Communications Blockset™ 4
Reference

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
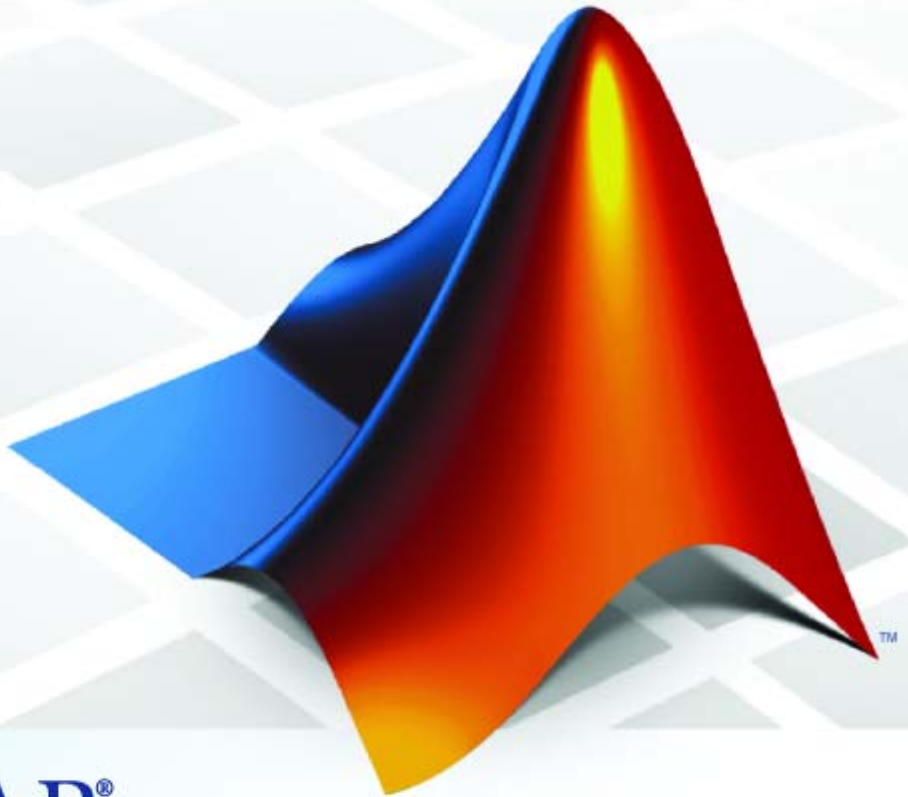
How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Communications Blockset™ Reference

© COPYRIGHT 2001–2009 by 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

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

Revision History

Contents

Block Reference

1

Blocks — Alphabetical List

2

Function Reference

3

Algorithm Used to Decode BCH and Reed-Solomon Codes

A

1

Block Reference

Communications Sources

These are the sublibraries of Communications Sources:

Random Data Sources

Bernoulli Binary Generator	Generate Bernoulli-distributed random binary numbers
Poisson Integer Generator	Generate Poisson-distributed random integers
Random Integer Generator	Generate integers randomly distributed in range [0, M-1]

Noise Generators

Gaussian Noise Generator	Generate Gaussian distributed noise with given mean and variance values
Rayleigh Noise Generator	Generate Rayleigh distributed noise
Rician Noise Generator	Generate Rician distributed noise
Uniform Noise Generator	Generate uniformly distributed noise between upper and lower bounds

Sequence Generators

Barker Code Generator	Generate Barker Code
Gold Sequence Generator	Generate Gold sequence from set of sequences

Hadamard Code Generator	Generate Hadamard code from orthogonal set of codes
Kasami Sequence Generator	Generate Kasami sequence from set of Kasami sequences
OVSF Code Generator	Generate orthogonal variable spreading factor (OVSF) code from set of orthogonal codes
PN Sequence Generator	Generate pseudonoise sequence
Walsh Code Generator	Generate Walsh code from orthogonal set of codes

Communications Sinks

Discrete-Time Eye Diagram Scope	Display multiple traces of modulated signal
Discrete-Time Scatter Plot Scope	Display in-phase and quadrature components of modulated signal constellation
Discrete-Time Signal Trajectory Scope	Plot modulated signal's in-phase component versus its quadrature component
Error Rate Calculation	Compute bit error rate or symbol error rate of input data

Source Coding

| A-Law Compressor | Implement A-law compressor for source coding |
| A-Law Expander | Implement A-law expander for source coding |

Differential Decoder	Decode binary signal using differential coding
Differential Encoder	Encode binary signal using differential coding
Mu-Law Compressor	Implement µ-law compressor for source coding
Mu-Law Expander	Implement µ-law expander for source coding
Quantizing Decoder	Decode quantization index according to codebook
Quantizing Encoder	Quantize signal using partition and codebook

Error Detection and Correction

These are the sublibraries of Error Detection and Correction:

Block Coding (p. 1-4)

Convolutional Coding (p. 1-5)

Cyclic Redundancy Check Coding (p. 1-6)

Block Coding

BCH Decoder	Decode BCH code to recover binary vector data
BCH Encoder	Create BCH code from binary vector data
Binary Cyclic Decoder	Decode systematic cyclic code to recover binary vector data

Binary Cyclic Encoder	Create systematic cyclic code from binary vector data
Binary Linear Decoder	Decode linear block code to recover binary vector data
Binary Linear Encoder	Create linear block code from binary vector data
Binary-Input RS Encoder	Create Reed-Solomon code from binary vector data
Binary-Output RS Decoder	Decode Reed-Solomon code to recover binary vector data
Hamming Decoder	Decode Hamming code to recover binary vector data
Hamming Encoder	Create Hamming code from binary vector data
Integer-Input RS Encoder	Create Reed-Solomon code from integer vector data
Integer-Output RS Decoder	Decode Reed-Solomon code to recover integer vector data
LDPC Decoder	Decode binary low-density parity-check code specified by parity-check matrix
LDPC Encoder	Encode binary low-density parity-check code specified by parity-check matrix

Convolutional Coding

APP Decoder	Decode convolutional code using a posteriori probability (APP) method
Convolutional Encoder	Create convolutional code from binary data
Viterbi Decoder	Decode convolutionally encoded data using Viterbi algorithm

Cyclic Redundancy Check Coding

CRC-N Generator	Generate CRC bits according to CRC method and append to input data frames
CRC-N Syndrome Detector	Detect errors in input data frames according to selected CRC method
General CRC Generator	Generate CRC bits according to generator polynomial and append to input data frames
General CRC Syndrome Detector	Detect errors in input data frames according to generator polynomial

Interleaving

These are the sublibraries of Interleaving:

Block Interleaving

Algebraic Deinterleaver	Restore ordering of input symbols using algebraically derived permutation
Algebraic Interleaver	Reorder input symbols using algebraically derived permutation table
General Block Deinterleaver	Restore ordering of symbols in input vector
General Block Interleaver	Reorder symbols in input vector

Matrix Deinterleaver	Permute input symbols by filling matrix by columns and emptying it by rows
Matrix Helical Scan Deinterleaver	Restore ordering of input symbols by filling matrix along diagonals
Matrix Helical Scan Interleaver	Permute input symbols by selecting matrix elements along diagonals
Matrix Interleaver	Permute input symbols by filling matrix by rows and emptying it by columns
Random Deinterleaver	Restore ordering of input symbols using random permutation
Random Interleaver	Reorder input symbols using random permutation

Convolutional Interleaving

Convolutional Deinterleaver	Restore ordering of symbols that were permuted using shift registers
Convolutional Interleaver	Permute input symbols using set of shift registers
General Multiplexed Deinterleaver	Restore ordering of symbols using specified-delay shift registers
General Multiplexed Interleaver	Permute input symbols using set of shift registers with specified delays
Helical Deinterleaver	Restore ordering of symbols permuted by helical interleaver
Helical Interleaver	Permute input symbols using helical array

MIMO

OSTBC Combiner	Combine inputs for received signals and channel estimate according to orthogonal space-time block code (OSTBC)
OSTBC Encoder	Encode input message using orthogonal space-time block code (OSTBC)

Modulation

These are the sublibraries of Modulation:

Digital Baseband AM Sublibrary

General QAM Demodulator Baseband	Demodulate QAM-modulated data
General QAM Modulator Baseband	Modulate using quadrature amplitude modulation

M-PAM Demodulator Baseband	Demodulate PAM-modulated data
M-PAM Modulator Baseband	Modulate using M-ary pulse amplitude modulation
Rectangular QAM Demodulator Baseband	Demodulate rectangular-QAM-modulated data
Rectangular QAM Modulator Baseband	Modulate using rectangular quadrature amplitude modulation

Digital Baseband PM Sublibrary

BPSK Demodulator Baseband	Demodulate BPSK-modulated data
BPSK Modulator Baseband	Modulate using binary phase shift keying method
DBPSK Demodulator Baseband	Demodulate DBPSK-modulated data
DBPSK Modulator Baseband	Modulate using differential binary phase shift keying method
DQPSK Demodulator Baseband	Demodulate DQPSK-modulated data
DQPSK Modulator Baseband	Modulate using differential quaternary phase shift keying method
M-DPSK Demodulator Baseband	Demodulate DPSK-modulated data
M-DPSK Modulator Baseband	Modulate using M-ary differential phase shift keying method
M-PSK Demodulator Baseband	Demodulate PSK-modulated data
M-PSK Modulator Baseband	Modulate using M-ary phase shift keying method
OQPSK Demodulator Baseband	Demodulate OQPSK-modulated data
OQPSK Modulator Baseband	Modulate using offset quadrature phase shift keying method

| QPSK Demodulator Baseband | Demodulate QPSK-modulated data |
| QPSK Modulator Baseband | Modulate using quaternary phase shift keying method |

Digital Baseband FM Sublibrary

| M-FSK Demodulator Baseband | Demodulate FSK-modulated data |
| M-FSK Modulator Baseband | Modulate using M-ary frequency shift keying method |

Digital Baseband CPM Sublibrary

CPFSK Demodulator Baseband	Demodulate CPFSK-modulated data
CPFSK Modulator Baseband	Modulate using continuous phase frequency shift keying method
CPM Demodulator Baseband	Demodulate CPM-modulated data
CPM Modulator Baseband	Modulate using continuous phase modulation
GMSK Demodulator Baseband	Demodulate GMSK-modulated data
GMSK Modulator Baseband	Modulate using Gaussian minimum shift keying method
MSK Demodulator Baseband	Demodulate MSK-modulated data
MSK Modulator Baseband	Modulate using minimum shift keying method

Digital Baseband TCM Sublibrary

General TCM Decoder	Decode trellis-coded modulation data, mapped using arbitrary constellation
General TCM Encoder	Convolutionally encode binary data and map using arbitrary constellation
M-PSK TCM Decoder	Decode trellis-coded modulation data, modulated using PSK method
M-PSK TCM Encoder	Convolutionally encode binary data and modulate using PSK method
Rectangular QAM TCM Decoder	Decode trellis-coded modulation data, modulated using QAM method
Rectangular QAM TCM Encoder	Convolutionally encode binary data and modulate using QAM method

Analog Passband Modulation

DSB AM Demodulator Passband	Demodulate DSB-AM-modulated data
DSB AM Modulator Passband	Modulate using double-sideband amplitude modulation
DSBSC AM Demodulator Passband	Demodulate DSBSC-AM-modulated data
DSBSC AM Modulator Passband	Modulate using double-sideband suppressed-carrier amplitude modulation
FM Demodulator Passband	Demodulate FM-modulated data
FM Modulator Passband	Modulate using frequency modulation
PM Demodulator Passband	Demodulate PM-modulated data
PM Modulator Passband	Modulate using phase modulation

SSB AM Demodulator Passband — Demodulate SSB-AM-modulated data

SSB AM Modulator Passband — Modulate using single-sideband amplitude modulation

Communications Filters

Gaussian Filter — Filter input signal, possibly downsampling, using Gaussian FIR filter

Ideal Rectangular Pulse Filter — Shape input signal using ideal rectangular pulses

Integrate and Dump — Integrate discrete-time signal, resetting to zero periodically

Raised Cosine Receive Filter — Filter input signal, possibly downsampling, using raised cosine FIR filter

Raised Cosine Transmit Filter — Upsample and filter input signal using raised cosine FIR filter

Windowed Integrator — Integrate over time window of fixed length

Channels

AWGN Channel — Add white Gaussian noise to input signal

Binary Symmetric Channel — Introduce binary errors

| Multipath Rayleigh Fading Channel | Simulate multipath Rayleigh fading propagation channel |
| Multipath Rician Fading Channel | Simulate multipath Rician fading propagation channel |

RF Impairments

Free Space Path Loss	Reduce amplitude of input signal by amount specified
I/Q Imbalance	Create complex baseband model of signal impairments caused by imbalances between in-phase and quadrature receiver components
Memoryless Nonlinearity	Apply memoryless nonlinearity to complex baseband signal
Phase Noise	Apply receiver phase noise to complex baseband signal
Phase/Frequency Offset	Apply phase and frequency offsets to complex baseband signal
Receiver Thermal Noise	Apply receiver thermal noise to complex baseband signal

Synchronization

These are the sublibraries of Synchronization:

Carrier Phase Recovery

CPM Phase Recovery	Recover carrier phase using 2P-Power method
M-PSK Phase Recovery	Recover carrier phase using M-Power method

Timing Phase Recovery

Early-Late Gate Timing Recovery	Recover symbol timing phase using early-late gate method
Gardner Timing Recovery	Recover symbol timing phase using Gardner's method
MSK-Type Signal Timing Recovery	Recover symbol timing phase using fourth-order nonlinearity method
Mueller-Muller Timing Recovery	Recover symbol timing phase using Mueller-Muller method
Squaring Timing Recovery	Recover symbol timing phase using squaring method

Synchronization Components

Baseband PLL	Implement baseband phase-locked loop
Charge Pump PLL	Implement charge pump phase-locked loop using digital phase detector
Continuous-Time VCO	Implement voltage-controlled oscillator
Discrete-Time VCO	Implement voltage-controlled oscillator in discrete time

Linearized Baseband PLL   Implement linearized version of
                                   baseband phase-locked loop

Phase-Locked Loop   Implement phase-locked loop to
                                   recover phase of input signal

Equalizers

CMA Equalizer   Equalize using constant modulus
                                   algorithm

LMS Decision Feedback Equalizer   Equalize using decision feedback
                                   equalizer that updates weights with
                                   LMS algorithm

LMS Linear Equalizer   Equalize using linear equalizer
                                   that updates weights with LMS
                                   algorithm

MLSE Equalizer   Equalize using Viterbi algorithm

Normalized LMS Decision Feedback   Equalize using decision feedback
Equalizer   equalizer that updates weights with
                                   normalized LMS algorithm

Normalized LMS Linear Equalizer   Equalize using linear equalizer that
                                   updates weights with normalized
                                   LMS algorithm

RLS Decision Feedback Equalizer   Equalize using decision feedback
                                   equalizer that updates weights with
                                   RLS algorithm

RLS Linear Equalizer   Equalize using linear equalizer
                                   that updates weights using RLS
                                   algorithm

Sign LMS Decision Feedback   Equalize using decision feedback
Equalizer   equalizer that updates weights with
                                   signed LMS algorithm

Sign LMS Linear Equalizer	Equalize using linear equalizer that updates weights with signed LMS algorithm
Variable Step LMS Decision Feedback Equalizer	Equalize using decision feedback equalizer that updates weights with variable-step-size LMS algorithm
Variable Step LMS Linear Equalizer	Equalize using linear equalizer that updates weights with variable-step-size LMS algorithm

Sequence Operations

Deinterlacer	Distribute elements of input vector alternately between two output vectors
Derepeat	Reduce sampling rate by averaging consecutive samples
Descrambler	Descramble input signal
Insert Zero	Distribute input elements in output vector
Interlacer	Alternately select elements from two input vectors to generate output vector
Puncture	Output elements which correspond to 1s in binary Puncture vector
Scrambler	Scramble input signal

Utility Blocks

Align Signals	Align two signals by finding delay between them
Bipolar to Unipolar Converter	Map bipolar signal into unipolar signal in range [0, M-1]
Bit to Integer Converter	Map vector of bits to corresponding vector of integers
Complex Phase Difference	Output phase difference between two complex input signals
Complex Phase Shift	Shift phase of complex input signal by second input value
Data Mapper	Map integer symbols from one coding scheme to another
EVM Measurement	Calculate vector magnitude difference between ideal reference signal and measured signal
Find Delay	Find delay between two signals
Integer to Bit Converter	Map vector of integers to vector of bits
MER Measurement	Measure signal-to-noise ratio (SNR) in digital modulation applications
Unipolar to Bipolar Converter	Map unipolar signal in range [0, M-1] into bipolar signal

Blocks — Alphabetical List

A-Law Compressor

Purpose   Implement A-law compressor for source coding

Library   Source Coding

Description The A-Law Compressor block implements an A-law compressor for the input signal. The formula for the A-law compressor is
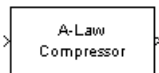
$$
y = \begin{cases} \dfrac{A|x|}{1+\log A}\,\mathrm{sgn}(x) & \text{for } 0 \le |x| \le \dfrac{V}{A} \\[2ex] \dfrac{V\left(1+\log(A|x|/V)\right)}{1+\log A}\,\mathrm{sgn}(x) & \text{for } \dfrac{V}{A} < |x| \le V \end{cases}
$$

where A is the A-law parameter of the compressor, V is the peak signal magnitude for x, log is the natural logarithm, and sgn is the signum function (`sign` in MATLAB® software).

The most commonly used A value is 87.6.

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box
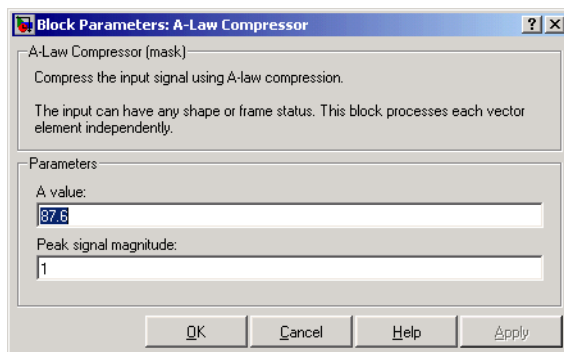
A value

  The A-law parameter of the compressor.

Peak signal magnitude
> The peak value of the input signal. This is also the peak value of
> the output signal.

Pair Block   A-Law Expander

See Also   Mu-Law Compressor

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J., Prentice-Hall, 1988.

A-Law Expander

Purpose   Implement A-law expander for source coding

Library   Source Coding

Description   The A-Law Expander block recovers data that the A-Law Compressor block compressed. The formula for the A-law expander, shown below, is the inverse of the compressor function.
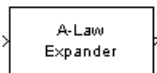
A-Law
Expander

$$
x = \begin{cases} \dfrac{y(1 + \log A)}{A} & \text{for } 0 \le |y| \le \dfrac{V}{1+\log A} \\ \exp\bigl(|y|(1 + \log A)/V - 1\bigr)\dfrac{V}{A}\,\text{sgn}(y) & \text{for } \dfrac{V}{1+\log A} < |y| \le V \end{cases}
$$

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box
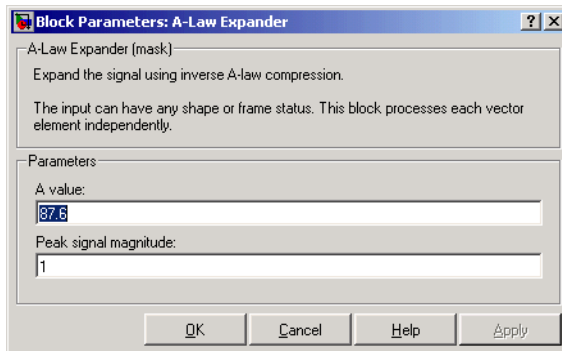
A value
    The A-law parameter of the compressor.

Peak signal magnitude
    The peak value of the input signal. This is also the peak value of the output signal.

Match these parameters to the ones in the corresponding A-Law Compressor block.

**Pair Block**   A-Law Compressor

See Also   Mu-Law Expander

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications.* Englewood Cliffs, N.J., Prentice-Hall, 1988.

Algebraic Deinterleaver

Purpose   Restore ordering of input symbols using algebraically derived permutation

Library   Block sublibrary of Interleaving
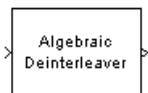
Description   The Algebraic Deinterleaver block restores the original ordering of a sequence that was interleaved using the Algebraic Interleaver block. In typical usage, the parameters in the two blocks have the same values.

The **Number of elements** parameter, N, indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

The **Type** parameter indicates the algebraic method that the block uses to generate the appropriate permutation table. Choices are `Takeshita-Costello` and `Welch-Costas`. Each of these methods has parameters and restrictions that are specific to it; these are described on the reference page for the Algebraic Interleaver block.

Dialog Box
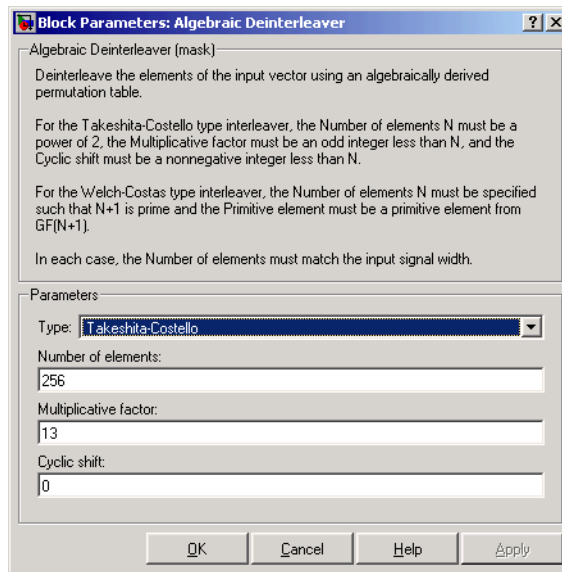
Type

> The type of permutation table that the block uses for deinterleaving. Choices are `Takeshita-Costello` and `Welch-Costas`.

Number of elements

> The number of elements, N, in the input vector.

Multiplicative factor

> The factor used to compute the corresponding interleaver's cycle vector. This field appears only if **Type** is set to `Takeshita-Costello`.

Cyclic shift

> The amount by which the block shifts indices when creating the corresponding interleaver's permutation table. This field appears only if **Type** is set to `Takeshita-Costello`.

Algebraic Deinterleaver

Primitive element
An element of order N in the finite field GF(N+1). This field appears only if **Type** is set to `Welch-Costas`.

Pair Block   Algebraic Interleaver

See Also   General Block Deinterleaver

References   [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

[2] Takeshita, O. Y. and D. J. Costello, Jr. "New Classes Of Algebraic Interleavers for Turbo-Codes." *Proc. 1998 IEEE International Symposium on Information Theory*, Boston, Aug. 16-21, 1998. 419.

Purpose   Reorder input symbols using algebraically derived permutation table

Library   Block sublibrary of Interleaving

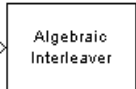**Description**   The Algebraic Interleaver block rearranges the elements of its input
vector using a permutation that is algebraically derived. The **Number
of elements** parameter, N, indicates how many numbers are in the
input vector. If the input is frame-based, then it must be a column
vector.

The block can accept the data types int8, uint8, int16, uint16, int32,
uint32, boolean, single, double, and fixed-point. The data type of this
output will be the same as that of the input signal.

The **Type** parameter indicates the algebraic method that the block
uses to generate the appropriate permutation table. Choices are
Takeshita-Costello and Welch-Costas. Each of these methods has
parameters and restrictions that are specific to it:

- If **Type** is set to Welch-Costas, then N+1 must be prime. The
  Primitive element parameter is an integer, A, between 1 and N
  that represents a primitive element of the finite field GF(N+1). This
  means that every nonzero element of GF(N+1) can be expressed as
  A raised to some integer power.

  In a Welch-Costas interleaver, the permutation maps the integer k
  to $\mod(A^k, N+1) - 1$.

- If **Type** is set to Takeshita-Costello, then N must be 2^m for some
  integer m. The **Multiplicative factor** parameter, h, must be an
  odd integer less than N. The **Cyclic shift** parameter, k, must be a
  nonnegative integer less than N.

  A Takeshita-Costello interleaver uses a length-N *cycle vector* whose
  nth element is

  $\mod(k*(n-1)*n/2, N)$

  for integers n between 1 and N. The block creates a permutation
  vector by listing, for each element of the cycle vector in ascending

order, one plus the element's successor. The interleaver's actual permutation table is the result of shifting the elements of the permutation vector left by the **Cyclic shift** parameter. (The block performs all computations on numbers and indices modulo N.)

Dialog Box
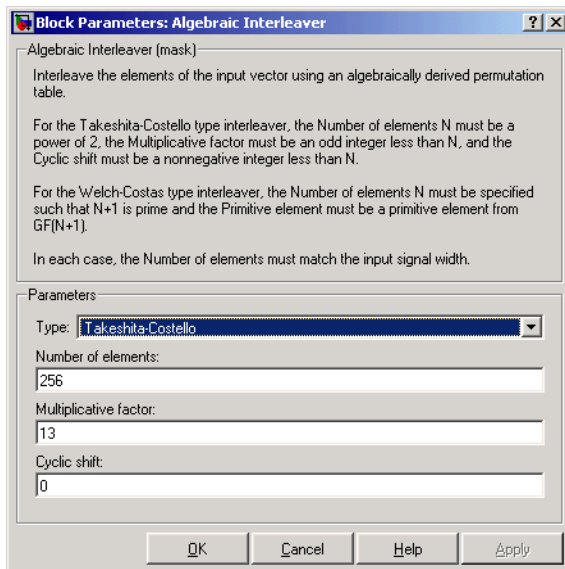
Type

    The type of permutation table that the block uses for interleaving.

Number of elements

    The number of elements, N, in the input vector.

Multiplicative factor

    The factor used to compute the interleaver's cycle vector. This field appears only if **Type** is set to `Takeshita-Costello`.

Cyclic shift

    The amount by which the block shifts indices when creating the permutation table. This field appears only if **Type** is set to `Takeshita-Costello`.

Primitive element
An element of order N in the finite field GF(N+1). This field appears only if **Type** is set to Welch-Costas.

Pair Block   Algebraic Deinterleaver

See Also   General Block Interleaver

References   [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding.* Boston: Kluwer Academic Publishers, 1999.

[2] Takeshita, O. Y. and D. J. Costello, Jr. "New Classes Of Algebraic Interleavers for Turbo-Codes." *Proc. 1998 IEEE International Symposium on Information Theory*, Boston, Aug. 16-21, 1998. 419.

Align Signals

Purpose   Align two signals by finding delay between them

Library   Utility Blocks

Description   The Align Signals block aligns a signal with a delayed, and possibly
distorted, version of itself. The block is particularly useful when you
want to compare a transmitted and received signal to find the bit error
rate, but do not know the delay in the received signal.

The input port labeled s1 receives the original signal, while the input
port labeled s2 receives the delayed version of the signal. The two input
signals must have the same sample times. The block calculates the
delay between the two signal, and then

- Delays the first signal, s1, by the calculated value, and outputs it
  through the port labeled s1 del.

- Outputs the second signal s2 without change through the port
  labeled s2.

- Outputs the delay value through the port labeled delay.

See "Computing Delays" in the Communications Blockset™ online
documentation for more information about signal delays.

The block's **Correlation window length** parameter specifies
how many samples of the signals the block uses to calculate the
cross-correlation. The delay output is a nonnegative integer less than
the **Correlation window length**.

As the **Correlation window length** is increased, the reliability of
the computed delay also increases. However, the processing time to
compute the delay increases as well.

You can make the Align Signals block stop updating the delay after it
computes the same delay value for a specified number of samples. To
do so, select the **Disable recurring updates** check box, and enter a
positive integer in the **Number of constant delay outputs to disable
updates** field. For example, if you set **Number of constant delay
outputs to disable updates** to 20, the block will stop recalculating

and updating the delay after it calculates the same value 20 times in succession. Disabling recurring updates causes the simulation to run faster after the target number of constant delays occurs.

Tips for Using the Block Effectively

- Set the **Correlation window length** parameter sufficiently large so that the computed delay eventually stabilizes at a constant value. If the computed delay is not constant, you should increase **Correlation window length**. If the increased value of **Correlation window length** exceeds the duration of the simulation, then you should also increase the duration of the simulation accordingly.

- If the cross-correlation between the two signals is broad, then **Correlation window length** should be much larger than the expected delay, or else the algorithm might stabilize at an incorrect value. For example, a CPM signal has a broad autocorrelation, so it has a broad cross-correlation with a delayed version of itself. In this case, the **Correlation window length** value should be much larger than the expected delay.

- If the block calculates a delay that is greater than 75 percent of **Correlation window length**, the signal s1 is probably delayed relative to the signal s2. In this case, you should switch the signal lines leading into the two input ports.

- If you use the Align Signals block with the Error Rate Calculation block, you should set the **Receive delay** parameter of the Error Rate Calculation block to 0 because the Align Signals block compensates for the delay. Also, you might want to set the Error Rate Calculation block's **Computation delay** parameter to a nonzero value to account for the possibility that the Align Signals block takes a nonzero amount of time to stabilize on the correct amount by which to delay one of the signals.

Examples   See the "Computing Delays" section of *Communications Blockset User's Guide* for an example that uses the Align Signals block in conjunction with the Error Rate Calculation block.

Align Signals

See Setting the Correlation Window Length, on the reference page for the Find Delay block, for an example that illustrates how to set the correlation window length properly.

Dialog Box
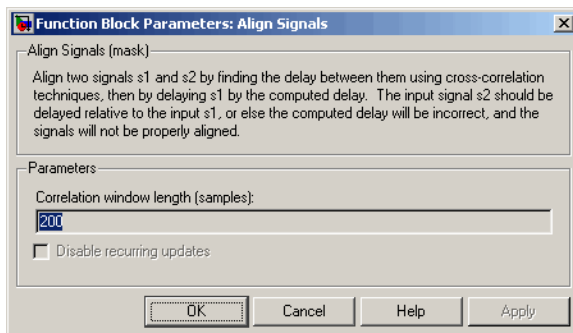
Correlation window length

    The number of samples the block uses to calculate the cross-correlations of the two signals.

Disable recurring updates

    Selecting this option causes the block to stop computing the delay after it computes the same delay value for a specified number of samples.

Number of constant delay outputs to disable updates

    A positive integer specifying how many times the block must compute the same delay before ceasing to update. This field appears only if **Disable recurring updates** is selected.

Algorithm   The Align Signals block finds the delay by calculating the cross-correlations of the first signal with time-shifted versions of the second signal, and then finding the index at which the cross-correlation is maximized.

See Also   Find Delay, Error Rate Calculation

Purpose   Decode convolutional code using a posteriori probability (APP) method

Library   Convolutional sublibrary of Channel Coding

Description The APP Decoder block performs a posteriori probability (APP) decoding
of a convolutional code.

Inputs and Outputs

The input L(u) represents the sequence of log-likelihoods of encoder
input bits, while the input L(c) represents the sequence of log-likelihoods
of code bits. The outputs L(u) and L(c) are updated versions of these
sequences, based on information about the encoder.

If the convolutional code uses an alphabet of 2^n possible symbols,
this block's L(c) vectors have length Q*n for some positive integer Q.
Similarly, if the decoded data uses an alphabet of 2^k possible output
symbols, then this block's L(u) vectors have length Q*k. The integer Q
is the number of frames that the block processes in each step.

The inputs can be either

- Sample-based vectors having the same dimension and orientation,
  with Q = 1

- Frame-based column vectors with any positive integer for Q

If you only need the input L(c) and output L(u), you can attach a
Simulink Ground block to the input L(u) and a Simulink Terminator
block to the output L(c).

This block accepts `single` and `double` data types. Both inputs,
however, must be of the same type. The output data type is the same as
the input data type.

Specifying the Encoder

To define the convolutional encoder that produced the coded
input, use the **Trellis structure** parameter. This parameter is a
MATLAB structure whose format is described in "Trellis Description

of a Convolutional Encoder" in the Communications Toolbox™ documentation. You can use this parameter field in two ways:

- If you have a variable in the MATLAB workspace that contains the trellis structure, enter its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink® to spend less time updating the diagram at the beginning of each simulation, compared to the usage described next.

