` and `comm.RSEncoder` have a different API. Refer to the following syntax examples when updating your legacy code:

fec.rsenc	comm.RSEncoder	Note
<pre>h=fec.rsenc</pre>	<pre>h = comm.RSEncoder('CodewordLength',3, ... 'PuncturePatternSource', 'default', 'PuncturePattern', ... [1 1 1].', 'GeneratorPolynomialSource', 'property', ... 'GeneratorPolynomial', [1 0 1 1]); h = comm.BCHEncoder('CodewordLength',7, 'MessageLength',4, ... 'PuncturePatternSource', 'property', 'PuncturePattern', ... [1 1 1].', 'GeneratorPolynomialSource', 'property', ... 'GeneratorPolynomial', [1 0 1 1]);</pre>	<p>Use this syntax to create the <code>comm.RSEncoder</code> object with the same properties as the <code>fec.rsenc</code> object.</p>

fec.rsenc	comm.RSEncoder	Note
<pre>enc = fec.rsenc(7,3); msg = [0 1 0]'; code = encode(enc,msg);</pre>	<pre>h = comm.RSEncoder('CodewordLength',3, 'PuncturePatternSource','PuncturePatternSource', [1 1 1 1].', 'GeneratorPolynomialSource','GeneratorPolynomial', [1 3 1 2 3]) msg = [0 1 0]'; code = step(h,msg)</pre>	<p>The <code>step</code> method replaces use of the <code>encode</code> function.</p>
<pre>encShort = fec.rsenc(7,3) encShort.ShortenedLength = 1; msgShort = [0 1]'; codeShort = encode(encShort,msgShort);</pre>	<pre>h = comm.RSEncoder('CodewordLength',2, 'PuncturePatternSource','PuncturePatternSource', [1 1 1 1].', 'GeneratorPolynomialSource','GeneratorPolynomial', [1 3 1 2 3]) msg = [0 1]'; code = step(h,msg)</pre>	<p>The <code>step</code> method replaces use of the <code>encode</code> function.</p>

Map fec.rsdec Properties to comm.RSDecoder

fec.rsdec	comm.RSDecoder	Note
N	CodewordLength	
K	MessageLength	
T	The ErrorCorrectionCapability element of the Info method	
ShortenedLength	N/A	This information is included in the CodewordLength and MessageLength properties.
ParityPosition	N/A	Always 'end'.
PuncturePattern	PuncturePattern	This property appears when you set PuncturePatternSource to Property.
GenPoly	GeneratorPolynomial	This property appears when you set GeneratorPolynomialSource to Property.
Type	N/A	This read-only property was removed.

Note `fec.rsdec` and `comm.RSDecoder` have a different API. Refer to the following syntax examples when updating your legacy code:

fec.rsdec	comm.RSDecoder	Note
<pre>h=fec.rsdec</pre>	<pre>h = comm.RSDecoder('CodewordLength',3, 'PuncturePatternSource','PuncturePatternSource', [1 1 1 1].', 'GeneratorPolynomialSource','GeneratorPolynomial', [1 3 1 2 3])</pre>	<p>Use this syntax to create the default configuration of <code>fec.rsdec</code>.</p>

fec.rsdec	comm.RSDecoder	Note
<pre>dec = fec.rsdec(7,3); code = [0 1 1 0 0 0 1].'; msg = decode(dec,code);</pre>	<pre>h = comm.RSDecoder('CodewordLength',7,'PuncturePatternSource','[1 1 1 1].','GeneratorPolynomialSource','[1 3 1 2 3]'); code = [0 1 1 0 0 0 1].'; msg = step(h,code)</pre>	<p>The short message length property is included in the CodewordLength and MessageLength properties.</p> <ul style="list-style-type: none"> The step method replaces use of the encode function.
<pre>decShort = fec.rsdec(7,3) decShort.ShortenedLength = 1; code = [0 1 1 1 0 1].'; msg = decode(decShort,code);</pre>	<pre>h = comm.RSDecoder('CodewordLength',7,'PuncturePatternSource','[1 1 1 1].','GeneratorPolynomialSource','[1 3 1 2 3]'); code = [0 1 1 1 0 1].'; msg = step(h,code)</pre>	<p>The short message length property is included in the CodewordLength and MessageLength properties.</p> <ul style="list-style-type: none"> The step method replaces use of the encode function.