- If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, use a poly2trellis command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7,[171 133],171)
```

To indicate how the encoder treats the trellis at the beginning and end of each frame, set the **Termination method** parameter to either Truncated or Terminated. The Truncated option indicates that the encoder resets to the all-zeros state at the beginning of each frame. The Terminated option indicates that the encoder forces the trellis to end each frame in the all-zeros state. If you use the Convolutional Encoder block with the **Operation mode** parameter set to Truncated (reset every frame), use the Truncated option in this block. If you use the Convolutional Encoder block with the **Operation mode** parameter set to Terminate trellis by appending tail bits, use the Terminated option in this block.

Specifying Details of the Algorithm

You can control part of the decoding algorithm using the **Algorithm** parameter. The True APP option implements a posteriori probability decoding as per equations 20–23 in section V of [1]. To gain speed, both the Max* and Max options approximate expressions like

$$\log \sum_i \exp(a_i)$$

by other quantities. The Max option uses $\max(a_i)$ as the approximation, while the Max* option uses $\max(a_i)$ plus a correction term given by

$\ln(1 + \exp(-|a_{i-1} - a_i|))$ [3].

The Max* option enables the **Scaling bits** parameter in the dialog box. This parameter is the number of bits by which the block scales the data it processes internally (multiplies the input by (2^numScalingBits) and divides the pre-output by the same factor). Use this parameter to avoid losing precision during the computations.

Dialog Box
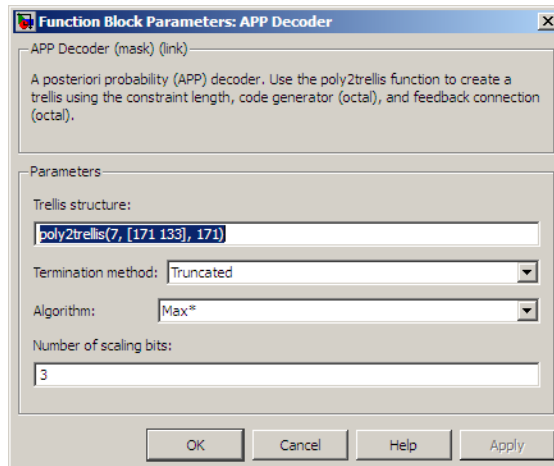
Trellis structure
> MATLAB structure that contains the trellis description of the convolutional encoder.

Termination method
> Either Truncated or Terminated. This parameter indicates how the convolutional encoder treats the trellis at the beginning and end of frames.

Algorithm

Either `True APP`, `Max*`, or `Max`.

Number of scaling bits

An integer between 0 and 8 that indicates by how many bits the decoder scales data in order to avoid losing precision. This field is active only when **Algorithm** is set to `Max*`.

See Also

Viterbi Decoder, Convolutional Encoder; `poly2trellis`

References

[1] Benedetto, S., G. Montorsi, D. Divsalar, and F. Pollara, "A Soft-Input Soft-Output Maximum A Posterior (MAP) Module to Decode Parallel and Serial Concatenated Codes," *JPL TDA Progress Report*, Vol. 42-127, November 1996.

[2] Benedetto, Sergio and Guido Montorsi, "Performance of Continuous and Blockwise Decoded Turbo Codes." *IEEE Communications Letters*, Vol. 1, May 1997, 77–79.
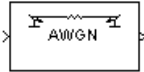
[3] Viterbi, Andrew J., "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE Journal on Selected Areas in Communications*, Vol. 16, February 1998, 260–264.

Purpose   Add white Gaussian noise to input signal

Library   Channels

Description   The AWGN Channel block adds white Gaussian noise to a real or
                     complex input signal. When the input signal is real, this block adds
                     real Gaussian noise and produces a real output signal. When the input
                     signal is complex, this block adds complex Gaussian noise and produces
                     a complex output signal. This block inherits its sample time from the
                     input signal.

                     This block uses the Signal Processing Blockset™ Random Source block
                     to generate the noise. Random numbers are generated using the
                     Ziggurat method, which is the same method used by the MATLAB
                     randn function. The **Initial seed** parameter in this block initializes the
                     noise generator. **Initial seed** can be either a scalar or a vector whose
                     length matches the number of channels in the input signal. For details
                     on **Initial seed**, see the Random Source block reference page in the
                     Signal Processing Blockset documentation set.

                     The signal inputs can only be of type single or double. The port data
                     types are inherited from the signals that drive the block.

                     Note All values of power assume a nominal impedance of 1 ohm.

                     ### Frame-Based Processing and Input Dimensions

                     This block can process multichannel signals that are frame-based or
                     sample-based. The guidelines below indicate how the block interprets
                     your data, depending on the data's shape and frame status:

                     • If your input is a sample-based scalar, then the block adds scalar
                       Gaussian noise to your signal.

                     • If your input is a sample-based vector or a frame-based row vector,
                       then the block adds independent Gaussian noise to each channel.

- If your input is a frame-based column vector, then the block adds a frame of Gaussian noise to your single-channel signal.

- If your input is a frame-based m-by-n matrix, then the block adds a length-m frame of Gaussian noise independently to each of the n channels.

The input cannot be a sample-based m-by-n matrix if both m and n are greater than 1.

Specifying the Variance Directly or Indirectly

You can specify the variance of the noise generated by the AWGN Channel block using one of these modes:

- `Signal to noise ratio (Eb/No)`, where the block calculates the variance from these quantities that you specify in the dialog box:

  - **Eb/No**, the ratio of bit energy to noise power spectral density

  - **Number of bits per symbol**

  - **Input signal power**, the actual power of the symbols at the input of the block

  - **Symbol period**

- `Signal to noise ratio (Es/No)`, where the block calculates the variance from these quantities that you specify in the dialog box:

  - **Es/No**, the ratio of signal energy to noise power spectral density

  - **Input signal power**, the actual power of the symbols at the input of the block

  - **Symbol period**

- `Signal to noise ratio (SNR)`, where the block calculates the variance from these quantities that you specify in the dialog box:

  - **SNR**, the ratio of signal power to noise power

  - **Input signal power**, the actual power of the samples at the input of the block

- `Variance from mask`, where you specify the variance in the dialog box. The value must be positive.

- `Variance from port`, where you provide the variance as an input to the block. The variance input must be positive, and its sampling rate must equal that of the input signal. If the first input signal is sample-based, then the variance input must be sample-based. If the first input signal is frame-based, then the variance input can be either frame-based with exactly one row, or sample-based.

Changing the symbol period in the `AWGN Channel` block affects the variance of the noise added per sample, which also causes a change in the final error rate.

$$NoiseVariance = \frac{SignalPower \times SymbolPeriod}{SampleTime \times 10^{\frac{Es/No}{10}}}$$

A good rule of thumb for selecting the **Symbol period** value is to set it to be what you model as the symbol period in the model. The value would depend upon what constitutes a symbol and what the oversampling applied to it is (e.g., a symbol could have 3 bits and be oversampled by 4).

In both `Variance from mask` mode and `Variance from port` mode, these rules describe how the block interprets the variance:

- If the variance is a scalar, then all signal channels are uncorrelated but share the same variance.

- If the variance is a vector whose length is the number of channels in the input signal, then each element represents the variance of the corresponding signal channel.

AWGN Channel

Note If you apply complex input signals to the AWGN Channel block, then it adds complex zero-mean Gaussian noise with the calculated or specified variance. The variance of each of the quadrature components of the complex noise is half of the calculated or specified value.

Relationship Among Eb/No, Es/No, and SNR Modes

For complex input signals, the AWGN Channel block relates E_b/N_0, E_s/N_0, and SNR according to the following equations:

$$E_s/N_0 = (T_{sym}/T_{samp}) \cdot SNR$$

$$E_s/N_0 = E_b/N_0 + 10\log_{10}(k) \quad \text{in dB}$$

where

- E_s = Signal energy (Joules)
- E_b = Bit energy (Joules)
- N_0 = Noise power spectral density (Watts/Hz)
- T_{sym} is the **Symbol period** parameter of the block in `Es/No` mode
- k is the number of information bits per input symbol
- T_{samp} is the inherited sample time of the block, in seconds

For real signal inputs, the AWGN Channel block relates E_s/N_0 and SNR according to the following equation:

$$E_s/N_0 = 0.5 \ (T_{sym}/T_{samp}) \cdot SNR$$

Note that the equation for the real case differs from the corresponding equation for the complex case by a factor of 2. This is so because the block uses a noise power spectral density of $N_0/2$ Watts/Hz for real input signals, versus N_0 Watts/Hz for complex signals.

For more information about these quantities, see "Describing the Noise Level of an AWGN Channel" in the Communications Toolbox documentation.

Tunable Block Parameters

The following table indicates which parameters are tunable, for different block modes.

Mode	Tunable Parameters
Eb/No	**Eb/No**, **Input signal power**
Es/No	**Es/No**, **Input signal power**
SNR	**SNR**, **Input signal power**
Variance from mask	**Variance**

You can tune parameters in normal mode, Accelerator mode and the Rapid Accelerator mode.

If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameters listed in the previous table without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.
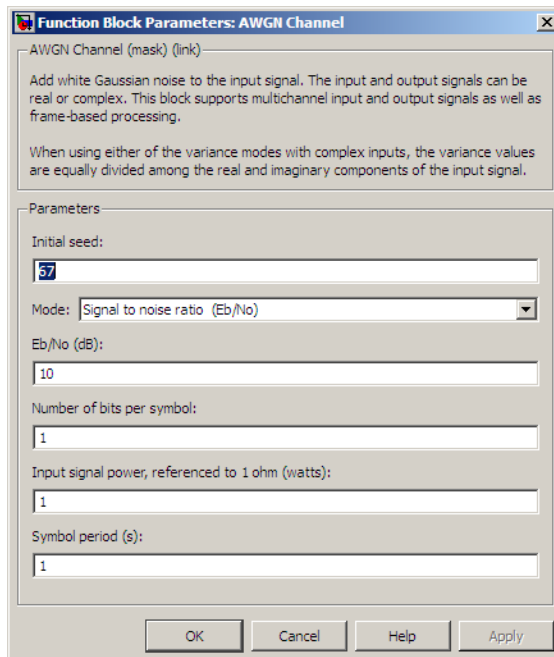
**Dialog
Box**

Initial seed

    The seed for the Gaussian noise generator.

Mode

    The mode by which you specify the noise variance: `Signal to noise ratio (Eb/No)`, `Signal to noise ratio (Es/No)`, `Signal to noise ratio (SNR)`, `Variance from mask`, or `Variance from port`.

Eb/No (dB)

    The ratio of bit energy per symbol to noise power spectral density, in decibels. This field appears only if **Mode** is set to `Eb/No`.

Es/No (dB)

    The ratio of signal energy per symbol to noise power spectral density, in decibels. This field appears only if **Mode** is set to `Es/No`.

SNR (dB)

The ratio of signal power to noise power, in decibels. This field appears only if **Mode** is set to SNR.

Number of bits per symbol

The number of bits in each input symbol. This field appears only if **Mode** is set to Eb/No.

Input signal power, referenced to 1 ohm (watts)

The mean square power of the input symbols (if **Mode** is Eb/No or Es/No) or input samples (if **Mode** is SNR), in watts. This field appears only if **Mode** is set to Eb/No, Es/No, or SNR.

Symbol period (s)

The duration of a channel symbol, in seconds. This field appears only if **Mode** is set to Eb/No or Es/No.

Variance

The variance of the white Gaussian noise. This field appears only if **Mode** is set to Variance from mask.

Examples   Many demonstration models and documentation examples use this block, including:

- Gray Coded 8-PSK demo, commgraycode (EbNo mode)

- Phase Noise Effects in 256-QAM demo, commphasenoise (EsNo mode)

- "Building a Frequency-Shift Keying Model" (EsNo mode)

- "Example: Using Raised Cosine Filters" (SNR mode)

- Discrete Multitone Signaling Demo, commdmt (Variance from mask mode)

See Also   Random Source (Signal Processing Blockset documentation)
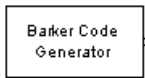
Reference   [1] Proakis, John G., *Digital Communications*, 4th Ed., McGraw-Hill, 2001.

Barker Code Generator

Purpose   Generate Barker Code

Library   Sequence Generators sublibrary of Comm Sources

Description   Barker codes, which are subsets of PN sequences, are commonly used
for frame synchronization in digital communication systems. Barker

Barker Code
Generator

codes have length at most 13 and have low correlation sidelobes. A
correlation sidelobe is the correlation of a codeword with a time-shifted
version of itself. The correlation sidelobe, C_k, for a k-symbol shift of an
N-bit code sequence, $\{X_j\}$, is given by

$$C_k = \sum_{j=1}^{N-k} X_j X_{j+k}$$

where X_j is an individual code symbol taking values +1 or -1 for
j=1, 2, 3,..., N, and the adjacent symbols are assumed to be zero.

The Barker Code Generator block provides the codes listed in the
following table:

Code length	Barker Code
1	[-1]
2	[-1 1];
3	[-1 -1 1]
4	[-1 -1 1 -1]
5	[-1 -1 -1 1 -1]
7	[-1 -1 -1 1 1 -1 1]
11	[-1 -1 -1 1 1 1 -1 1 1 -1 1]
13	[-1 -1 -1 -1 -1 1 1 -1 -1 1 -1 1 -1]

**Dialog
Box**
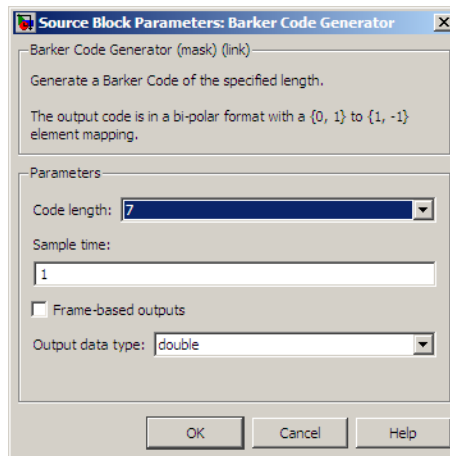
Opening this dialog box causes a running simulation to pause.
See " Changing Source Block Parameters " in the online Simulink
documentation for details.

Code length
The length of the Barker code.

Sample time
Period of each element of the output signal.

Frame-based outputs
Determines whether the block's output is frame-based or
sample-based.

Samples per frame
The number of samples in a frame-based output signal. This field
appears if you select the **Frame-based outputs** check box.

Output data type
The output type of the block can be specified as an int8 or double.
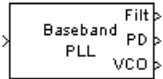By default, the block sets this to double.

See Also   PN Sequence Generator

Baseband PLL

Purpose   Implement baseband phase-locked loop

Library   Components sublibrary of Synchronization

Description   The Baseband PLL (phase-locked loop) block is a feedback control
system that automatically adjusts the phase of a locally generated
signal to match the phase of an input signal. Unlike thePhase-Locked
Loop block, this block uses a baseband method and does not depend on
a carrier frequency.

This PLL has these three components:

- An integrator used as a phase detector.

- A filter. You specify the filter's transfer function using the **Lowpass
  filter numerator** and **Lowpass filter denominator** parameters.
  Each is a vector that gives the respective polynomial's coefficients in
  order of descending powers of *s*.

  To design a filter, you can use the Signal Processing Toolbox™
  functions `cheby1`, and `cheby2`. The default filter is a Chebyshev type
  II filter whose transfer function arises from the command below.

  `[num, den] = cheby2(3,40,100,'s')`

- A voltage-controlled oscillator (VCO). You specify the sensitivity
  of the VCO signal to its input using the **VCO input sensitivity**
  parameter. This parameter, measured in Hertz per volt, is a scale
  factor that determines how much the VCO shifts from its quiescent
  frequency.

The input signal represents the received signal. The input must be a
sample-based scalar signal. The three output ports produce:

- The output of the filter

- The output of the phase detector

- The output of the VCO

This model is nonlinear; for a linearized version, use theLinearized
Baseband PLL block.

**Dialog
Box**
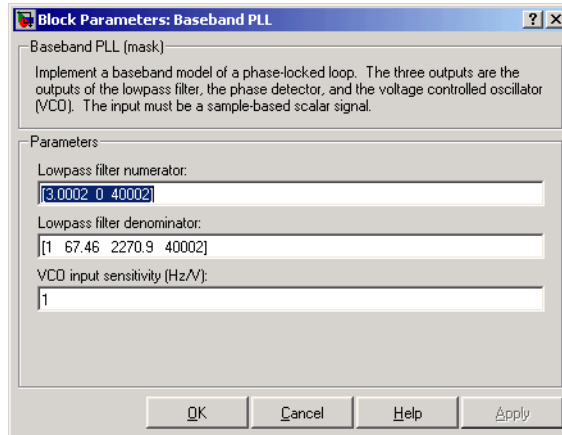
Lowpass filter numerator

The numerator of the lowpass filter's transfer function,
represented as a vector that lists the coefficients in order of
descending powers of *s*.

Lowpass filter denominator

The denominator of the lowpass filter's transfer function,
represented as a vector that lists the coefficients in order of
descending powers of *s*.

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the
shift from the VCO's quiescent frequency.

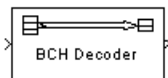**See Also**   Linearized Baseband PLL, Phase-Locked Loop

References   For more information about phase-locked loops, see the works listed
in "Selected Bibliography for Synchronization" in *Communications
Blockset User's Guide*.

BCH Decoder

Purpose   Decode BCH code to recover binary vector data

Library   Block sublibrary of Channel Coding

Description   The BCH Decoder block recovers a binary message vector from a binary
BCH codeword vector. For proper decoding, the first two parameter
values in this block should match the parameters in the corresponding
BCH Encoder block.

This block supports punctures and erasures ("Shortening, Puncturing,
and Erasures" provides a tutorial).

Note 1s and 0s have precisely opposite meanings for the puncture and
erasure vectors. For a puncture vector, a 1 means that the data symbol
is passed unaltered, and a 0 means that the data symbol is punctured
(i.e., removed) from the data stream. For an erasure vector, a 1 means
that the data symbol is to be replaced with an erasure symbol, and a 0
means that the data symbol is passed unaltered. These conventions are
carried for both the encoder and the decoder.

The input must be a frame-based column vector with an integer
multiple of (*N* - the number of punctures) elements. Each group of *N*
input elements represents one codeword to be decoded. The values of (*N*
+ shortening length) and (*K* + shortening length) must produce a valid
narrow-sense BCH code.

If the decoder is processing multiple codewords per frame, then the
same puncture pattern holds for all codewords.

For a given codeword length *N*, only specific message lengths *K* are
valid for a BCH code. For a full length BCH code, *N* must be of the
form 2^M-1, where $3 \leq M \leq 16$. If *N* is less than 2^M-1, the block assumes
that the code has been shortened by length 2^{M-1} - *N*. However, if *N* is
greater than or equal to 2^{M-1}, **Primitive polynomial** must be specified
to appropriately set the value of *M*.

No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability. For a list of some valid values of *K* corresponding to values of *N* up to 511, see the bchenc reference page in the Communications Toolbox documentation.

The primitive and generator polynomials may be specified in their respective fields, which appear after selecting their corresponding check boxes.

To have the block output error information, select **Output number of corrected errors**. Selecting this option causes a second output port to appear. The second output is the number of errors detected during decoding of the codeword. A negative integer indicates that the block detected more errors than it could correct using the coding scheme.
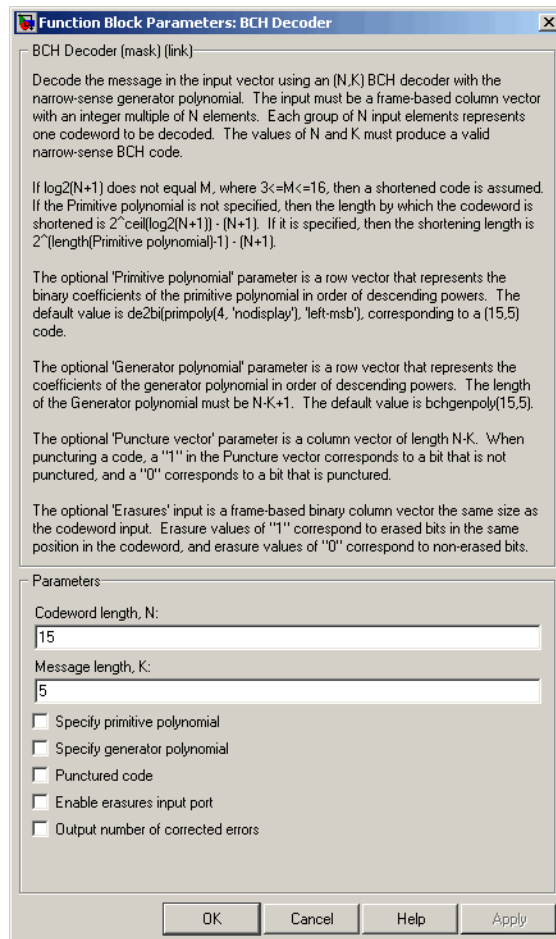
In the case of a decoder failure, the message portion of the decoder input is returned unchanged as the decoder output.

The sample times of all input and output signals are equal.

This block supports double and boolean data types.

BCH Decoder

Dialog Box

Function Block Parameters: BCH Decoder dialog box showing the BCH Decoder (mask) (link) description:

Decode the message in the input vector using an (N,K) BCH decoder with the narrow-sense generator polynomial. The input must be a frame-based column vector with an integer multiple of N elements. Each group of N input elements represents one codeword to be decoded. The values of N and K must produce a valid narrow-sense BCH code.

If log2(N+1) does not equal M, where 3<=M<=16, then a shortened code is assumed. If the Primitive polynomial is not specified, then the length by which the codeword is shortened is 2^ceil(log2(N+1)) - (N+1). If it is specified, then the shortening length is 2^(length(Primitive polynomial)-1) - (N+1).

The optional 'Primitive polynomial' parameter is a row vector that represents the binary coefficients of the primitive polynomial in order of descending powers. The default value is de2bi(primpoly(4, 'nodisplay'), 'left-msb'), corresponding to a (15,5) code.

The optional 'Generator polynomial' parameter is a row vector that represents the coefficients of the generator polynomial in order of descending powers. The length of the Generator polynomial must be N-K+1. The default value is bchgenpoly(15,5).

The optional 'Puncture vector' parameter is a column vector of length N-K. When puncturing a code, a "1" in the Puncture vector corresponds to a bit that is not punctured, and a "0" corresponds to a bit that is punctured.

The optional 'Erasures' input is a frame-based binary column vector the same size as the codeword input. Erasure values of "1" correspond to erased bits in the same position in the codeword, and erasure values of "0" correspond to non-erased bits.

Parameters:
Codeword length, N: 15
Message length, K: 5
☐ Specify primitive polynomial
☐ Specify generator polynomial
☐ Punctured code
☐ Enable erasures input port
☐ Output number of corrected errors

OK | Cancel | Help | Apply

Codeword length, N
> The codeword length.

Message length, K
> The message length.

Specify primitive polynomial

Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial

A row vector that represents the binary coefficients of the primitive polynomial in order of descending powers.

This field defaults to `de2bi(primpoly(4, 'nodisplay'), 'left-msb')`, corresponding to a (15,5) code.

This field appears only when you select **Specify primitive polynomial**.

Specify generator polynomial

Selecting this check box enables the field **Generator polynomial**.

Generator polynomial

A row vector that represents the binary coefficients of the generator polynomial in order of descending powers.

The length of the **Generator polynomial** must be *N*-*K*+1.

This field defaults to `bchgenpoly(15,5)`.

This field appears only when you select **Specify generator polynomial**.

Disable generator polynomial checking

This check box appears only when you select **Specify generator polynomial**.

Each time a model initializes, the block performs a polynomial check. This check verifies that $X^N + 1$ is divisible by the user-defined generator polynomial, where *N* represents the full code word length. Selecting this check box disables the polynomial check. For larger codes, disabling the check speeds up the simulation process. You should always run the check at least once before disabling this feature.

BCH Decoder

Puncture code
> Selecting this check box enables the field **Puncture vector**.

Puncture vector
> This field is available only when **Puncture code** is selected.
>
> A column vector of length *N-K*. A value of 1 in the **Puncture vector** corresponds to a bit that is not punctured, and a 0 corresponds to a bit that is punctured.
>
> The default value is [ones(8,1); zeros(2,1)].

Enable erasures input port
> Selecting this check box will open the ports, Era and Err.
>
> Through the Era port, you can input a frame-based binary column vector the same size as the codeword input. The number of errors corrected is output through the Err port.
>
> Erasure values of 1 correspond to erased bits in the same position in the codeword, and values of 0 correspond to nonerased bits.

Output number of corrected errors
> Selecting this check box gives the block an additional output port, which indicates the number of errors the block detected in the input codeword.

Pair Block   BCH Encoder

References
[1] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

[2] Berlekamp, Elwyn R., *Algebraic Coding Theory*, New York, McGraw-Hill, 1968.

[3] Clark, George C., Jr., and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

See Also   bchdec (in Communications Toolbox documentation)
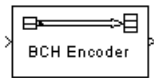
BCH Encoder

Purpose   Create BCH code from binary vector data

Library   Block sublibrary of Channel Coding

Description   This block supports punctures ("Shortening, Puncturing, and Erasures" provides a tutorial).

The BCH Encoder block creates a BCH code with message length *K* and codeword length (*N* - number of punctures). You specify both *N* and *K* directly in the dialog box.

The input must be a frame-based column vector with an integer multiple of *K* elements. Each group of *K* input elements represents one message word to be encoded.

If the encoder is processing multiple codewords per frame, then the same puncture pattern holds for all codewords.

For a given codeword length *N*, only specific message lengths *K* are valid for a BCH code. For a full length BCH code, *N* must be of the form 2^M-1, where $3 \leq M \leq 16$. If *N* is less than 2^M-1, the block assumes that the code has been shortened by length 2^{M-1} - *N*. However, if *N* is greater than or equal to 2^{M-1}, **Primitive polynomial** must be specified to appropriately set the value of *M*.

No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability. For a list of some valid values of *K* corresponding to values of *N* up to 511, see the bchenc reference page in the Communications Toolbox documentation.

This block supports double and boolean data types.

The primitive and generator polynomials may be specified in their respective fields, which appear after selecting their corresponding check boxes.

**Dialog
Box**

Note For a puncture vector, a 1 means that the data symbol is passed unaltered, and a 0 means that the data symbol is punctured (i.e., removed) from the data stream. This convention is carried for both the encoder and the decoder.

Codeword length, N
> The codeword length.

BCH Encoder

Message length, K
> The message length.

Specify primitive polynomial
> Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial
> A row vector that represents the binary coefficients of the primitive polynomial in order of descending powers.
>
> This field defaults to `de2bi(primpoly(4, 'nodisplay'), 'left-msb')`, corresponding to a (15,5) code.
>
> This field is available only when you select **Specify primitive polynomial**.

Specify generator polynomial
> Selecting this check box enables the field **Generator polynomial**.

Generator polynomial
> A row vector that represents the binary coefficients of the generator polynomial in order of descending powers.
>
> The length of the **Generator polynomial** must be *N-K*+1.
>
> This field defaults to `bchgenpoly(15,5)`.
>
> This field appears only when you select **Specify generator polynomial**.

Disable generator polynomial checking
> This check box appears only when you select **Specify generator polynomial**.
>
> Each time a model initializes, the block performs a polynomial check. This check verifies that $X^N + 1$ is divisible by the user-defined generator polynomial, where *N* represents the full code word length. Selecting this check box disables the polynomial check. For larger codes, disabling the check speeds up

the simulation process. You should always run the check at least once before disabling this feature.

Puncture code

Selecting this check box enables the field **Puncture vector**.

Puncture vector

A column vector of length *N-K*. A value of 1 in the **Puncture vector** corresponds to a bit that is not punctured, and a 0 corresponds to a bit that is punctured.

The field defaults to [ones(8,1); zeros(2,1)].

This field appears only when you select **Puncture code**.

Pair Block	BCH Decoder
References	[1] Clark, George C., Jr., and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.
See Also	bchenc (in Communications Toolbox documentation)

Bernoulli Binary Generator

Purpose　　　　　Generate Bernoulli-distributed random binary numbers

Library　　　　　Random Data Sources sublibrary of Comm Sources
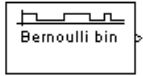
Description　　　The Bernoulli Binary Generator block generates random binary numbers using a Bernoulli distribution. The Bernoulli distribution with parameter p produces zero with probability p and one with probability 1-p. The Bernoulli distribution has mean value 1-p and variance p(1-p). The **Probability of a zero** parameter specifies p, and can be any real number between zero and one.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See "Signal Attribute Parameters for Random Sources" in *Communications Blockset User's Guide* for more details.

The number of elements in the **Initial seed** and **Probability of a zero** parameters becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** and **Probability of a zero** parameters becomes the shape of a sample-based two-dimensional output signal.

**Dialog
Box**

Source Block Parameters: Bernoulli Binary Generator ☒

┌─ Bernoulli Binary Generator ─────────────────────────────
Generate a Bernoulli random binary number.
To generate a vector output, specify the probability as a vector.

┌─ Parameters ───

Probability of a zero: 0.5

Initial seed: 61

Sample time: 1

☐ Frame-based outputs

☐ Interpret vector parameters as 1-D

Output data type: double ▼

OK   Cancel   Help

Opening this dialog box causes a running simulation to pause.
See "Changing Source Block Parameters" in the online Simulink
documentation for details.

Probability of a zero

The probability with which a zero output occurs.

Initial seed

The initial seed value for the random number generator. The seed
can be either a vector of the same length as the **Probability of
a zero** parameter, or a scalar.

Sample time

The period of each sample-based vector or each row of a
frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based.
This box is active only if **Interpret vector parameters as 1-D**
is unchecked.

Bernoulli Binary Generator

Samples per frame

> The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

> If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

> The output type of the block can be specified as a `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, or `double`. By default, the block sets this to `double`. Single outputs may lead to different results when compared with double outputs for the same set of parameters.

See Also   Random Integer Generator, Binary Symmetric Channel; `randint` (in Communications Toolboxdocumentation), `rand` (built-in MATLAB function)

Purpose   Decode systematic cyclic code to recover binary vector data

Library   Block sublibrary of Channel Coding
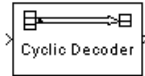
Description   The Binary Cyclic Decoder block recovers a message vector from
a codeword vector of a binary systematic cyclic code. For proper
decoding, the parameter values in this block should match those in the
correspondingBinary Cyclic Encoder block.

If the cyclic code has message length K and codeword length N, then N
must have the form 2^M-1 for some integer M greater than or equal to 3.

The input must contain exactly N elements. If it is frame-based, then it
must be a column vector. The output is a vector of length K.

You can determine the systematic cyclic coding scheme in one of two
ways:

- To create an [N,K] code, enter N and K as the first and second
  dialog parameters, respectively. The block computes an appropriate
  generator polynomial, namely, `cyclpoly(N,K,'min')`.

- To create a code with codeword length N and a particular
  degree-(N-K) binary *generator polynomial*, enter N as the first
  parameter and a binary vector as the second parameter. The vector
  represents the generator polynomial by listing its coefficients in order
  of ascending exponents. You can create cyclic generator polynomials
  using the Communications Toolbox `cyclpoly` function.

This block supports `double` and `boolean` data types.

Binary Cyclic Decoder

Dialog Box
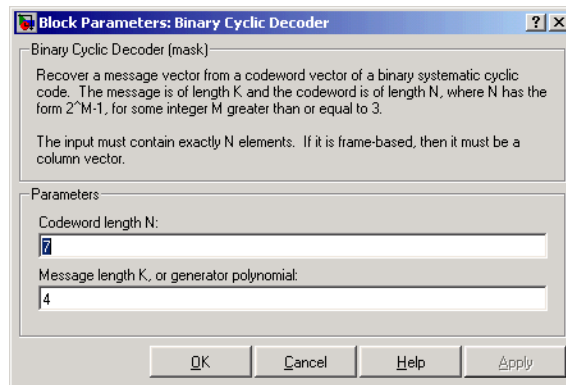
Codeword length N

The codeword length N, which is also the input vector length.

Message length K, or generator polynomial

Either the message length, which is also the output vector length; or a binary vector that represents the generator polynomial for the code.

Pair Block   Binary Cyclic Encoder

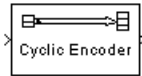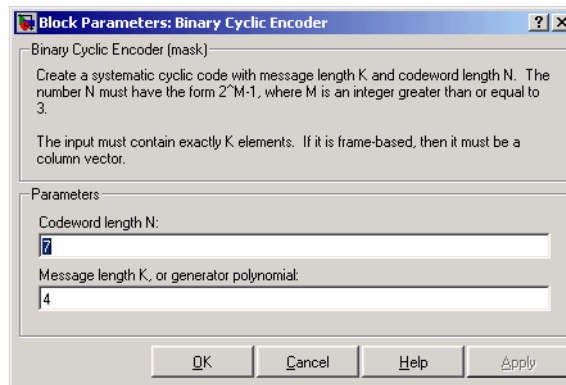**See Also**   cyclpoly (Communications Toolbox)

Purpose   Create systematic cyclic code from binary vector data

Library   Block sublibrary of Channel Coding

Description   The Binary Cyclic Encoder block creates a systematic cyclic code with
message length K and codeword length N. The number N must have the
form 2^M-1, where M is an integer greater than or equal to 3.

The input must contain exactly K elements. If it is frame-based, then it
must be a column vector. The output is a vector of length N.

You can determine the systematic cyclic coding scheme in one of two
ways:

- To create an [N,K] code, enter N and K as the first and second
  dialog parameters, respectively. The block computes an appropriate
  generator polynomial, namely, `cyclpoly(N,K,'min')`.

- To create a code with codeword length N and a particular
  degree-(N-K) binary *generator polynomial*, enter N as the first
  parameter and a binary vector as the second parameter. The vector
  represents the generator polynomial by listing its coefficients in order
  of ascending exponents. You can create cyclic generator polynomials
  using the Communications Toolbox `cyclpoly` function.

This block supports `double` and `boolean` data types.

Binary Cyclic Encoder

Dialog Box

Codeword length N
> The codeword length, which is also the output vector length.

Message length K, or generator polynomial
> Either the message length, which is also the input vector length; or a binary vector that represents the generator polynomial for the code.

Pair Block Binary Cyclic Decoder

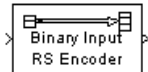**See Also** cyclpoly (in the Communications Toolbox documentation)

Purpose   Create Reed-Solomon code from binary vector data

Library   Block sublibrary of Channel Coding

Description   This block supports punctures ("Shortening, Puncturing, and Erasures" provides a tutorial).

Note For a puncture vector, a 1 means that the data symbol is passed unaltered, and a 0 means that the data symbol is punctured (i.e., removed) from the data stream. This convention is carried for both the encoder and the decoder.

The Binary-Input RS Encoder block creates a Reed-Solomon code with message length, K, and codeword length, (N - number of punctures). You specify both N and K directly in the dialog box. The symbols for the code are binary sequences of length M, corresponding to elements of the Galois field $GF(2^M)$, where the first bit in each sequence is the most significant bit. Restrictions on M and N are given in "Restrictions on the M and the Codeword Length N" on page 2-49 below. The difference N-K must be an even integer.

This block can output shortened codewords when N and K are appropriately specified. To specify output codewords that are shortened by a length S, N and K must be specified in the dialog box as $N_{full} - S$ and $K_{full} - S$, where N_{full} and K_{full} are the N and K of an unshortened code. If $S < (N_{full} + 1)/2$, the encoder can automatically determine the value of N_{full} and K_{full}. However, if $S \geq (N_{full} + 1)/2$, **Primitive polynomial** must be specified in order to properly define the extension field for the code.

The input and output are binary-valued signals that represent messages and codewords, respectively. The input must be a frame-based column vector whose length is an integer multiple of M*K. The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. The output is a frame-based column vector whose length is the

same integer multiple of M*(N - number of punctures), and whose data type is inherited from the input. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon Only)" in *Communications Blockset User's Guide*.

If the encoder is processing multiple codewords per frame, then the same puncture pattern holds for all codewords.

The default value of M is the smallest integer that is greater than or equal to log2(N+1), that is, `ceil(log2(N+1))`. You can change the value of M from the default by specifying the primitive polynomial for $GF(2^M)$, as described in "Specifying the Primitive Polynomial" on page 2-48 below. If N is less than 2^M-1, the block assumes that the code has been shortened by length 2^{M-1} - N.

Each M*K input bits represent K integers between 0 and 2^M-1. Similarly, each M*(N - number of punctures) output bits represent N integers between 0 and 2^M-1. These integers in turn represent elements of the Galois field $GF(2^M)$.

An (N,K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword.

Specifying the Primitive Polynomial

You can specify the primitive polynomial that defines the finite field $GF(2^M)$, corresponding to the integers that form messages and codewords. To do so, first select **Specify primitive polynomial**. Then, set **Primitive polynomial** to a binary row vector that represents a primitive polynomial over GF(2) of degree M, in descending order of powers. For example, to specify the polynomial x^3+x+1, enter the vector `[1 0 1 1]`.

If you do not select **Specify primitive polynomial**, the block uses the default primitive polynomial of degree M = ceil(log2(N+1)). You can display the default polynomial by entering `primpoly(ceil(log2(N+1)))` at the MATLAB prompt.

Restrictions on the M and the Codeword Length N

The restrictions on the degree M of the primitive polynomial and the codeword length N are as follows:

- If you do not select **Specify primitive polynomial**, N must lie in the range $3 < N < 2^{16}-1$.

- If you do select **Specify primitive polynomial**, N must lie in the range $3 \leq N < 2^{16}-1$ and M must lie in the range $3 \leq M \leq 16$.

Specifying the Generator Polynomial

You can specify the generator polynomial for the Reed-Solomon code. To do so, first select **Specify generator polynomial**. Then, in the **Generator polynomial** field, enter an integer row vector whose entries are between 0 and 2^M-1. The vector represents a polynomial, in descending order of powers, whose coefficients are elements of $GF(2^M)$ represented in integer format. See the section "Integer Format (Reed-Solomon Only)" for more information about integer format. The generator polynomial must be equal to a polynomial with a factored form

$$g(x) = (x+A^b)(x+A^{b+1})(x+A^{b+2})...(x+A^{b+N-K-1})$$

where A is the primitive element of the Galois field over which the input message is defined, and *b* is a non-negative integer.

If you do not select **Specify generator polynomial**, the block uses the default generator polynomial, corresponding to *b*=1, for Reed-Solomon encoding. You can display the default generator polynomial by entering `rsgenpoly(N1,K1)`, where N1=2^M-1 and K1=K+(N1-N), at the MATLAB prompt, if you are using the default primitive polynomial. If the **Specify primitive polynomial** box is selected, and you specify the primitive polynomial specified as `poly`, the default generator polynomial is `rsgenpoly(N1,K1,poly)`.
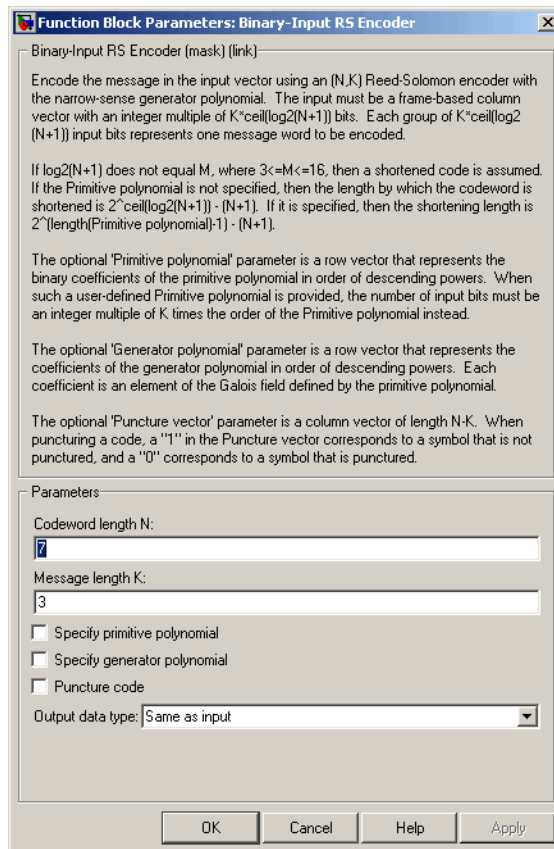
Examples   Suppose M = 3, N = 2^3-1 = 7, and K = 5. Then a message is a binary vector of length 15 that represents 5 three-bit integers. A corresponding codeword is a binary vector of length 21 that represents 7 three-bit

integers. The following figure shows the codeword that would result from a particular message word. The integer format equivalents illustrate that the highest order bit is at the left.

Message input: [0 1 1 1 1 1 0 0 1 0 0 0 0 0 1] ——— [3 7 1 0 1]
in integer format

Binary-Input RS Encoder
with N = 7, K = 5

Code output: [0 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1] ——— [3 7 1 0 1 3 3]
in integer format
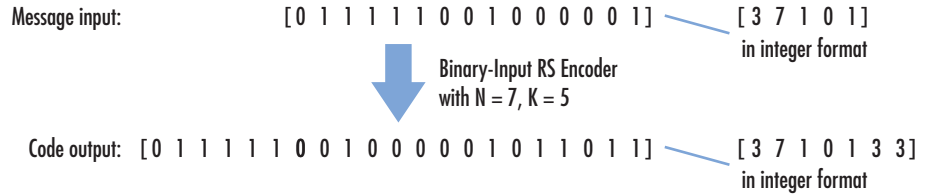
Dialog Box

Codeword length N

 The codeword length. The output has vector length NC*M*(N - NP), where NC is the number of codewords being output, and NP is the number of punctures per codeword.

Message length K

 The message length. The input has vector length NM*M*K, where NM is the number of messages per frame being input.

Binary-Input RS Encoder

Specify primitive polynomial
> Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial
> This field is available only when **Specify primitive polynomial** is selected.
>
> Binary row vector representing the primitive polynomial in descending order of powers.

Specify generator polynomial
> Selecting this check box enables the field **Generator polynomial**.

Generator polynomial
> This field is available only when **Specify generator polynomial** is selected.
>
> Integer row vector, whose entries are in the range from 0 to 2^M-1, representing the generator polynomial in descending order of powers.

Puncture code
> Selecting this check box enables the field **Puncture vector**.

Puncture vector
> This field is available only when **Puncture code** is selected.
>
> A column vector of length N-K. A value of 1 in the **Puncture vector** corresponds to an M-bit symbol that is not punctured, and a `0` corresponds to an M-bit symbol that is punctured.
>
> The default value is `[ones(2,1); zeros(2,1)]`.

Output data type
> The output type of the block can be specified as `Same as input`, `boolean`, or `double`. By default, the block sets this to `Same as input`.

Pair Block   Binary-Output RS Decoder

See Also   Integer-Input RS Encoder

Binary Linear Decoder

Purpose   Decode linear block code to recover binary vector data

Library   Block sublibrary of Channel Coding
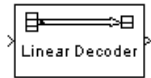
Description   The Binary Linear Decoder block recovers a binary message vector from
   a binary codeword vector of a linear block code.

The **Generator matrix** parameter is the generator matrix for the block
code. For proper decoding, this should match the **Generator matrix**
parameter in the corresponding Binary Linear Encoder block. If N is
the codeword length of the code, then **Generator matrix** must have N
columns. If K is the message length of the code, then the **Generator
matrix** parameter must have K rows.

The input must contain exactly N elements. If it is frame-based, then it
must be a column vector. The output is a vector of length K.

The decoder tries to correct errors, using the **Decoding table**
parameter. If **Decoding table** is the scalar 0, then the block defaults to
the table produced by the Communications Toolbox function `syndtable`.
Otherwise, **Decoding table** must be a 2^{N-K}-by-N binary matrix. The *r*th
row of this matrix is the correction vector for a received binary codeword
whose syndrome has decimal integer value *r*-1. The syndrome of a
received codeword is its product with the transpose of the parity-check
matrix.

This block supports `double` and `boolean` data types.

**Dialog
Box**
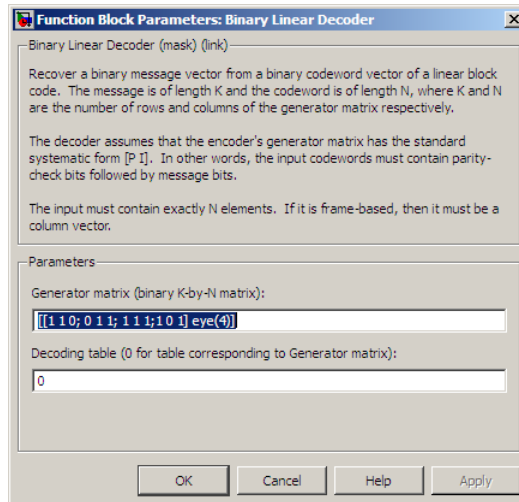
Generator matrix

Generator matrix for the code; same as in Binary Linear Encoder block.

Decoding table

Either a 2^{N-K}-by-N matrix that lists correction vectors for each codeword's syndrome; or the scalar 0, in which case the block defaults to the table corresponding to the **Generator matrix** parameter.
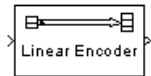
Pair Block   Binary Linear Encoder

Binary Linear Encoder

Purpose   Create linear block code from binary vector data

Library   Block sublibrary of Channel Coding

Description

The Binary Linear Encoder block creates a binary linear block code using a generator matrix that you specify. If K is the message length of the code, then the **Generator matrix** parameter must have K rows. If N is the codeword length of the code, then **Generator matrix** must have N columns.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N.

This block supports `double` and `boolean` data types.

Dialog Box
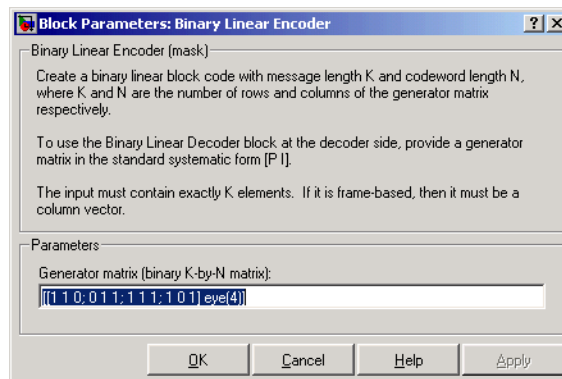
Generator matrix
    A K-by-N matrix, where K is the message length and N is the codeword length.

Pair Block   Binary Linear Decoder

Purpose   Decode Reed-Solomon code to recover binary vector data

Library   Block sublibrary of Channel Coding
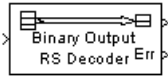
Description   The Binary-Output RS Decoder block recovers a binary message vector from a binary Reed-Solomon codeword vector. For proper decoding, the parameter values in this block should match those in the corresponding Binary-Input RS Encoder block.

This block supports punctures ("Shortening, Puncturing, and Erasures" provides a tutorial).

Note 1s and 0s have precisely opposite meanings for the puncture and erasure vectors. For a puncture vector, a 1 means that the data symbol is passed unaltered, and a 0 means that the data symbol is punctured (i.e., removed) from the data stream. For an erasure vector, a 1 means that the data symbol is to be replaced with an erasure symbol, and a 0 means that the data symbol is passed unaltered. These conventions are carried for both the encoder and the decoder.

The Reed-Solomon code has message length, K, and codeword length, (N - number of punctures). You specify both N and K directly in the dialog box. The symbols for the code are binary sequences of length M, corresponding to elements of the Galois field $GF(2^M)$, where the first bit in each sequence is the most significant bit. Restrictions on M and N are described in "Restrictions on the M and the Codeword Length N" on page 2-49. The difference N-K must be an even integer.

This block can output shortened codewords when N and K are appropriately specified. To specify output codewords that are shortened by a length S, N and K must be specified in the dialog box as $N_{full} - S$ and $K_{full} - S$, where N_{full} and K_{full} are the N and K of an unshortened code. If $S < (N_{full} + 1)/2$, the encoder can automatically determine the value of N_{full} and K_{full}. However, if $S \geq (N_{full} + 1)/2$, **Primitive**

Binary-Output RS Decoder

polynomial must be specified in order to properly define the extension field for the code.

The input and output are binary-valued signals that represent codewords and messages, respectively. The input must be a frame-based column vector whose length is an integer multiple of M*(N - number of punctures). The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. The output is a frame-based column vector whose length is the same integer multiple of M*K, and whose data type is inherited from the input. For more information on representing data for Reed-Solomon codes, see "Integer Format (Reed-Solomon Only)" in *Communications Blockset User's Guide*.

If the decoder is processing multiple codewords per frame, then the same puncture pattern holds for all codewords.

The default value of M is `ceil(log2(N+1))`, that is, the smallest integer greater than or equal to log2(N+1). You can change the value of M from the default by specifying the primitive polynomial for GF(2^M), as described in "Specifying the Primitive Polynomial" on page 2-48 below. If N is less than 2^M-1, the block uses a shortened Reed-Solomon code.

You can also specify the generator polynomial for the Reed-Solomon code, as described in "Specifying the Generator Polynomial" on page 2-49.
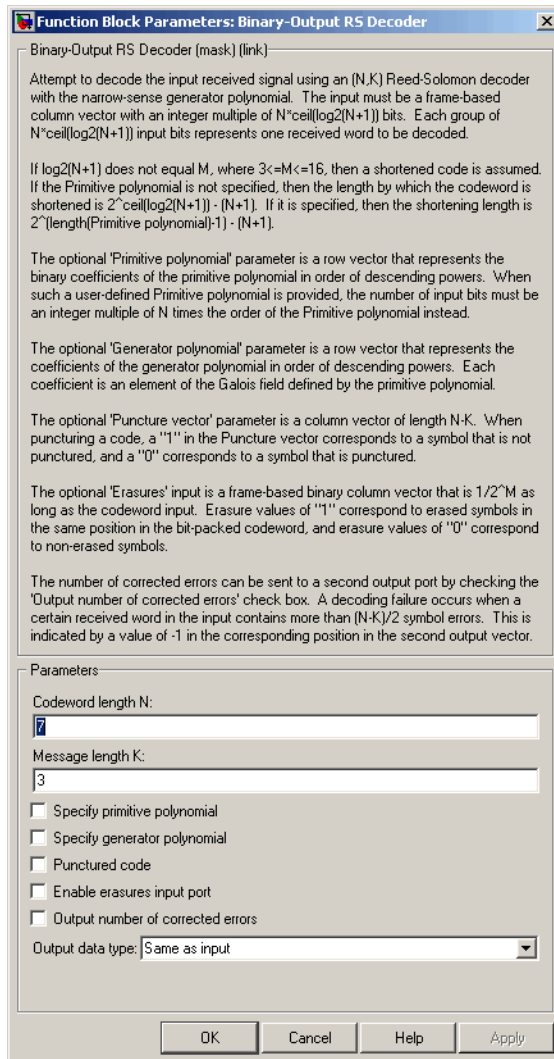
Each M*K input bits represent K integers between 0 and 2^M-1. Similarly, each M*(N - number of punctures) output bits represent N integers between 0 and 2^M-1. These integers in turn represent elements of the Galois field GF(2^M).

The second output is a vector of the number of errors detected during decoding of the codeword. A -1 indicates that the block detected more errors than it could correct using the coding scheme. An (N,K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword. The data type of this output is also inherited from the input signal.

You can disable the second output by deselecting **Output port for number of corrected errors.** This removes the block's second output port.

Binary-Output RS Decoder

Dialog Box

Binary-Output RS Decoder (mask) (link)

Attempt to decode the input received signal using an (N,K) Reed-Solomon decoder with the narrow-sense generator polynomial. The input must be a frame-based column vector with an integer multiple of N*ceil(log2(N+1)) bits. Each group of N*ceil(log2(N+1)) input bits represents one received word to be decoded.

If log2(N+1) does not equal M, where 3<=M<=16, then a shortened code is assumed. If the Primitive polynomial is not specified, then the length by which the codeword is shortened is 2^ceil(log2(N+1)) - (N+1). If it is specified, then the shortening length is 2^(length(Primitive polynomial)-1) - (N+1).

The optional 'Primitive polynomial' parameter is a row vector that represents the binary coefficients of the primitive polynomial in order of descending powers. When such a user-defined Primitive polynomial is provided, the number of input bits must be an integer multiple of N times the order of the Primitive polynomial instead.

The optional 'Generator polynomial' parameter is a row vector that represents the coefficients of the generator polynomial in order of descending powers. Each coefficient is an element of the Galois field defined by the primitive polynomial.

The optional 'Puncture vector' parameter is a column vector of length N-K. When puncturing a code, a "1" in the Puncture vector corresponds to a symbol that is not punctured, and a "0" corresponds to a symbol that is punctured.

The optional 'Erasures' input is a frame-based binary column vector that is 1/2^M as long as the codeword input. Erasure values of "1" correspond to erased symbols in the same position in the bit-packed codeword, and erasure values of "0" correspond to non-erased symbols.

The number of corrected errors can be sent to a second output port by checking the 'Output number of corrected errors' check box. A decoding failure occurs when a certain received word in the input contains more than (N-K)/2 symbol errors. This is indicated by a value of -1 in the corresponding position in the second output vector.

Parameters

Codeword length N:

7

Message length K:

3

☐ Specify primitive polynomial

☐ Specify generator polynomial

☐ Punctured code

☐ Enable erasures input port

☐ Output number of corrected errors

Output data type: Same as input ▼

| OK | Cancel | Help | Apply |

Codeword length N

The codeword length. The input has vector length NC*M*(N - NP), where NC is the number of codewords being output, and NP is the number of punctures per codeword.

Message length K

The message length. The first output has vector length NM*M*K, where NM is the number of messages per frame being output.

Specify primitive polynomial

Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial

This field is available only when **Specify primitive polynomial** is selected.

Binary row vector representing the primitive polynomial in descending order of powers.

Specify generator polynomial

Selecting this check box enables the field **Generator polynomial**.

Generator polynomial

This field is available only when **Specify generator polynomial** is selected.

Integer row vector, whose entries are in the range from 0 to 2^M-1, representing the generator polynomial in descending order of powers.

Puncture code

Selecting this check box enables the field **Puncture vector**.

Puncture vector

This field is available only when **Puncture code** is selected.

A column vector of length N-K. A value of 1 in the **Puncture vector** corresponds to an M-bit symbol that is not punctured, and a 0 corresponds to an M-bit symbol that is punctured.

Binary-Output RS Decoder

The default value is `[ones(2,1); zeros(2,1)]`.

Enable erasures input port

Selecting this check box will open the port, `Era`.

Through the port, you can input a frame-based binary column vector that is 1/M times as long as the codeword input.

Erasure values of `1` correspond to erased symbols in the same position in the bit-packed codeword, and values of `0` correspond to nonerased symbols.

Output number of corrected errors

When you select this box, the block outputs the number of corrected errors in each word through a second output port.

Output data type

The output type of the block can be specified as `Same as input`, `boolean`, or `double`. By default, the block sets this to `Same as input`.

Algorithm

This block uses the Berlekamp-Massey decoding algorithm. For information about this algorithm, see the references listed below.

Pair Block

Binary-Input RS Encoder

References

[1] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

[2] Berlekamp, Elwyn R., *Algebraic Coding Theory*, New York, McGraw-Hill, 1968.

[3] Clark, George C., Jr., and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

See Also

Integer-Output RS Decoder

Purpose   Introduce binary errors

Library   Channels
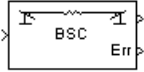
Description The Binary Symmetric Channel block introduces binary errors to the signal transmitted through this channel.

The input port is the transmitted binary signal. The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently, and introduces an error in a given spot with probability **Error probability**.

This block uses the Signal Processing Blockset Random Source block to generate the noise. Random numbers are generated using the Ziggurat method, which is the same method used by the MATLAB randn function. The **Initial seed** parameter in this block initializes the noise generator. **Initial seed** can be either a scalar or a vector whose length matches the number of channels in the input signal. For details on **Initial seed**, see the Random Source block reference page in the Signal Processing Blockset documentation set.

The first output port is the binary signal that has passed through the channel. The second output port is the vector of errors that were introduced. To suppress the second output port, clear the **Output error vector** check box.

Binary Symmetric Channel

Dialog Box
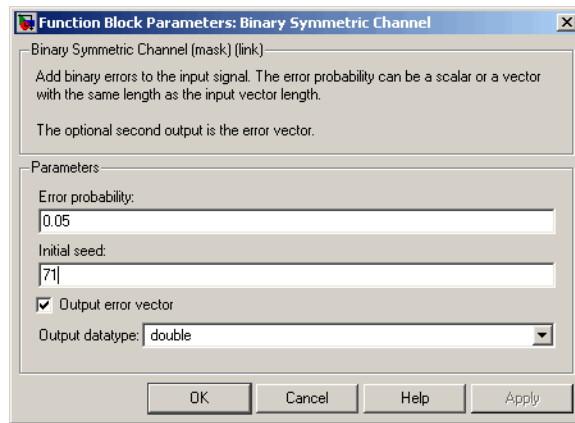
Error probability

The probability that a binary error will occur. The value of this parameter must be between zero and one.

Initial seed

The initial seed value for the random number generator.

Output error vector

If this box is checked, then the block outputs the vector of errors.

Output datatype

You can set the output data type to double or boolean.

See Also   Bernoulli Binary Generator

Purpose   Map bipolar signal into unipolar signal in range [0, M-1]

Library   Utility Blocks
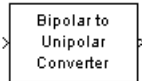
Description   The Bipolar to Unipolar Converter block maps the bipolar input signal to a unipolar output signal. If the input consists of integers in the set {-M+1, -M+3, -M+5,..., M-1}, where M is the **M-ary number** parameter, then the output consists of integers between 0 and M-1. This block is only designed to work when the input value is within the set {-M+1, -M+3, -M+5,..., M-1}, where M is the **M-ary number** parameter. If the input value is outside of this set of integers the output may not be valid.

The table below shows how the block's mapping depends on the **Polarity** parameter.

Polarity Parameter Value	Output Corresponding to Input Value of k
Positive	(M-1+k)/2
Negative	(M-1-k)/2

Dialog Box
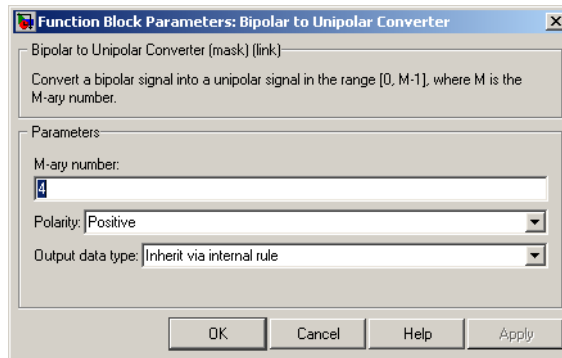
M-ary number
    The number of symbols in the bipolar or unipolar alphabet.

Bipolar to Unipolar Converter

Polarity

A value of `Positive` causes the block to maintain the relative ordering of symbols in the alphabets. A value of `Negative` causes the block to reverse the relative ordering of symbols in the alphabets.

Output Data Type

The type of bipolar signal produced at the block's output.

The block supports the following output data types:

- `Inherit via internal rule`
- `Same as input`
- `double`
- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean`

When the parameter is set to its default setting, `Inherit via internal rule`, the block determines the output data type based on the input data type.

- If the input signal is floating-point (either `single` or `double`), the output data type is the same as the input data type.

- If the input data type is not floating-point:

  - Based on the **M-ary number** parameter, the output data type is the ideal unsigned integer output word length required to contain the range [0 M-1] and is computed as follows:

ideal word length = ceil(log2(M))

- The block sets the output data type to be an unsigned integer, based on the smallest word length (in bits) that can fit best the computed ideal word length.

Note The selections in the **Hardware Implementation** pane pertaining to word length constraints do not affect how this block determines output data types.

Examples If the input is [-3; -1; 1; 3], the **M-ary number** parameter is 4, and the **Polarity** parameter is Positive, then the output is [0; 1; 2; 3]. Changing the **Polarity** parameter to Negative changes the output to [3; 2; 1; 0].

If the value for the **M-ary number** is 2^8 the block gives an output of uint8.

If the value for the **M-ary number** is 2^8+1 the block gives an output of uint16.

Pair Block Unipolar to Bipolar Converter

Bit to Integer Converter

Purpose   Map vector of bits to corresponding vector of integers

Library   Utility Blocks

Description The Bit to Integer Converter block maps groups of bits in the input vector to integers in the output vector.
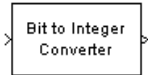
For unsigned integers, if M is the **Number of bits per integer**, then the block maps each group of M bits to an integer between 0 and 2^M-1. As a result, the output vector length is 1/M times the input vector length. For signed integers, if M is the **Number of bits per integer**, then the block maps each group of M bits to an integer between -2^{M-1} and 2^{M-1}-1.

If the input signal is a sample-based input, then it must be a vector whose length equals the value for the **Number of bits per integer** parameter. If the input is frame-based, then it must be a column vector whose length is an integer multiple of **Number of bits per integer**.

The block can accept input data types `double`, `single`, `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` and `ufix1`.

Dialog Box

Function Block Parameters: Bit to Integer Converter

Bit to Integer Converter (mask) (link)

Map a vector of bits to a corresponding vector of integer values. The 'Number of bits per integer' parameter defines how many bits are mapped for each output.

This block is single-rate and single-channel. The input can be either a scalar, a sample-based vector, or a frame-based column vector. For sample-based inputs, the input length must equal the 'Number of bits per integer'. For frame-based inputs, the input length must be an integer multiple of the 'Number of bits per integer'. The output is always a column vector with a length equal to 'Number of bits per integer' times smaller than the input signal length.

Parameters

Number of bits per integer:

3

Input bit order: MSB first

After bit packing, treat resulting integer values as: Unsigned

Output data type: Inherit via internal rule

[OK] [Cancel] [Help] [Apply]

Number of bits per integer

> The number of input bits that the block maps to each integer of the output. This parameter must be an integer between 1 and 32.

Input bit order

> Defines whether the first bit of the input signal is the most significant bit (MSB) or the least significant bit (LSB). The default selection is MSB.

After bit packing, treat resulting integer value as

> Indicates if the integer value input ranges should be treated as signed or unsigned. The default setting is Unsigned.

> **Note** This parameter setting determines which **Output data type** selections are available.

Output data type

> If the input values are unsigned integers, you can choose from the following **Output data type** options:
>
> - Inherit via internal rule
> - Smallest integer
> - Same as input
> - double
> - single
> - int8
> - uint8
> - int16
> - uint16
> - int32
> - uint32

Bit to Integer Converter

If the input values are signed integers, you can choose from the following **Output data type** options:

- `Inherit via internal rule`
- `Smallest integer`
- `double`
- `single`
- `int8`
- `int16`
- `int32`

The default selection for this parameter is `Inherit via internal rule`.

When the parameter is set to `Inherit via internal rule`, the block determines the output data type based on the input data type.

- If the input signal is floating-point (either `double` or `single`), the output data type is the same as the input data type.
- If the input data type is not floating-point, the output data type is determined as if the parameter is set to `Smallest integer`.

When the parameter is set to `Smallest integer`, the software selects the output data type based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box.

- If `ASIC/FPGA` is selected, the output data type is the smallest ideal integer or fixed-point data type, based on the setting for the **Number of bits per integer** parameter.
- For all other selections, the output data type is the smallest available (signed or unsigned) integer word length that is large enough to fit the ideal minimum bit size.

Examples   Refer to the example on the Integer to Bit Converter reference page:
Fixed-Point Integer To Bit and Bit To Integer Conversion (Audio
Scrambling and Descrambling Example)

See Also

bi2de, bin2dec

Pair Block   Integer to Bit Converter

BPSK Demodulator Baseband

Purpose   Demodulate BPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The BPSK Demodulator Baseband block demodulates a signal that was
                    modulated using the binary phase shift keying method. The input is
                    a baseband representation of the modulated signal. The input can be
                    either a scalar or a frame-based column vector. The block accepts the
                    data types `double`, `single`, and signed fixed-point (in hard-decision
                    modes only).

                    The input must be a discrete-time complex signal. The block maps
                    the points $\exp(j\theta)$ and $-\exp(j\theta)$ to 0 and 1, respectively, where θ is the
                    Phase offset parameter.

Algorithm
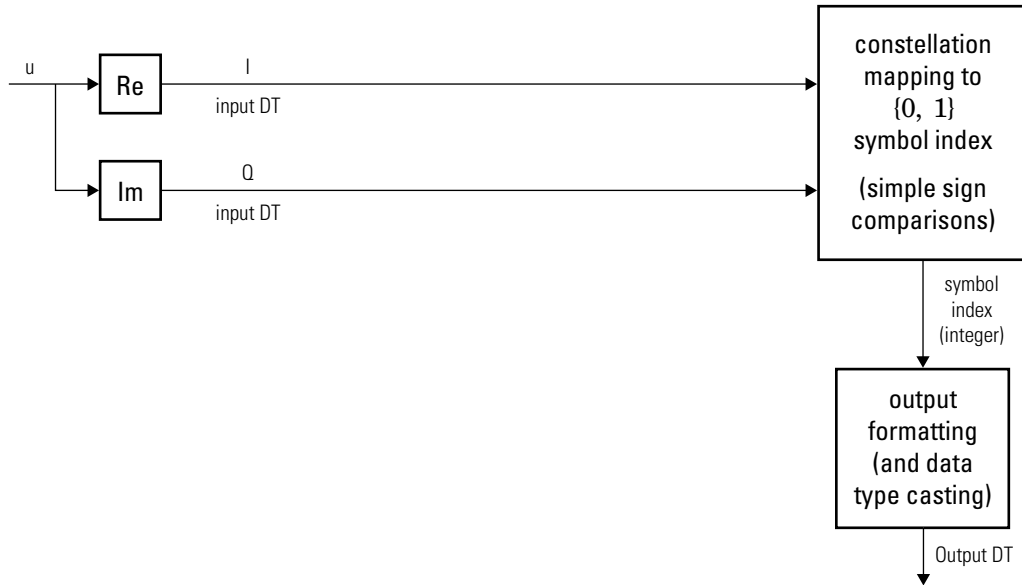
Hard-Decision BPSK Demodulator Signal Diagram for Trivial Phase

Offset (multiple of $\frac{\pi}{2}$)

BPSK Demodulator Baseband
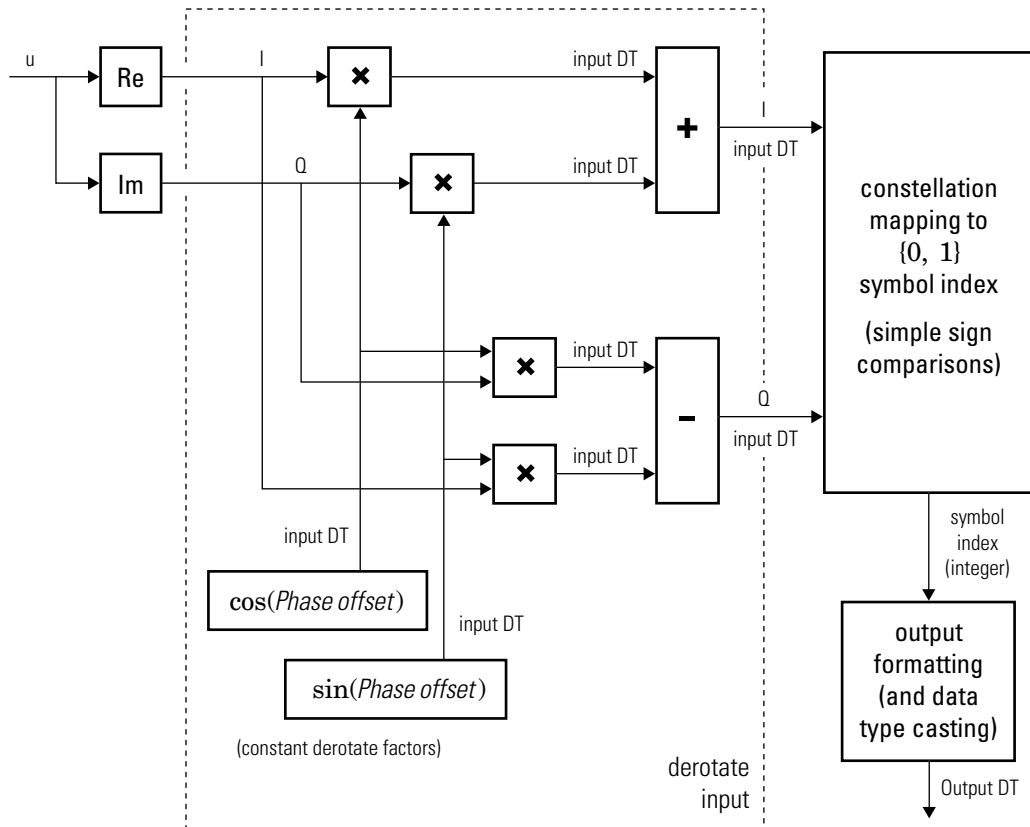
Hard-Decision BPSK Demodulator Floating-Point Signal Diagram for Nontrivial Phase Offset
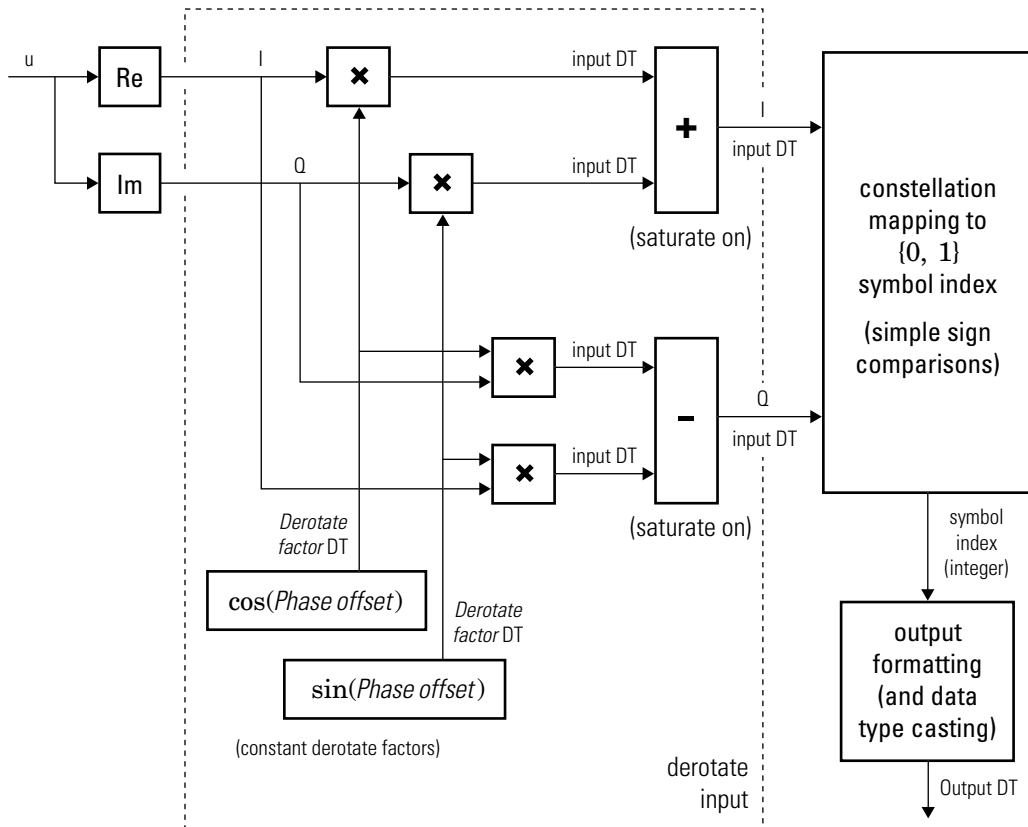
Hard-Decision BPSK Demodulator Fixed-Point Signal Diagram for Nontrivial Phase Offset

The exact LLR and approximate LLR cases (soft-decision) are described in "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications ToolboxUser's Guide*.

BPSK Demodulator Baseband

Dialog Box
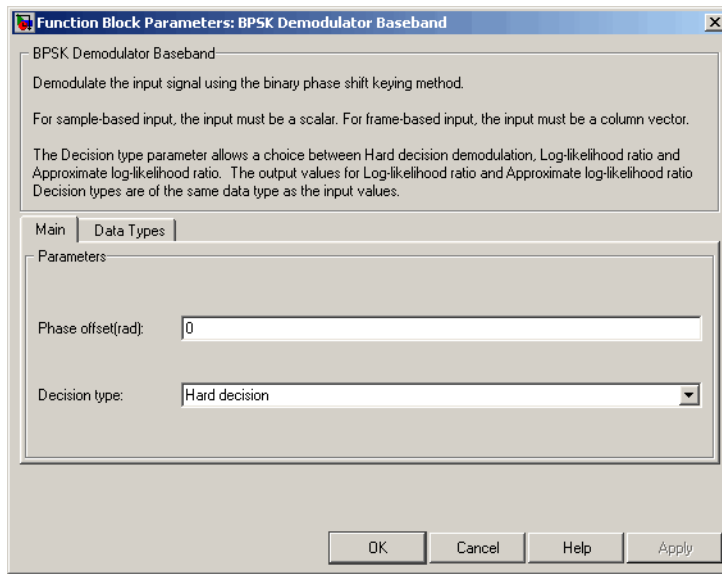
Phase offset (rad)
> The phase of the zeroth point of the signal constellation.

Decision type
> Specifies the use of hard decision, LLR, or approximate LLR during demodulation. See "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications ToolboxUser's Guide* for algorithm details.

Noise variance source
> This field appears when `Approximate log-likelihood ratio` or `Log-likelihood ratio` is selected for **Decision type**.
>
> When set to `Dialog`, the noise variance can be specified in the **Noise variance** field. When set to `Port`, a port appears on the block through which the noise variance can be input.

Noise variance

This parameter appears when the **Noise variance source** is set to Dialog and specifies the noise variance in the input signal. This parameter is tunable in normal mode, Accelerator mode and Rapid Accelerator mode.

If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameter without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

The LLR algorithm involves computing exponentials of very large or very small numbers using finite precision arithmetic and would yield:

- Inf to -Inf if **Noise variance** is very high
- NaN if **Noise variance** and signal power are both very small

In such cases, use approximate LLR, as its algorithm does not involve computing exponentials.

BPSK Demodulator Baseband
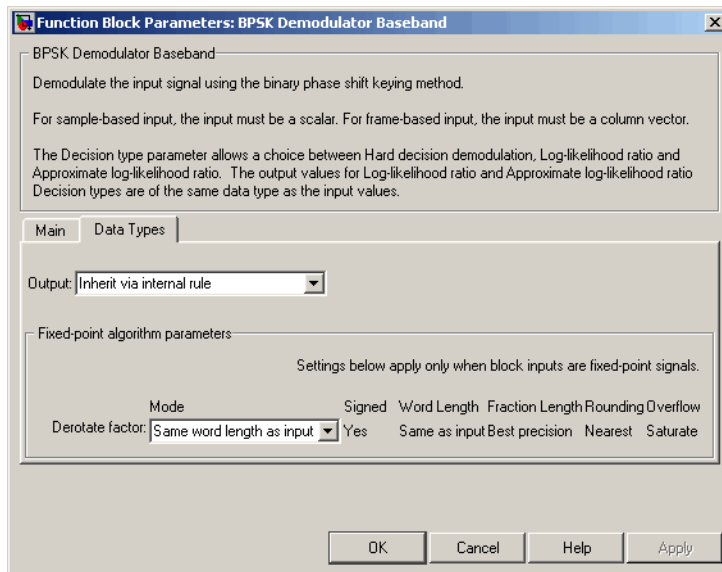
Data Types Pane for Hard-Decision

Output

When **Decision type** is set to `Hard decision`, the output data type can be set to `'Inherit via internal rule'`, `'Smallest unsigned integer'`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`.

When this parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is a floating-point type (`single` or `double`). If the input data type is fixed-point, the output data type will work as if this parameter is set to `'Smallest unsigned integer'`.

When this parameter is set to `'Smallest unsigned integer'`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model. If `ASIC/FPGA` is selected in

the **Hardware Implementation** pane, the output data type is the ideal minimum one-bit size, i.e., `ufix(1)`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., `uint8`).

Derotate factor

This parameter only applies when the input is fixed-point and

Phase offset is not a multiple of $\frac{\pi}{2}$.

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input.

Data Types Pane for Soft-Decision

When **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, the output data type is inherited from the input (e.g., if the input is of data type `double`, the output is also of data type `double`).

BPSK Demodulator Baseband

Pair Block   BPSK Modulator Baseband

See Also   M-PSK Demodulator Baseband, QPSK Demodulator Baseband, DBPSK
Demodulator Baseband

Purpose   Modulate using binary phase shift keying method

Library   PM, in Digital Baseband sublibrary of Modulation

Description The BPSK Modulator Baseband block modulates using the binary phase
shift keying method. The output is a baseband representation of the
modulated signal. For both integer and bit inputs, this block can accept
the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`,
`single`, and `double`.

The input must be a discrete-time binary-valued signal. If the input bit
is 0 or 1, respectively, then the modulated symbol is exp(jθ) or -exp(jθ)
respectively, where θ is the **Phase offset** parameter.

Constellation Visualization

The BPSK Modulator Baseband block provides the capability to
visualize a signal constellation from the block mask. This Constellation
Visualization feature allows you to visualize a signal constellation for
specific block parameters. For more information, see the Constellation
Visualization section of the *Communications Blockset User's Guide*.

Dialog Box
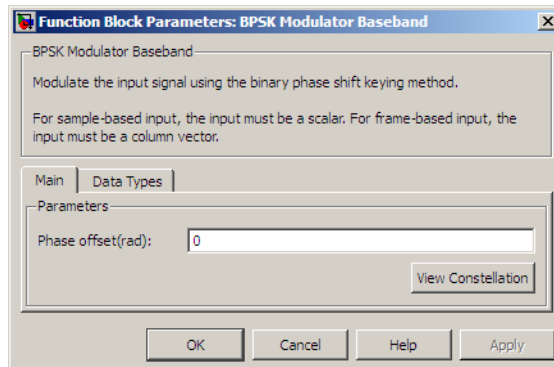
Phase offset (rad)

The phase of the zeroth point of the signal constellation.

BPSK Modulator Baseband

Output data type

The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Setting this parameter to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this parameter to `Inherit via back propagation`, sets the output data type and scaling to match the following block.

Output word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Set output fraction length to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter or when you select `User-defined` and the specified output data type is a fixed-point data type.

User-defined data type

Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point software. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Output fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for

the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   BPSK Demodulator Baseband

See Also   M-PSK Modulator Baseband, QPSK Modulator Baseband, DBPSK Modulator Baseband

Charge Pump PLL

Purpose   Implement charge pump phase-locked loop using digital phase detector

Library   Components sublibrary of Synchronization

Description   The Charge Pump PLL (phase-locked loop) block automatically adjusts
                  the phase of a locally generated signal to match the phase of an input
                  signal. It is suitable for use with digital signals.
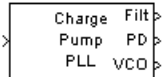
This PLL has these three components:

- A sequential logic phase detector, also called a digital phase detector
  or a phase/frequency detector.

- A filter. You specify the filter's transfer function using the **Lowpass
  filter numerator** and **Lowpass filter denominator** parameters.
  Each is a vector that gives the respective polynomial's coefficients in
  order of descending powers of *s*.

  To design a filter, you can use functions such as butter, cheby1, and
  cheby2 in Signal Processing Toolbox software. The default filter is
  a Chebyshev type II filter whose transfer function arises from the
  command below.