Frame-Based Processing

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. For more information, see “Frame-Based Processing” on page 24-5.

R2012a

Version: 5.2

New Features

Compatibility Considerations

MIMO Multipath Fading Channel System Objects

The Communications System Toolbox product now includes a Multiple Input Multiple Output (MIMO) Multipath Fading Channel System object, `comm.MIMOChannel`. Multipath MIMO fading channels allow for design of communication systems with multiple antenna elements at the transmitter and receiver. For more information, see the `comm.MIMOChannel` Help page.

The product also includes an LTE MIMO Multipath Fading Channel System object, `comm.LTEMIMOChannel`. This object allows for design of communication systems with multiple antenna elements at the transmitter and receiver using the 3GPP Long Term Evolution (LTE) standard. For more information, see the `comm.LTEMIMOChannel` Help page.

Multi-H Support for CPM Modulator and Demodulator Simulink Blocks and MATLAB System Objects

The CPM Modulator Baseband and CPM Demodulator Baseband blocks and System objects now support Multi-H CPM modulation. These enhancements allow you to perform research and development work for communication systems designed with the ARTM, JTRS, or MIL-STD-188-181C communications standards. For more information, see:

- `comm.CPModulator`
- `comm.CPMDemodulator`
- CPM Modulator Baseband
- CPM Demodulator Baseband

GPU System Objects

This release adds new GPU System objects, which use a graphics processing unit (GPU) to procure simulation results more quickly than a CPU. These new objects include:

- `comm.gpu.ConvolutionalInterleaver`
- `comm.gpu.ConvolutionalDeinterleaver`
- `comm.gpu.ConvolutionalEncoder`
- `comm.gpu.PSKDemodulator`
- `comm.gpu.TurboDecoder`

MATLAB Compiler Support for GPU System Objects

In Release 2012a, you can use the MATLAB Compiler product with GPU System objects. With this capability, MATLAB Compiler software can generate standalone applications from MATLAB files, including files that contain GPU System objects.

Code Generation Support

The following System objects now support C code generation:

- `comm.BCHEncoder`
- `comm.RSEncoder`

The following function now supports C code generation:

- `bchgenpoly`

HDL Code Generation from MATLAB code

The following System objects now support HDL code generation:

- `comm.ViterbiDecoder`
- `comm.PSKModulator`
- `comm.BPSKModulator`
- `comm.QPSKModulator`
- `comm.rectangularQAMmodulator`
- `comm.ConvolutionalInterleaver`
- `comm.ConvolutionalDeinterleaver`

See also HDL Code Generation from MATLAB.

HDL Support For HDL CRC Generator Block

Release R2012a provides HDL code generation support for the new HDL CRC Generator block.

Enhancements for System Objects Defined by Users

This release contains enhancements for System objects defined by users.

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 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 runtime.

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 a System object's 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`.

Compatibility Considerations

Compatibility Considerations

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.

New and Enhanced Demos

The following demos are new or enhanced for this release:

- IEEE 802.11 WLAN - Beacon Frame simulates packetized, non-streaming transmission and reception of beacon frames in an 802.11-based wireless local area network (WLAN).
- IEEE 802.16-2009 WirelessMAN-OFDMA PHY Downlink PUSC simulates a downlink partial usage of subchannels (PUSC) Physical Layer communication from base station (BS) to two mobile stations. This demo uses variable-size signals to model dynamic channel allocation between the two users.
- QPSK Transmitter and Receiver implements a QPSK transmitter and receiver, including carrier and timing recovery.
- Digital Video Broadcasting - Cable (DVB-C) models part of the ETSI (European Telecommunications Standards Institute) EN 300 429 standard for cable system transmission of digital television signals.
- Downlink Transport Channel (DL-SCH) Processing models part of the transport channel processing for the Downlink Shared Channel (eNodeB to UE) of the Long Term Evolution (LTE) specifications developed by the Third Generation Partnership Project (3GPP) .
- Using GPUs To Accelerate Turbo Coding Bit Error Rate Simulations shows how you can use GPUs to dramatically accelerate bit error rate simulations.
- End to End System Simulation Acceleration Using GPUs compares four techniques that can be used to accelerate bit error rate (BER) simulations.