  ```
  [num, den] = cheby2(3,40,100,'s')
  ```

- A voltage-controlled oscillator (VCO). You specify characteristics
  of the VCO using the **VCO input sensitivity**, **VCO quiescent
  frequency**, **VCO initial phase**, and **VCO output amplitude**
  parameters.

The input signal represents the received signal. The input must be a
sample-based scalar signal. The three output ports produce:

- The output of the filter
- The output of the phase detector
- The output of the VCO

A sequential logic phase detector operates on the zero crossings of the signal waveform. The equilibrium point of the phase difference between the input signal and the VCO signal equals π. The sequential logic detector can compensate for any frequency difference that might exist between a VCO and an incoming signal frequency. Hence, the sequential logic phase detector acts as a frequency detector.

Dialog Box

Lowpass filter numerator
> The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of *s*.

Lowpass filter denominator
> The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of *s*.

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the **VCO quiescent frequency** value. The units of **VCO input sensitivity** are Hertz per volt.

VCO quiescent frequency (Hz)

The frequency of the VCO signal when the voltage applied to it is zero. This should match the frequency of the input signal.

VCO initial phase (rad)

The initial phase of the VCO signal.

VCO output amplitude

The amplitude of the VCO signal.

See Also   Phase-Locked Loop

References   For more information about digital phase-locked loops, see the works listed in"Selected Bibliography for Synchronization" in *Communications Blockset User's Guide*.

Purpose   Equalize using constant modulus algorithm

Library   Equalizers
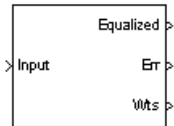
Description   The CMA Equalizer block uses a linear equalizer and the constant modulus algorithm (CMA) to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the CMA to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

When using this block, you should initialize the equalizer weights with a nonzero vector. Typically, CMA is used with differential modulation; otherwise, the initial weights are very important. A typical vector of initial weights has a 1 corresponding to the center tap and zeros elsewhere.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of the extra ports listed in the table below.

Port	Meaning	How to Enable
Err output	$y(R - |y|^2)$, where y is the equalized signal and R is a constant related to the signal constellation	Check the **Output error** check box.
Wts output	A vector listing the weights after the block has processed either the current input frame or, in	Check the **Output weights** check box.

Port	Meaning	How to Enable
	sample-based mode, the current input sample.	

Equalizer Delay

The delay between the transmitter's modulator output and the CMA equalizer output is typically unknown (unlike the delay for other adaptive equalizers in this blockset). If you need to determine the delay, you can use the Find Delay block.

Dialog Box

Function Block Parameters: CMA Equalizer

CMA Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using the CMA algorithm.

The block computes filter weights with the CMA algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a fractionally spaced (i.e. T/N-spaced) equalizer.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Parameters

Number of taps:

4

Number of samples per symbol:

1

Signal constellation:

[-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j]

Step size:

0.01

Leakage factor:

1

Initial weights:

[1 0 0 0]

☑ Output error

☐ Output weights

OK   Cancel   Help   Apply

Number of taps
> The number of taps in the filter of the equalizer.

Number of samples per symbol
> The number of input samples for each symbol.

Signal constellation
> A vector of complex numbers that specifies the constellation for the modulation.

CMA Equalizer

Step size
> The step size of the CMA.

Leakage factor
> The leakage factor of the CMA, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights
> A vector that lists the initial weights for the taps.

Output error
> If you check this box, the block outputs the error signal described in the table above.

Output weights
> If you check this box, the block outputs the current weights.

References

[1] Haykin, Simon, *Adaptive Filter Theory*, Third Ed., Upper Saddle River, N.J., Prentice-Hall, 1996.

[2] Johnson, Richard C. Jr., Philip Schniter, Thomas. J. Endres, et al., "Blind Equalization Using the Constant Modulus Criterion: A Review," *Proceedings of the IEEE*, vol. 86, pp. 1927-1950, October 1998.

See Also
LMS Linear Equalizer, LMS Decision Feedback Equalizer, RLS Linear Equalizer, RLS Decision Feedback Equalizer

Purpose   Output phase difference between two complex input signals

Library   Utility Blocks
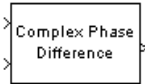
Description   The Complex Phase Difference block accepts two complex input signals
that have the same size and frame status. The output is the phase
difference from the second to the first, measured in radians. The
elements of the output are between -π and π.

The input signals can have any size or frame status. This block
processes each pair of elements independently.

**Dialog
Box**
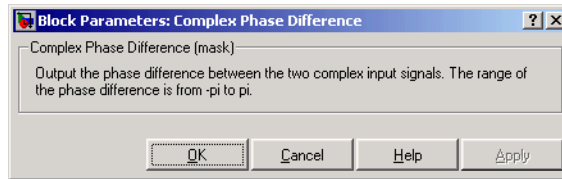
See Also   Complex Phase Shift

Complex Phase Shift

Purpose   Shift phase of complex input signal by second input value

Library   Utility Blocks

Description   The Complex Phase Shift block accepts a complex signal at the port labeled In. The output is the result of shifting this signal's phase by an amount specified by the real signal at the input port labeled Ph. The Ph input is measured in radians, and must have the same size and frame status as the In input.

The input signals can have any size or frame status. This block processes each pair of corresponding elements independently.

Dialog Box
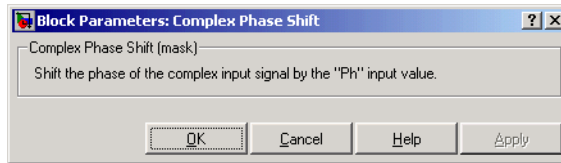
See Also   Complex Phase Difference

Purpose　　　　Implement voltage-controlled oscillator

Library　　　　Components sublibrary of Synchronization

Description　　The Continuous-Time VCO (voltage-controlled oscillator) block
generates a signal whose frequency shift from the **Quiescent
frequency** parameter is proportional to the input signal. The input
signal is interpreted as a voltage. If the input signal is $u(t)$, then the
output signal is
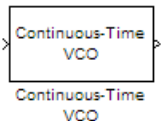
$$y(t) = A_c \cos\left(2\pi f_c t + 2\pi k_c \int_0^t u(\tau)d\tau + \varphi \right)$$

where A_c is the **Output amplitude** parameter, f_c is the **Quiescent
frequency** parameter, k_c is the **Input sensitivity** parameter, and φ
is the **Initial phase** parameter.

This block uses a continuous-time integrator to interpret the equation
above.

The input and output signals are both sample-based scalars.

**Dialog
Box**
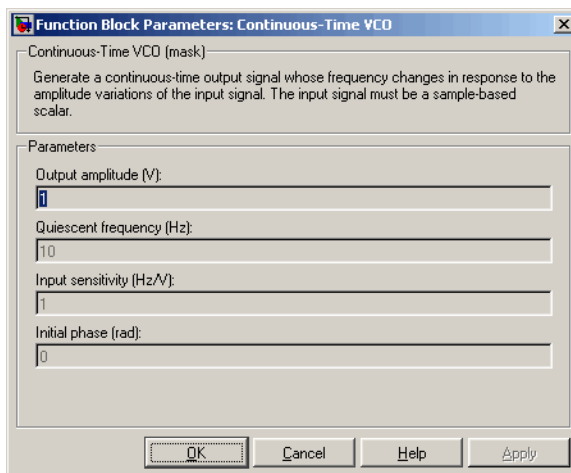
Continuous-Time VCO

Output amplitude

    The amplitude of the output.

Quiescent frequency

    The frequency of the oscillator output when the input signal is zero.

Input sensitivity

    This value scales the input voltage and, consequently, the shift from the **Quiescent frequency** value. The units of **Input sensitivity** are Hertz per volt.

Initial phase

    The initial phase of the oscillator in radians.

See Also   Discrete-Time VCO

Purpose   Restore ordering of symbols that were permuted using shift registers

Library   Convolutional sublibrary of Interleaving
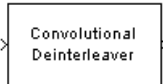
Description   The Convolutional Deinterleaver block recovers a signal that was interleaved using theConvolutional Interleaver block. The parameters in the two blocks should have the same values.

Convolutional
Deinterleaver

The input can be either a scalar or a frame-based column vector. It can be real or complex. The sample times of the input and output signals are the same.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box

Block Parameters: Convolutional Deinterleaver

Convolutional Deinterleaver (mask)

A convolutional deinterleaver consists of N shift registers. The ith register has delay (N-i)*B where B is a specified register length step. With each new input symbol, a commutator switches to a new register and the new symbol is shifted in while the oldest symbol in that register is shifted out. When the commutator reaches the Nth register, upon the next new input, it returns to the first register.

Parameters

Rows of shift registers:

6

Register length step:

2

Initial conditions:

0

OK   Cancel   Help   Apply
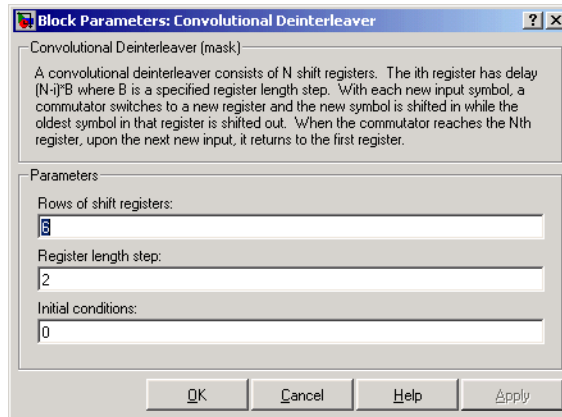
Rows of shift registers
    The number of shift registers that the block uses internally.

Register length step
    The difference in symbol capacity of each successive shift register, where the last register holds zero symbols.

Convolutional Deinterleaver

Initial conditions
The values that fill each shift register when the simulation begins.

Examples   For an example that uses this block, see "Example: Convolutional Interleavers".

Pair Block   Convolutional Interleaver

See Also   General Multiplexed Deinterleaver, Helical Deinterleaver

References   [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.

[2] Forney, G., D., Jr. "Burst-Correcting Codes for the Classic Bursty Channel." *IEEE Transactions on Communications*, vol. COM-19, October 1971. 772-781.

[3] Ramsey, J. L. "Realization of Optimum Interleavers." *IEEE Transactions on Information Theory*, IT-16 (3), May 1970. 338-345.

Purpose   Create convolutional code from binary data

Library   Convolutional sublibrary of Channel Coding
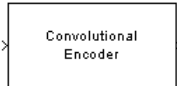
Description   The Convolutional Encoder block encodes a sequence of binary input vectors to produce a sequence of binary output vectors. This block can process multiple symbols at a time.

Input and Output Sizes

If the encoder takes k input bit streams (that is, can receive 2^k possible input symbols), this block's input vector length is L*k for some positive integer L. Similarly, if the encoder produces n output bit streams (that is, can produce 2^n possible output symbols), this block's output vector length is L*n.

The input can be a sample-based vector with L = 1, or a frame-based column vector with any positive integer for L.

For both its inputs and outputs for the data ports, the block supports `double`, `single`, `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `ufix1`. The port data types are inherited from the signals that drive the block. The input reset port supports `double` and `boolean` typed signals.

Note This block works for cases $k \geq 1$, where it has the same values for constraint lengths in each input stream (e.g., constraint lengths of [2 2] or [7 7] will work, but [5 4] will not).

Specifying the Encoder

To define the convolutional encoder, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in "Trellis Description of a Convolutional Encoder" in the Communications Toolbox documentation. You can use this parameter field in two ways:

Convolutional Encoder

- If you have a variable in the MATLAB workspace that contains the trellis structure, enter its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink to spend less time updating the diagram at the beginning of each simulation, compared to the usage described next.

- If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, use a `poly2trellis` command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7,[171 133],171)
```

The encoder registers begin in the all-zeros state. You can configure the encoder so that it resets its registers to the all-zeros state during the course of the simulation. To do this, set the `Operation mode` to **Reset on nonzero input via port**. The block then opens a second input port, labeled `Rst`. The signal at the `Rst` port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.

**Dialog
Box**
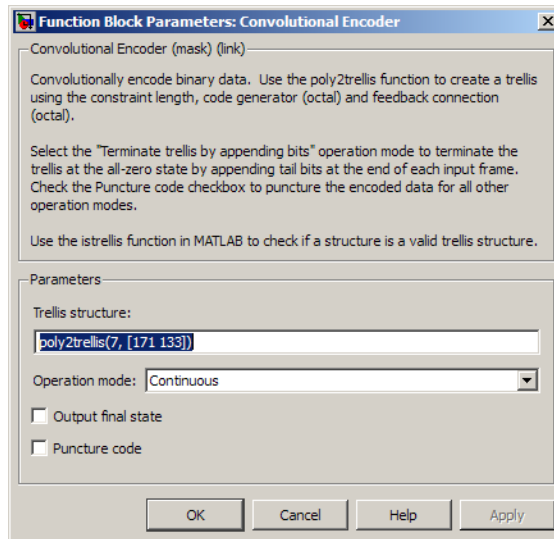
Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

Operation mode

In `Continuous` mode, the block retains the encoder states at the end of each frame, for use with the next frame.

In `Truncated (reset every frame)` mode, the block treats each frame independently. I.e., the encoder states are reset to all-zeros state at the start of each frame.

In `Terminate trellis by appending bits` mode, the block treats each frame independently. For each input frame, extra bits are used to set the encoder states to all-zeros state at the end of the frame. The output length is given by

$y = n \cdot (x + s) / k$, where x is the number of input bits, and

$s = $ constraint length $- 1$ (or, in the case of multiple constraint

lengths, s =sum(ConstraintLength(i)-1)). The block supports this mode for frame-based inputs only.

In Reset on nonzero input via port mode, the block has an additional input port, labeled Rst. When the Rst input is nonzero, the encoder resets to the all-zeros state.

Output final state
When you select **Output final state**, the second output port signal specifies the output state for the block. The output signal is a scalar, integer value. You can select **Output final state** for all operation modes except Terminate trellis by appending bits .

Specify initial state via input port
When you select **Specify initial state via input port** the second input port signal specifies the starting state for every frame in the block. The input signal must be a scalar, integer value. **Specify initial state via input port** appears only in Truncated operation mode.

Puncture code
Selecting this option opens the field **Puncture vector**.

Puncture vector
Vector used to puncture the encoded data. The puncture vector is a pattern of 1s and 0s where the 0s indicate the punctured bits. This field appears when you select **Punctured code**.

Puncture Pattern Examples   For some commonly used puncture patterns for specific rates and polynomials, see the last three references listed on this page.

See Also   Viterbi Decoder, APP Decoder

References   [1] Clark, George C. Jr. and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

[2] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein, *Data Communications Principles*, New York, Plenum, 1992.

[3] Yasuda, Y., et. al., "High rate punctured convolutional codes for soft decision Viterbi decoding," *IEEE Transactions on Communications*, Vol. COM-32, No. 3, pp 315–319, March 1984.

[4] Haccoun, D., and Begin, G., "High-rate punctured convolutional codes for Viterbi and Sequential decoding," *IEEE Transactions on Communications*, Vol. 37, No. 11, pp 1113–1125, Nov. 1989.

[5] Begin, G., et.al., "Further results on high-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Transactions on Communications*, Vol. 38, No. 11, pp 1922–1928, Nov. 1990.

Convolutional Interleaver

Purpose   Permute input symbols using set of shift registers

Library   Convolutional sublibrary of Interleaving
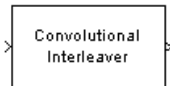
Description The Convolutional Interleaver block permutes the symbols in the
input signal. Internally, it uses a set of shift registers. The delay
value of the kth shift register is (k-1) times the **Register length step**
parameter. The number of shift registers is the value of the **Rows of
shift registers** parameter.

The **Initial conditions** parameter indicates the values that fill each
shift register at the beginning of the simulation (except for the first shift
register, which has zero delay). If **Initial conditions** is a scalar, then
its value fills all shift registers except the first; if **Initial conditions**
is a column vector whose length is the **Rows of shift registers**
parameter, then each entry fills the corresponding shift register. The
value of the first element of the **Initial conditions** parameter is
unimportant, since the first shift register has zero delay.

The input can be either a scalar or a frame-based column vector. It can
be real or complex. The sample times of the input and output signals
are the same.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`,
`uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this
output will be the same as that of the input signal.

**Dialog
Box**
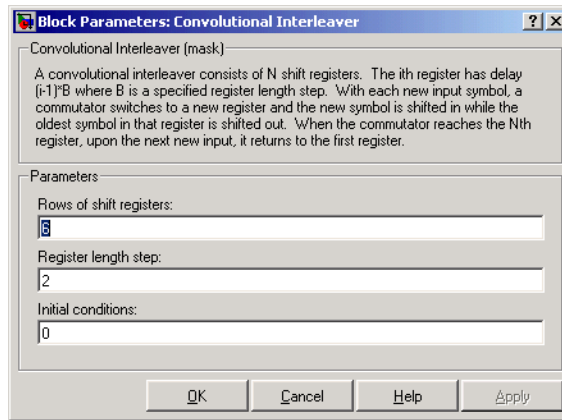
Rows of shift registers
    The number of shift registers that the block uses internally.

Register length step
    The number of additional symbols that fit in each successive shift register, where the first register holds zero symbols.

Initial conditions
    The values that fill each shift register when the simulation begins.

Examples   For an example that uses this block, see "Example: Convolutional Interleavers".

Pair Block   Convolutional Deinterleaver

See Also   General Multiplexed Interleaver, Helical Interleaver

References   [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.

[2] Forney, G., D., Jr. "Burst-Correcting Codes for the Classic Bursty Channel." *IEEE Transactions on Communications*, vol. COM-19, October 1971. 772-781.

[3] Ramsey, J. L. "Realization of Optimum Interleavers." *IEEE Transactions on Information Theory*, IT-16 (3), May 1970. 338-345.

Purpose   Demodulate CPFSK-modulated data

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The CPFSK Demodulator Baseband block demodulates a signal that
was modulated using the continuous phase frequency shift keying
method. The input is a baseband representation of the modulated
signal. The **M-ary number** parameter, M, is the size of the input
alphabet. M must have the form 2^K for some positive integer K.

The **Modulation index** parameter times π radians is the phase shift
in the modulated signal due to the latest symbol, when that symbol is
the integer 1. The **Phase offset** parameter is the initial phase of the
modulated waveform.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation
scheme and uses the Viterbi algorithm. The **Traceback length**
parameter, D, in this block is the number of trellis branches used
to construct each traceback path. D influences the output delay,
which is the number of zero symbols that precede the first meaningful
demodulated value in the output.

- If the input signal is sample-based, then the delay consists of D+1
  zero symbols.

- If the input signal is frame-based, then the delay consists of D zero
  symbols.

The optimal **Traceback length** parameter value choice is dependent
on minimum squared Euclidean distance calculations. Alternatively, a
typical value can be chosen using the "five-times-the-constraint-length"
rule which depends on the number of states given by:

$$numStates = \begin{cases} p * m^{(L-1)} < 0 \\ 2p * m^{(L-1)} > 0 \end{cases}$$

CPFSK Demodulator Baseband

- h = m/p is the modulation index proper rational form

  - m = numerator of modulation index

  - p = denominator of modulation index

- L is the Pulse length

- M is M-ary number

Outputs and Symbol Sets

If the **Output type** parameter is set to `Integer`, then the block produces odd integers between -(M-1) and M-1.

If the **Output type** parameter is set to `Bit`, then the block produces groupings of K bits. Each grouping is called a binary *word*.

In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer (k+M-1)/2. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether the **Symbol set ordering** parameter is set to `Binary` or `Gray`. For more information about Gray and binary coding, see "Binary-Valued and Integer-Valued Signals" in *Communications Blockset User's Guide*.

The input can be either a scalar or a frame-based column vector and must be of type `single` or `double`.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

**Dialog
Box**
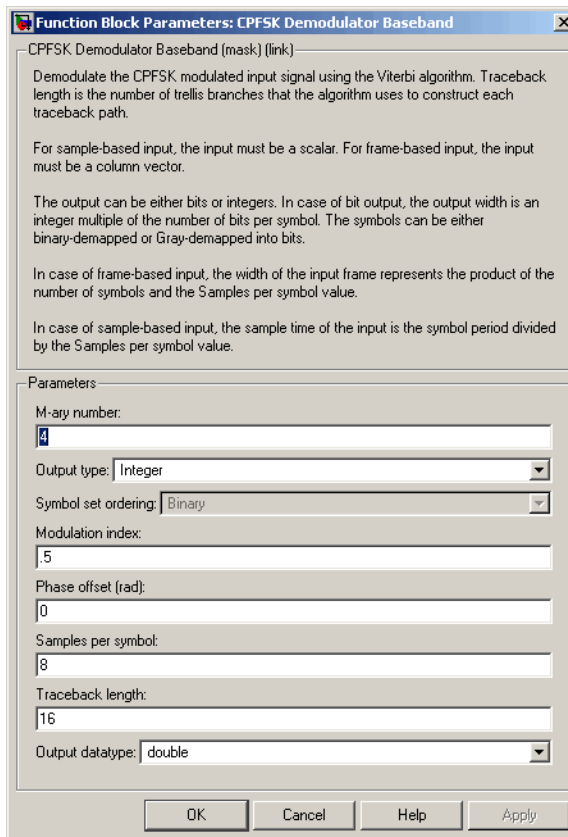
M-ary number

The size of the alphabet.

Output type

Determines whether the output consists of integers or groups
of bits.

Symbol set ordering

Determines how the block maps each integer to a group of output
bits. This field is active only when **Output type** is set to Bit.

CPFSK Demodulator Baseband

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the CPFSK Demodulator Baseband block uses to construct each traceback path.

Output datatype

The output data type can be `boolean`, `int8`, `int16`, `int32`, or `double`.

Pair Block	CPFSK Modulator Baseband
See Also	CPM Demodulator Baseband, Viterbi Decoder, M-FSK Demodulator Baseband
References	[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose   Modulate using continuous phase frequency shift keying method

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The CPFSK Modulator Baseband block modulates using the continuous phase frequency shift keying method. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the size of the input alphabet. M must have the form 2^K for some positive integer K.

The **Modulation index** parameter times π radians is the phase shift due to the latest symbol when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

For the exact definitions of the rectangular pulse shape that this block uses, see the work by Anderson, Aulin, and Sundberg among the references listed below.

Inputs and Symbol Sets

If the **Input type** parameter is set to Integer, then the block accepts odd integers between -(M-1) and M-1.

If the **Input type** parameter is set to Bit, then the block accepts groupings of K bits. Each grouping is called a binary *word*. The input vector length must be an integer multiple of K.

In binary input mode, the block maps each binary word to an integer between 0 and M-1, using a mapping that depends on whether the **Symbol set ordering** parameter is set to Binary or Gray. The block then maps the integer k to the intermediate value 2k-(M-1) and proceeds as in the integer input mode. For more information, see "Binary-Valued and Integer-Valued Signals" in *Communications Blockset User's Guide*.

The input can be either a scalar or a frame-based column vector. If **Input type** is Bit, then the input can also be a vector of length K.

CPFSK Modulator Baseband

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

Dialog Box

M-ary number
　　The size of the alphabet.

Input type
　　Indicates whether the input consists of integers or groups of bits.

Symbol set ordering
Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to Bit.

Modulation index
The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Phase offset (rad)
The initial phase of the output waveform.

Samples per symbol
The number of output samples that the block produces for each integer or binary word in the input.

Output datatype
The output data type can be single or double.

Pair Block   CPFSK Demodulator Baseband

See Also   CPM Modulator Baseband, M-FSK Modulator Baseband

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.
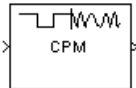
CPM Demodulator Baseband

Purpose   Demodulate CPM-modulated data

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The CPM Demodulator Baseband block demodulates a signal that was modulated using continuous phase modulation. The input is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the size of the input alphabet. M must have the form 2^K for some positive integer K.

The input can be either a scalar or a frame-based column vector and must be of type `single` or `double`.

The **Modulation index**, **Frequency pulse shape**, **Rolloff**, **BT product**, **Pulse length**, **Symbol prehistory**, and **Phase offset** parameters are as described on the reference page for theCPM Modulator Baseband block.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D, in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of D+1 zero symbols.

- If the input signal is frame-based, then the delay consists of D zero symbols.

The optimal **Traceback length** parameter value choice is dependent on minimum squared Euclidean distance calculations. Alternatively, a typical value can be chosen using the "five-times-the-constraint-length" rule which depends on the number of states given by:

$$numStates = \begin{cases} p*m^{(L-1)} < 0 \\ 2p*m^{(L-1)} > 0 \end{cases}$$

- h = m/p is the modulation index proper rational form

  - m = numerator of modulation index

  - p = denominator of modulation index

- L is the Pulse length

- M is M-ary number

For a binary raised cosine pulse shape with a pulse length of 3, h=2/3, this rule $(5*\log 2(3*2^2) = 18)$ gives close to the optimum value of 20.

Outputs and Symbol Sets

If the **Output type** parameter is set to Integer, then the block produces odd integers between -(M-1) and M-1.

If the **Output type** parameter is set to Bit, then the block produces groupings of K bits. Each grouping is called a binary *word*.

In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer (k+M-1)/2. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether the **Symbol set ordering** parameter is set to Binary or Gray. For more information about Gray and binary coding, see "Binary-Valued and Integer-Valued Signals" in *Communications Blockset User's Guide*.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

CPM Demodulator Baseband

Dialog Box

M-ary number

   The size of the alphabet.

Output type

   Determines whether the output consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to `Bit`.

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Frequency pulse shape

The type of pulse shaping that the corresponding modulator uses to smooth the phase transitions of the modulated signal.

Main lobe pulse duration (symbol intervals)

Number of symbol intervals of the largest lobe of the spectral raised cosine pulse. This field is active only when **Frequency pulse shape** is set to `Spectral Raised Cosine`.

Rolloff

The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to `Spectral Raised Cosine`.

BT product

The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to `Gaussian`.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

CPM Demodulator Baseband

Traceback length

The number of trellis branches that the CPM Demodulator block uses to construct each traceback path.

Output datatype

The output data type can be `boolean`, `int8`, `int16`, `int32`, or `double`.

Pair Block   CPM Modulator Baseband

See Also   CPFSK Demodulator Baseband, GMSK Demodulator Baseband, MSK Demodulator Baseband, Viterbi Decoder

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose   Modulate using continuous phase modulation

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The CPM Modulator Baseband block modulates using continuous phase modulation. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the size of the input alphabet. M must have the form 2^K for some positive integer K.

Continuous phase modulation uses pulse shaping to smooth the phase transitions of the modulated signal. Using the **Frequency pulse shape** parameter, you can choose these types of pulse shapes:

- Rectangular

- Raised Cosine

- Spectral Raised Cosine

  This option requires an additional parameter, **Rolloff**. The **Rolloff** parameter, which affects the spectrum of the pulse, is a scalar between zero and one.

- Gaussian

  This option requires an additional parameter, **BT product**. The **BT product** parameter, which represents bandwidth multiplied by time, is a nonnegative scalar. It is used to reduce the bandwidth at the expense of increased intersymbol interference.

- Tamed FM (tamed frequency modulation)

For the exact definitions of these pulse shapes, see the work by Anderson, Aulin, and Sundberg among the references listed below. Each pulse shape has a correponding pulse duration. The **Pulse length** parameter measures this quantity in symbol intervals.

The **Modulation index** parameter times π radians is the phase shift due to the latest symbol when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

The **Symbol prehistory** parameter is a scalar or vector that specifies the data symbols used before the start of the simulation, in reverse chronological order. If it is a vector, then its length must be one less than the **Pulse length** parameter.

Inputs and Symbol Sets

If the **Input type** parameter is set to `Integer`, then the block accepts odd integers between -(M-1) and M-1.

If the **Input type** parameter is set to `Bit`, then the block accepts groupings of K bits. Each grouping is called a binary *word*. The input vector length must be an integer multiple of K.

In binary input mode, the block maps each binary word to an integer between 0 and M-1, using a mapping that depends on whether the **Symbol set ordering** parameter is set to `Binary` or `Gray`. The block then maps the integer k to the intermediate value 2k-(M-1) and proceeds as in the integer input mode. For more information, see "Binary-Valued and Integer-Valued Signals" in *Communications Blockset User's Guide*.

The input can be either a scalar or a frame-based column vector. If **Input type** is `Bit`, then the input can also be a vector of length K.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications BlocksetUser's Guide*.

Dialog Box

M-ary number

The size of the alphabet.

Input type

Indicates whether the input consists of integers or groups of bits.

Symbol set ordering
> Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to `Bit`.

Modulation index
> The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Frequency pulse shape
> The type of pulse shaping that the block uses to smooth the phase transitions of the modulated signal.

Main lobe pulse duration (symbol intervals)
> Number of symbol intervals of the largest lobe of the spectral raised cosine pulse. This field is active only when **Frequency pulse shape** is set to `Spectral Raised Cosine`.

Rolloff
> The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to `Spectral Raised Cosine`.

BT product
> The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to `Gaussian`.

Pulse length (symbol intervals)
> The length of the frequency pulse shape.

Symbol prehistory
> The data symbols used before the start of the simulation, in reverse chronological order.

Phase offset (rad)
> The initial phase of the output waveform.

Samples per symbol
> The number of output samples that the block produces for each integer or binary word in the input.

Output data type
>   This block supports `double` and `single` data types.

Pair Block   CPM Demodulator Baseband

See Also   CPFSK Modulator Baseband, GMSK Modulator Baseband, MSK Modulator Baseband

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

CPM Phase Recovery

Purpose　　　　Recover carrier phase using 2P-Power method

Library　　　　Carrier Phase Recovery sublibrary of Synchronization

Description

```
       CPM      Sig
  Phase Recovery Ph
```

The CPM Phase Recovery block recovers the carrier phase of the input signal using the 2P-Power method. This feedforward, non-data-aided, clock-aided method is suitable for systems that use these types of baseband modulation: continuous phase modulation (CPM), minimum shift keying (MSK), continuous phase frequency shift keying (CPFSK), and Gaussian minimum shift keying (GMSK). This block is suitable for use with blocks in the Baseband Continuous Phase Modulation library.

If you express the modulation index for CPM as a proper fraction, h = K / P, then P is the number to which the name "2P-Power" refers.

The 2P-Power method assumes that the carrier phase is constant over a series of consecutive symbols, and returns an estimate of the carrier phase for the series. The **Observation interval** parameter is the number of symbols for which the carrier phase is assumed constant. This number must be an integer multiple of the input signal's vector length.

Input and Outputs

The input signal must be a frame-based column vector or a sample-based scalar of type `double` or `single`. The input signal represents a baseband signal at the symbol rate, so it must be complex-valued and must contain one sample per symbol.

The outputs are as follows:

- The output port labeled `Sig` gives the result of rotating the input signal counterclockwise, where the amount of rotation equals the carrier phase estimate. The `Sig` output is thus a corrected version of the input signal, and has the same sample time and vector size as the input signal.

- The output port labeled `Ph` outputs the carrier phase estimate, in degrees, for all symbols in the observation interval. The `Ph` output is a scalar signal.

> **Note** Because the block internally computes the argument of a complex number, the carrier phase estimate has an inherent ambiguity. The carrier phase estimate is between -90/P and 90/P degrees and might differ from the actual carrier phase by an integer multiple of 180/P degrees.

Delays and Latency

The block's algorithm requires it to collect symbols during a period of length **Observation interval** before computing a single estimate of the carrier phase. Therefore, each estimate is delayed by **Observation interval** symbols and the corrected signal has a latency of **Observation interval** symbols, relative to the input signal.

Dialog Box
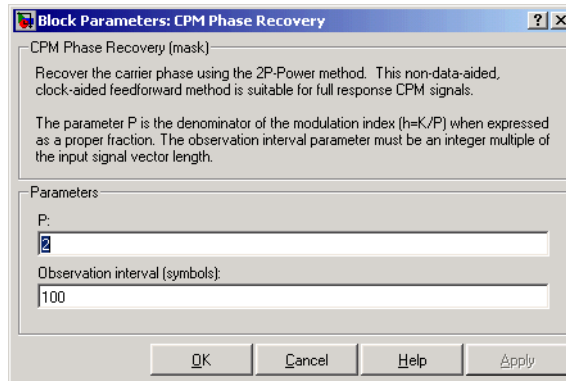
P

> The denominator of the modulation index for CPM (h = K / P) when expressed as a proper fraction.

Observation interval

> The number of symbols for which the carrier phase is assumed constant.

CPM Phase Recovery

Algorithm   If the symbols occurring during the observation interval are x(1), x(2), x(3),..., x(L), then the resulting carrier phase estimate is

$$\frac{1}{2P}\arg\left\{\sum_{k=1}^{L}(x(k))^{2P}\right\}$$

where the arg function returns values between -180 degrees and 180 degrees.

References   [1] Mengali, Umberto, and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

See Also   M-PSK Phase Recovery, CPM Modulator Baseband

Purpose   Generate CRC bits according to CRC method and append to input data frames

Library   CRC sublibrary of Error Detection and Correction
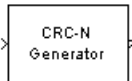
Description   The CRC-N Generator block generates cyclic redundancy code (CRC) bits for each input data frame and appends them to the frame. The CRC-N Generator block is a simplified version of the General CRC Generator block. With the CRC-N Generator block, you can select the generator polynomial for the CRC algorithm from a list of commonly used polynomials, given in the **CRC-N method** field in the block's dialog. N is degree of the generator polynomial. The table below lists the options for the generator polynomial.

CRC Method	Generator Polynomial	Number of Bits
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	32
CRC-24	$x^{24}+x^{23}+x^{14}+x^{12}+x^8+1$	24
CRC-16	$x^{16}+x^{15}+x^2+1$	16
Reversed CRC-16	$x^{16}+x^{14}+x+1$	16
CRC-8	$x^8+x^7+x^6+x^4+x^2+1$	8
CRC-4	$x^4+x^3+x^2+x+1$	4

You specify the initial state of the internal shift register using the **Initial states** parameter. You specify the number of checksums that the block calculates for each input frame using the **Checksums per frame** parameter. For more detailed information, see the reference page for the General CRC Generator block.

This block supports double and boolean data types. The output data type is inherited from the input.

CRC-N Generator

Signal Attributes

The General CRC Generator block has one input port and one output port. Both ports allow frame based binary column vectors only.
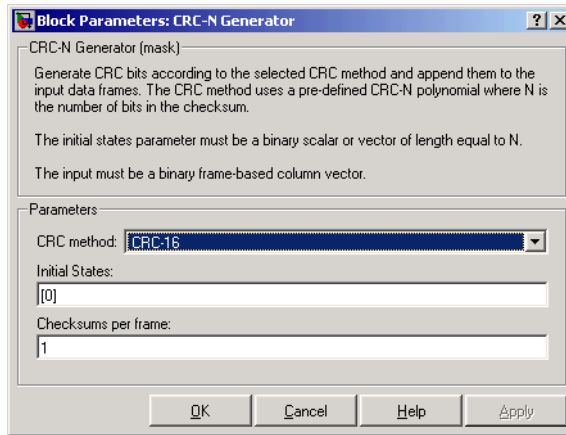
Dialog Box

CRC-N method
The generator polynomial for the CRC algorithm.

Initial states
A binary scalar or a binary row vector of length equal to the degree of the generator polynomial, specifying the initial state of the internal shift register.

Checksums per frame
A positive integer specifying the number of checksums the block calculates for each input frame.

Algorithm   For a description of the CRC algorithm as implemented by this block, see "Cyclic Redundancy Check Coding" in *Communications Blockset User's Guide*.

Schematic of the CRC Implementation

The above circuit divides the polynomial

$$a(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + a_0 \quad \text{by}$$

$$g(x) = g_{r-1}x^{r-1} + g_{r-2}x^{r-2} + \cdots + g_1 x + g_0 \text{, and returns the remainder}$$

$$d(x) = d_{r-1}x^{r-1} + d_{r-2}x^{r-2} + \cdots + d_1 x + d_0 \text{.}$$

The input symbols $\{a_{k-1}, a_{k-2}, \ldots, a_2, a_1, a_0\}$ are fed into the shift register one at a time in order of decreasing index. When the last symbol (a_0) works its way out of the register (achieved by augmenting the message with r zeros), the register contains the coefficients of the remainder polynomial $d(x)$.

This remainder polynomial is the checksum that is appended to the original message, which is then transmitted.

References

[1] Sklar, Bernard, *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J., Prentice-Hall, 1988.

[2] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

Pair Block   CRC-N Syndrome Detector

See Also   General CRC Generator, General CRC Syndrome Detector

CRC-N Syndrome Detector

Purpose   Detect errors in input data frames according to selected CRC method

Library   CRC sublibrary of Error Detection and Correction
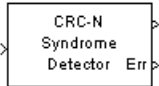
Description   The CRC-N Syndrome Detector block computes checksums for its
entire input frame. The block's second output is a vector whose size is
the number of checksums, and whose entries are 0 if the checksum
computation yields a zero value, and 1 otherwise. The block's first
output is the set of message words with the checksums removed.

The CRC-N Syndrome Detector block is a simplified version of the
General CRC Syndrome Detector block. You can select the generator
polynomial for the CRC algorithm from a list of commonly used
polynomials, given in the **CRC-N method** field in the block's dialog. N
is the degree of the generator polynomial. The reference page for the
CRC-N Generator block contains a list of the options for the generator
polynomial.

The parameter settings for the CRC-N Syndrome Detector block should
match those of the CRC-N Generator block.

You specify the initial state of the internal shift register by the **Initial
states** parameter. You specify the number of checksums that the
block calculates for each input frame by the **Checksums per frame**
parameter. For more detailed information, see the reference page for
the General CRC Syndrome Detector block.

This block supports `double` and `boolean` data types. The output data
type is inherited from the input.

Dialog Box
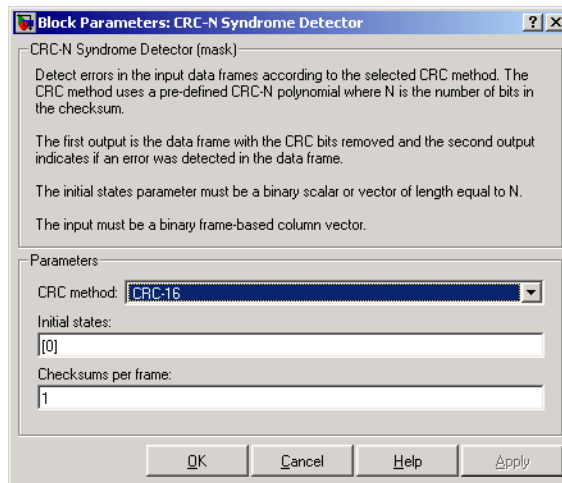
CRC-N method
> The generator polynomial for the CRC algorithm.

Initial states
> A binary scalar or a binary row vector of length equal to the degree of the generator polynomial, specifying the initial state of the internal shift register.

Checksums per frame
> A positive integer specifying the number of checksums the block calculates for each input frame.

Algorithm   For a description of the CRC algorithm as implemented by this block, see "Cyclic Redundancy Check Coding" in *Communications Blockset User's Guide*.

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications.* Englewood Cliffs, N.J., Prentice-Hall, 1988.

CRC-N Syndrome Detector

[2] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

Pair Block　　CRC-N Generator

See Also　　General CRC Generator, General CRC Syndrome Detector

Purpose   Map integer symbols from one coding scheme to another

Library   Utility Blocks
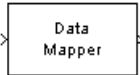
Description   The Data Mapper block accepts integer inputs and produces integer
outputs. You can select one of four mapping modes: `Binary to Gray`,
`Gray to Binary`, `User Defined`, or `Straight Through`.

The input can be a scalar, a sample-based vector, or a frame-based
signal. The block can accept multichannel inputs and allows for input
and output data types of `double`, `single`, `int32`, `int16`, `int8`, `uint32`,
`uint16`, and `uint8`. If the input is `double` or `single`, then it must be
non-negative in value. Note that although the block will provide outputs
for non-integer valued inputs, the results will likely be meaningless.

Gray coding is an ordering of binary numbers such that all adjacent
numbers differ by only one bit. However, the inputs and outputs of this
block are integers, not binary vectors. As a result, the first two mapping
modes perform code conversions as follows:

- In the `Binary to Gray` mode, the output from this block is the
  integer equivalent of the Gray code bit representation for the input
  integer.

- In the `Gray to Binary` mode, the output from this block is the
  integer position of the binary equivalent of the input integer in a
  Gray code ordering.

As an example, the table below shows both the `Binary to Gray` and
`Gray to Binary` mappings for integers in the range 0 to 7. In the
Binary to Gray Mode Output column, notice that binary representations
in successive rows differ by exactly one bit. In the Gray to Binary Mode
columns, notice that sorting the rows by Output value creates a Gray
code ordering of Input binary representations.

Data Mapper

Binary to Gray Mode		Gray to Binary Mode	
Input	**Output**	**Input**	**Output**
0	0 *(000)*	0 *(000)*	0
1	1 *(001)*	1 *(001)*	1
2	3 *(011)*	2 *(010)*	3
3	2 *(010)*	3 *(011)*	2
4	6 *(110)*	4 *(100)*	7
5	7 *(111)*	5 *(101)*	6
6	5 *(101)*	6 *(110)*	4
7	4 *(100)*	7 *(111)*	5

When you select the User Defined mode, you can use any arbitrary mapping by providing a vector to specify the output ordering. For example, the vector [1,5,0,4,2,3] defines the following mapping:

$$0 \rightarrow 1$$
$$1 \rightarrow 5$$
$$2 \rightarrow 0$$
$$3 \rightarrow 4$$
$$4 \rightarrow 2$$
$$5 \rightarrow 3$$

When you select the Straight Through mode, the output equals the input.

**Dialog
Box**
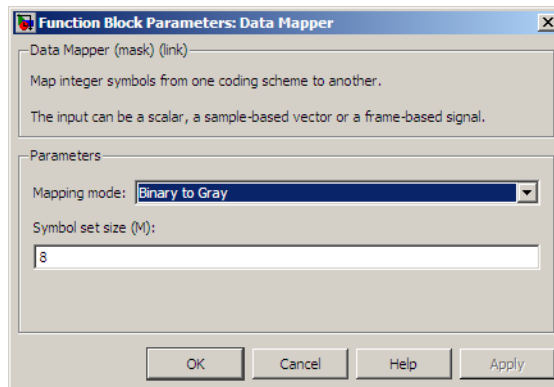
Mapping mode
> The type of data mapping that the block performs.

Symbol set size
> Symbol set size of M restricts this block's inputs and outputs to
> integers in the range 0 to M-1.

Mapping vector
> A vector of length M that contains the integers from 0 to M-1.
> The order of the elements of this vector specifies the mapping of
> inputs to outputs. This field appears if **Mapping mode** is set
> to User Defined.

DBPSK Demodulator Baseband

Purpose   Demodulate DBPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The DBPSK Demodulator Baseband block demodulates a signal that
was modulated using the differential binary phase shift keying method.
The input is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The block compares
the current symbol to the previous symbol. It maps phase differences of
θ and π+θ, respectively, to outputs of 0 and 1, respectively, where θ is
the **Phase rotation** parameter. The first element of the block's output
is the initial condition of zero because there is no previous symbol with
which to compare the first symbol.

The input can be either a scalar or a frame-based column vector. The
block accepts input of data types `single` and `double`.

Dialog Box
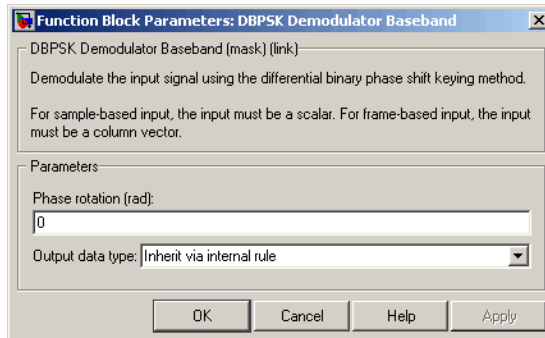
Phase rotation (rad)
    This phase difference between the current and previous modulated
    symbols results in an output of zero.

Output data type
    When the parameter is set to `'Inherit via internal rule'`
    (default setting), the block will inherit the output data type from

the input port. The output data type will be the same as the input data type if the input is of type `single` or `double`.

For both integer and bit inputs, this block can output the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`.

Pair Block   DBPSK Modulator Baseband

See Also   M-DPSK Demodulator Baseband, DQPSK Demodulator Baseband, BPSK Demodulator Baseband

DBPSK Modulator Baseband

Purpose   Modulate using differential binary phase shift keying method

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The DBPSK Modulator Baseband block modulates using the differential
binary phase shift keying method. The output is a baseband
representation of the modulated signal.

The input must be a discrete-time binary-valued signal. The input can
be either a scalar or a frame-based column vector. For both integer
and bit inputs, the block can accept the data types `int8`, `uint8`, `int16`,
`uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`. These rules
govern this modulation method when the **Phase rotation** parameter
is θ:

- If the first input bit is 0 or 1, respectively, then the first modulated
  symbol is exp(jθ) or -exp(jθ), respectively.

- If a successive input bit is 0 or 1, respectively, then the modulated
  symbol is the previous modulated symbol multiplied by exp(jθ) or
  -exp(jθ), respectively.

**Dialog
Box**
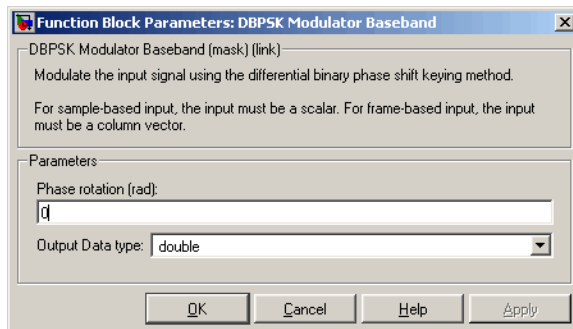
Phase rotation (rad)
The phase difference between the previous and current modulated
symbols when the input is zero.

Output Data type
> The output data type can be either single or double. By default, the block sets this to double.

Pair Block   DBPSK Demodulator Baseband

See Also   DQPSK Modulator Baseband, BPSK Modulator Baseband

Deinterlacer

Purpose Distribute elements of input vector alternately between two output vectors

Library Sequence Operations

Description The Deinterlacer block accepts an input vector that has an even number of elements. The block alternately places the elements in each of two output vectors. As a result, each output vector size is half the input vector size. The output vectors have the same complexity and sample time of the input.

The input can be either a sample-based vector of length two, or a frame-based column vector whose length is any even integer. The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data types of this output will be the same as that of the input signal.

This block can be useful for separating in-phase and quadrature information from a single vector into separate vectors.

Dialog Box
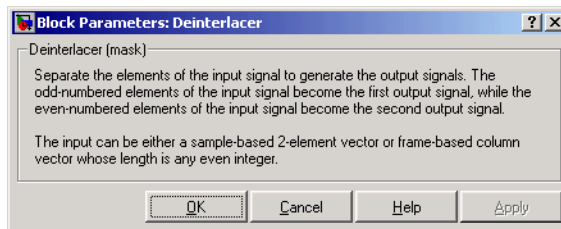
Examples If the input vector is frame-based with value [1; 5; 2; 6; 3; 7; 4; 8], then the two output vectors are [1; 2; 3; 4] and [5; 6; 7; 8]. Notice that this is the inverse of the example on the reference page for the Interlacer block.

If the input vector is frame-based with value [1; 2; 3; 4; 5; 6], then the two output vectors are [1; 3; 5] and [2; 4; 6].

Pair Block Interlacer

See Also   Demux (Simulink documentation)

Derepeat

Purpose   Reduce sampling rate by averaging consecutive samples

Library   Sequence Operations

Description

```
Derepeat
/5
```
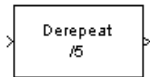
The Derepeat block resamples the discrete input at a rate 1/N times the input sample rate by averaging N consecutive samples. This is one possible inverse of the Signal Processing Blockset Repeat block. The positive integer N is the **Derepeat factor** parameter in the Derepeat dialog.

The **Initial condition** parameter prescribes elements of the output when it is still too early for the input data to show up in the output. If the dimensions of the **Initial condition** parameter match the output dimensions, then the parameter represents the initial output value. If **Initial condition** is a scalar, then it represents the initial value of each element in the output. The block does not support empty matrices for initial conditions.

The input can have any shape or frame status. The block can accept the data types single and double. The data type of the output will be the same as that of the input signal.

This block will work within a triggered subsystem, as long as it is used in the single-rate mode.

Sample-Based Operation

If the input is sample-based, and **Framing** is set to Maintain input size, then the block assumes that the input is a vector or matrix whose elements represent samples from independent channels. The block averages samples from each channel independently over time. The output period is N times the input period, and the input and output sizes are identical. The output is delayed by one output period, and the first output value is the **Initial condition** value. If **Framing** is set to Maintain input rate, the block will generate an error message.

Frame-Based Operation

If the input is frame-based, then the block derepeats each frame, treating distinct channels independently. Each element of the output

is the average of N consecutive elements along a *column* of the input matrix. The **Derepeat factor** must be less than the frame size.

The **Framing** parameter determines how the block adjusts the rate at the output to accommodate the reduced number of samples. The two options are:

- Maintain input frame size

  The block reduces the sampling rate by using a proportionally longer frame *period* at the output port than at the input port. For derepetition by a factor of N, the output frame period is N times the input frame period, but the input and output frame sizes are equal. The output is delayed by one output frame, and the first output frame is determined only by the **Initial condition** value.

  For example, if a single-channel input with a frame period of 1 second is derepeated by a factor of 4, then the output has a frame period of 4 seconds. The input and output frame sizes are equal.

- Maintain input frame rate

  The block reduces the sampling rate by using a proportionally smaller frame *size* than the input. **Derepeat factor** should be an integer factor of the number of rows in the input vector or matrix. For derepetition by a factor of N, the output frame size is 1/N times the input frame size, but the input and output frame rates are equal. When you use this option, the **Initial condition** parameter does not apply and the block incurs no delay, because the input data immediately shows up in the output.

  For example, if a single-channel input with 64 elements is derepeated by a factor of 4, then the output contains 16 elements. The input and output frame periods are equal.

Derepeat

**Dialog
Box**
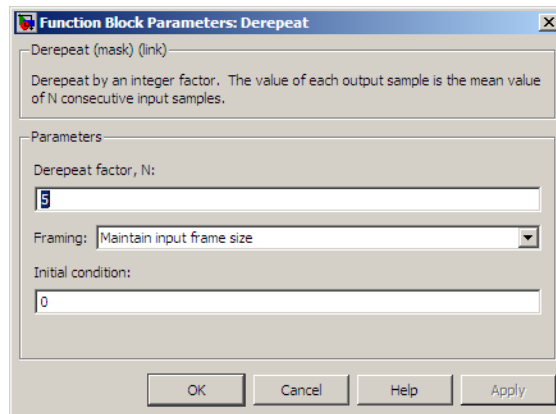
Derepeat factor, N

The number of consecutive input samples to average in order to produce each output sample.

Framing

For frame-based operation, the method by which to reduce the amount of data. One method decreases the frame rate while maintaining frame size, while the other decreases the frame size while maintaining frame rate.

Initial condition

The value with which to initialize the block.

See Also   Repeat (Signal Processing Blockset documentation), Downsample (Signal Processing Blockset documentation)

Purpose   Descramble input signal

Library   Sequence Operations
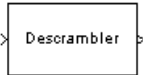
Description   The Descrambler block descrambles the input signal, which must be a
                   scalar or a frame-based column vector. The Descrambler block is the
                   inverse of theScrambler block. If you use the Scrambler block in the
                   transmitter, then you should use the Descrambler block in the receiver.

> Descrambler >

Below is a schematic of the descrambler. All adders and the one
subtracter operate modulo N, where N is the **Calculation base**
parameter. The input values must be integers between 0 and N-1.
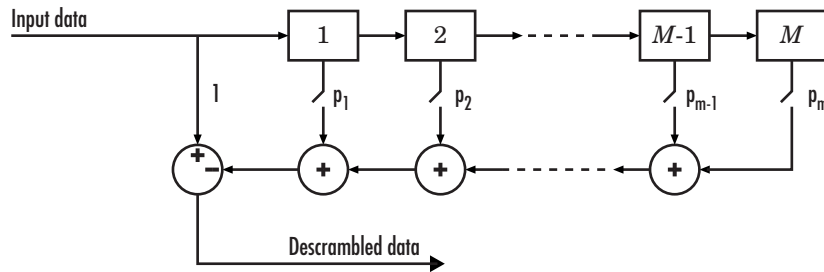
At each time step, the input causes the contents of the registers to shift
sequentially. Each switch in the descrambler is on or off as defined
by the **Scramble polynomial** parameter. To make the Descrambler
block reverse the operation of the Scrambler block, use the same
Scramble polynomial parameters in both blocks. If there is no signal
delay between the scrambler and the descrambler, then the **Initial
states** in the two blocks must be the same. See the reference page for
theScrambler block for more information about these parameters.

Descrambler
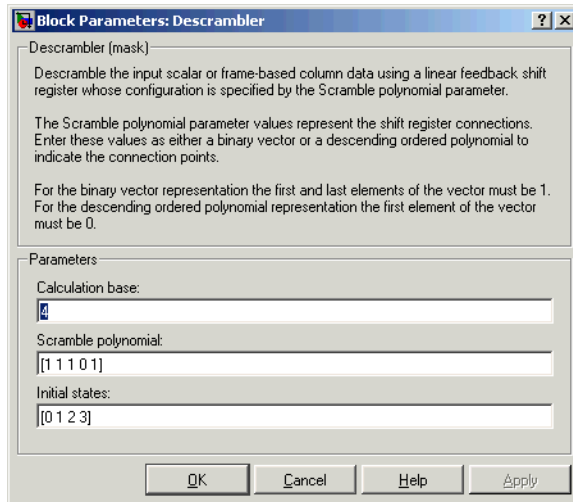
Dialog Box

Calculation base
The calculation base N. The input and output of this block are integers in the range [0, N-1].

Scramble polynomial
A polynomial that defines the connections in the scrambler.

Initial states
The states of the scrambler's registers when the simulation starts.

Pair Block   Scrambler

Purpose Decode binary signal using differential coding

Library Source Coding
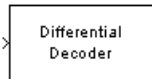
Description The Differential Decoder block decodes the binary input signal. The output is the logical difference between the present input and the previous input. More specifically, the block's input and output are related by

$m(t_0) = d(t_0)$ XOR **Initial condition** parameter value

$m(t_k) = d(t_k)$ XOR $d(t_{k-1})$

where

- d is the differentially encoded input.
- m is the output message.
- t_k is the kth time step.
- XOR is the logical exclusive-or operator.

The input can be either a scalar, a vector, or a frame-based matrix. This block processes each vector element independently.

Differential Decoder
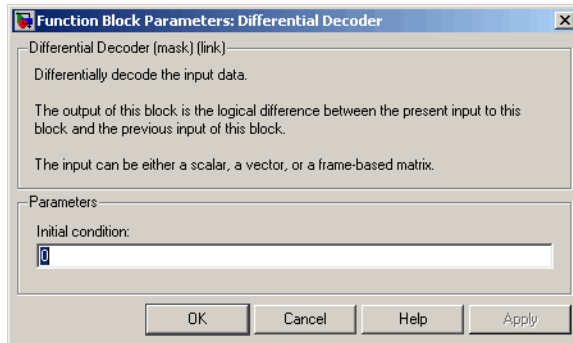
Dialog Box

Initial condition

The logical exclusive-or of this value with the initial input value forms the initial output value.

References   [1] Couch, Leon W., II, *Digital and Analog Communication Systems*, Sixth edition, Upper Saddle River, N. J., Prentice Hall, 2001.

Pair Block   Differential Encoder

Purpose   Encode binary signal using differential coding

Library   Source Coding
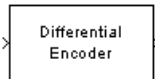
Description   The Differential Encoder block encodes the binary input signal. The output is the logical difference between the present input and the previous output. More specifically, the input and output are related by

> $d(t_0)$ = $m(t_0)$ XOR **Initial condition** parameter value

> $d(t_k)$ = $d(t_{k-1})$ XOR $m(t_k)$

where

- m is the input message
- d is the differentially encoded output.
- t_k is the kth time step.
- XOR is the logical exclusive-or operator.

The input can be either a scalar or a vector.

Differential Encoder
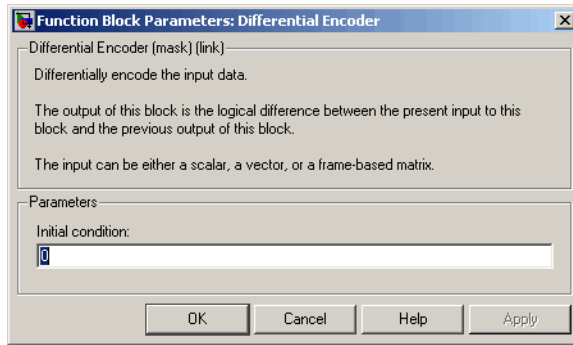
**Dialog
Box**

Initial condition

The logical exclusive-or of this value with the initial input value
forms the initial output value.

References [1] Couch, Leon W., II, *Digital and Analog Communication Systems*,
Sixth edition, Upper Saddle River, N. J., Prentice Hall, 2001.

Pair Block Differential Decoder

Purpose   Display multiple traces of modulated signal

Library   Comm Sinks

Description   The Discrete-Time Eye Diagram Scope block displays multiple traces of a modulated signal to produce an eye diagram. You can use the block to reveal the modulation characteristics of the signal, such as pulse shaping or channel distortions.
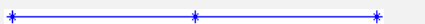
The Discrete-Time Eye Diagram Scope block has one input port. The block accepts signal of type double, single, boolean, base integer, and fixed-point data types for input, but will cast as double prior to display. In sample-based mode, the input signal must be a scalar value. In frame-based mode, the input signal must be a column vector or a scalar value.

Warning

Neither the Rapid Accelerator nor external mode allows for the uploading of data to this block, preventing any display on this scope. Run the model in accelerator or normal mode to take full advantage of this scope block.
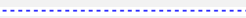
Marker and Line Styles
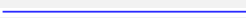
The **Marker**, **Line style**, and **Line color** parameters, on the **Rendering Properties** panel, control the appearance of the signal trajectory. The **Marker** parameter specifies the marker style for points in the eye diagram. The following table lists some of the available line markers.

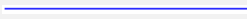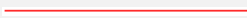| Marker Style	Parameter Symbol	Appearance
Plus	+	
Circle	o	
Asterisk	*	
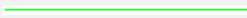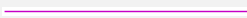
Marker Style	Parameter Symbol	Appearance
Point	.	
Cross	x	

The **Line style** parameter specifies the style for lines in the eye diagram. The following lists some of the available line styles.

Line Style	Appearance
Solid	
Dashed	
Dotted	
Dash-dot	

The **Line color** parameter specifies the color of the eye diagram. These settings plot the signal channels in the following colors (8-bit RGB equivalents are shown in the center column).

Color	RGB Equivalent	Appearance
Black	(0,0,0)	
Blue	(0,0,255)	
Red	(255,0,0)	
Green	(0,255,0)	
Dark purple	(192,0,192)	

See the line function in the MATLAB documentation for more information about the available markers, colors, and line styles.