Functionality Being Changed or Removed

The following functions will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
rsdecof	Warns	comm.RSDecoder	Replace all instances of rsdecof with comm.RSDecoder.
rsencof	Warns	comm.RSEncoder	Replace all instances of rsencof with comm.RSEncoder.

The following functions, which were previously announced for removal in a future release, now warn at run time. You should not use these functions.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
rcosflt	Warns	fdesign.pulseshaping	<ul style="list-style-type: none"> Use fdesign.interpolator and fdesign.decimator to design multirate filters. Use fdesign.pulseshaping to design a single-rate raised cosine filter. Does not support IIR.
rcosiir	Warns	N/A	Do not use.
rcosine	Warns	fdesignpulseshaping	<ul style="list-style-type: none"> Use fdesign.interpolator and fdesign.decimator to design multirate filters. Use fdesign.pulseshaping to design a single-rate raised cosine filter. Does not support IIR.
bchdec	Warns	comm.BCHDecoder	
bchenc	Warns	comm.BCHEncoder	
rsdec	Warns	comm.RSDecoder	
rsenc	Warns	comm.RSEncoder	
randint	Warns	randi	Use randi to generate matrix of uniformly distributed random integers

Several functions, which were previously announced for removal in a future release and warned at run time, have been removed from the Communications System Toolbox product. To see the full list of these removed functions, expand the following section.

Removed Functions

- ademod
- ademodce
- amod
- amodce
- apkconst
- bchdeco
- bchenco
- bchpoly
- constlay
- convdeco
- convenco
- ddemod
- ddemodce
- demodmap
- dmod
- dmodce
- eyescat
- flxor
- gen2abcd
- gfplus
- htruthb
- imp2sys
- lineprob
- modmap
- oct2gen
- qaskdeco
- qaskenco
- randbit
- rscore
- rsdeco
- rsdecode
- rsenco
- rsencode
- rspoly
- sim2gen

-
- `sim2gen2`
 - `sim2logi`
 - `sim2tran`
 - `simpassbandex`
 - `simsum`
 - `simsum2`
 - `viterbi`
 - `vitshort`

The following function, which was previously announced for removal in a future release, will remain in the Communications System Toolbox product.

- `rcosfir`

Frame-Based Processing

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. For more information, see “Frame-Based Processing” on page 24-5.

Inherited Option of the Input Processing Parameter Now Warns

Some Communications 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 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

To see a list of Communications System Toolbox blocks that contain the **Input processing** parameter, expand the following section.

Blocks with Input Processing Parameter

- AWGN Channel (with only two options)
- Derepeat

- Gaussian Filter
- Ideal Rectangular Pulse Filter
- Raised Cosine Receive Filter
- Raised Cosine Transmit Filter
- Windowed Integrator

Compatibility Considerations

Compatibility Considerations

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 `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Inherited Option of the Rate Options Parameter Now Warns

Some Communications System Toolbox blocks support single-rate or multirate processing. After the transition to the new paradigm for handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both single-rate and multirate processing have a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks received a new **Rate options** parameter in previous releases. You can set this parameter to `Enforce single-rate processing` or `Allow multirate processing`. The third choice, `Inherit from input (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 met for any block in your model:

- The **Rate options** parameter set to `Inherit from input (this choice will be removed - see release notes)`
- The input signal is sample-based
- The input signal is a scalar

To see a full list of Communications System Toolbox blocks that have a new **Rate options** parameter, expand the following section.

Blocks with Rate Options Parameter

- OQPSK Modulator Baseband
- OQPSK Demodulator Baseband
- CPM Modulator Baseband
- CPM Demodulator Baseband
- MSK Modulator Baseband
- MSK Demodulator Baseband
- GMSK Modulator Baseband
- GMSK Demodulator Baseband
- CPFSK Modulator Baseband
- CPFSK Demodulator Baseband
- M-FSK Demodulator Baseband
- M-FSK Modulator Baseband

Compatibility Considerations

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the `supdate` function. The function detects all blocks that have `Inherit from input` (this choice will be removed - see release notes) selected for the **Rate options** 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 **Rate options** parameter to `Enforce single-rate processing`. If the bit is 0 (samples), the function sets the parameter to `Allow multirate processing`.