Recommended Settings

The following table summarizes the recommended parameter settings for the Discrete-Time Eye Diagram Scope.

Parameter	Recommended Setting
Samples per symbol	Same as the **Samples per symbol** setting in the modulator block, or the **Interpolation factor** setting in the interpolation block
Offset (samples)	0 to view the open part of the eye (**Samples per symbol**)/2 to view the closed part of the eye
Symbols per trace	An integer between 1 and 4
Traces displayed	10 times the alphabet size of the modulator, M
New traces per display	Same as **Traces displayed** for greater speed A small positive integer for best animation
Marker	None or a point (.) to see where the samples are plotted
Line style	Solid dash (-)
Line color	Blue (b)
Duplicate points at trace boundary	Check **Duplicate points at trace boundary** for modulations such as PSK and QAM. Clear to display the phase trees for MSK, CPFSK, GFSK, GMSK, and other continuous phase modulations.

Parameter	Recommended Setting
Color fading	Check **Color fading** for animation that resembles an oscilloscope.
	Clear for greater speed and animation that resembles a plot.
High quality rendering	Check **High quality rendering** for better animation.
	Clear for greater speed.
Eye diagram to display	Select `In-phase and Quadrature` to view real and imaginary components.
	Select `In-phase Only` to view real component only and for greater speed.
	When the input is real and you choose `In-phase and Quadrature`, the quadrature component of the eye diagram is zero.
Open at start of simulation	Check **Open at start of simulation** to view the signal at the start of simulation.
	Clear to view the signal after convergence to steady state and for greater initial speed.

Parameter	Recommended Setting
Y-axis minimum	Approximately 10% less than the expected minimum value of the signal
Y-axis maximum	Approximately 10% greater than the expected maximum value of the signal

To use the scope in single-rate mode, the following conditions must be true:

$$sps * ((td * (spt-1)) + ntpd) = Sf$$

where:

- sps = Samples per symbol

- td = Traces displayed

- spt = Symbols per trace

- ntpd = New traces per display

- Sf = Input frame size, in samples

Scope Options

The scope title (in the window title bar) is the same as the block title. You can set the axis scaling by setting the y-axis minimum and y-axis maximum parameters on the **Axes Properties** panel.

In addition to the standard MATLAB figure window menus (**File**, **Edit**, **Window**, **Help**), the Vector Scope window has an **Axes** and a **Channels** menu.

The properties listed in the **Axes** menu apply to all channels. Many of the parameters in this menu are also accessible through the block parameter dialog box. These are **Autoscale**, **Show grid**, **Frame #**, and **Save Position**. Below are descriptions of the other parameters listed in the **Axes** menu:
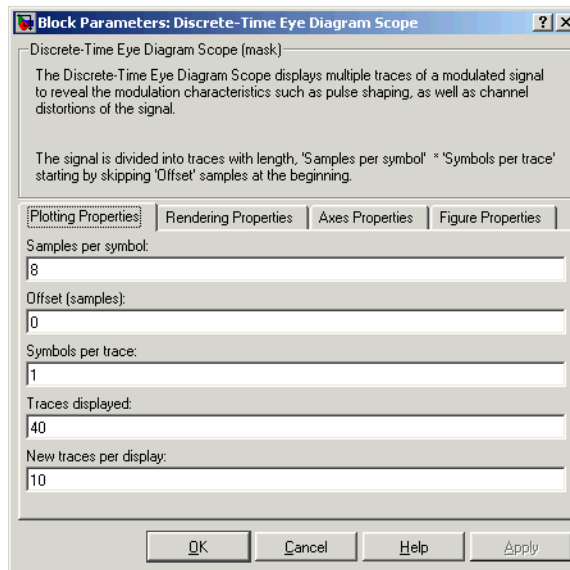
- **Autoscale** resizes the *y*-axis to best fit the vertical range of the data. The numerical limits selected by the autoscale feature are displayed in the **Minimum Y-limit** and **Maximum Y-limit** parameters in the parameter dialog box. You can change them by editing those values.

- **Show grid** - When selected, the scope displays a grid according to tick marks on the *x*- and *y*-axes.

- **Frame #** - When selected, the scope displays the current frame number at the bottom of the scope window.

- **Save Position** automatically updates the **Scope position** parameter in the **Figure properties** panel to reflect the scope window's current position and size. To make the scope window open at a particular location on the screen when the simulation runs, simply drag the window to the desired location, resize it as needed, and select **Save Position**.

The properties listed in the **Channels** menu apply to a particular channel. The parameters listed in this menu are **Style**, **Marker**, and **Color**. They correspond to the parameters **Line style**, **Marker**, and **Line color**, respectively.

You can also access many of these options by right-clicking with the mouse anywhere on the scope display. The menu that pops up contains a combination of the options available in both the **Axes** and **Channels** menus.

**Dialog
Box**

Block Parameters: Discrete-Time Eye Diagram Scope

Discrete-Time Eye Diagram Scope (mask)

The Discrete-Time Eye Diagram Scope displays multiple traces of a modulated signal to reveal the modulation characteristics such as pulse shaping, as well as channel distortions of the signal.

The signal is divided into traces with length, 'Samples per symbol' * 'Symbols per trace' starting by skipping 'Offset' samples at the beginning.

| Plotting Properties | Rendering Properties | Axes Properties | Figure Properties |

Samples per symbol:

8

Offset (samples):

0

Symbols per trace:

1

Traces displayed:

40

New traces per display:

10

OK   Cancel   Help   Apply

Samples per symbol
   Number of samples per symbol. Use with **Symbols per trace** to determine the number of samples per trace.

Offset (samples)
   Nonnegative integer less than the product of **Samples per symbol** and **Symbols per trace**, specifying the number of samples to omit before plotting the first point. Tunable.

Symbols per trace
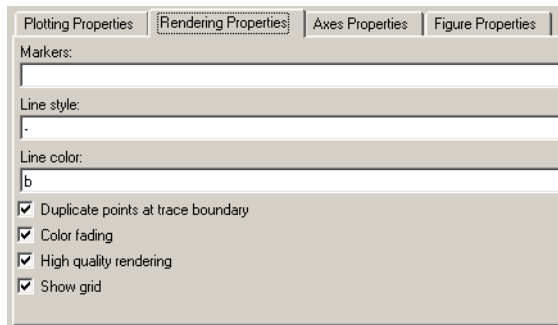   Positive integer specifying the number of symbols plotted per trace.

Traces displayed
   Number of traces plotted.

New traces per display
   Positive integer less than **Traces displayed**, specifying the number of new traces that appear in each display.

Discrete-Time Eye Diagram Scope

Markers

    The marker for points in the eye diagram. Tunable.

Line style

    The line style in the eye diagram. Tunable.

Line color

    The line color in the eye diagram. Tunable.

Duplicate points at trace boundary

    Check to enable duplicate points at the trace boundary. Clear to disable.

Color fading

    When selected, the points in the eye diagram fade as the interval of time after they are first plotted increases. Tunable.

High quality rendering

    When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid

    Toggles the scope grid on and off. Tunable.

Y-axis minimum

Minimum signal value the scope displays. Tunable.

Y-axis maximum

Maximum signal value the scope displays. Tunable.

In-phase Y-axis label

Label for *y*-axis of the in-phase diagram. Tunable.

Quadrature Y-axis label

Label for *y*-axis of the quadrature diagram. Tunable.

Open scope at start of simulation

When selected, the scope opens at the start of simulation. When cleared, you must double-click the block after the start of simulation to open the scope. Tunable.

Discrete-Time Eye Diagram Scope

Eye diagram to display

Type of eye diagram to display. Choose In-phase and Quadrature to display real and complex components, or In-phase Only to display only the real component. Tunable.

Trace number

Displays the number of the current trace in the input sequenced. Tunable.

Scope position

A four-element vector of the form [left bottom width height] specifying the position of the scope window. (0,0) is the lower left corner of the display. Tunable.

Title

Title of eye diagram figure window. Tunable.

Examples

For documentation examples that use this block, see "Example: Viewing a Sinusoid" and "Example: Viewing a Modulated Signal".

Also, the following Communications Blockset demos illustrate how to use the Discrete-Time Eye Diagram Scope block:

- CPM Phase Tree Example
- Filtered Offset QPSK vs. Filtered QPSK
- Rayleigh Fading Channel
- QPSK vs. MSK

See Also

Discrete-Time Scatter Plot Scope, Discrete-Time Signal Trajectory Scope

Purpose   Display in-phase and quadrature components of modulated signal constellation

Library   Comm Sinks

Description   The Discrete-Time Scatter Plot Scope block displays scatter plots of a modulated signal, to reveal the modulation characteristics, such as pulse shaping or channel distortions of the signal.

The Discrete-Time Scatter Plot Scope block has one input port. The input signal must be complex. The block accepts signal of type `double`, `single`, base integer, and fixed-point for input, but will cast it as `double`. In sample-based mode, the input signal must be a scalar value. In frame-based mode, the input signal must be a column vector or a scalar value.

See the reference page for the Discrete-Time Signal Trajectory Scope block to compare the preceding scatter plot with the trajectory of the same signal. The Discrete-Time Signal Trajectory Scope block connects the points displayed by the Discrete-Time Scatter Plot Scope block to display the signal trajectory.

Setting **Samples per symbol** to 8, increasing **Points displayed** to 100, and running the model for 100 seconds produces the following scatter plot.

Discrete-Time Scatter Plot Scope
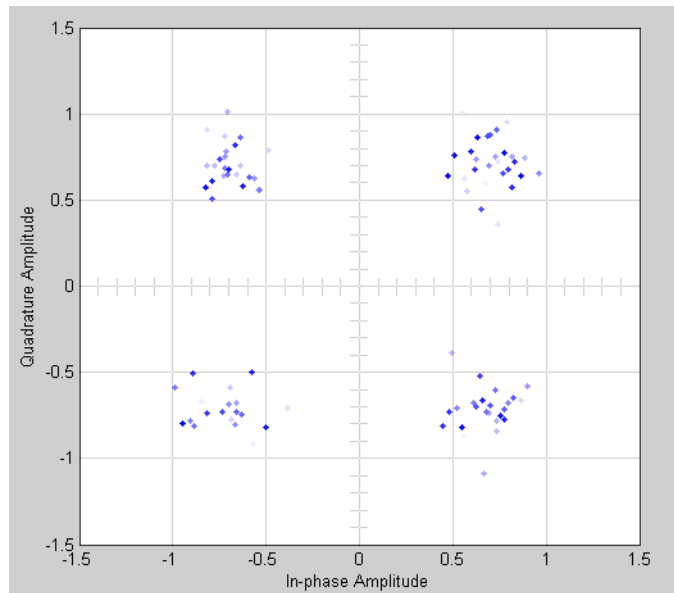
Warning

Neither the Rapid Accelerator nor external mode allows for the uploading of data to this block, preventing any display on this scope. Run the model in accelerator or normal mode to take full advantage of this scope block.

Markers and Color

The **Markers** and **Color** parameters, on the **Rendering Properties** panel, specify the style and color of markers in the scatter plot. For details on the options for these parameters, see the reference page for the Discrete-Time Eye Diagram Scope block.

Recommended Settings

The following table summarizes the recommended parameter settings for the Discrete-Time Scatter Plot Scope.

Parameter	Recommended Setting
Samples per symbol	Same as the **Samples per symbol** setting in the modulator block, or the **Interpolation factor** setting in the interpolation block
Points displayed	10 times the alphabet size of the modulator
New points per display	Same as **Points displayed** for greater speed A small positive integer for best animation
Line style	Solid dash (-)
Line color	Blue (b)
Color fading	Check **Color fading** for animation that resembles an oscilloscope. Clear for greater speed and animation that resembles a plot.
High quality rendering	Check **High quality rendering** for higher quality rendering. Clear for greater speed.
Open at start of simulation	Check **Open at start of simulation** to view the signal at the start of simulation. Clear to view the signal after convergence to steady state and for greater initial speed.

Discrete-Time Scatter Plot Scope

| Parameter	Recommended Setting
X-axis minimum	Approximately 10% less than the expected minimum value of the signal
X-axis maximum	Approximately 10% greater than the expected maximum value of the signal

To use the scope in single-rate mode, the following conditions must be true:

sps * nppd = Sf

where:

- sps = Samples per symbol

- nppd = New points per display

- Sf = Input frame size, in samples

**Dialog
Box**
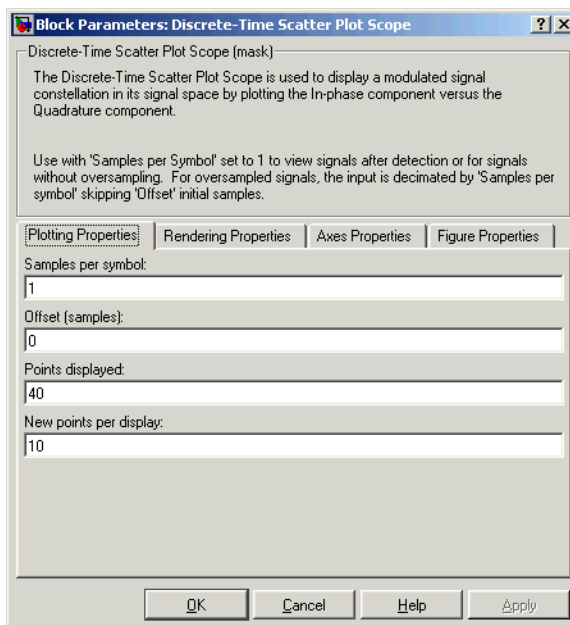
Samples per symbol
> Number of samples per symbol.

Offset (samples)
> Nonnegative integer less than the number of samples per symbol,
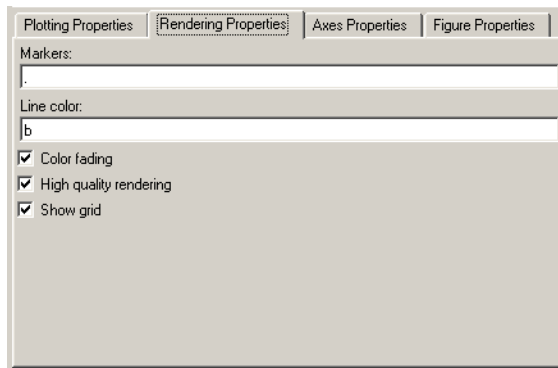> specifying the number of samples to skip before plotting points.

Points displayed
> Total number of points plotted.

New points per display
> Number of new points that appear in each display.

Discrete-Time Scatter Plot Scope

Markers

Line markers used in the scatter plot. Tunable.

Line color

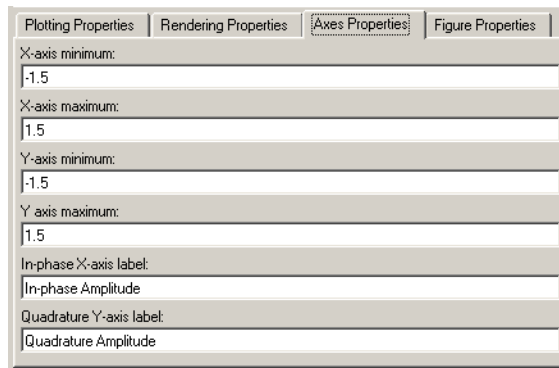The line color used in the scatter plot. Tunable.

Color fading

When selected, the points in the scatter plot fade as the interval of time after they are first plotted increases. Tunable.

High quality rendering

When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid

Toggles the scope grid on and off. Tunable.

X-axis minimum

Minimum value the scope displays on the *x*-axis. Tunable.

X-axis maximum

Maximum value the scope displays on the *x*-axis. Tunable.

Y-axis minimum

Minimum signal value the scope displays on the *y*-axis. Tunable.

Y-axis maximum

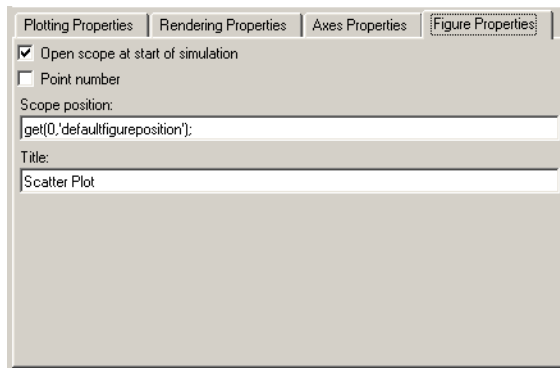Maximum signal value the scope displays on the *y*-axis. Tunable.

In-phase X-axis label

Label for *x*-axis. Tunable.

Quadrature Y-axis label

Label for *y*-axis. Tunable.

Discrete-Time Scatter Plot Scope

Open at start of simulation

When selected, the scope opens at the start of simulation. When cleared, you must double-click the block after the start of simulation to open the scope.

Point number

Displays the number of the current point in the input sequence. Tunable.

Scope position

A four-element vector of the form [left bottom width height] specifying the position of the scope window. (0,0) is the lower left corner of the display. Tunable.

Title

Title of scatter plot. Tunable.

Examples   For documentation examples that use this block, see "Example: Viewing a Sinusoid" and "Example: Viewing a Modulated Signal".

The following demos in Communications Blockset software illustrate how to use the Discrete-Time Scatter Plot Scope block:

- Digital Video Broadcasting Model — Terrestrial

- HiperLAN/2

- Phase Noise Effects in 256 QAM

- Multipath Rayleigh Fading Channel

See Also   Discrete-Time Eye Diagram Scope, Discrete-Time Signal Trajectory
Scope, Real-Imag to Complex

Discrete-Time Signal Trajectory Scope

Purpose   Plot modulated signal's in-phase component versus its quadrature component

Library   Comm Sinks

Description   The Discrete-Time Signal Trajectory Scope displays the trajectory of a modulated signal in its signal space by plotting its in-phase component versus its quadrature component.

The Discrete-Time Signal Trajectory Scope block has one input port. The input signal must be complex. The block accepts signal of type `double`, `single`, base integer, and fixed-point for input, but will cast it as `double`. In sample-based mode, the input signal must be a scalar value. In frame-based mode, the input signal must be a column vector or a scalar value.

Warning

Neither the Rapid Accelerator nor external mode allows for the uploading of data to this block, preventing any display on this scope. Run the model in accelerator or normal mode to take full advantage of this scope block.

Line Style and Color

The **Line style** and **Line color** parameters on the **Rendering Properties** panel control the appearance of the signal trajectory. The **Line style** parameter specifies the style for lines in the signal trajectory. For details on the options for these parameters, see the reference page for the Discrete-Time Eye Diagram Scope block.

Recommended Settings

The following table summarizes the recommended parameter settings for the Discrete-Time Signal Trajectory Scope.

Parameter	Recommended Setting
Samples per symbol	Same as the **Samples per symbol** setting in the modulator block, or the **Interpolation factor** used in the interpolation block
Symbols displayed	10 times the alphabet size of the modulator, M
New symbols per display	Same as **Symbols displayed** for greater speed A small positive integer for best animation
Line style	Solid dash (-)
Line color	Blue (b)
Color fading	Check **Color fading** for animation that resembles an oscilloscope. Clear for greater speed and animation that resembles a plot.
High quality rendering	Check **High quality rendering** for higher quality rendering. Clear for greater speed.
Open at start of simulation	Check **Open at start of simulation** to view the signal at the start of simulation. Clear to view the signal after convergence to steady state and for greater initial speed.

Parameter	Recommended Setting
Y-axis minimum	Approximately 10% less than the expected minimum value of the signal
Y-axis maximum	Approximately 10% greater than the expected maximum value of the signal

To use the scope in single-rate mode, the following conditions must be true:
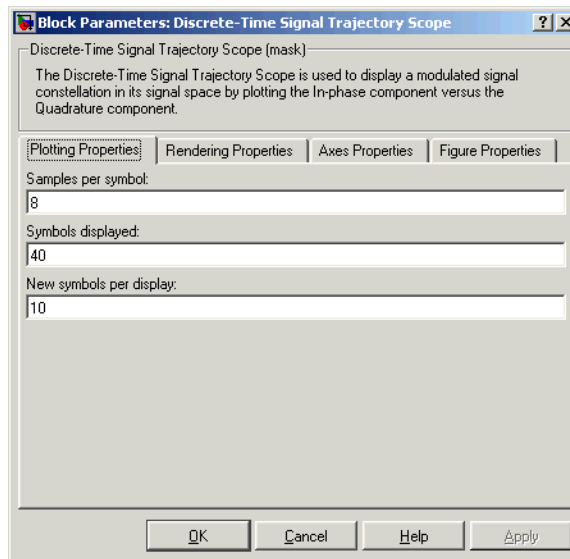
where:

$$sps * nspd = Sf$$

where:

- sps = Samples per symbol
- nspd = New symbols per display
- Sf = Input frame size, in samples

**Dialog
Box**

Samples per symbol
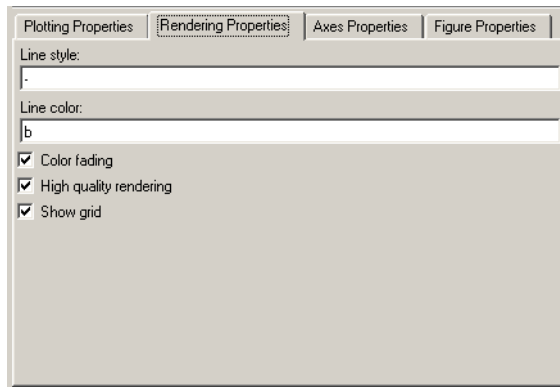>   Number of samples per symbol.

Symbols displayed
>   Total number of symbols plotted.

New symbols per display
>   Number of new symbols that appear in each display.

Discrete-Time Signal Trajectory Scope

Line markers
> The line markers used in the signal trajectory. Tunable.

Line color
> The line color used in the signal trajectory. Tunable.

Color fading
> When selected, the points in the signal trajectory fade as the interval of time after they are first plotted increases. Tunable.
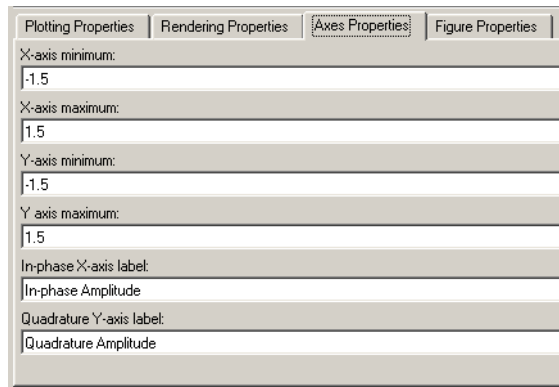
High quality rendering
> When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid
> Toggles the scope grid on and off. Tunable.

X-axis minimum

Minimum value the scope displays on the x-axis. Tunable.

X-axis maximum

Maximum value the scope displays on the *x*-axis. Tunable.

Y-axis minimum

Minimum signal value the scope displays on the *y*-axis. Tunable.

Y-axis maximum

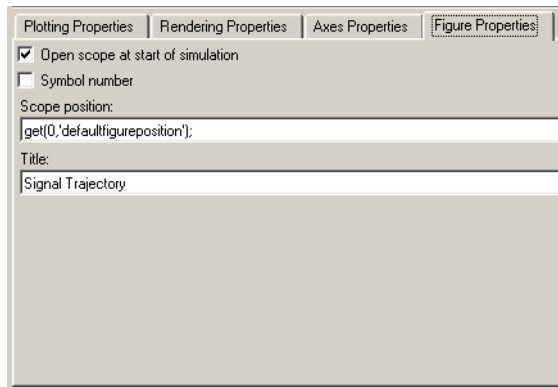Maximum signal value the scope display on the *y*-axis. Tunable.

In-phase X-axis label

Label for *x*-axis. Tunable.

Quadrature Y-axis label

Label for *y*-axis. Tunable.

Discrete-Time Signal Trajectory Scope

Open at start of simulation

When selected, the scope opens at the start of simulation.
When cleared, you must double-click the block after the start of
simulation to open the scope. Tunable

Symbol number

Displays the number of the current symbol in the input sequence.
Tunable.

Scope position

A four-element vector of the form [left bottom width height]
specifying the position of the scope window. (0,0) is the lower left
corner of the display. Tunable.

Title

Title of signal trajectory plot. Tunable.

Examples   For documentation examples that use this block, see "Example: Viewing
a Sinusoid" and "Example: Viewing a Modulated Signal".

Also, the following demos in Communications Blockset software
illustrate how to use the Discrete-Time Signal Trajectory Scope:

- Filtered Offset QPSK vs. Filtered QPSK

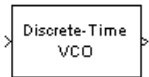- GMSK vs. MSK

See Also   Discrete-Time Eye Diagram Scope, Discrete-Time Scatter Plot Scope

Discrete-Time VCO

Purpose   Implement voltage-controlled oscillator in discrete time

Library   Components sublibrary of Synchronization

Description   The Discrete-Time VCO (voltage-controlled oscillator) block generates a
signal whose frequency shift from the **Quiescent frequency** parameter
is proportional to the input signal. The input signal is interpreted as a
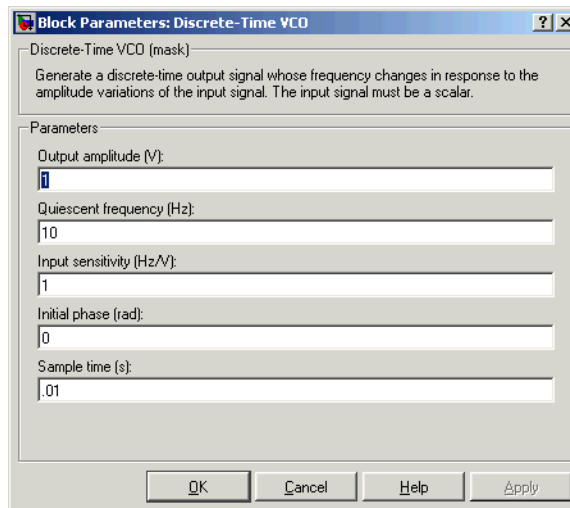voltage. If the input signal is $u(t)$, then the output signal is

$$y(t) = A_c \cos\left(2\pi f_c t + 2\pi k_c \int_0^t u(\tau)d\tau + \varphi \right)$$

where A_c is the **Output amplitude**, f_c is the **Quiescent frequency**, k_c
is the **Input sensitivity**, and φ is the **Initial phase**

This block uses a discrete-time integrator to interpret the equation
above.

The input and output signals can be scalars of data type `single` or
`double`. The data type of the output will be the same as that of the input
signal. The block supports double precision only for code generation.

Dialog Box

Output amplitude
> The amplitude of the output.

Quiescent frequency (Hz)
> The frequency of the oscillator output when the input signal is zero.

Input sensitivity
> This value scales the input voltage and, consequently, the shift from the **Quiescent frequency** value. The units of **Input sensitivity** are Hertz per volt.

Initial phase (rad)
> The initial phase of the oscillator in radians.

Sample time
> The calculation sample time.

See Also   Continuous-Time VCO

DQPSK Demodulator Baseband

Purpose   Demodulate DQPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The DQPSK Demodulator Baseband block demodulates a signal that
was modulated using the differential quaternary phase shift keying
method. The input is a baseband representation of the modulated
signal.

The input must be a discrete-time complex signal. The output depends
on the phase difference between the current symbol and the previous
symbol. The first integer (or binary pair, if the **Output type** parameter
is set to Bit) in the block's output is the initial condition of zero because
there is no previous symbol.

The input can be either a scalar or a frame-based column vector. The
block accepts the input data types single and double.
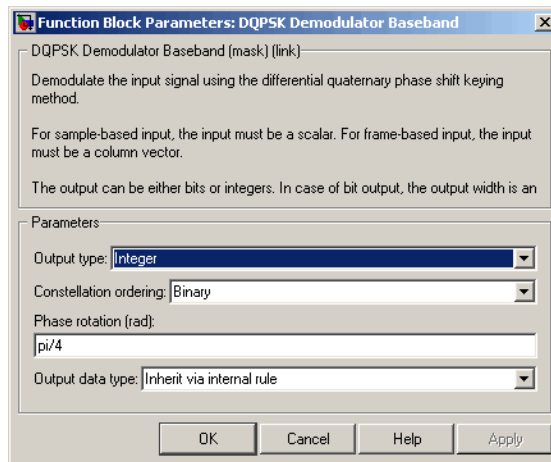
Outputs and Constellation Types

If the **Output type** parameter is set to Integer, then the block maps a
phase difference of

$$\theta + \pi m/2$$

to m, where θ is the **Phase rotation** parameter and m is 0, 1, 2, or 3.

If the **Output type** parameter is set to Bit, then the output contains
pairs of binary values. The reference page for theDQPSK Modulator
Baseband block shows which phase differences map to each binary pair,
for the cases when the **Constellation ordering** parameter is either
Binary or Gray.

Dialog Box

Output type
> Determines whether the output consists of integers or pairs of bits.

Constellation ordering
> Determines how the block maps each integer to a pair of output bits. This field is active only when **Output type** is set to Bit.

Phase rotation (rad)
> This phase difference between the current and previous modulated symbols results in an output of zero.

Output data type
> When the parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is of type single or double.
>
> For integer outputs, this block can output the data types int8, uint8, int16, uint16, int32, uint32, single, and double. For bit outputs, output can be int8, uint8, int16, uint16, int32, uint32, boolean, single, or double.

Pair Block   DQPSK Modulator Baseband

DQPSK Demodulator Baseband

See Also M-DPSK Demodulator Baseband, DBPSK Demodulator Baseband, QPSK Demodulator Baseband

Purpose   Modulate using differential quaternary phase shift keying method

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The DQPSK Modulator Baseband block modulates using the differential
quaternary phase shift keying method. The output is a baseband
representation of the modulated signal.

The input must be a discrete-time signal. For integer inputs, the block
can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`,
`single`, and `double`. For bit inputs, the block can accept `int8`, `uint8`,
`int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`.

### Inputs and Constellation Types

If the **Input type** parameter is set to `Integer`, then valid input values
are 0, 1, 2, and 3. In this case, the input can be either a scalar or a
frame-based column vector. If the first input is m, then the modulated
symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase rotation** parameter. If a successive input is m,
then the modulated symbol is the previous modulated symbol multiplied
by $\exp(j\theta + j\pi m/2)$.

If the **Input type** parameter is set to `Bit`, then the input contains pairs
of binary values. The input can be either a vector of length two or a
frame-based column vector whose length is an even integer. The figure
below shows the complex numbers by which the block multiples the
previous symbol to compute the current symbol, depending on whether
the **Constellation ordering** parameter is set to `Binary` or `Gray`. The
figure assumes that the **Phase rotation** parameter is set to `pi/4`; in
other cases, the two schematics would be rotated accordingly.

DQPSK Modulator Baseband

Binary   Gray

01   00   01   00

10   11   11   10
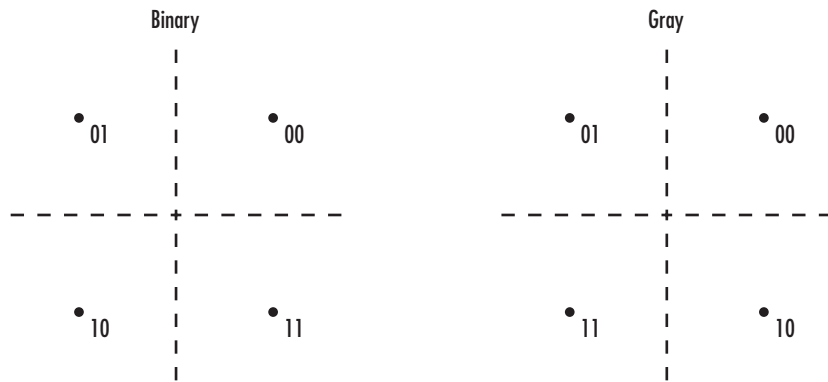
The figure below shows the signal constellation for the DQPSK modulation method when the **Phase rotation** parameter is π/4. The arrows indicate the four possible transitions from each symbol to the next symbol. The Binary and Gray options determine which transition is associated with each pair of input values.

●   Constellation point

◄——►   Transition to next point
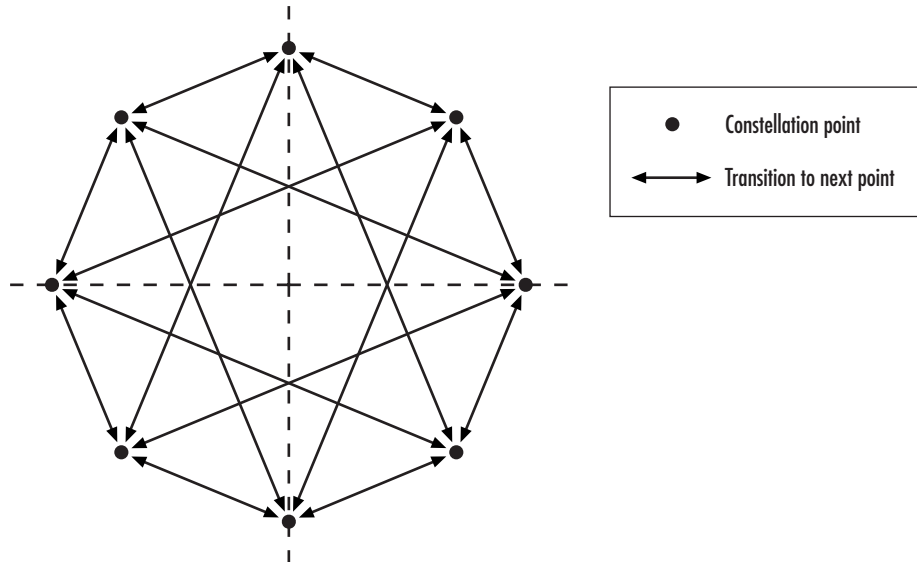
More generally, if the **Phase rotation** parameter has the form π/k for some integer k, then the signal constellation has 2k points.

Dialog Box
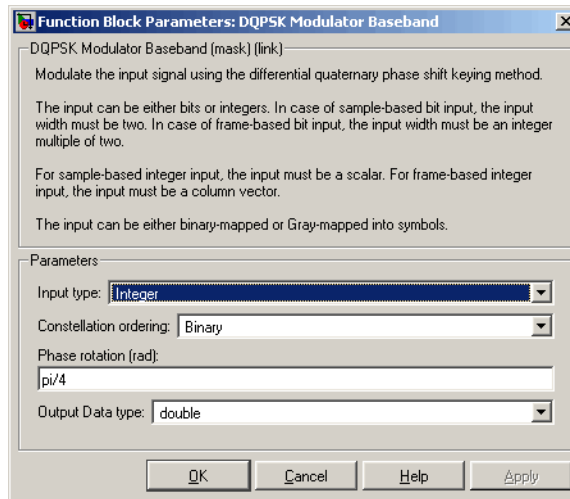
Input type

    Indicates whether the input consists of integers or pairs of bits.

Constellation ordering

    Determines how the block maps each pair of input bits to a corresponding integer. This field is active only when **Input type** is set to Bit.

Phase rotation (rad)

    The phase difference between the previous and current modulated symbols when the input is zero.

Output Data type

    The output data type can be either single or double. By default, the block sets this to double.

Pair Block   DQPSK Demodulator Baseband
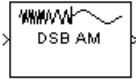
DQPSK Modulator Baseband

See Also   M-DPSK Modulator Baseband, DBPSK Modulator Baseband, QPSK
Modulator Baseband

Purpose   Demodulate DSB-AM-modulated data

Library   Analog Passband Modulation, in Modulation

Description The DSB AM Demodulator Passband block demodulates a signal that was modulated using double-sideband amplitude modulation. The block uses the envelope detection method. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

In the course of demodulating, this block uses a filter whose order, coefficients, passband ripple and stopband ripple are described by their respective lowpass filter parameters.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

This block works only with real inputs of type double. This block is not suited to be placed inside a triggered subsystem.

DSB AM Demodulator Passband

Dialog Block
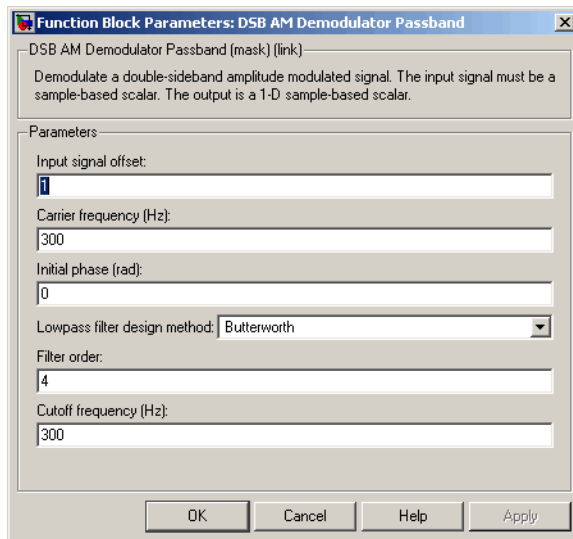
Input signal offset

The same as the **Input signal offset** parameter in the corresponding DSB AM Modulator Passband block.

Carrier frequency (Hz)

The frequency of the carrier in the corresponding DSB AM Modulator Passband block.

Initial phase (rad)

The initial phase of the carrier in radians.

Lowpass filter design method

The method used to generate the filter. Available methods are Butterworth, Chebyshev type I, Chebyshev type II, and Elliptic.

Filter order

The order of the lowpass digital filter specified in the **Lowpass filter design method** field .

Cutoff frequency (Hz)

> The cutoff frequency of the lowpass digital filter specified in the **Lowpass filter design method** field in Hertz.

Passband ripple (dB)

> Applies to Chebyshev type I and Elliptic filters only. This is peak-to-peak ripple in the passband in dB.

Stopband ripple (dB)

> Applies to Chebyshev type II and Elliptic filters only. This is the peak-to-peak ripple in the stopband in dB.
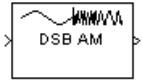
Pair Block   DSB AM Modulator Passband

DSB AM Modulator Passband

Purpose   Modulate using double-sideband amplitude modulation

Library   Analog Passband Modulation, in Modulation

Description   The DSB AM Modulator Passband block modulates using double-sideband amplitude modulation. The output is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

If the input is $u(t)$ as a function of time t, then the output is

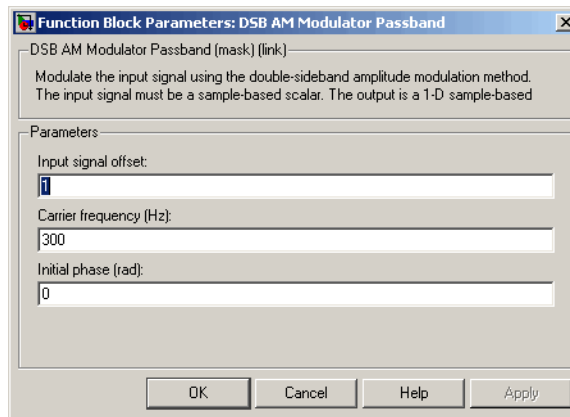$$(u(t) + k)\cos(2\pi f_c t + \theta)$$

where:

- k is the **Input signal offset** parameter.

- f_c is the **Carrier frequency** parameter.

- θ is the **Initial phase** parameter.

It is common to set the value of k to the maximum absolute value of the negative part of the input signal $u(t)$.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

This block works only with real inputs of type `double`. This block is not suited to be placed inside a triggered subsystem.

Dialog Box

Function Block Parameters: DSB AM Modulator Passband

DSB AM Modulator Passband (mask) (link)

Modulate the input signal using the double-sideband amplitude modulation method. The input signal must be a sample-based scalar. The output is a 1-D sample-based

Parameters

Input signal offset:

1

Carrier frequency (Hz):

300

Initial phase (rad):

0

OK   Cancel   Help   Apply

Input signal offset
The offset factor *k*. This value should be greater than or equal to the absolute value of the minimum of the input signal.

Carrier frequency (Hz)
The frequency of the carrier.

Initial phase (rad)
The initial phase of the carrier.

Pair Block   DSB AM Demodulator Passband

See Also   DSBSC AM Modulator Passband, SSB AM Modulator Passband

DSBSC AM Demodulator Passband

Purpose   Demodulate DSBSC-AM-modulated data

Library   Analog Passband Modulation, in Modulation

Description   The DSBSC AM Demodulator Passband block demodulates a signal that
                   was modulated using double-sideband suppressed-carrier amplitude
                   modulation. The input is a passband representation of the modulated
                   signal. Both the input and output signals are real sample-based scalar
                   signals.

   In the course of demodulating, this block uses a filter whose order,
                   coefficients, passband ripple and stopband ripple are described by the
                   their respective lowpass filter parameters.

                   Typically, an appropriate **Carrier frequency** value is much higher
                   than the highest frequency of the input signal. By the Nyquist sampling
                   theorem, the reciprocal of the model's sample time (defined by the
                   model's signal source) must exceed twice the **Carrier frequency**
                   parameter.

                   This block works only with real inputs of type `double`. This block is not
                   suited to be placed inside a triggered subsystem.

Dialog Box
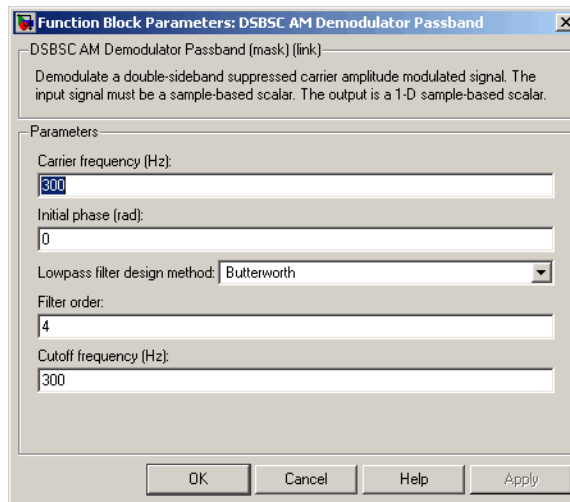
Carrier frequency (Hz)

The carrier frequency in the corresponding DSBSC AM Modulator Passband block.

Initial phase (rad)

The initial phase of the carrier in radians.

Lowpass filter design method

The method used to generate the filter. Available methods are Butterworth, Chebyshev type I, Chebyshev type II, and Elliptic.

Filter order

The order of the lowpass digital filter specified in the **Lowpass filter design method** field .

Cutoff frequency (Hz)

The cutoff frequency of the lowpass digital filter specified in the Lowpass filter design method field in Hertz.

Passband Ripple (dB)

Applies to Chebyshev type I and Elliptic filters only. This is peak-to-peak ripple in the passband in dB.

DSBSC AM Demodulator Passband

Stopband Ripple (dB)
   Applies to Chebyshev type II and Elliptic filters only. This is the peak-to-peak ripple in the stopband in dB.

Pair Block   DSBSC AM Modulator Passband

See Also   DSB AM Demodulator Passband, SSB AM Demodulator Passband

Purpose
Modulate using double-sideband suppressed-carrier amplitude modulation

Library
Analog Passband Modulation, in Modulation

Description

The DSBSC AM Modulator Passband block modulates using double-sideband suppressed-carrier amplitude modulation. The output is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

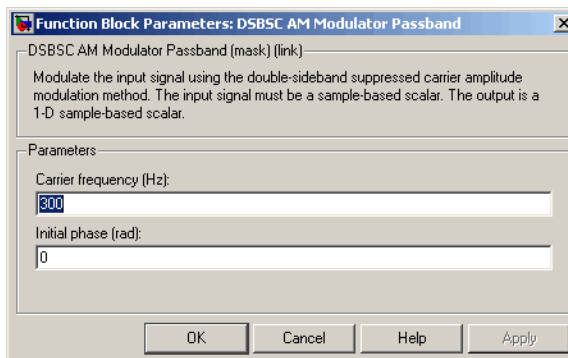If the input is $u(t)$ as a function of time t, then the output is

$$u(t)\cos(2\pi f_c t + \theta)$$

where f_c is the **Carrier frequency** parameter and θ is the **Initial phase** parameter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

This block works only with real inputs of type double. This block is not suited to be placed inside a triggered subsystem.

Dialog Box

2-193

DSBSC AM Modulator Passband

Carrier frequency (Hz)
> The frequency of the carrier.

Initial phase (rad)
> The initial phase of the carrier in radians.

Pair Block   DSBSC AM Demodulator Passband

See Also   DSB AM Modulator Passband, SSB AM Modulator Passband

Purpose   Recover symbol timing phase using early-late gate method

Library   Timing Phase Recovery sublibrary of Synchronization

Description The Early-Late Gate Timing Recovery block recovers the symbol timing
phase of the input signal using the early-late gate method. This block
implements a non-data-aided feedback method.

Inputs

By default, the block has one input port. Typically, the input signal is
the output of a receive filter that is matched to the transmitting pulse
shape. For best results, the input signal power should be normalized.
The input must be a scalar or a frame-based column vector. The input
uses N samples to represent each symbol, where N > 1 is the **Samples
per symbol** parameter. If the input is frame-based, then its vector
length is N*R, where R is a positive integer that indicates the number
of symbols per frame. If the input is sample-based, then its sample time
is 1/N times the underlying symbol period.

If the **Reset** parameter is set to `On nonzero input via port`, then the
block has a second input port, labeled `Rst`. The `Rst` input determines
when the timing estimation process restarts, and must be a scalar. The
sample time of the `Rst` input equals the symbol period if the input signal
is sample-based, and the frame period if the input signal is frame-based.

Typically, **Samples per symbol** is at least 4 and the input signal is
shaped using a raised cosine filter.

Outputs

The block has two output ports, labeled `Sym` and `Ph`:

- The `Sym` output is the result of applying the estimated phase
  correction to the input signal. This output is the signal value for each
  symbol, which can be used for decision purposes. The values in the
  `Sym` output occur at the symbol rate:

Early-Late Gate Timing Recovery

- If the input signal is a frame-based column vector of length N*R, then the `Sym` output is a frame-based column vector of length R having the same frame period.

- If the input signal is a sample-based scalar with sample time T/N, then the `Sym` output is a sample-based scalar with sample time T.

- The `Ph` output gives the phase estimate for each symbol in the input signal.

  The `Ph` output contains nonnegative real numbers less than N. Noninteger values for the phase estimate correspond to interpolated values that lie between two values of the input signal. The sample time or frame period of the `Ph` output is the same as that of the `Sym` output.
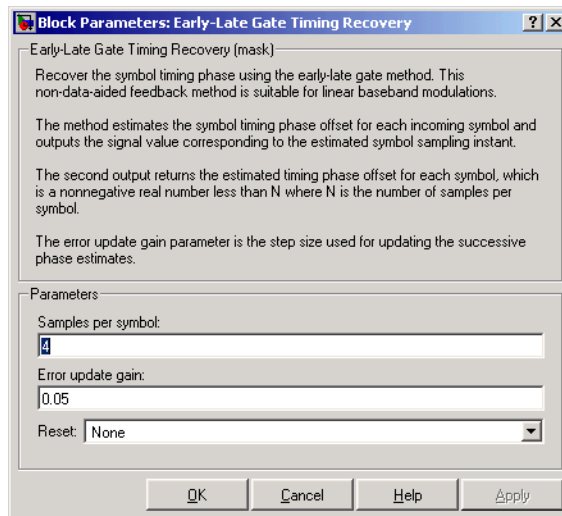

Note If the `Ph` output is very close to either zero or **Samples per symbol**, or if the actual timing phase offset in your input signal is very close to zero, then the block's accuracy might be compromised by small amounts of noise or jitter. The block works well when the timing phase offset is significant rather than very close to zero.

Delays

This block incurs a delay of two symbols when the input is frame-based and three symbols when the input is sample-based.

Dialog Box

Block Parameters: Early-Late Gate Timing Recovery

Early-Late Gate Timing Recovery (mask)

Recover the symbol timing phase using the early-late gate method. This non-data-aided feedback method is suitable for linear baseband modulations.

The method estimates the symbol timing phase offset for each incoming symbol and outputs the signal value corresponding to the estimated symbol sampling instant.

The second output returns the estimated timing phase offset for each symbol, which is a nonnegative real number less than N where N is the number of samples per symbol.

The error update gain parameter is the step size used for updating the successive phase estimates.

Parameters

Samples per symbol:

4

Error update gain:

0.05

Reset: None

OK   Cancel   Help   Apply

Samples per symbol

The number of samples, N, that represent each symbol in the input signal. This must be greater than 1.

Error update gain

A positive real number representing the step size that the block uses for updating successive phase estimates. Typically, this number is less than 1/N, which corresponds to a slowly varying phase.

Reset

Determines whether and under what circumstances the block restarts the phase estimation process. Choices are None, Every frame, and On nonzero input via port. The last option causes the block to have a second input port, labeled Rst.

Algorithm   This block uses a timing error detector whose result for the kth symbol is e(k), given by

Early-Late Gate Timing Recovery

$$e(k) = a_I(k) + a_Q(k)$$

$$a_I(k) = y_I(kT + d_k)\{y_I(kT + T/2 + d_k) - y_I(kT - T/2 + d_{k-1})\}$$

$$a_Q(k) = y_Q(kT + d_k)\{y_Q(kT + T/2 + d_k) - y_Q(kT - T/2 + d_{k-1})\}$$

where

- y_I and y_Q are the in-phase and quadrature components, respectively, of the block's input signal

- T is the symbol period

- d_k is the phase estimate for the kth symbol

For more information about the role that e(k) plays in this block's algorithm, see "Feedback Methods for Timing Phase Recovery" in *Communications Blockset User's Guide*.

References [1] Mengali, Umberto and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

[2] Sklar, Bernard. *Digital Communications: Fundamentals and Applications.* Englewood Cliffs, N.J., Prentice-Hall, 1988.

See Also Gardner Timing Recovery, Squaring Timing Recovery, Mueller-Muller Timing Recovery
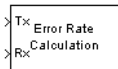
Purpose   Compute bit error rate or symbol error rate of input data

Library   Comm Sinks

Description   The Error Rate Calculation block compares input data from a
transmitter with input data from a receiver. It calculates the error
rate as a running statistic, by dividing the total number of unequal
pairs of data elements by the total number of input data elements from
one source.

You can use this block to compute either symbol or bit error rate,
because it does not consider the magnitude of the difference between
input data elements. If the inputs are bits, then the block computes the
bit error rate. If the inputs are symbols, then it computes the symbol
error rate.

This block inherits the sample time of its inputs.

Note This block does not generate C code when the block parameter
Output data is set to Workspace. Consider changing **Output data**
to Port instead.

Input Data

This block has between two and four input ports, depending on how
you set the dialog parameters. The inports marked Tx and Rx accept
transmitted and received signals, respectively. The Tx and Rx signals
must share the same sampling rate.

The Tx and Rx inputs can be either scalars or frame-based column
vectors of fixed-point data type, or int8, uint8, int16, uint16, int32,
uint32, boolean, single, or double. If Tx is a scalar and Rx is a vector,
or vice-versa, then the block compares the scalar with each element of
the vector. (Overall, the block behaves as if you had preprocessed the
scalar signal with the Communications Blockset Repeat block using the
Maintain input frame rate option.)

Error Rate Calculation

If you check the **Reset port** box, then an additional inport appears, labeled Rst. The Rst input must be a sample-based scalar signal (of type double or boolean) and must have the same sampling rate as the Tx and Rx signals. When the Rst input is nonzero, the block clears its error statistics and then computes them anew.

If you set the **Computation mode** parameter to Select samples from port, then an additional inport appears, labeled Sel. The Sel input indicates which elements of a frame are relevant for the computation; this is explained further, in the last subbullet below. The Sel input can be either a sample-based column vector or a one-dimensional vector of type double.

The guidelines below indicate how you should configure the inputs and the dialog parameters depending on how you want this block to interpret your Tx and Rx data.

- If both data signals are scalar, then this block compares the Tx scalar signal with the Rx scalar signal. You should leave the **Computation mode** parameter at its default value, Entire frame.

- If both data signals are vectors, then this block compares some or all of the Tx and Rx data:

  - If you set the **Computation mode** parameter to Entire frame, then the block compares all of the Tx frame with all of the Rx frame.

  - If you set the **Computation mode** parameter to Select samples from mask, then the **Selected samples from frame** field appears in the dialog. This parameter field accepts a vector that lists the indices of those elements of the Rx frame that you want the block to consider. For example, to consider only the first and last elements of a length-six receiver frame, set the **Selected samples from frame** parameter to [1 6]. If the **Selected samples from frame** vector includes zeros, then the block ignores them.

  - If you set the **Computation mode** parameter to Select samples from port, then an additional input port, labeled Sel, appears on the block icon. The data at this input port must have the same

format as that of the **Selected samples from frame** parameter described above.

- If one data signal is a scalar and the other is a vector, then this block compares the scalar with each entry of the vector. The three subbullets above are still valid for this mode, except that if Rx is a scalar, then the phrase "Rx frame" above refers to the vector expansion of Rx.

Note Simulink requires that input signals have constant length throughout a simulation. If you choose the Select samples from port option and want the number of elements in the subframe to vary during the simulation, then you should pad the Sel signal with zeros. The Error Rate Calculation block ignores zeros in the Sel signal.

Output Data

This block produces a vector of length three, whose entries correspond to:

- The error rate
- The total number of errors, that is, comparisons between unequal elements
- The total number of comparisons that the block made

The block sends this output data to the base MATLAB workspace or to an output port, depending on how you set the **Output data** parameter:

- If you set the **Output data** parameter to Workspace and fill in the **Variable name** parameter, then that variable in the base MATLAB workspace contains the current value when the simulation *ends*. Pausing the simulation does not cause the block to write interim data to the variable.

  If you plan to use this block along with the Real-Time Workshop® software, then you should not use the Workspace option; instead, use

the Port option below and connect the output port to a Simulink
To Workspace block.

- If you set the **Output data** parameter to Port, then an output port
  appears. This output port contains the *running* error statistics.

Delays

The **Receive delay** and **Computation delay** parameters implement
two different types of delays for this block. One is useful when part of
your model causes a lag in the received data, and the other is useful
when you want to ignore the transient behavior of both input signals:

- The **Receive delay** parameter is the number of samples by which
  the received data lags behind the transmitted data. This parameter
  tells the block which samples "correspond" to each other and should
  be compared. The receive delay persists throughout the simulation.

- The **Computation delay** parameter tells the block to ignore the
  specified number of samples at the beginning of the comparison.

If you do not know the receive delay in your model, you can use the
Align Signals block, which automatically compensates for the delay. If
you use the Align Signals block, you should set the **Receive delay** in
the Error Rate Calculation block to 0.

Alternatively, you can use the Find Delay block to find the value of the
delay, and then set the **Receive delay** parameter in the Error Rate
Calculation block to that value.

Note The Version 1.4 Error Rate Calculation block considers a vector
input to be a sample, whereas the current block considers a vector input
to be a frame of multiple samples. For vector inputs of length n, a
Receive delay of k in the Version 1.4 block is equivalent to a **Receive
delay** of k*n in the current block.

If you use the `Select samples from mask` or `Select samples from port` option, then each delay parameter refers to the number of samples that the block receives, whether the block ultimately ignores some of them or not.

Stopping the Simulation Based on Error Statistics

You can configure this block so that its error statistics control the duration of simulation. This is useful for computing reliable steady-state error statistics without knowing in advance how long transient effects might last. To use this mode, check the **Stop simulation** check box. The block attempts to run the simulation until it detects **Target number of errors** errors. However, the simulation stops before detecting enough errors if the time reaches the model's **Stop time** setting (in the **Configuration Parameters** dialog box), if the Error Rate Calculation block makes **Maximum number of symbols** comparisons, or if another block in the model directs the simulation to stop.

To ignore either of the two stopping criteria in this block, set the corresponding parameter (**Target number of errors** or **Maximum number of symbols**) to `Inf`. For example, to reach a target number of errors without stopping the simulation early, set **Maximum number of symbols** to `Inf` and set the model's **Stop time** to `Inf`.

Examples The figure below shows how the block compares pairs of elements and counts the number of error events. This example assumes that the sample time of each input signal is 1 second and that the block's parameters are as follows:

- **Receive delay** = 2
- **Computation delay** = 0
- **Computation mode** = `Entire frame`

The input signals are both frame-based column vectors of length three. However, the schematic arranges each column vector horizontally and aligns pairs of vectors so as to reflect a receive delay of two samples. At

each time step, the block compares elements of the Rx signal with those of the Tx signal that appear directly above them in the schematic. For instance, at time 1, the block compares 2, 4, and 1 from the Rx signal with 2, 3, and 1 from the Tx signal.

The values of the first two elements of Rx appear as asterisks because they do not influence the output. Similarly, the 6 and 5 in the Tx signal do not influence the output up to time 3, though they *would* influence the output at time 4.

In the error rates on the right side of the figure, each numerator at time t reflects the number of errors when considering the elements of Rx up through time t.
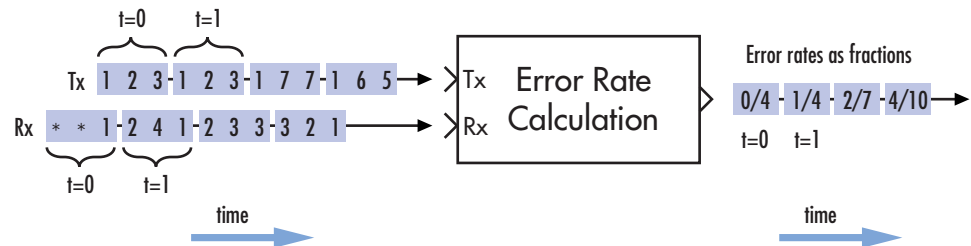
Note: Tx and Rx inputs are frame-based column vectors.

If the block's **Reset port** box had been checked and a reset had occurred at time = 3 seconds, then the last error rate would have been 2/3 instead of 4/10. This value 2/3 would reflect the comparison of 3, 2, and 1 from the Rx signal with 7, 7, and 1 from the Tx signal. The figure below illustrates this scenario.
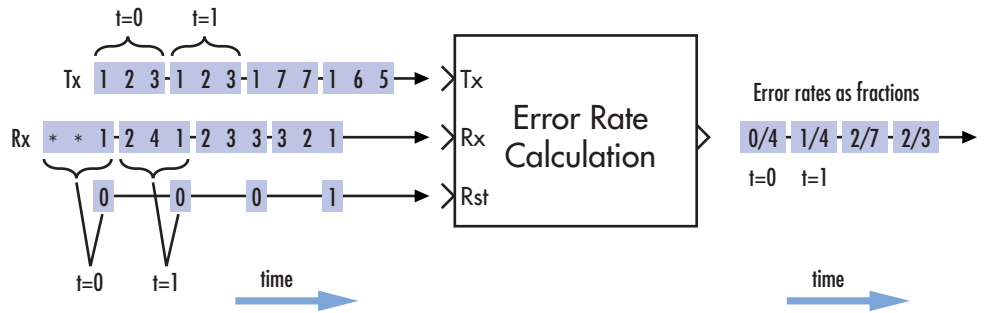
Note: Tx and Rx inputs are frame-based column vectors.

Tuning Parameters in an RSim Executable (Real-Time Workshop Software)

If you use the Real-Time Workshop rapid simulation (RSim) target to build an RSim executable, then you can tune the **Target number of errors** and **Maximum number of symbols** parameters without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

Error Rate Calculation
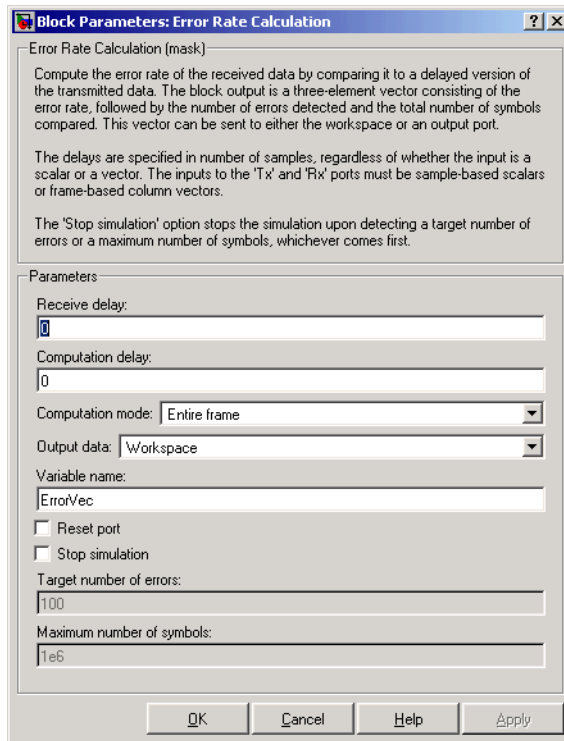
Dialog Box

Receive delay
> Number of samples by which the received data lags behind the transmitted data. (If Tx or Rx is a vector, then each entry represents a sample.)

Computation delay
> Number of samples that the block should ignore at the beginning of the comparison.

Computation mode
> Either Entire frame, Select samples from mask, or Select samples from port, depending on whether the block should consider all or only part of the input frames.

Selected samples from frame

A vector that lists the indices of the elements of the Rx frame vector that the block should consider when making comparisons. This field appears only if **Computation mode** is set to Select samples from mask.

Output data

Either Workspace or Port, depending on where you want to send the output data.

Variable name

Name of variable for the output data vector in the base MATLAB workspace. This field appears only if **Output data** is set to Workspace.

Reset port

If you check this box, then an additional input port appears, labeled Rst.

Stop simulation

If you check this box, then the simulation runs only until this block detects a specified number of errors or performs a specified number of comparisons, whichever comes first.

Target number of errors

The simulation stops after detecting this number of errors. This field is active only if **Stop simulation** is checked.

Maximum number of symbols

The simulation stops after making this number of comparisons. This field is active only if **Stop simulation** is checked.
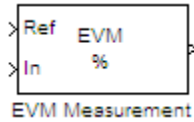
See Also Align Signals, Find Delay

EVM Measurement

Purpose   Calculate vector magnitude difference between ideal reference signal and measured signal

Library   Utility Blocks

Description   Error Vector Magnitude (EVM) is a measurement of modulator or demodulator performance in an impaired signal.

While certain mask selections can change EVM block behavior, the block always has two input signals: a reference signal (at the reference port, `Ref`) and a corrupted signal (at the input port, `In`). You must select which normalization method the block uses when performing EVM calculations and which calculations you want as outputs.

The block either normalizes to the average reference signal power, average constellation power, or peak constellation power. For RMS EVM, Max EVM, and X-percentile EVM, the output computations reflect the normalization method.

The default EVM output is RMS EVM in percent, with an option of `Output maximum EVM` or `Output X-percentile EVM` values. The maximum EVM represents the worst-case EVM value per burst. For the X-percentile option, you can select to output the number of symbols processed in the percentile computations.

The following table shows the output type, the activation (what selects the output computation), computation units, and the corresponding computation duration.

Output	Activation	Units	Computation Duration
RMS EVM	Default	Percentage	Per burst
Max EVM	Parameter setting	Percentage	Per burst

Output	Activation	Units	Computation Duration
Percentile EVM	Parameter setting	Percentage	Continuous
Number of symbols	Parameter setting if you select **Output X-percentile EVM**	None	Continuous

The computation duration in per burst mode spans the symbols in the present burst. The computation duration in continuous mode spans all the symbols across multiple bursts.

Dimension

The block computes measurements for bursts of data. The data must be of length *N*, where *N* is the size of the burst (sample-based or frame-based). When computing RMS EVM or Max EVM, the block computes a unique output for each incoming burst; therefore, the computation duration is per burst.

The block computes the X-percentile for all incoming symbols across multiple bursts. This computation duration is the continuous mode (in contrast to the per burst duration for RMS EVM or Max EVM).

Input Signals

The input signals must be 1-D or 2-D sample-based column vectors or 2-D frame-based column vectors. The input and reference signals must have identical dimensions.

Output Signals

The output is always a scalar value.

Data Type

The block accepts double, single, and fixed-point data types. The output of the block is always double type.

EVM Measurement

Algorithms

The EVM block provides three different normalization methods. You can normalize measurements according to the average power of the reference signal, average constellation power, or peak constellation power. Different industry standards follow one of these normalization methods.

The following table lists how the block calculates the RMS EVM value for different normalization methods.

EVM Normalization Method	Algorithm
Reference Signal	$$EVM_{RMS} = \sqrt{\dfrac{\dfrac{1}{N}\displaystyle\sum_{k=1}^{N}(e_k)}{\dfrac{1}{N}\displaystyle\sum_{k=1}^{N}(I_k^2 + Q_k^2)}} * 100$$
Average Power	$$EVM_{RMS} = \sqrt{\dfrac{\dfrac{1}{N}\displaystyle\sum_{k=1}^{N}(e_k)}{P_{avg}}} * 100$$
Peak Power	$$EVM_{RMS} = \sqrt{\dfrac{\dfrac{1}{N}\displaystyle\sum_{k=1}^{N}(e_k)}{P_{max}}} * 100$$

where,

$$e_k = (I_k - \tilde{I}_k)^2 + (Q_k - \tilde{Q}_k)^2$$

I_k = In-phase measurement of the *kth* symbol in the burst

Q_k = Quadrature phase measurement of the *kth* symbol in the burst

N = Input vector length

P_{avg} = The value for **Average constellation power**

P_{max} = The value for **Peak constellation power**

I_k and Q_k represent ideal (reference) values. $\tilde{I}_k$ and $\tilde{Q}_k$ represent measured (received) symbols.

The max EVM is the maximum EVM value in a frame or

$$EVM_{\max} = \max_{k \in [1,...,N]} \{EVM_k\}$$

where k is the kth symbol in a burst of length N.

The definition for EVM_k varies depending upon which normalization method you select for computing measurements. The block supports the algorithms in the following table.

EVM Normalization	Algorithm
Reference Signal	$EVM_k = \sqrt{\dfrac{e_k}{\dfrac{1}{N}\displaystyle\sum_{k=1}^{N}(I_k^2 + Q_k^2)}} *100$

EVM Normalization	Algorithm
Average Power	$$EVM_k = \sqrt{\frac{e_k}{P_{avg}}} * 100$$
Peak Power	$$EVM_k = \sqrt{\frac{e_k}{P_{max}}} * 100$$

The block computes X-percentile EVM by creating a histogram of all the incoming EVM_k values. The output provides the EVM value below which X% of the EVM values lay.

Function Block Parameters: EVM Measurement ☒

─ EVM Measurement (mask) (link) ─────────────────────

Calculate the RMS Error Vector Magnitude (EVM) between a reference signal and a
corrupted received signal of an input frame. The RMS EVM measurement is defined
in different standards as the (%) ratio of the RMS error to the square root of
either one of the following:
1) Average reference signal power
2) Average constellation power
3) Peak constellation power

All power values are given in watts, referenced to 1 ohm.

The optional maximum EVM is the maximum EVM value in an input frame.
The optional X-percentile output is the EVM value below which X% of the computed
EVM values lie.

─ Parameters ─────────────────────────────────────

Normalize RMS error vector by: `Average reference signal power` ▼

☐ Output maximum EVM

☐ Output X-percentile EVM

[OK]   [Cancel]   [Help]   [Apply]

**Dialog
Box**

Normalize RMS error vector by:
Selects the method by which the block normalizes measurements:

- `Average reference signal power`

- `Average constellation power`

- `Peak constellation power`
This parameter defaults to `Average reference signal power`.

Average constellation power:
Normalizes EVM measurement by the average constellation
power. This parameter only appears if you set **Normalize RMS
error vector** to `Average constellation power`.

Peak constellation power:
Normalizes EVM measurement by the peak constellation power.
This parameter only appears if you set **Normalize RMS error
vector** to `Peak constellation power`.

EVM Measurement

Output maximum EVM

  Outputs the maximum EVM of an input vector or frame.

Output X-percentile EVM

  Enables an output X-percentile EVM measurement. When you select this option, specify **X-percentile value (%)**.

X-percentile value (%)

  This parameter only appears when you select **Output X-percentile EVM**. The Xth percentile is the EVM value below which X% of all the computed EVM values lie. The parameter defaults to the 95th percentile. Therefore, 95% of all EVM values are below this output.

Output the number of symbols processed

  Outputs the number of symbols that the block uses to compute the **Output X-percentile EVM**. This parameter only appears when you select **Output X-percentile EVM**.

Examples   To see an example using the EVM block, refer to Measuring Modulator Accuracy in the Communications Blockset User's Guide.

References   [1] IEEE Standard 802.16-2004: "Part 16: Air interface for fixed broadband wireless access systems," October 2004. http://ieee802.org/16/published.html

[2] 3 GPP TS 45.005 V8.1.0 (2008-05): "Radio Access Network: Radio transmission and reception"

[3] IEEE Standard 802.11a-1999: "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band," 1999.

See Also   commmeasure.EVM, MER Measurement

Purpose   Find delay between two signals

Library   Utility Blocks

Description   The Find Delay block finds the delay between a signal and a delayed, and possibly distorted, version of itself. The block is particularly useful when you want to compare a transmitted and received signal to find the bit error rate, but do not know the delay in the received signal. See "Computing Delays" for more information about signal delays.

The input port labeled sRef receives the original signal, while the input port labeled sDel receives the delayed version of the signal. The two input signals must have the same sample times.

The output port labeled delay outputs the delay in units of samples. If you select Include "change signal" output port, then an output port labeled chg appears. The chg output port outputs 1 when there is a change from the delay computed at the previous sample, and 0 when there is no change. The delay output port outputs signals of type double, and the chg output port outputs signals of type boolean.

The block's **Correlation window length** parameter specifies how many samples of the signals the block uses to calculate the cross-correlation. The delay output is a nonnegative integer less than the **Correlation window length**.

As the **Correlation window length** is increased, the reliability of the computed delay also increases. However, the processing time to compute the delay increases as well.

You can make the Find Delay block stop updating the delay after it computes the same delay value for a specified number of samples. To do so, select the **Disable recurring updates** check box, and enter a positive integer in the **Number of constant delay outputs to disable updates** field. For example, if you set **Number of constant delay outputs to disable updates** to 20, the block will stop recalculating and updating the delay after it calculates the same value 20 times in succession. Disabling recurring updates causes the simulation to run faster after the target number of constant delays occurs.

Find Delay

Tips for Using the Block Effectively

- Set **Correlation window length** sufficiently large so that the computed delay eventually stabilizes at a constant value. When this occurs, the signal from the optional chg output port stabilizes at the constant value of zero. If the computed delay is not constant, you should increase **Correlation window length**. If the increased value of **Correlation window length** exceeds the duration of the simulation, then you should also increase the duration of the simulation accordingly.

- If the cross-correlation between the two signals is broad, then the **Correlation window length** value should be much larger than the expected delay, or else the algorithm might stabilize at an incorrect value. For example, a CPM signal has a broad autocorrelation, so it has a broad cross-correlation with a delayed version of itself. In this case, the **Correlation window length** value should be much larger than the expected delay.

- If the block calculates a delay that is greater than 75 percent of the **Correlation window length**, the signal sRef is probably delayed relative to the signal sDel. In this case, you should switch the signal lines leading into the two input ports.

Examples   ### Finding the Delay Before Calculating an Error Rate

A typical use of this block is to determine the correct **Receive delay** parameter in the Error Rate Calculation block. This is illustrated in "Finding the Delay in a Model". In that example, the modulation/demodulation operation introduces a computational delay into the received signal and the Find Delay block determines that the delay is 6 samples. This value of 6 becomes a parameter in the Error Rate Calculation block, which computes the bit error rate of the system.

Another example of this usage is in "Computing Delays".

Finding the Delay to Help Align Words

Another typical use of this block is to determine how to align the boundaries of frames with the boundaries of codewords or other types of

data blocks. "Manipulating Delays" describes when such alignment is necessary and also illustrates, in the "Aligning Words of a Block Code" discussion, how to use the Find Delay block to solve the problem.

Setting the Correlation Window Length

The next example illustrates how to tell when the **Correlation window length** is not sufficiently large.
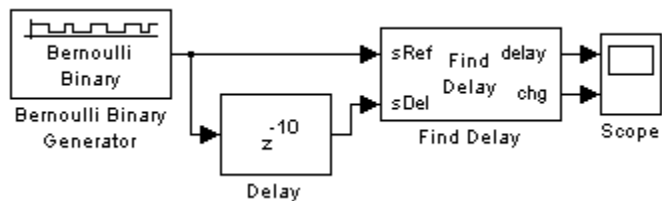
The model uses a Delay block to delay a signal by 10 samples, and uses the Find Delay block to compare the original signal with the delayed version. The model then displays the output of the Find Delay block in a scope. If the **Correlation window length** is 15, the scope shows that the calculated delay is not constant over time, as you can see below.
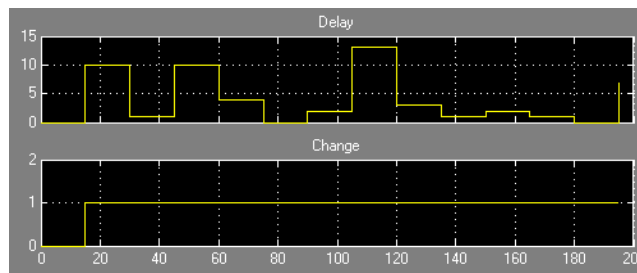
This result tells you to increase the **Correlation window length**. If you increase it to 50, the calculated delay stabilizes at 10, as shown below.

Find Delay
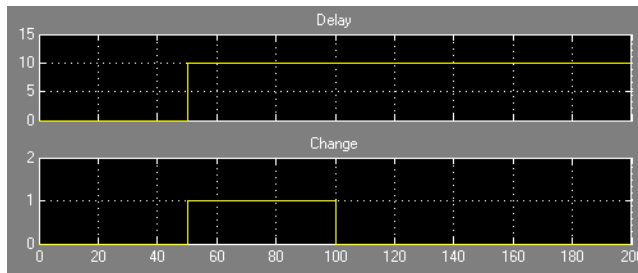
Dialog Box
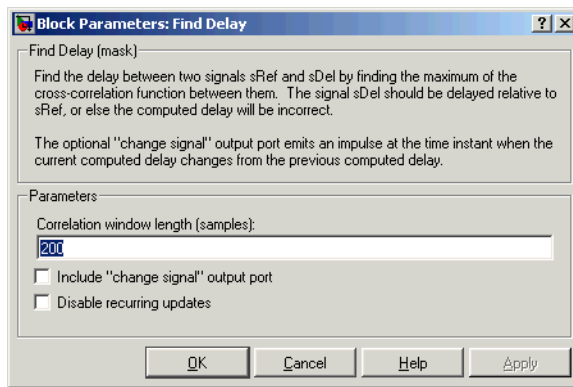
Correlation window length

The number of samples the block uses to calculate the cross-correlations of the two signals.

Include "change signal" output port

If you select this option, then the block has an extra output port that emits an impulse when the current computed delay differs from the previous computed delay.

Disable recurring updates

Selecting this option causes the block to stop computing the delay after it computes the same delay value for a specified number of samples.

Number of constant delay outputs to disable updates
A positive integer specifying how many times the block must compute the same delay before ceasing to update. This field appears only if **Disable recurring updates** is selected.

Algorithm   The Find Delay block finds the delay by calculating the cross-correlations of the first signal with time-shifted versions of the second signal, and then finding the index at which the cross-correlation is maximized.

See Also   Align Signals, Error Rate Calculation

FM Demodulator Passband

Purpose   Demodulate FM-modulated data

Library   Analog Passband Modulation, in Modulation
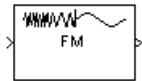
Description   The FM Demodulator Passband block demodulates a signal that was modulated using frequency modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

For best results, use a carrier frequency which is estimated to be larger than 10% of the reciprocal of your input signal's sample time. This is due to the implementation of the Hilbert transform by means of a filter.

In the following example, we sample a 10Hz input signal at 8000 samples per second. We then designate a Hilbert Transform filter of order 100. Below is the response of the Hilbert Transform filter as returned by `fvtool`.
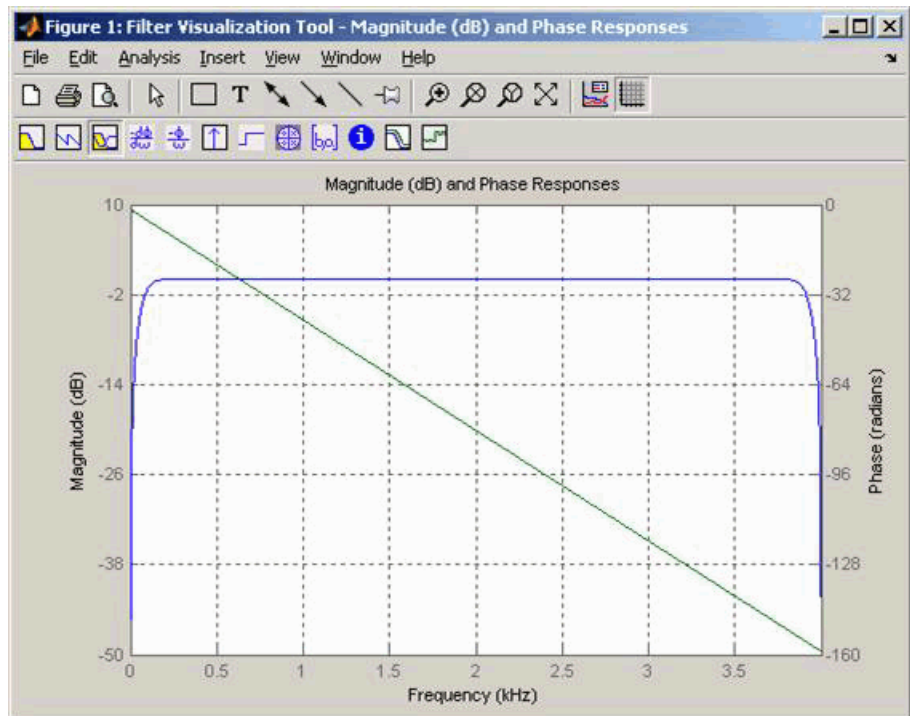
Note the bandwidth of the filter's magnitude response. By choosing a
carrier frequency larger than 10% (but less than 90%) of the reciprocal
of your input signal's sample time (8000 samples per second, in this
example) or equivalently, a carrier frequency larger than 400Hz, we
ensure that the Hilbert Transform Filter will be operating in the flat
section of the filter's magnitude response (shown in blue), and that our
modulated signal will have the desired magnitude and form.

Typically, an appropriate **Carrier frequency** value is much higher
than the highest frequency of the input signal. By the Nyquist sampling
theorem, the reciprocal of the model's sample time (defined by the
model's signal source) must exceed twice the **Carrier frequency**
parameter.

FM Demodulator Passband

This block works only with real inputs of type `double`. This block is not suited to be placed inside a triggered subsystem.

Dialog Box
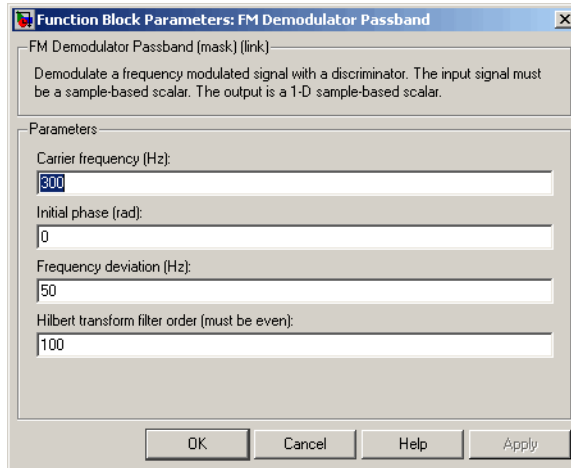
Carrier frequency (Hz)
The frequency of the carrier.

Initial phase (rad)
The initial phase of the carrier in radians.

Frequency deviation (Hz)
The frequency deviation of the carrier frequency in Hertz. Sometimes it is referred to as the "variation" in the frequency.

Hilbert transform filter order
The length of the FIR filter used to compute the Hilbert transform.

Pair Block   FM Modulator Passband

Purpose   Modulate using frequency modulation

Library   Analog Passband Modulation, in Modulation

Description
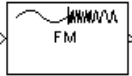
The FM Modulator Passband block modulates using frequency modulation. The output is a passband representation of the modulated signal. The output signal's frequency varies with the input signal's amplitude. Both the input and output signals are real sample-based scalar signals.

If the input is $u(t)$ as a function of time t, then the output is

$$\cos\left(2\pi f_c t + 2\pi K_c \int_0^t u(\tau)d\tau + \theta \right)$$

where:

- f_c is the **Carrier frequency** parameter.
- θ is the **Initial phase** parameter.
- K_c is the **Modulation constant** parameter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal.

By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

This block works only with real inputs of type `double`. This block is not suited to be placed inside a triggered subsystem.

FM Modulator Passband

Dialog Box
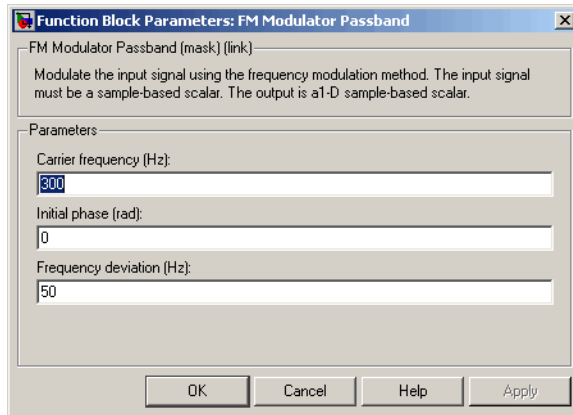
Carrier frequency (Hz)
> The frequency of the carrier.

Initial phase (rad)
> The initial phase of the carrier in radians.

Frequency deviation (Hz)
> The frequency deviation of the carrier frequency in Hertz.
> Sometimes it is referred to as the "variation" in the frequency.

Pair Block   FM Demodulator Passband

Purpose   Reduce amplitude of input signal by amount specified

Library   RF Impairments

Description The Free Space Path Loss block simulates the loss of signal power due
                 to the distance between transmitter and receiver. The block reduces
                 the amplitude of the input signal by an amount that is determined in
                 either of two ways:

```
Free Space
Path Loss
  10 dB
```

- By the **Distance (km)** and **Carrier frequency (MHz)** parameters,
  if you specify `Distance and Frequency` in the **Mode** field

- By the **Loss (dB)** parameter, if you specify `Decibels` in the **Mode**
  field

The input to this block must be a complex signal.

**Dialog
Box**
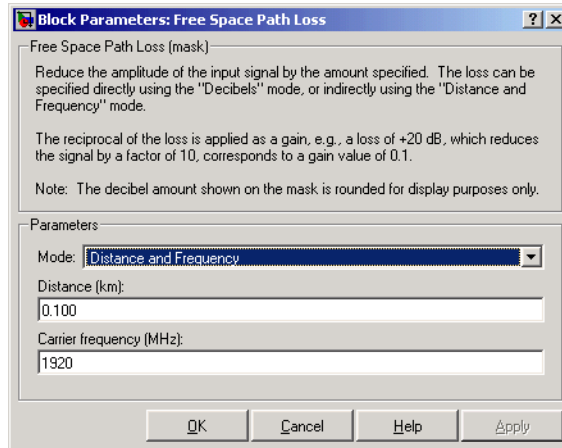
Mode
> Method of specifying the amount by which the signal power
> is reduced. The choices are `Decibels` and `Distance and
> Frequency`.

Free Space Path Loss

Loss

The signal loss in decibels. This parameter appears when you set **Mode** to `Decibels`.

Distance

Distance between transmitter and receiver in kilometers. This parameter appears when you set **Mode** to `Distance and Frequency`.

Carrier frequency (MHz)

The carrier frequency in megahertz. This parameter appears when you set **Mode** to `Distance and Frequency`.

Examples   The model below illustrates the effect of the Free Space Path Loss block with the following parameter settings:

- **Mode** is set to `Distance and Frequency`.

- **Distance (km)** is set to `0.5`

- **Carrier frequency (MHz)** is set to `180`
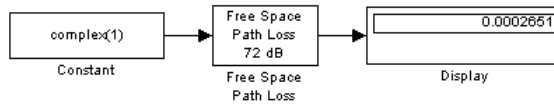
See Also   Memoryless Nonlinearity

Purpose   Recover symbol timing phase using Gardner's method

Library   Timing Phase Recovery sublibrary of Synchronization

Description   The Gardner Timing Recovery block recovers the symbol timing phase
of the input signal using Gardner's method. This block implements a
non-data-aided feedback method that is independent of carrier phase
recovery. The timing error detector that forms part of this block's
algorithm requires at least two samples per symbol, one of which is the
point at which the decision can be made.

Inputs

By default, the block has one input port. Typically, the input signal is
the output of a receive filter that is matched to the transmitting pulse
shape. For best results, the input signal power should be less than 1.
The input must be a scalar or a frame-based column vector. The input
uses N samples to represent each symbol, where N > 1 is the **Samples
per symbol** parameter. If the input is frame-based, then its vector
length is N*R, where R is a positive integer that indicates the number
of symbols per frame. If the input is sample-based, then its sample time
is 1/N times the underlying symbol period.

If the **Reset** parameter is set to On nonzero input via port, then the
block has a second input port, labeled Rst. The Rst input determines
when the timing estimation process restarts, and must be a scalar. The
sample time of the Rst input equals the symbol period if the input signal
is sample-based, and the frame period if the input signal is frame-based.