In a future release, the frame bit and the `Inherit from input` (this choice will be removed - see release notes) option will be removed. At that time, the **Rate options** parameter in models that have not been upgraded will automatically be set to either `Enforce single-rate processing` or `Allow multirate processing`. 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.

R2011b

Version: 5.1

New Features

Bug Fixes

Compatibility Considerations

New Demos

- The Transceiver Simulation Acceleration demo illustrates simulation acceleration improvements by comparing simulation times using System objects with simulation times using MATLAB functions.
- The Parallel Concatenated Convolutional Coding: Turbo Codes demo now uses the Turbo Encoder and Turbo Decoder blocks and the accompanying MATLAB script uses the `comm.TurboEncoder` and `comm.TurboDecoder` System objects.

Turbo Codes

Communications System Toolbox now supports turbo codes. These error correction codes approach the Shannon limit, resulting in low error rates for transmission schemes with low signal-to-noise ratios. You can implement turbo codes using either MATLAB System objects or Simulink blocks:

- `comm.TurboDecoder`
- `comm.TurboEncoder`
- Turbo Decoder
- Turbo Encoder

USRP2 Migration

Support for the UDP-based USRP2 Transmitter and USRP2 Receiver blocks is being removed in release R2011b. New USRP™ blocks and System objects that work with USRP™ radios using the Universal Hardware Driver™ from Ettus Research™ are now available. These new blocks and objects support buffers with arbitrary frame size. If you have Communications System Toolbox, you can download and use these new blocks and System objects.

GPU System Objects

This release adds new GPU System objects, which use a graphics processing unit (GPU) to procure simulation results more quickly than a CPU. These new objects include:

- `comm.gpu.AWGNChannel`
- `comm.gpu.BlockDeinterleaver`
- `comm.gpu.BlockInterleaver`
- `comm.gpu.PSKModulator`
- `comm.gpu.ViterbiDecoder`

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.

Variable-Size Support

The following blocks now support variable-size input and/or output signals:

- APP Decoder
- AWGN Channel (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)
- CRC-N Generator
- CRC-N Syndrome Detector
- Error Rate Calculation
- General CRC Generator
- General CRC Syndrome Detector
- OSTBC Combiner
- OSTBC Encoder
- Turbo Decoder (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)
- Turbo Encoder (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)

The following blocks now support puncturing with variable-size signals:

- Convolutional Encoder
- Viterbi Decoder

The following System objects now support variable-size input and/or output signals:

- `comm.APPDecoder`
- `comm.ConvolutionalEncoder`
- `comm.CRCDetector`
- `comm.CRCGenerator`
- `comm.ErrorRate`
- `comm.OSTBCCombiner`
- `comm.OSTBCEncoder`
- `comm.TurboDecoder`
- `comm.TurboEncoder`
- `comm.ViterbiDecoder`

System Object Code Generation Support

The following System objects support code generation:

- `comm.BarkerCode`
- `comm.DifferentialDecoder`
- `comm.DifferentialEncoder`
- `comm.DiscreteTimeVCO`

- `comm.HadamardCode`
- `comm.OVSFCode`
- `comm.TurboEncoder`
- `comm.TurboDecoder`
- `comm.WalshCode`

Delayed Reset for Viterbi Decoder

The Viterbi Decoder block and Viterbi Decoder System object now have a delayed reset option. The delay in the reset action allows the block to support HDL code generation. To generate HDL code, you must have an HDL Coder license.

For the Viterbi Decoder block:

- Select **Enable reset input port**
- Select **Delay reset action to next time step**. This parameter only appears when you set the **Operation mode** parameter to `Continuous`.

The Viterbi Decoder block resets its internal state after decoding the incoming data.

For the `comm.ViterbiDecoder` System object

- Set `ResetInputPort` to `true`
- Set `DelayedResetAction` to `true`. This property only appears when you set the `ResetInputPort` property to `true`.
- Set `TerminationMethod` to `Continuous`

The Viterbi Decoder System object resets its internal state after decoding the incoming data.

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 inputs.

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`

This change affects the following System objects:

- `comm.IntegrateAndDumpFilter`
- `comm.PAMDemodulator`
- `comm.RectangularQAMDemodulator`

-
- `comm.GeneralQAMDemodulator`

Compatibility Considerations

Compatibility Consideration

All these System objects have their new `FullPrecisionOverride` property set to the default, `true`. If you had set any fixed-point properties to nondefault 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`.