Outputs

The block has two output ports, labeled Sym and Ph:

- The Sym output is the result of applying the estimated phase
  correction to the input signal. This output is the signal value for each
  symbol, which can be used for decision purposes. The values in the
  Sym output occur at the symbol rate:

- If the input signal is a frame-based column vector of length N*R, then the `Sym` output is a frame-based column vector of length R having the same frame period.

- If the input signal is a sample-based scalar with sample time T/N, then the `Sym` output is a sample-based scalar with sample time T.

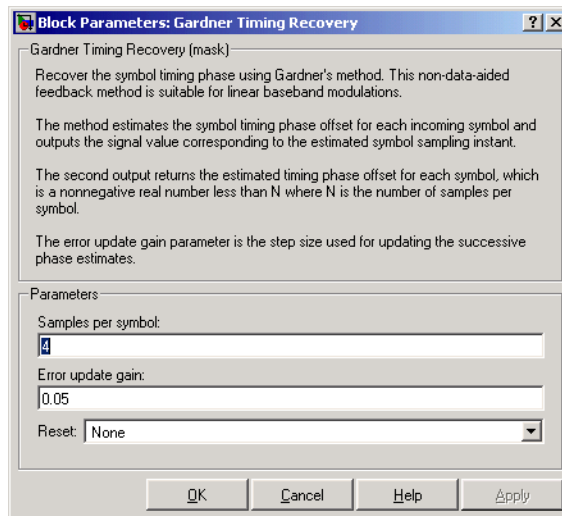- The `Ph` output gives the phase estimate for each symbol in the input.

  The `Ph` output contains nonnegative real numbers less than N. Noninteger values for the phase estimate correspond to interpolated values that lie between two values of the input signal. The sample time or frame period of the `Ph` output is the same as that of the `Sym` output.

Note If the `Ph` output is very close to either zero or **Samples per symbol**, or if the actual timing phase offset in your input signal is very close to zero, then the block's accuracy might be compromised by small amounts of noise or jitter. The block works well when the timing phase offset is significant rather than very close to zero.

Delays

This block incurs a delay of two symbols when the input is frame-based and three symbols when the input is sample-based.

Dialog Box

Samples per symbol

The number of samples, N, that represent each symbol in the input signal. This must be greater than 1.

Error update gain

A positive real number representing the step size that the block uses for updating successive phase estimates. Typically, this number is less than 1/N, which corresponds to a slowly varying phase.

Reset

Determines whether and under what circumstances the block restarts the phase estimation process. Choices are None, Every frame, and On nonzero input via port. The last option causes the block to have a second input port, labeled Rst.

Algorithm   This block uses a timing error detector whose result for the kth symbol is e(k), given by

Gardner Timing Recovery

$$e(k) = a_I(k) + a_Q(k)$$

$$a_I(k) = \left\{ y_I((k-1)T + d_{k-1}) - y_I(kT + d_k) \right\} y_I(kT - T/2 + d_{k-1})$$

$$a_Q(k) = \left\{ y_Q((k-1)T + d_{k-1}) - y_Q(kT + d_k) \right\} y_Q(kT - T/2 + d_{k-1})$$

where

- y_I and y_Q are the in-phase and quadrature components, respectively, of the block's input signal
- T is the symbol period
- d_k is the phase estimate for the kth symbol

Notice from the expressions in curly braces above that the timing error detector approximates the derivative of y using finite differences.

For more information about the role that e(k) plays in this block's algorithm, see "Feedback Methods for Timing Phase Recovery" in *Communications Blockset User's Guide*.

Examples   The `commgardnerphrecov` demonstration model uses this block.

References   [1] Gardner, F. M., "A BPSK/QPSK Timing-Error Detector for Sampled Receivers", *IEEE Transactions on Communications*, Vol. COM-34, No. 5, May 1986, pp. 423-429.

[2] Mengali, Umberto and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

[3] Meyr, Heinrich, Marc Moeneclaey, and Stefan A. Fechtel, *Digital Communication Receivers*, Vol 2, New York, Wiley, 1998.

[4] Oerder, M., "Derivation of Gardner's Timing-Error Detector from the ML principle", *IEEE Transactions on Communications*, Vol. COM-35, No. 6, June 1987, pp. 684-685.

See Also   Early-Late Gate Timing Recovery, Squaring Timing Recovery,
Mueller-Muller Timing Recovery

Gaussian Filter

Purpose Filter input signal, possibly downsampling, using Gaussian FIR filter

Library Comm Filters

Description The Gaussian Filter block filters the input signal using a Gaussian FIR filter. The block expects the input signal to be upsampled, so that the **Input samples per symbol** parameter, N, is at least 2. The block's icon shows the filter's impulse response."

Characteristics of the Filter

The impulse response of the Gaussian filter is

$$h(t) = \frac{\exp\left(\dfrac{-t^2}{2\delta^2}\right)}{\sqrt{2\pi} \cdot \delta}$$

where

$$\delta = \frac{\sqrt{\ln(2)}}{2\pi BT}$$

and B is the filter's 3-dB bandwidth. The **BT product** parameter is B times the input signal's symbol period. For a given BT product, the Signal Processing Toolbox `gaussfir` function generates a filter that is half the bandwidth of the filter generated by the Communications Blockset Gaussian Filter block.

The **Group delay** parameter is the number of symbol periods between the start of the filter's response and the peak of the filter's response. The group delay and N determine the length of the filter's impulse response, which is 2 * N * **Group delay** + 1.

The **Filter coefficient normalization** parameter indicates how the block scales the set of filter coefficients:

- Sum of coefficients means that the sum of the coefficients equals 1.

- `Filter energy` means that the sum of the squares of the coefficients equals 1.

- `Peak amplitude` means that the maximum coefficient equals 1.

After the block normalizes the set of filter coefficients as above, it multiplies all coefficients by the **Linear amplitude filter gain** parameter. As a result, the output is scaled by $\sqrt{N}$. If the output of this block feeds the input to the AWGN Channel block, specify the AWGN signal power parameter to be 1/N.

Input and Output Signals

This block filters each channel of the input signal independently over time. The output frame status and dimensions are always the same as those of the input signal. When inputs are frame based, the block treats each column as an independent channel; the block filters each column. When inputs are sample based, the block treats each element of the input as an individual channel. `double`, `single`, and fixed-point data types are supported.

Exporting Filter Coefficients to the MATLAB Workspace

To examine or manipulate the coefficients of the filter that this block designs, select **Export filter coefficients to workspace**. Then set the **Coefficient variable name** parameter to the name of a variable that you want the block to create in the MATLAB workspace. Running the simulation causes the block to create the variable, overwriting any previous contents in case the variable already exists.

Gaussian Filter

Dialog Box
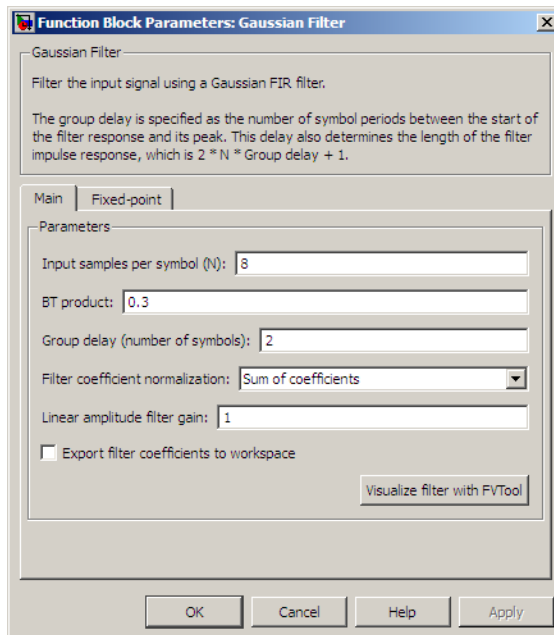
Input samples per symbol
> A positive integer representing the number of samples per symbol in the input signal.

BT product
> The product of the filter's 3-dB bandwidth and the input signal's symbol period

Group delay
> A positive integer that represents the number of symbol periods between the start of the filter response and its peak.

Filter coefficient normalization
> The block scales the set of filter coefficients so that this quantity equals 1. Choices are `Sum of coefficients`, `Filter energy`, and `Peak amplitude`.

Linear amplitude filter gain

A positive scalar used to scale the filter coefficients after the block uses the normalization specified in the **Filter coefficient normalization** parameter.

Export filter coefficients to workspace

If you check this box, then the block creates a variable in the MATLAB workspace that contains the filter coefficients.

Coefficient variable name

The name of the variable to create in the MATLAB workspace. This field appears only if **Export filter coefficients to workspace** is selected.

Visualize filter with FVTool

If you click this button, then MATLAB launches the Filter Visualization Tool, `fvtool`, to analyze the Gaussian filter whenever you apply any changes to the block's parameters. If you launch `fvtool` for the filter, and subsequently change parameters in the mask, `fvtool` will not update. You will need to launch a new `fvtool` in order to see the new filter characteristics. Also note that if you have launched `fvtool`, then it will remain open even after the model is closed.
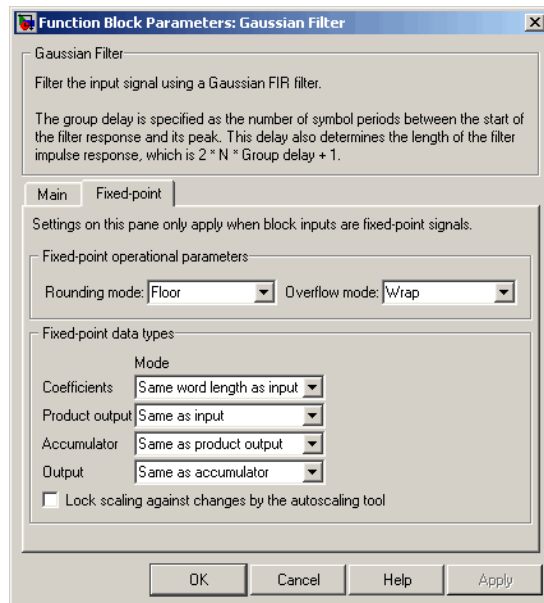
Rounding mode

    Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`. The block uses the **Rounding** selection if a number cannot be represented exactly by the specified data type and scaling, it is rounded to a representable number. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink® Fixed Point™ documentation.

Overflow mode

    Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

    Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator). See "Filter Structure Diagrams" in the *Signal Processing Blockset Reference*

Guide for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you are able to enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the coefficients, in bits. If applicable, you are able to enter separate fraction lengths for the numerator and denominator coefficients.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the coefficients. If applicable, you are able to enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.

- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" in the *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool.

See Also Raised Cosine Receive Filter, `gaussfir`

References [1] 3GPP TS 05.04 V8.4.0 — 3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Digital cellular telecommunications system (Phase 2+); Modulation (Release 1999)

Gaussian Noise Generator

Purpose   Generate Gaussian distributed noise with given mean and variance values

Library   Noise Generators sublibrary of Comm Sources

Description   The Gaussian Noise Generator block generates discrete-time white Gaussian noise. You must specify the **Initial seed** vector in the simulation.
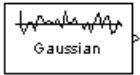
The **Mean Value** and the **Variance** can be either scalars or vectors. If either of these is a scalar, then the block applies the same value to each element of a sample-based output or each column of a frame-based output. Individual elements or columns, respectively, are uncorrelated with each other.

When the **Variance** is a vector, its length must be the same as that of the **Initial seed** vector. In this case, the covariance matrix is a diagonal matrix whose diagonal elements come from the **Variance** vector. Since the off-diagonal elements are zero, the output Gaussian random variables are uncorrelated.

When the **Variance** is a square matrix, it represents the covariance matrix. Its off-diagonal elements are the correlations between pairs of output Gaussian random variables. In this case, the **Variance** matrix must be positive definite, and it must be N-by-N, where N is the length of the **Initial seed**.

The probability density function of n-dimensional Gaussian noise is

$$f(x) = \left((2\pi)^n \det K\right)^{-1/2} \exp\left(-(x-\mu)^T K^{-1}(x-\mu)/2\right)$$

where x is a length-n vector, K is the n-by-n covariance matrix, μ is the mean value vector, and the superscript T indicates matrix transpose.

Initial Seed

The **Initial seed** parameter initializes the random number generator that the Gaussian Noise Generator block uses to add noise to the input signal. For best results, the **Initial seed** should be a prime number

greater than 30. Also, if there are other blocks in a model that have an **Initial seed** parameter, you should choose different initial seeds for all such blocks.

You can choose seeds for the Gaussian Noise Generator block using the Communications Blockset randseed function. At the MATLAB prompt, enter

```
randseed
```

This returns a random prime number greater than 30. Entering randseed again produces a different prime number. If you supply an integer argument, randseed always returns the same prime for that integer. For example, randseed(5) always returns the same answer.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See "Signal Attribute Parameters for Random Sources" in the *Communications Blockset User's Guide* for more details.

If the **Initial seed** parameter is a vector, then its length becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. In this case, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal. If the **Initial seed** parameter is a scalar but either the **Mean value** or **Variance** parameter is a vector, then the vector length determines the output attributes mentioned above.

Gaussian Noise Generator
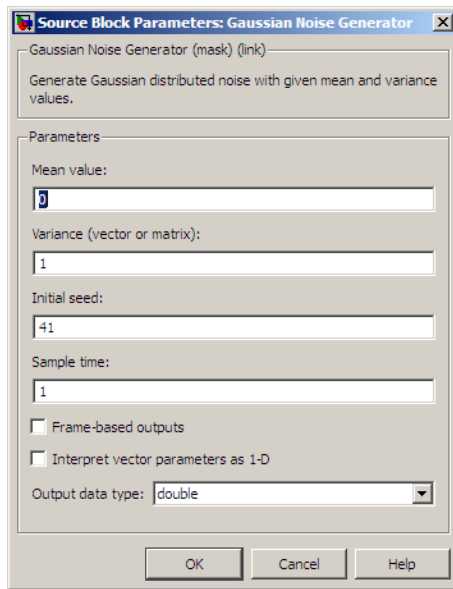
**Dialog
Box**

Opening this dialog box causes a running simulation to pause.
See "Changing Source Block Parameters" in the online Simulink
documentation for details.

Mean value
> The mean value of the random variable output.

Variance
> The covariance among the output random variables.

Initial seed
> The initial seed value for the random number generator.

Sample time
> The period of each sample-based vector or each row of a
> frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

The output can be set to `double` or `single` data types.

See Also   Random Source (Signal Processing Blockset documentation), AWGN Channel, `rand` (built-in MATLAB function), randseed

General Block Deinterleaver

Purpose   Restore ordering of symbols in input vector

Library   Block sublibrary of Interleaving

Description   The General Block Deinterleaver block rearranges the elements of its input vector without repeating or omitting any elements. The input can be real or complex. If the input contains N elements, then the **Elements** parameter is a vector of length N that indicates the indices, in order, of the output elements that came from the input vector. That is, for each integer k between 1 and N,
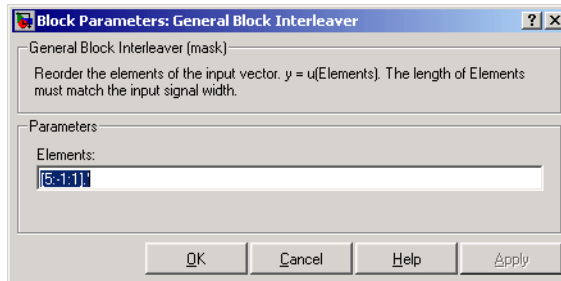
Output(**Elements**(k)) = Input(k)

The **Elements** parameter must contain unique integers between 1 and N.

If the input is frame-based, then both it and the **Elements** parameter must be column vectors.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

To use this block as an inverse of theGeneral Block Interleaver block, use the same **Elements** parameter in both blocks. In that case, the two blocks are inverses in the sense that applying the General Block Interleaver block followed by the General Block Deinterleaver block leaves data unchanged.

Dialog Box
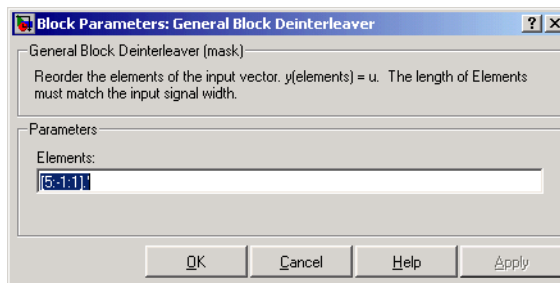
Elements
    A vector of length N that lists the indices of the output elements
that came from the input vector.

Examples   This example reverses the operation in the example on the General
Block Interleaver block reference page. If **Elements** is `[4,1,3,2]` and
the input to the General Block Deinterleaver block is `[1;40;59;32]`,
then the output of the General Block Deinterleaver block is
`[40;32;59;1]`.

Pair Block   General Block Interleaver

See Also   `perms` (MATLAB function)

General Block Interleaver

Purpose   Reorder symbols in input vector

Library   Block sublibrary of Interleaving

Description   The General Block Interleaver block rearranges the elements of its input vector without repeating or omitting any elements. The input can be real or complex. If the input contains N elements, then the **Elements** parameter is a vector of length N that indicates the indices, in order, of the input elements that form the length-N output vector; that is,

Output(k) = Input(**Elements**(k))

for each integer k between 1 and N. The contents of **Elements** must be integers between 1 and N, and must have no repetitions.

If the input is frame-based, then both it and the **Elements** parameter must be column vectors.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box

Elements
   A vector of length N that lists the indices of the input elements that form the output vector.

Examples   If **Elements** is [4,1,3,2] and the input vector is [40;32;59;1], then the output vector is [1;40;59;32]. Notice that all of these vectors have the same length and that the vector **Elements** is a permutation of the vector [1:4].

Pair Block   General Block Deinterleaver

See Also   perms (MATLAB function)

General CRC Generator

Purpose   Generate CRC bits according to generator polynomial and append to input data frames

Library   CRC sublibrary of Error Correction and Detection

Description

General
CRC
Generator
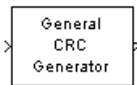
The General CRC Generator block generates cyclic redundancy code (CRC) bits for each input data frame and appends them to the frame. You specify the generator polynomial for the CRC algorithm using the **Generator polynomial** parameter. This block is general in the sense that the degree of the polynomial does not need to be a power of two. You represent the polynomial in one of these ways:

- As a binary row vector containing the coefficients in descending order of powers. For example, [1 1 0 1] represents the polynomial $x^3 + x^2 + 1$.

- As an integer row vector containing the powers of nonzero terms in the polynomial, in descending order. For example, [3 2 0] represents the polynomial $x^3 + x^2 + 1$.

You specify the initial state of the internal shift register by the **Initial states** parameter. The **Initial states** parameter is either a scalar or a binary row vector of length equal to the degree of the generator polynomial. A scalar value is expanded to a row vector of length equal to the degree of the generator polynomial. For example, the default initial state of [0] is expanded to a row vector of all zeros.

You specify the number of checksums that the block calculates for each input frame by the **Checksums per frame** parameter. The **Checksums per frame** value must evenly divide the size of the input frame. If the value of **Checksums per frame** is k, the block does the following:

1 Divides each input frame into k subframes of equal size

2 Prefixes the **Initial states** vector to each of the k subframes

3 Applies the CRC algorithm to each augmented subframe

4 Appends the resulting checksums at the end of each subframe

5 Outputs concatenated subframes

If the size of the input frame is m and the degree of the generator polynomial is r, the output frame has size m + k * r.

This block supports `double` and `boolean` data types. The output data type is inherited from the input.

Example

Suppose the size of the input frame is 10, the degree of the generator polynomial is 3, **Initial states** is [0], and **Checksums per frame** is 2. The block divides each input frame into two subframes of size 5 and appends a checksum of size 3 to each subframe, as shown below. The initial states are not shown in this example, because an initial state of [0] does not affect the output of the CRC algorithm. The output frame then has size 5 + 3 + 5 + 3 = 16.
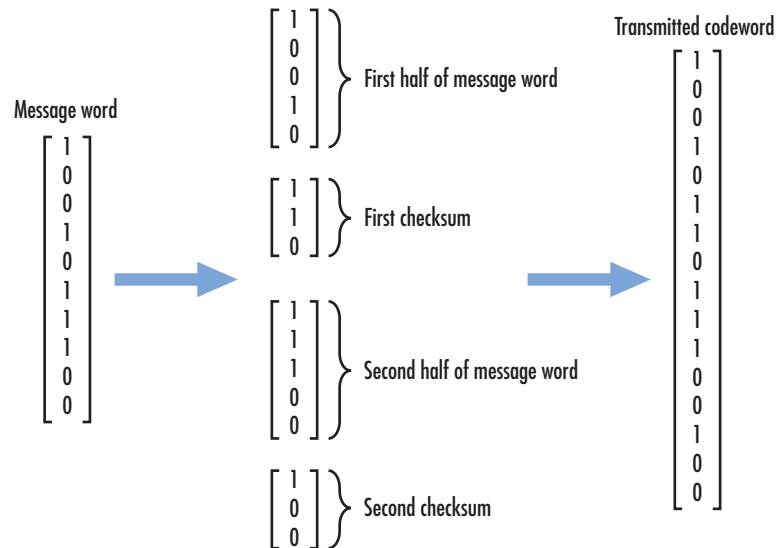
General CRC Generator

Example of Cyclic Redundancy Check Encoding

This example clarifies the operation of the General CRC Generator block by comparing the generated CRC bits from the library block with those generated from primitive Simulink blocks. The model is located at: *matlabroot*/help/toolbox/commblks/examples/doc_crcgen.
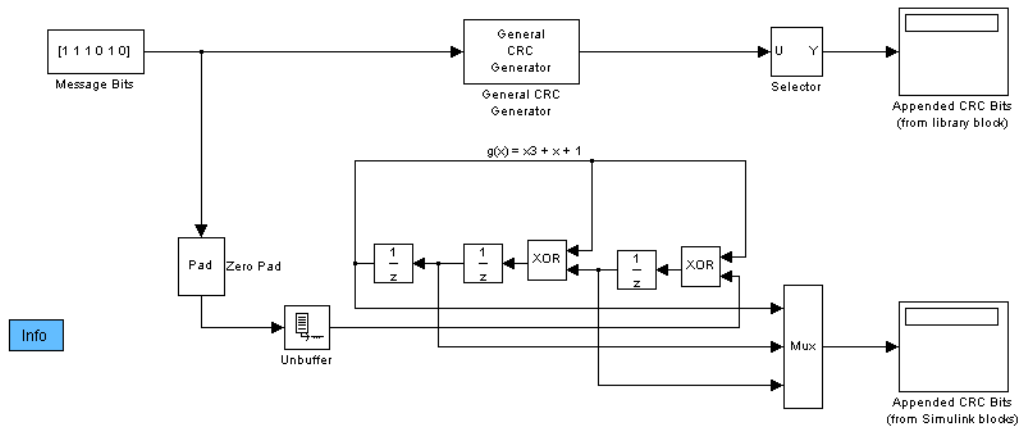
For a known input message with a length of 6 bits, the model generates CRC bits for a generator polynomial, $g(x) = x^3 + x + 1$, and a specific initial state of the register.

You can experiment with different initial states by changing the value of **Initial states** prior to running the simulation. For all values, the comparison (generated CRC bits from the library block with those generated from primitive Simulink blocks) holds true.

Using the General CRC Generator block allows you to easily specify the generator polynomial (especially for higher order polynomials).

Signal Attributes

The General CRC Generator block has one input port and one output port. Both ports allow only frame-based binary column vectors.

**Dialog
Box**
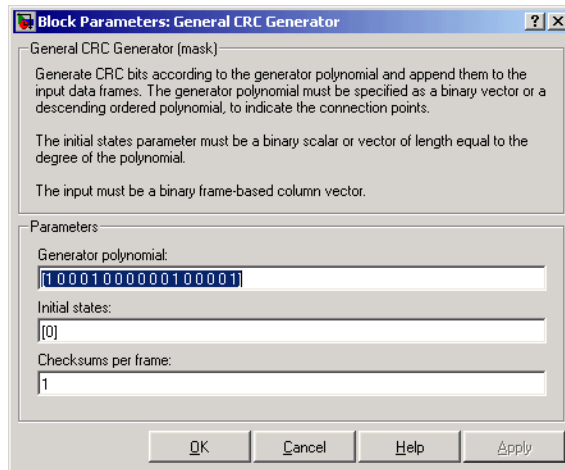
Generator polynomial

> A binary or integer row vector specifying the generator
> polynomial, in descending order of powers.

Initial states

> Binary scalar or a binary row vector of length equal to the degree
> of the generator polynomial, specifying the initial state of the
> internal shift register.

Checksums per frame

> Positive integer specifying the number of checksums the block
> calculates for each input frame.

Algorithm For a description of the CRC algorithm as implemented by this block,
see "Cyclic Redundancy Check Coding" in *Communications Blockset
User's Guide*.

General CRC Generator

Schematic of the CRC Implementation

The above circuit divides the polynomial

$$a(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + a_0 \quad \text{by}$$

$$g(x) = g_{r-1}x^{r-1} + g_{r-2}x^{r-2} + \cdots + g_1 x + g_0 \text{, and returns the remainder}$$

$$d(x) = d_{r-1}x^{r-1} + d_{r-2}x^{r-2} + \cdots + d_1 x + d_0 \text{.}$$

The input symbols $\{a_{k-1}, a_{k-2}, \ldots, a_2, a_1, a_0\}$ are fed into the shift register one at a time in order of decreasing index. When the last symbol (a_0) works its way out of the register (achieved by augmenting the message with r zeros), the register contains the coefficients of the remainder polynomial $d(x)$.

This remainder polynomial is the checksum that is appended to the original message, which is then transmitted.

References

[1] Sklar, Bernard, *Digital Communications: Fundamentals and Applications.* Englewood Cliffs, N.J., Prentice-Hall, 1988.

[2] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

Pair Block   General CRC Syndrome Detector

See Also   CRC-N Generator, CRC-N Syndrome Detector

Purpose   Detect errors in input data frames according to generator polynomial

Library   CRC sublibrary of Error Correction and Detection
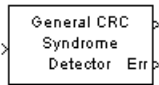
Description   The General CRC Syndrome Detector block computes checksums for its entire input frame. The block's second output is a vector whose size is the number of checksums, and whose entries are 0 if the checksum computation yields a zero value, and 1 otherwise. The block's first output is the set of message words with the checksums removed.

General CRC
Syndrome
Detector Err

The block's parameter settings should agree with those in the General CRC Generator block.

You specify the number of checksums the block calculates for each frame by the **Checksums per frame** parameter. If the **Checksums per frame** value is k, the size of the input frame is n, and the degree of the generator polynomial is r, then k must divide n - k*r, which is the size of the message word.

This block supports `double` and `boolean` data types. The output data type is inherited from the input.

Example

Suppose the received codeword has size 16, the generator polynomial has degree 3, **Initial states** is [0], and **Checksums per frame** is 2. The block computes the two checksums of size 3, one from the first half of the received codeword, and the other from the second half of the received codeword, as shown in the following figure. The initial states are not shown in this example, because an initial state of [0] does not affect the output of the CRC algorithm. The block concatenates the two halves of the message word as a single vector of size 10 and outputs this vector through the first output port. The block outputs a 2-by-1 binary frame vector whose entries depend on whether the computed checksums are zero. The following figure shows an example in which the first checksum is nonzero and the second checksum is zero. This indicates that an error occurred in transmitting the first half of the codeword.
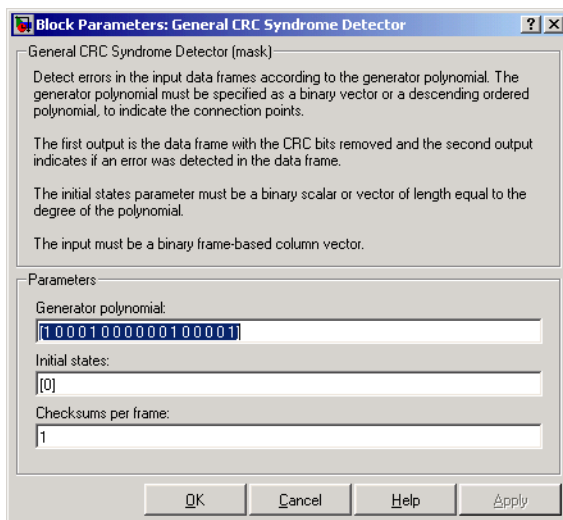
General CRC Syndrome Detector
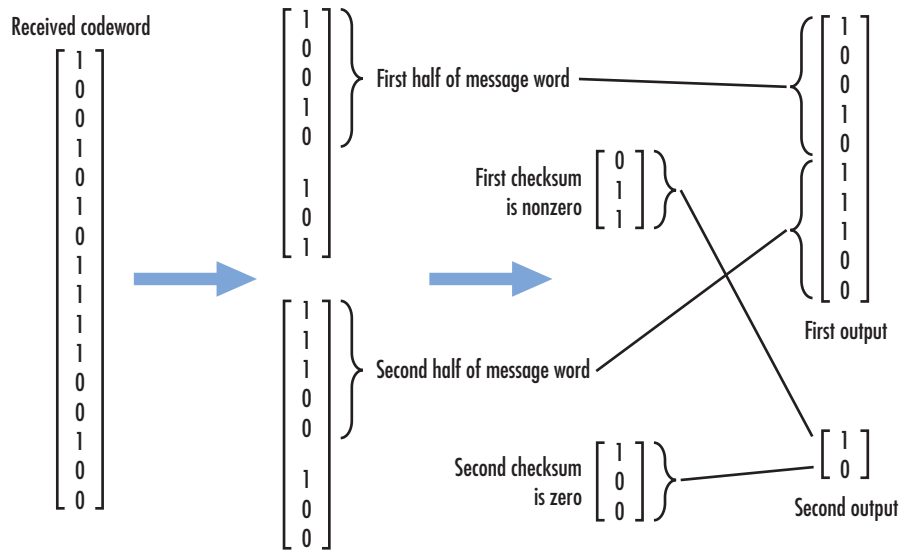
Signal Attributes

The General CRC Syndrome Detector block has one input port and two output ports. All ports allow frame-based binary column vectors only.

**Dialog
Box**

Generator polynomial

A binary or integer row vector specifying the generator
polynomial, in descending order of powers.

Initial states

A binary scalar or a binary row vector of length equal to the
degree of the generator polynomial, specifying the initial state of
the internal shift register.

Checksums per frame

A positive integer specifying the number of checksums the block
calculates for each input frame.

Algorithm   For a description of the CRC algorithm as implemented by this block,
see "Cyclic Redundancy Check Coding" in *Communications Blockset
User's Guide*.

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and
Applications*. Englewood Cliffs, N.J., Prentice-Hall, 1988.

General CRC Syndrome Detector

[2] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

Pair Block   General CRC Generator

See Also   CRC-N Generator, CRC-N Syndrome Detector

Purpose   Restore ordering of symbols using specified-delay shift registers

**Library**   Convolutional sublibrary of Interleaving
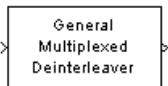
Description   The General Multiplexed Deinterleaver block restores the original ordering of a sequence that was interleaved using theGeneral Multiplexed Interleaver block.

> General
> Multiplexed
> Deinterleaver

In typical usage, the parameters in the two blocks have the same values. As a result, the **Interleaver delay** parameter, V, specifies the delays for each shift register in the corresponding *interleaver*, so that the delays of the deinterleaver's shift registers are actually `max(V)-V`.

The input can be either a scalar or a frame-based column vector. It can be real or complex. The input and output signals share the same sample time.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box
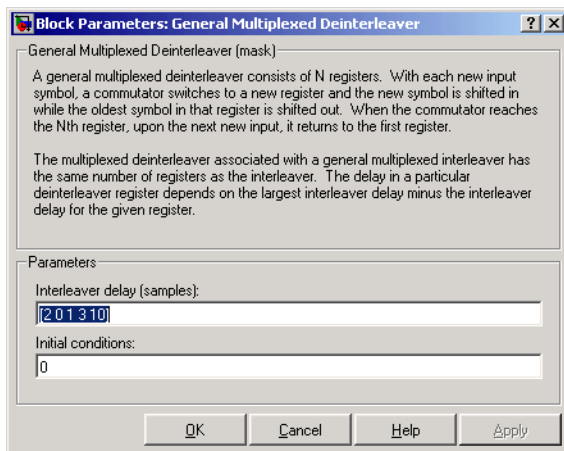
General Multiplexed Deinterleaver

Interleaver delay (samples)
> A vector that lists the number of symbols that fit in each shift register of the corresponding interleaver. The length of this vector is the number of shift registers.

Initial conditions
> The values that fill each shift register when the simulation begins.

Pair Block   General Multiplexed Interleaver

See Also   Convolutional Deinterleaver, Helical Deinterleaver

References   [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

**Purpose**   Permute input symbols using set of shift registers with specified delays

Library   Convolutional sublibrary of Interleaving
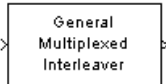
Description   The General Multiplexed Interleaver block permutes the symbols in
the input signal. Internally, it uses a set of shift registers, each with
its own delay value.

The input can be either a scalar or a frame-based column vector. It
can be real or complex. The input and output signals share the same
sample time.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`,
`uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this
output will be the same as that of the input signal.

The **Interleaver delay** parameter is a column vector whose entries
indicate how many symbols can fit into each shift register. The length
of the vector is the number of shift registers. (In sample-based mode, it
can also be a row vector.)

The **Initial conditions** parameter indicates the values that fill each
shift register at the beginning of the simulation. If **Initial conditions**
is a scalar, then its value fills all shift registers; if **Initial conditions** is
a column vector, then each entry fills the corresponding shift register.
(In sample-based mode, **Initial conditions** can also be a row vector.) If
a given shift register has zero delay, then the value of the corresponding
entry in the **Initial conditions** vector is unimportant.

General Multiplexed Interleaver

Dialog Box
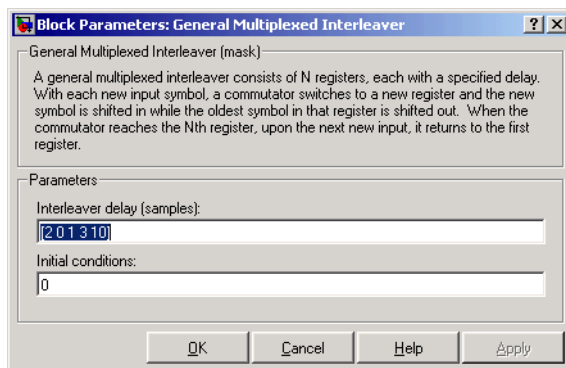
Interleaver delay (samples)

A vector that lists the number of symbols that fit in each shift register. The length of this vector is the number of shift registers.

Initial conditions

The values that fill each shift register when the simulation begins.

Pair Block   General Multiplexed Deinterleaver

See Also   Convolutional Interleaver, Helical Interleaver

References   [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

Purpose   Demodulate QAM-modulated data

Library   AM, in Digital Baseband sublibrary of Modulation
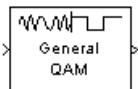
Description   The General QAM Demodulator Baseband block demodulates a signal
                   that was modulated using quadrature amplitude modulation. The input
                   is a baseband representation of the modulated signal.

                   The input must be a discrete-time complex signal. The **Signal
                   constellation** parameter defines the constellation by listing its points
                   in a vector of complex numbers. The block maps the mth point in the
                   Signal constellation vector to the integer m-1.

                   The input can be either a scalar or a frame-based column vector and
                   must be of data types single or double.

General QAM Demodulator Baseband
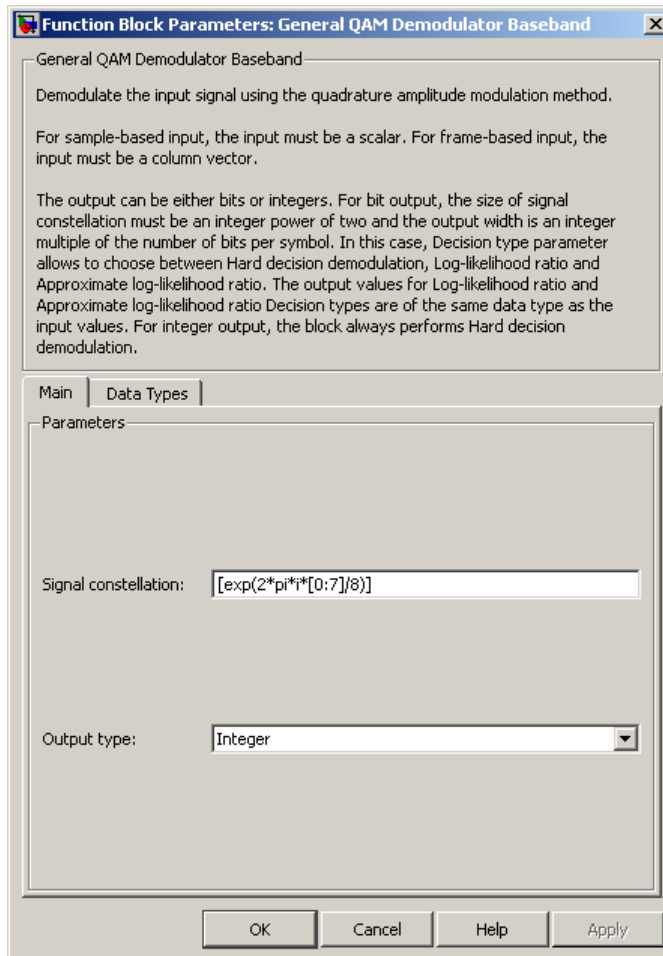
Dialog Box

Signal constellation

    A real or complex vector that lists the constellation points.

Output type

    Determines whether the block produces integers or binary representations of integers.

If you set this parameter to `Integer`, the block produces integers.

If you set this parameter to `Bit`, the block produces a group of K bits, called a *binary word*, for each symbol, when **Decision type** is set to `Hard decision`. If **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, the block outputs bitwise LLR and approximate LLR, respectively.

Decision type

This field appears when `Bit` is selected in the pull-down list **Output type**.

Specifies the use of hard decision, LLR, or approximate LLR during demodulation. See "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide* for algorithm details.

Noise variance source

This field appears when you set `Approximate log-likelihood ratio` or `Log-likelihood ratio` for **Decision type**.

When you set this parameter to `Dialog`, you can then specify the noise variance in the **Noise variance** field. When you set this option to `Port`, a port appears on the block through which the noise variance can be input.

Noise variance

This parameter appears when the **Noise variance source** is set to `Dialog` and specifies the noise variance in the input signal. This parameter is tunable in normal mode, Accelerator mode and Rapid Accelerator mode.

If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameter without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

The LLR algorithm involves computing exponentials of very large or very small numbers using finite precision arithmetic and would yield:

- `Inf` to `-Inf` if **Noise variance** is very high

- `NaN` if **Noise variance** and signal power are both very small

In such cases, use approximate LLR, as its algorithm does not involve computing exponentials.
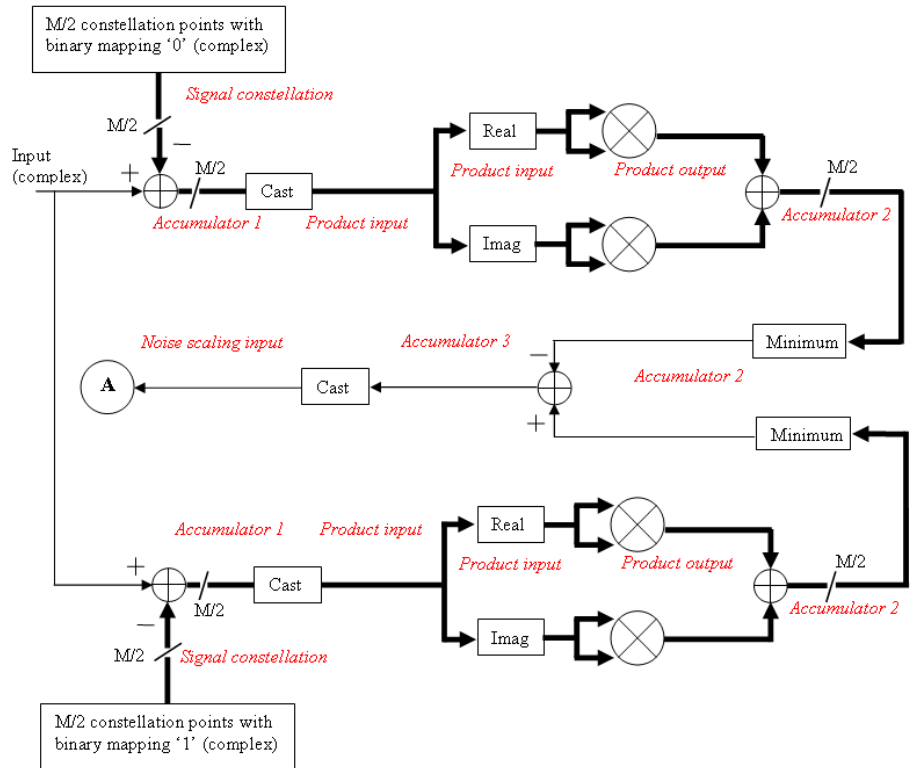
Fixed-Point Signal Flow Diagrams
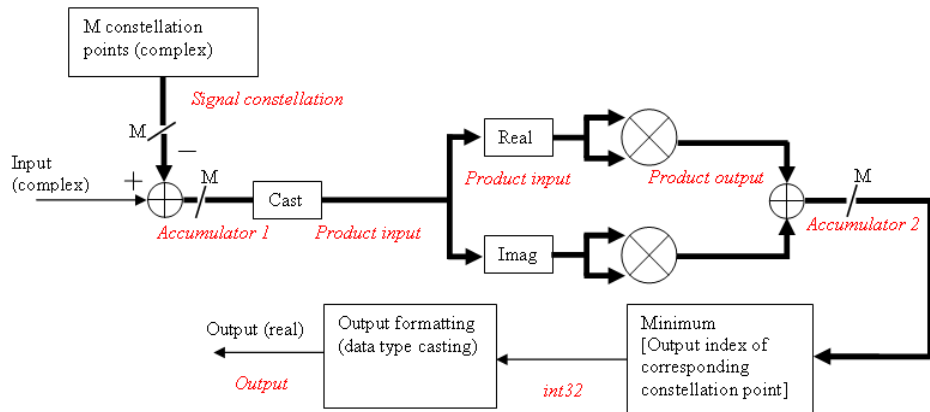
Fixed-Point Signal Flow Diagram for Hard Decision Mode

Note In the figure above, *M* represents the size of the **Signal constellation** .