APP Decoder System Object Parameter Change

For the `comm.APPDecoder` System object, the `Algorithm` property replaces the `MetricMethod` property. At this time, existing customer code continues to work; however, a warning prompts you to update the code.

Compatibility Considerations

Compatibility Consideration

If you have any existing System object code that uses the `MetricMethod` property, you should use the `sysobjupdate` function to update your code. For more information, type `help sysobjupdate` at the MATLAB command line.

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.

Compatibility Considerations

Compatibility Considerations

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.

Conversion of System Object Error and Warning Message Identifiers

For R2011b, error and warning message identifiers for System objects have changed in Communications System Toolbox software.

Compatibility Considerations

Compatibility Considerations

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, 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.

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, 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.

Frame-Based Processing

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. For more information, see “Frame-Based Processing” on page 24-5.

R2011a

Version: 5.0

New Features

Bug Fixes

Compatibility Considerations

Product Restructuring

The Communications System Toolbox product replaces two pre-existing products: Communications Blockset and Communications Toolbox. You can access archived documentation for both products on the MathWorks Web site.

LDPC Encoder and Decoder System Objects

This release adds new `comm.LDPCDecoder` and `comm.LDPCDecoder` System objects. These new System objects provide simulation of low-density, parity-check codes.

LDPC GPU Decoder System Object

This release adds a new `comm.gpu.LDPCDecoder` System object, which uses a graphics processing unit (GPU) to decode low-density, parity-check codes. This new System object procures simulation results more quickly than a CPU.

Variable-Size Support

The following blocks now support variable-size input signals:

- M-PSK Modulator Baseband
- QPSK Modulator Baseband
- BPSK Modulator Baseband
- M-PAM Modulator Baseband
- Rectangular QAM Modulator Baseband
- General QAM Modulator Baseband
- M-PSK Demodulator Baseband
- QPSK Demodulator Baseband
- BPSK Demodulator Baseband
- M-PAM Demodulator Baseband
- Rectangular QAM Demodulator Baseband
- General QAM Demodulator Baseband
- Bit to Integer Converter
- Integer to Bit Converter
- Convolutional Encoder
- Viterbi Decoder

The following source blocks can now output variable-size signals:

- Gold Sequence Generator
- Kasami Sequence Generator
- PN Sequence Generator

The following System objects now support variable-size input signals:

-
- comm.PSKModulator
 - comm.QPSKModulator
 - comm.BPSKModulator
 - comm.PAMModulator
 - comm.RectangularQAMModulator
 - comm.GeneralQAMModulator
 - comm.PSKDemodulator
 - comm.QPSKDemodulator
 - comm.BPSKDemodulator
 - comm.PAMDemodulator
 - comm.RectangularQAMDemodulator
 - comm.GeneralQAMDemodulator
 - comm.IntegerToBit
 - comm.BitToInteger

The following System objects now output variable-size signals:

- comm.GoldSequence
- comm.KasamiSequence
- comm.PNSequence

Algorithm Improvements for CRC Blocks

This release introduces a new encoding algorithm for all blocks in the CRC sublibrary residing in the Error Detection and Correction library. In this new implementation, the block processes multiple input bits in one step, resulting in faster processing times. The previous implementation always processed one input bit at each step.

MATLAB Compiler Support for System Objects

The Communications 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:

'Internal rule' System Object Property Values Changed to 'Full precision'

To clarify the value of many DataType properties, the 'Internal rule' option has been changed to 'Full precision'.

Compatibility Considerations

Compatibility Consideration

The objects allow you to enter either 'Internal rule' or 'Full precision'. If you enter 'Internal rule', that option is stored as 'Full precision'.

System Object Code Generation Support

The following System objects support code generation:

- comm.PSKTCMModulator
- comm.RectangularQAMTCMModulator
- comm.GeneralQAMTCMModulator
- comm.EarlyLateGateTimingSynchronizer
- comm.GardnerTimingSynchronizer
- comm.GMSKTimingSynchronize
- comm.MSKTimingSynchronizer
- comm.MuellerMullerTimingSynchronizer
- comm.KasamiSequence

LDPC Decoder Block Warnings

Communications System Toolbox software uses a new implementation of the LDPC Decoder block. If you open a previously existing model that contains the LDPC block, the model generates a warning at the MATLAB command line. Simply resave the model to prevent any subsequent warnings.

Phase/Frequency Offset Block and System Object Change

In previous releases, when the frequency offset input signal to the Phase/Frequency Offset block or `comm.PhaseFrequencyOffset` System object was constant, or time-invariant, the block and System object generated the correct output. However, the block and System object produced incorrect results for a time-varying frequency offset input signal. The new implementation generates the correct output for a time-varying frequency offset input signal.

Derepeat Block Changes

The Derepeat block now contains the **Input processing** and **Rate options** parameters. See “Sample- and Frame-Based Concepts” for more information.

Version 2, 2.5, and 3.0 Obsolete Blocks Removed

All the obsolete block libraries associated with Communications Blockset version 2 Release 12, version 2.5 Release 13, and version 3.0 Release 14 have been removed from this product. The removal includes the following libraries:

- `commanabbnd2`
- `commcontsrc2`
- `commdigpbndam2`
- `commdigpbndcpm2`

-
- `commdigpbndfm2`
 - `commdigpbndpm2`
 - `comminteg2`
 - `commanapbnd2`
 - `commchan2`
 - `commdigbbndam2`
 - `commdigbbndpm2`

Compatibility Considerations

Compatibility Considerations

Communications System Toolbox software does not support any of the blocks from Release 12 and Release 13. The Communications System Toolbox block libraries provide some of the same functionality in the form of upgraded blocks.

System Objects Input and Property Warnings Changed to Errors

When a System object is locked (for example, 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

Compatibility Considerations

Compatibility Consideration

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.

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. Communications System Toolbox documentation refers to the former as frame-based processing and the latter as sample-based processing (see “Sample- and Frame-Based Concepts”). 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 transition to the new paradigm of frame-based processing, many blocks have received new parameters. The following sections provide more detailed information about the specific Communications 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 24-7
- “Multirate Processing Parameter Changes” on page 24-8
- “Sample-Based Row Vector Processing Changes” on page 24-9

Compatibility Considerations

Compatibility Considerations

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 R2010b 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 the **Input processing** parameter, the parameter will be set to **Inherited** (this choice will be removed - see release notes) when you load your model. 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 R2010b frame-based processing changes, see the following Compatibility Considerations sections.

Blocks with a New Input Processing Parameter

Some Communications 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 received a new **Input processing** parameter. You can select `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, expand the following list.

Blocks with New Input Processing Parameter

- Derepeat
- Gaussian Filter
- Windowed Integrator
- AWGN Channel (with only two options)

Compatibility Considerations

Compatibility Considerations

When you load an existing model R2010b, 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 R2010b, 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 for the Gaussian Filter or Windowed Integrator, 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 `sIupdate` function. Therefore, you should upgrade your existing modes using `sIupdate` as soon as possible.

AWGN Channel Block Changes

The AWGN Channel block uses the new method of “Frame-Based Processing” on page 24-5. In previous releases, the frame status of the input signal determined how the AWGN Channel block

processed the signal. In R2010b, the default behavior of the AWGN Channel block is to always perform frame-based processing.

Unless you specify otherwise, the block now treats each column of the input signal as an individual channel, regardless of its frame status. To enable the behavior change in the AWGN Channel block while still allowing for backward compatibility, an **Input processing** parameter has been added. This parameter will be removed in a future release, at which point the block will always perform frame-based processing.

Compatibility Considerations

Compatibility Considerations

The **Input processing** parameter will be removed in a future release. At that point in time, the AWGN Channel block will always perform frame-based processing.

You can use the `sIupdate` function to upgrade your existing models that contain an AWGN Channel block. The function detects all AWGN Channel blocks in your model and, if you allow it to, performs the following actions:

- If the input to the block is an M -by-1 or unoriented sample-based signal, the `sIupdate` function performs three actions. First, a Transpose block is placed in front of the AWGN Channel 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. The `sIupdate` function also sets the **Input processing** parameter to `Columns as channels (frame based)`. This setting ensures that your model will continue to produce the same results when the **Input processing** parameter is removed in a future release. The `sIupdate` function also adds a Transpose block after the AWGN channel block in your model for an M -by-1 sample-based input and a Reshape block for unoriented inputs. By converting the row vector output of the AWGN channel to the input dimension, the model continues to behave as in prior releases.
- If the input to the block is *not* an M -by-1 or unoriented sample-based signal, the `sIupdate` function sets the **Input processing** parameter to `Columns as channels (frame based)`. 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 **Input processing** parameter is removed in a future release.

Multirate Processing Parameter Changes

In R2010a and earlier releases, many Communications System Toolbox blocks that supported multirate processing had a **Framing** parameter. This parameter allowed you to specify whether the block should `Maintain input frame size` or `Maintain input frame rate` when processing the input signal. Beginning in R2010b, a new **Rate options** parameter replaced the **Framing** parameter. The **Rate options** parameter allows you to specify whether the block should `Enforce single-rate processing` or `Allow multirate processing`.

Some blocks that supported multirate processing in R2010a and earlier releases did not have a **Framing** parameter. These blocks used the frame status of the input signal to determine whether they performed single-rate or multirate processing. Because of the upcoming frame-based processing changes, signals will no longer carry their frame status. Thus, multirate blocks can no longer rely on the frame status of the input signal to determine whether they perform single-rate or multirate processing. You must now specify a value for the **Rate options** parameter on the block dialog box.

To see a full list of blocks that have a new **Rate options** parameter, expand the following section.

Multirate Blocks with a New Rate Options Parameter

- Raised Cosine Receive Filter
- Raised Cosine Transmit Filter
- Ideal Rectangular Pulse Filter
- OQPSK Modulator Baseband
- OQPSK Demodulator Baseband
- CPM Modulator Baseband
- CPM Demodulator Baseband
- MSK Modulator Baseband
- MSK Demodulator Baseband
- GMSK Modulator Baseband
- GMSK Demodulator Baseband
- CPFSK Modulator Baseband
- CPFSK Demodulator Baseband
- M-FSK Demodulator Baseband
- M-FSK Modulator Baseband
- Derepeat

Sample-Based Row Vector Processing Changes

The following blocks do not process sample-based row vectors:

- APP Decoder
- Convolutional Encoder
- Viterbi Decoder
- Algebraic Deinterleaver
- Algebraic Interleaver
- General Block Deinterleaver
- General Block Interleaver
- Matrix Deinterleaver
- Matrix Helical Scan Deinterleaver
- Matrix Helical Scan Interleaver
- Matrix Interleaver
- Random Deinterleaver
- Random Interleaver
- M-PAM Modulator Baseband
- Rectangular QAM Modulator Baseband
- DQPSK Modulator Baseband
- M-DPSK Modulator Baseband

- M-PSK Modulator Baseband
- OQPSK Modulator Baseband
- QPSK Modulator Baseband
- M-FSK Modulator Baseband
- CPFSK Modulator Baseband
- CPM Modulator Baseband
- Insert Zero
- Puncture
- Bit to Integer Converter
- Integer to Bit Converter

Compatibility Considerations

Compatibility Considerations

Using existing models that contain these blocks to process sample-based row vectors generates an error message.

CMA Equalizer Changes

The CMA Equalizer block now handles input signals like the other equalizer blocks in the Communications Blockset library. Therefore, the block no longer accepts scalar input signals in symbol-spaced mode.

Differential Encoder Changes

The Differential Encoder block supports scalar-valued and column vector input signals. It does not support frame-based or sample-based row vectors.

Find Delay and Align Signal Block Changes

The **Correlation window length** parameter specifies the number of samples the block uses to calculate the cross-correlation of two signals. You must specify a window lengths of at least 2 for the cross-correlation calculations. If you set the **Correlation window length** parameter to 1, the block generates an error message. The following blocks contain the **Correlation window length** parameter:

- Find Delay
- Align Signals

New Demos

This release contains the following new demos:

- Parallel Concatenated Convolutional Coding: Turbo Codes
- Go-Back-N ARQ with PHY Layer
- Adaptive MIMO System with OSTBC
- CORDIC-Based QPSK Carrier Synchronization

-
- DVB-S.2 Link, Including LDPC Coding
 - DVB-S.2 System Simulation Using a GPU-Based LDPC Decoder System Object