The general QAM Demodulator Baseband block supports fixed-point operations for computing Hard Decision (**Output** type set to `Bit` and **Decision type** is set to `Hard decision`) and Approximate LLR (**Output type** is set to `Bit` and **Decision type** is set to `Approximate Log-Likelihood ratio`) output values. The input values must have fixed-point data type for fixed-point operations.
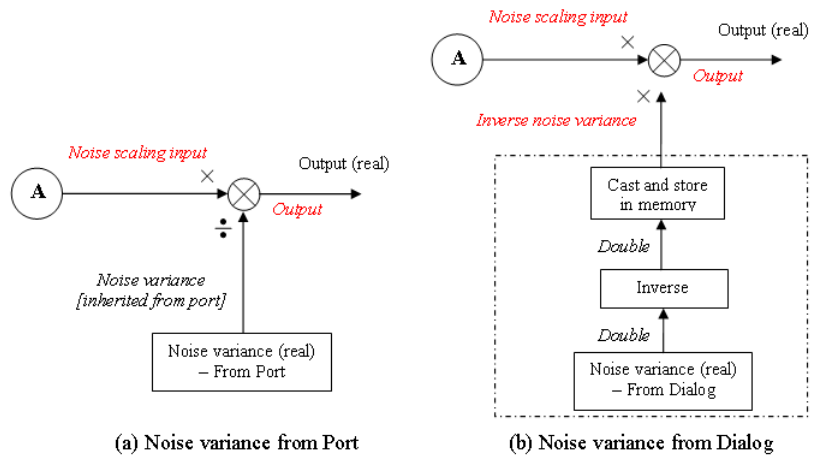
Note Fixed-Point operations are NOT yet supported for Exact LLR output values.

Fixed-Point Signal Flow Diagram for Approximate LLR Mode

**Note** In the figure above, M represents the size of the **Signal constellation**.

(a) Noise variance from Port   (b) Noise variance from Dialog

Fixed-Point Signal Flow Diagram for Approximate LLR Mode: Noise Variance Operation Modes

Note If **Noise variance** is set to `Dialog`, the block performs the operations shown inside the dotted line once during initialization. The block also performs these operations if the **Noise variance** value changes during simulation.

**Data
Types
Attributes**
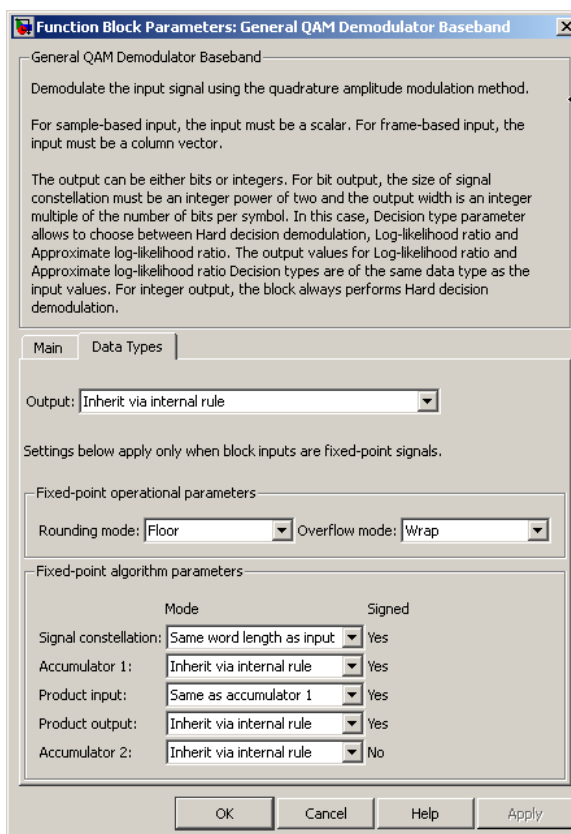
Fixed-Point Attributes for Hard Decision Mode

Output

> The block supports the following Output options:

> When you set the parameter to `Inherit via internal rule` (default setting), the block inherits the output data type from the input port. The output data type is the same as the input data type if the input is of type `single` or `double`.

For integer outputs, you can set this block's output to `Inherit via internal rule` (default setting), `Smallest unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`.

For bit outputs, when you set **Decision type** to `Hard decision`, you can set the output to `Inherit via internal rule`, `Smallest unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, or `double`.

When you set **Decision type** to `Log-likelihood ratio` or `Approximate log-likelihood ratio` and the input is a floating point data type, then the output inherits its data type from the input. For example, if the input is of data type `double`, the output is also of data type `double`. When you set **Decision type** to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, and the input is a fixed-point signal, the **Output** parameter, located in the Fixed-Point algorithm parameters region of the Data-Type tab, specifies the output data type.

When you set the parameter to `Smallest unsigned integer`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box. If you select `ASIC/FPGA` in the **Hardware Implementation** pane, the output data type is the ideal minimum size, i.e., `ufix(1)` for bit outputs, and `ufix(ceil(log2(M)))` for integer outputs. For all other choices, the **Output** data type is an unsigned integer with the smallest available word length large enough to fit the ideal minimum size, usually corresponding to the size of a char (e.g., `uint8`).

Rounding Mode Parameter

Use this parameter to specify the rounding method to be used when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result.

For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow Mode Parameter

Use this parameter to specify the method to be used if the magnitude of a fixed-point calculation result does not fit into the range of the data type and scaling that stores the result:

- `Saturate` represents positive overflows as the largest positive number in the range being used, and negative overflows as the largest negative number in the range being used.

- `Wrap` uses modulo arithmetic to cast an overflow back into the representable range of the data type. See Modulo Arithmetic for more information.

For more information, see the Rounding Mode Parameter subsection of Specifying Fixed-Point Attributes.

Signal constellation

Use this parameter to define the data type of the **Signal constellation** parameter.

- When you select `Same word length as input` the word length of the **Signal constellation** parameter matches that of the input to the block. The fraction length is computed to provide the best precision for given signal constellation values.

- When you select `Specify word length`, the **Word Length** field appears, and you may enter a value for the word length. The fraction length is computed to provide the best precision for given signal constellation values.

Accumulator 1

Use this parameter to specify the data type for **Accumulator 1**:

- When you select `Inherit via internal rule`, the block automatically calculates the output word and fraction lengths. For more information, see the `Inherit via Internal Rule` subsection of the *Signal Processing Blockset User's Guide*.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of **Accumulator 1**, in bits.

Product Input

Use this parameter to specify the data type for **Product input**.

- When you select `Same as accumulator 1`, the **Product Input** characteristics match those of **Accumulator 1**.

- When you select `Binary point scaling` you can enter the word length and the fraction length of **Product input**, in bits.

Product Output

Use this parameter to select the data type for Product output.

- When you select `Inherit via internal rule`, the block automatically calculates the output signal type. For more information, see the Inherit via Internal Rule subsection of the *Signal Processing Blockset User's Guide*.

- When you select `Binary point scaling` enter the word length and the fraction length for **Product output**, in bits.

Accumulator 2

Use this parameter to specify the data type for **Accumulator 2**:

- When you select `Inherit via internal rule`, the block automatically calculates the accumulator data type. The internal rule calculates the ideal, full-precision word length and fraction length as follows:

$$WL_{\text{ideal accumulator 2}} = WL_{\text{input to accumulator 2}}$$
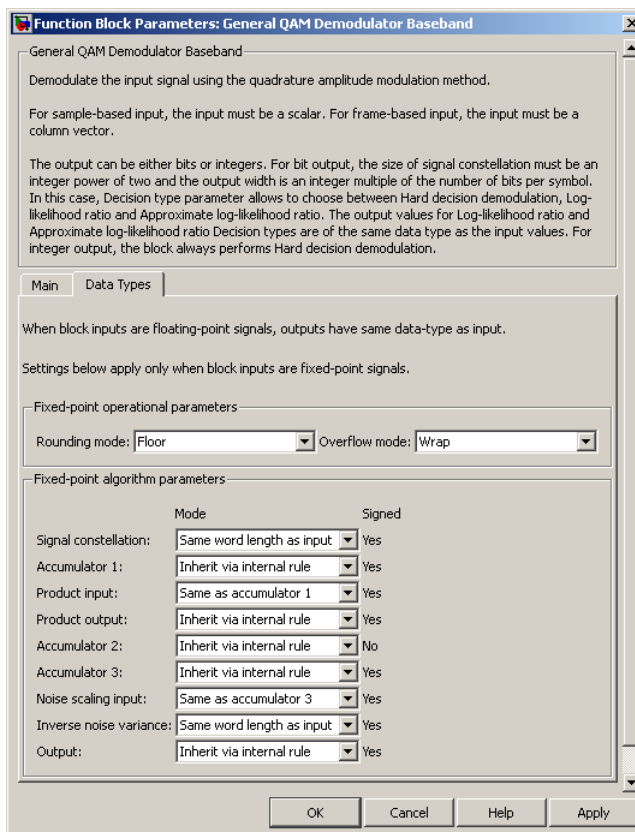
$$FL_{\text{ideal accumulator 2}} = FL_{\text{input to accumulator 2}}$$

After the full-precision result is calculated, your particular hardware may still affect the final word and fraction lengths set by the internal rule. For more information, see The Effect of the Hardware Implementation Pane on the Internal Rule subsection of the *Signal Processing Blockset User's Guide*.

The internal rule always sets the sign of data-type to `Unsigned`.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of **Accumulator 2**, in bits.

Fixed-Point Attributes for Approximate LLR Mode

The settings for the following fixed-point parameters only apply when you set **Decision type** to `Approximate log-likelihood ratio`.

Accumulator 3

When you select `Inherit via internal rule`, the block automatically calculates the accumulator data type. The internal rule first calculates ideal, full-precision word length and fraction length as follows:

$$WL_{\text{ideal accumulator 3}} = WL_{\text{input to accumulator 3}} + 1$$

$$FL_{\text{ideal accumulator 3}} = FL_{\text{input to accumulator 3}}.$$

After the full-precision result is calculated, your particular hardware may still affect the final word and fraction lengths set by the internal rule. For more information, see The Effect of the Hardware Implementation Pane on the Internal Rule subsection of the *Signal Processing Blockset User's Guide*.

The internal rule always sets the sign of data-type to `Signed`.

Noise scaling input

- When you select `Same as accumulator 3`, the **Noise scaling input** characteristics match those of **Accumulator 3**.

- When you select `Binary point scaling` you are able to enter the word length and the fraction length of **Noise scaling input**, in bits.

Inverse noise variance

This field appears when **Noise variance** source is set to Dialog.

- When you select `Same word length as input` the word length of the **Inverse noise variance** parameter matches that of the input to the block. The fraction length is computed to provide the best precision for a given inverse noise variance value.

- When you select `Specify word length`, the **Word Length** field appears, and you may enter a value for the word length. The fraction length is computed to provide the best precision for a given inverse noise variance value.

Output

> When you select `Inherit via internal rule`, the **Output data type** is automatically set for you.
>
> If you set the **Noise variance source** parameter to `Dialog`, the output is a result of product operation as shown in the Noise Variance Operation Modes Signal Flow DiagramFixed-Point Signal Flow Diagram for Approximate LLR Mode: Noise Variance Operation Modes on page 2-266. In this case, it follows the internal rule for Product data types specified in the Inherit via Internal Rule subsection of the *Signal Processing Blockset User's Guide*.
>
> If the **Noise variance source** parameter is set to `Port`, the output is a result of division operation as shown in the signal flow diagram. In this case, the internal rule calculates the ideal, full-precision word length and fraction length as follows:
>
> $$WL_{\text{output}} = \max(WL_{\text{Noise scaling input}}, WL_{\text{Noise variance}})$$
>
> $$FL_{\text{output}} = FL_{\text{Noise scaling input (dividend)}} - FL_{\text{Noise variance (divisor)}} \cdot$$
>
> After the full-precision result is calculated, your particular hardware may still affect the final word and fraction lengths set by the internal rule. For more information, see The Effect of the Hardware Implementation Pane on the Internal Rule subsection of the *Signal Processing Blockset User's Guide*.
>
> The internal rule for **Output** always sets the sign of data-type to `Signed`.

For additional information about the parameters pertaining to fixed-point applications, see Specifying Fixed-Point Attributes.

Pair Block   General QAM Modulator Baseband

See Also   Rectangular QAM Demodulator Baseband

General QAM Modulator Baseband

Purpose   Modulate using quadrature amplitude modulation

Library   AM, in Digital Baseband sublibrary of Modulation

Description The General QAM Modulator Baseband block modulates using quadrature amplitude modulation. The output is a baseband representation of the modulated signal.

The **Signal constellation** parameter defines the constellation by listing its points in a length-M vector of complex numbers. The input signal values must be integers between 0 and M-1. The block maps an input integer m to the (m+1)st value in the **Signal constellation** vector.

The input can be either a scalar or a frame-based column vector. For integer inputs, the block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. For bit inputs, the block can accept `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`.

Constellation Visualization

The General QAM Modulator Baseband block provides the capability to visualize a signal constellation from the block mask. This Constellation Visualization feature allows you to visualize a signal constellation for specific block parameters. For more information, see the Constellation Visualization section of the *Communications Blockset User's Guide*.

Dialog Box
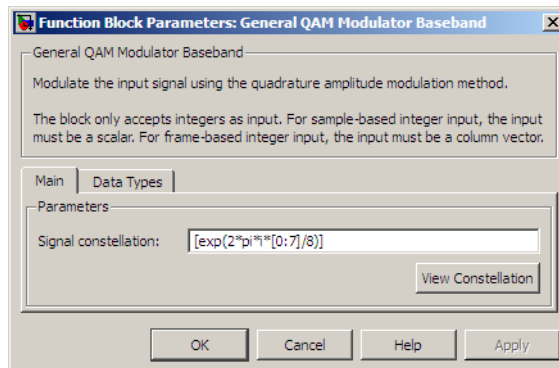
Signal constellation

A real or complex vector that lists the constellation points.

Output data type

The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Setting this to `Fixed-point` or `User-defined` will enable fields in which you can further specify details. Setting this to `Inherit via back propagation`, sets the output data type and scaling to match the following block..

Output word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type

Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point software. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

General QAM Modulator Baseband

Set output fraction length to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Output fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   General QAM Demodulator Baseband

See Also   Rectangular QAM Modulator Baseband
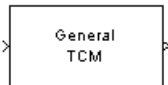
Purpose Decode trellis-coded modulation data, mapped using arbitrary constellation

Library Trellis-Coded Modulation

Description

The General TCM Decoder block uses the Viterbi algorithm to decode a trellis-coded modulation (TCM) signal that was previously modulated using an arbitrary signal constellation.

The **Trellis structure** and **Signal constellation** parameters in this block should match those in theGeneral TCM Encoder block, to ensure proper decoding. In particular, the **Signal constellation** parameter must be in set-partitioned order.

Input and Output Signals

The input signal must be a frame-based column vector containing complex numbers.

If the convolutional encoder described by the trellis structure represents a rate k/n code, then the General TCM Decoder block's output is a frame-based binary column vector whose length is k times the vector length of the input signal.

The input signal must be double or single. The reset port accepts double or boolean.

Operation Modes

The block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses. This parameter also affects the range of possible values for the **Traceback depth** parameter, D.

- In Continuous mode, the block initializes all state metrics to zero at the beginning of the simulation, waits until it accumulates D symbols, and then uses a sequence of D symbols to compute each of the traceback paths. D can be any positive integer. At the end of each frame, the block saves its internal state metric for use with the next frame.

If you select the **Enable the reset input port** check box, the block displays another input port, labeled Rst. This port receives an integer scalar signal. Whenever the value at the Rst port is nonzero, the block resets all state metrics to zero and sets the traceback memory to zero.
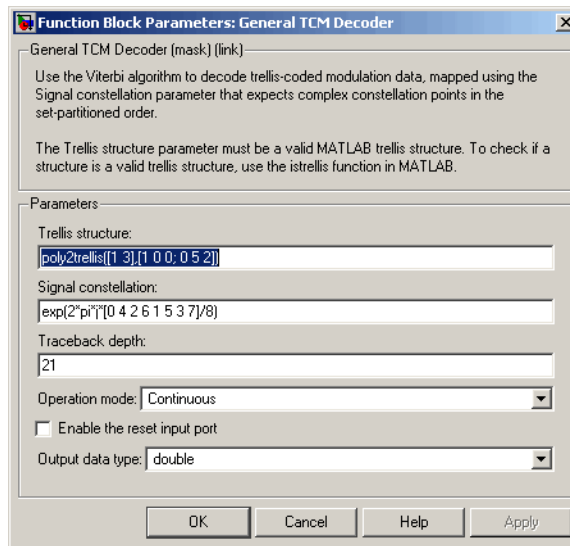
- In Truncated mode, the block treats each frame independently. The traceback path starts at the state with the lowest metric. D must be less than or equal to the vector length of the input.

- In Terminated mode, the block treats each frame independently. The traceback path always starts at the all-zeros state. D must be less than or equal to the vector length of the input. If you know that each frame of data typically ends at the all-zeros state, then this mode is an appropriate choice.

Decoding Delay

If you set **Operation mode** to Continuous, then this block introduces a decoding delay equal to **Traceback depth***k bits for a rate k/n convolutional code. The decoding delay is the number of zeros that precede the first decoded bit in the output.

The block incurs no delay for other values of **Operation mode**.

Dialog Box

Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

Signal constellation

A complex vector that lists the points in the signal constellation in set-partitioned order.

Traceback depth

The number of trellis branches (equivalently, the number of symbols) the block uses in the Viterbi algorithm to construct each traceback path.

Operation mode

The operation mode of the Viterbi decoder. The choices are Continuous, Truncated, and Terminated.

Enable the reset input port

When you check this box, the block has a second input port labeled Rst. Providing a nonzero value to this port causes the block to set its internal memory to the initial state before processing the

input data. This field appears only if you set **Operation mode** to `Continuous`.

Output data type
The output type of the block can be specified as a `boolean` or `double`. By default, the block sets this to `double`.

Pair Block   General TCM Encoder

See Also   M-PSK TCM Decoder, Rectangular QAM TCM Decoder, `poly2trellis`

References   [1] Biglieri, E., D. Divsalar, P. J. McLane, and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

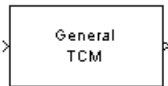[2] Proakis, John G., Digital Communications, Fourth edition, New York, McGraw-Hill, 2001.

Purpose   Convolutionally encode binary data and map using arbitrary constellation

Library   Trellis-Coded Modulation

Description   The General TCM Encoder block implements trellis-coded modulation (TCM) by convolutionally encoding the binary input signal and mapping the result to an arbitrary signal constellation. The points in the signal constellation are listed in set-partitioned order in the **Signal constellation** parameter. This parameter is a complex vector whose length, M, equals the number of possible output symbols from the convolutional encoder. (That is, $\log_2 M$ is equal to n for a rate k/n convolutional code.)

Input and Output Signals

If the convolutional encoder represents a rate k/n code, then the General TCM Encoder block's input must be a frame-based binary column vector whose length is L*k for some positive integer L.

The output from the General TCM Encoder block is a frame-based complex column vector of length L.

The input signal must be a binary value.

Specifying the Encoder

To define the convolutional encoder, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in the section "Trellis Description of a Convolutional Encoder" in the Communications Toolbox documentation. You can use this parameter field in two ways:

• If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, then use a `poly2trellis` command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7,[171 133],171)
```

- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is faster because it causes Simulink software to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the previous bulleted item.

The encoder registers begin in the all-zeros state. You can configure the encoder so that it resets its registers to the all-zeros state during the course of the simulation. To do this, set the `Operation mode` to **Reset on nonzero input via port**. The block then opens a second input port, labeled `Rst`. The signal at the `Rst` port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.
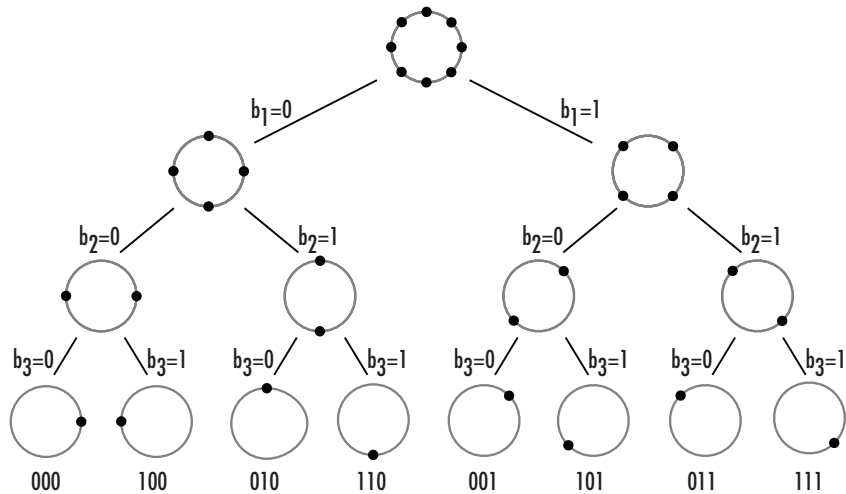
Signal Constellations

The trellis-coded modulation technique partitions the constellation into subsets called cosets so as to maximize the minimum distance between pairs of points in each coset.

Note When you set the **Signal constellation** parameter, you must ensure that the constellation vector is already in set-partitioned order. Otherwise, the block might produce unexpected or suboptimal results.

As an example, the diagram below shows one way to devise a set-partitioned order for the points for an 8-PSK signal constellation. The figure at the top of the tree is the entire 8-PSK signal constellation, while the eight figures at the bottom of the tree contain one constellation point each. Each level of the tree corresponds to a different bit in a binary sequence (b_3, b_2, b_1), while each branch in a given level of the tree corresponds to a particular value for that bit. Listing the constellation points using the sequence at the bottom of the tree leads to the vector

```
exp(2*pi*j*[0 4 2 6 1 5 3 7]/8)
```

which is a valid value for the **Signal constellation** parameter in this block.
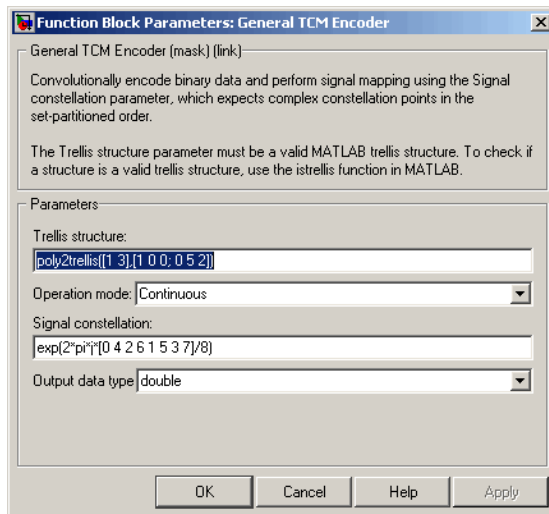

For other examples of signal constellations in set-partitioned order, see [1] or the reference pages for theM-PSK TCM Encoder andRectangular QAM TCM Encoder blocks.

Coding Gains

Coding gains of 3 to 6 decibels, relative to the uncoded case can be achieved in the presence of AWGN with multiphase trellis codes [3].

General TCM Encoder

Dialog Box

Function Block Parameters: General TCM Encoder

─ General TCM Encoder (mask) (link) ─

Convolutionally encode binary data and perform signal mapping using the Signal constellation parameter, which expects complex constellation points in the set-partitioned order.

The Trellis structure parameter must be a valid MATLAB trellis structure. To check if a structure is a valid trellis structure, use the istrellis function in MATLAB.

─ Parameters ─

Trellis structure:

poly2trellis([1 3],[1 0 0; 0 5 2])

Operation mode: Continuous

Signal constellation:

exp(2*pi*j*[0 4 2 6 1 5 3 7]/8)

Output data type: double

OK   Cancel   Help   Apply

Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

Operation mode

In Continuous mode (default setting), the block retains the encoder states at the end of each frame, for use with the next frame.

In Truncated (reset every frame) mode, the block treats each frame independently. I.e., the encoder states are reset to all-zeros state at the start of each frame.

In Terminate trellis by appending bits mode, the block treats each frame independently. For each input frame, extra bits are used to set the encoder states to all-zeros state at the end of the frame. The output length is given by

$y = n \cdot (x + s)/k$, where x is the number of input bits, and

$s = \text{constraint length} - 1$ (or, in the case of multiple constraint

lengths, s =sum(ConstraintLength(i)-1)). This mode is supported for frame based inputs only.

In Reset on nonzero input via port mode, the block has an additional input port, labeled Rst. When the Rst input is nonzero, the encoder resets to the all-zeros state.

Signal constellation
A complex vector that lists the points in the signal constellation in set-partitioned order.

Output data type
The output type of the block can be specified as a single or double. By default, the block sets this to double.

Pair Block   General TCM Decoder

See Also   M-PSK TCM Encoder, Rectangular QAM TCM Encoder, poly2trellis

References   [1] Biglieri, E., D. Divsalar, P. J. McLane, and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

[2] Proakis, John G., Digital Communications, Fourth edition, New York, McGraw-Hill, 2001.

[3] Ungerboeck, G., "Channel Coding with Multilevel/Phase Signals", *IEEE Trans. on Information Theory*, Vol IT28, Jan. 1982, pp. 55–67.

GMSK Demodulator Baseband

Purpose   Demodulate GMSK-modulated data

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The GMSK Demodulator Baseband block demodulates a signal that
was modulated using the Gaussian minimum shift keying method. The
input is a baseband representation of the modulated signal.

The **BT product**, **Pulse length**, **Symbol prehistory**, and **Phase
offset** parameters are as described on the reference page for theGMSK
Modulator Baseband block.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation
scheme and uses the Viterbi algorithm. The **Traceback length**
parameter, D, in this block is the number of trellis branches used
to construct each traceback path. D influences the output delay,
which is the number of zero symbols that precede the first meaningful
demodulated value in the output.

- If the input signal is sample-based, then the delay consists of D+1
  zero symbols.

- If the input signal is frame-based, then the delay consists of D zero
  symbols.

The optimal **Traceback length** parameter value choice is dependent
on minimum squared Euclidean distance calculations. Alternatively, a
typical value can be chosen using the "five-times-the-constraint-length"
rule which depends on the number of states given by:

$$numStates = \begin{cases} p*m^{(L-1)} < 0 \\ 2p*m^{(L-1)} > 0 \end{cases}$$

- h = m/p is the modulation index proper rational form

    - m = numerator of modulation index

- *p* = denominator of modulation index

- *L* is the Pulse length

- *M* is M-ary number

Inputs and Outputs

The input can be either a scalar or a frame-based column vector and must be of type `single` or `double`. If the **Output type** parameter is set to `Integer`, then the block produces values of 1 and -1. If the **Output type** parameter is set to `Bit`, then the block produces values of 0 and 1.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

GMSK Demodulator Baseband

Dialog Box
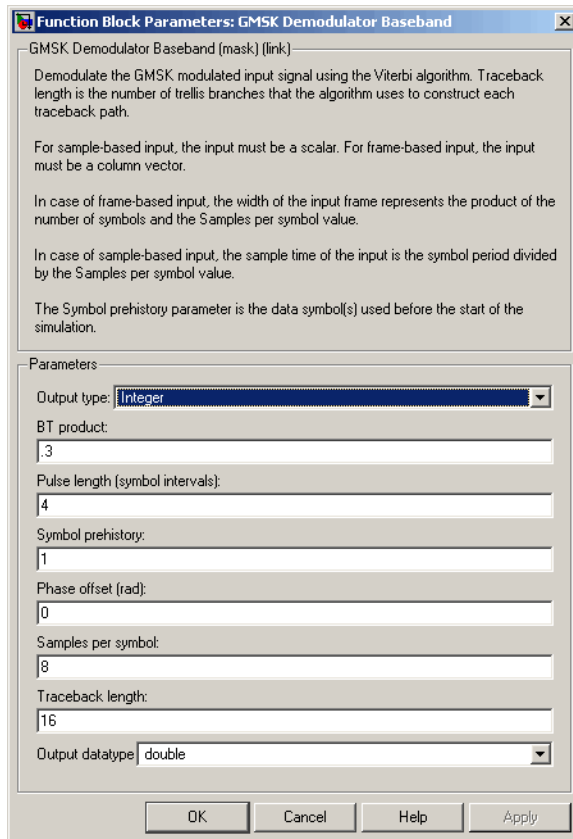
Output type

Determines whether the output consists of bipolar or binary values.

BT product

The product of bandwidth and time.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the GMSK Demodulator Baseband block uses to construct each traceback path.

Output data type

The output data type can be `boolean`, `int8`, `int16`, `int32`, or `double`.

Pair Block   GMSK Modulator Baseband

See Also   CPM Demodulator Baseband, Viterbi Decoder

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

GMSK Modulator Baseband

Purpose   Modulate using Gaussian minimum shift keying method

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The GMSK Modulator Baseband block modulates using the Gaussian minimum shift keying method. The output is a baseband representation of the modulated signal.

The **BT product** parameter represents bandwidth multiplied by time. This parameter is a nonnegative scalar. It is used to reduce the bandwidth at the expense of increased intersymbol interference. The **Pulse length** parameter measures the length of the Gaussian pulse shape, in symbol intervals. For an explanation of the pulse shape, see the work by Anderson, Aulin, and Sundberg among the references listed below. The frequency pulse shape is defined by the following equations.

$$g(t) = \frac{1}{2T}\left\{ Q\left[2\pi B_b \frac{t - \frac{T}{2}}{\sqrt{\ln(2)}} \right] - Q\left[2\pi B_b \frac{t + \frac{T}{2}}{\sqrt{\ln(2)}} \right] \right\}$$

$$Q(t) = \int_t^\infty \frac{1}{\sqrt{2\pi}} e^{-r^2/2} dt$$

The **Symbol prehistory** parameter is a scalar or vector that specifies the data symbols used before the start of the simulation, in reverse chronological order. If it is a vector, then its length must be one less than the **Pulse length** parameter.

In this block, a symbol of 1 causes a phase shift of π/2 radians. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

Input Attributes

The input can be either a scalar or a frame-based column vector. If the **Input type** parameter is set to Integer, then the block accepts values

of 1 and -1. If the **Input type** parameter is set to `Bit`, then the block accepts values of 0 and 1.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

Dialog Box
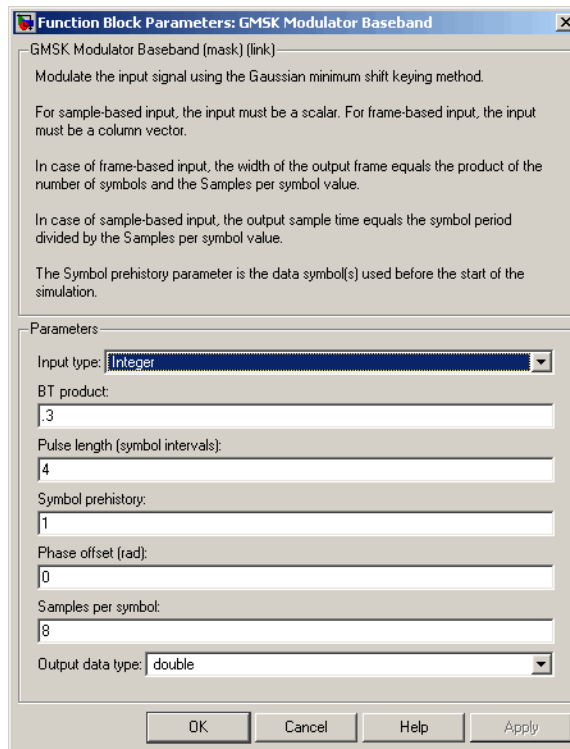
Input type
   Indicates whether the input consists of bipolar or binary values.

GMSK Modulator Baseband

BT product
> The product of bandwidth and time.

Pulse length (symbol intervals)
> The length of the frequency pulse shape.

Symbol prehistory
> The data symbols used before the start of the simulation, in reverse chronological order.

Phase offset (rad)
> The initial phase of the output waveform.

Samples per symbol
> The number of output samples that the block produces for each integer or bit in the input.

Output data type
> The output type of the block can be specified as a `single` or `double`. By default, the block sets this to `double`.

Pair Block   GMSK Demodulator Baseband

See Also   CPM Modulator Baseband

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose   Generate Gold sequence from set of sequences

Library   Sequence Generators sublibrary of Comm Sources
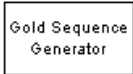
Description The Gold Sequence Generator block generates a Gold sequence. Gold
sequences form a large class of sequences that have good periodic
cross-correlation properties.

Gold Sequence
Generator

The Gold sequences are defined using a specified pair of sequences
u and v, of period $N = 2^n - 1$, called a *preferred pair*, as defined in
"Preferred Pairs of Sequences" on page 2-296 below. The set $G(u, v)$ of
Gold sequences is defined by

$$G(u,v) = \{u, v, u \oplus v, u \oplus Tv, u \oplus T^2 v, ..., u \oplus T^{N-1} v\}$$

where T represents the operator that shifts vectors cyclically to the
left by one place, and $\oplus$ represents addition modulo 2. Note that
G(u,v) contains $N + 2$ sequences of period N. The Gold Sequence
Generator block outputs one of these sequences according to the block's
parameters.

Gold sequences have the property that the cross-correlation between
any two, or between shifted versions of them, takes on one of three
values: $-t(n)$, -1, or $t(n) - 2$, where

$$t(n) = \begin{cases} 1 + 2^{(n+1)/2} & n \text{ even} \\ 1 + 2^{(n+2)/2} & n \text{ odd} \end{cases}$$

The Gold Sequence Generator block uses two PN Sequence Generator
blocks to generate the preferred pair of sequences, and then XORs these
sequences to produce the output sequence, as shown in the following
diagram.

Gold Sequence Generator
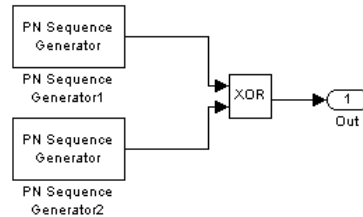
You can specify the preferred pair by the **Preferred polynomial [1]** and **Preferred polynomial [2]** parameters in the dialog for the Gold Sequence Generator block. These polynomials, both of which must have degree *n*, describe the shift registers that the PN Sequence Generator blocks use to generate their output. For more details on how these sequences are generated, see the reference page for the PN Sequence Generator block. You can specify the preferred polynomials using either of the following formats:

- A vector that lists the coefficients of the polynomial in descending order of powers. The first and last entries must be 1. Note that the length of this vector is one more than the degree of the generator polynomial.

- A vector containing the exponents of *z* for the nonzero terms of the polynomial in descending order of powers. The last entry must be 0.

For example, the vectors [5 2 0] and [1 0 0 1 0 1] both represent the polynomial $z^5 + z^2 + 1$.

The following table provides a short list of preferred pairs.

n	N	Preferred Polynomial[1]	Preferred Polynomial[2]
5	31	[5 2 0]	[5 4 3 2 0]
6	63	[6 1 0]	[6 5 2 1 0]
7	127	[7 3 0]	[7 3 2 1 0]

n	N	Preferred Polynomial[1]	Preferred Polynomial[2]
9	511	[9 4 0]	[9 6 4 3 0]
10	1023	[10 3 0]	[10 8 3 2 0]
11	2047	[11 2 0]	[11 8 5 2 0]

The **Initial states[1]** and **Initial states[2]** parameters are vectors specifying the initial values of the registers corresponding to **Preferred polynomial [1]** and **Preferred polynomial [2]**, respectively. These parameters must satisfy these criteria:

- All elements of the **Initial states[1]** and **Initial states[2]** vectors must be binary numbers.

- The length of the **Initial states[1]** vector must equal the degree of the **Preferred polynomial[1]**, and the length of the **Initial states[2]** vector must equal the degree of the **Preferred polynomial[2]**.

Note At least one element of the **Initial states** vectors must be nonzero in order for the block to generate a nonzero sequence. That is, the initial state of at least one of the registers must be nonzero.

The **Sequence index** parameter specifies which sequence in the set $G(u, v)$ of Gold sequences the block outputs. The range of **Sequence index** is [-2, -1, 0, 1, 2, ..., 2^n-2]. The correspondence between **Sequence index** and the output sequence is given in the following table.

Sequence Index	Output Sequence
-2	u
-1	v
0	$u \oplus v$

Gold Sequence Generator

Sequence Index	Output Sequence
1	$u \oplus Tv$
2	$u \oplus T^2v$
...	...
2^n-2	$u \oplus T^{2^n-2}v$

You can shift the starting point of the Gold sequence with the **Shift** parameter, which is an integer representing the length of the shift.

You can use an external signal to reset the values of the internal shift register to the initial state by selecting the **Reset on nonzero input** check box. This creates an input port for the external signal in the Gold Sequence Generator block. The way the block resets the internal shift register depends on whether its output signal and the reset signal are sample-based or frame-based. The following example demonstrates the possible alternatives. See "Example: Resetting a Signal" on page 2-574 for an example.

Preferred Pairs of Sequences

The requirements for a pair of sequences u, v of period $N = 2^n–1$ to be a preferred pair are as follows:

- n is not divisible by 4
- v = u[q], where
  - q is odd
  - q = 2^k+1 or q = $2^{2k}–2^k$+1
  - v is obtained by sampling every qth symbol of u

- $\gcd(n,k) = \begin{cases} 1 & n \equiv 1 \bmod 2 \\ 2 & n \equiv 2 \bmod 4 \end{cases}$

Dialog Box

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Gold Sequence Generator

Preferred polynomial[1]

Vector specifying the polynomial for the first sequence of the preferred pair.

Initial states[1]

Vector of initial states of the shift register for the first sequence of the preferred pair.

Preferred polynomial[2]

Vector specifying the polynomial for the second sequence of the preferred pair.

Initial states[2]

Vector of initial states of the shift register for the second sequence of the preferred pair.

Sequence index

Integer specifying the index of the output sequence from the set of sequences.

Shift

Integer scalar that determines the offset of the Gold sequence from the initial time.

Sample time

Period of each element of the output signal.

Frame-based outputs

Determines whether the output is frame-based or sample-based.

Samples per frame

The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box.

Reset on nonzero input

When selected, you can specify an input signal that resets the internal shift registers to the original values of the **Initial states** parameter

Output data type

The output type of the block can be specified as `boolean`, `double` or `Smallest unsigned integer`. By default, the block sets this to `double`.

When the parameter is set to `Smallest unsigned integer`, the output data type is selected based on the settings used in the Hardware Implementation pane of the Configuration Parameters dialog box of the model. If ASIC/FPGA is selected in the Hardware Implementation pane, the output data type is the ideal minimum one-bit size, i.e., ufix(1). For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., uint8).

**See Also**   Kasami Sequence Generator, PN Sequence Generator

References   [1] Proakis, John G., *Digital Communications*, Third edition, New York, McGraw Hill, 1995.

[2] Gold, R., "Maximal Recursive Sequences with 3-valued Recursive Cross-Correlation Functions," *IEEE Trans. Infor. Theory*, Jan., 1968, pp. 154-156.

[3] Gold, R., "Optimal Binary Sequences for Spread Spectrum Multiplexing, *IEEE Trans. Infor. Theory*, Oct., 1967, pp. 619-621.

[4] Sarwate, D.V., and M.B. Pursley, "Crosscorrelation Properties of Pseudorandom and Related Sequences," *Proc. IEEE*, Vol. 68, No. 5, May, 1980, pp. 583-619.

[5] Dixon, Robert, *Spread Spectrum Systems with Commercial Applications*, Third Edition, Wiley–Interscience, 1994.

Hadamard Code Generator

Purpose   Generate Hadamard code from orthogonal set of codes

Library   Sequence Generators sublibrary of Comm Sources
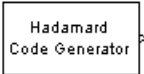
Description   The Hadamard Code Generator block generates a Hadamard code
                     from a Hadamard matrix, whose rows form an orthogonal set of codes.
                     Orthogonal codes can be used for spreading in communication systems
                     in which the receiver is perfectly synchronized with the transmitter.
                     In these systems, the despreading operation is ideal, as the codes are
                     decorrelated completely.

Hadamard
Code Generator

The Hadamard codes are the individual rows of a Hadamard matrix.
Hadamard matrices are square matrices whose entries are +1 or -1, and
whose rows and columns are mutually orthogonal. If N is a nonnegative
power of 2, the N-by-N Hadamard matrix, denoted H_N, is defined
recursively as follows.

$$H_1 = \begin{bmatrix} 1 \end{bmatrix}$$
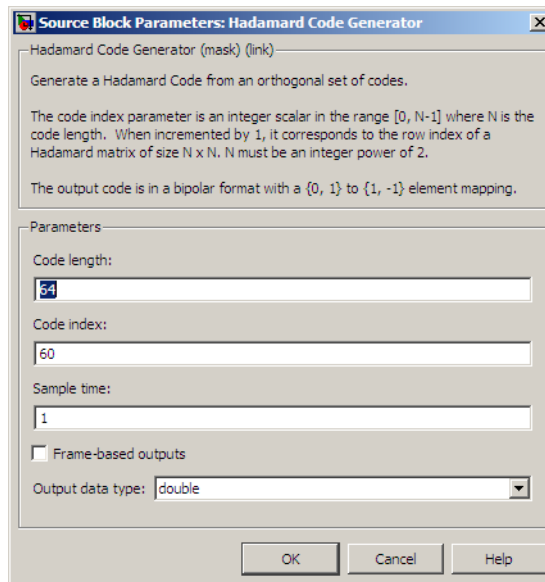$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

The N-by-N Hadamard matrix has the property that

$$H_N H_N{}^T = N I_N$$

where I_N is the N-by-N identity matrix.

The Hadamard Code Generator block outputs a row of H_N. The output
is bipolar. You specify the length of the code, N, by the **Code length**
parameter. The **Code length** must be a power of 2. You specify the
index of the row of the Hadamard matrix, which is an integer in the
range [0, 1, ... , N-1], by the **Code index** parameter.

**Dialog
Box**

Source Block Parameters: Hadamard Code Generator

Hadamard Code Generator (mask) (link)

Generate a Hadamard Code from an orthogonal set of codes.

The code index parameter is an integer scalar in the range [0, N-1] where N is the code length. When incremented by 1, it corresponds to the row index of a Hadamard matrix of size N x N. N must be an integer power of 2.

The output code is in a bipolar format with a {0, 1} to {1, -1} element mapping.

Parameters

Code length:

64

Code index:

60

Sample time:

1

☐ Frame-based outputs

Output data type: double

OK   Cancel   Help

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Code length

A positive integer that is a power of two specifying the length of the Hadamard code.

Code index

An integer between 0 and N-1, where N is the **Code length**, specifying a row of the Hadamard matrix.

Sample time

A positive real scalar specifying the sample time of the output signal.

Frame-based outputs

Determines whether the output is frame-based or sample-based.

Hadamard Code Generator

Samples per frame

The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box.

Output data type

The output type of the block can be specified as an `int8` or `double`. By default, the block sets this to `double`.

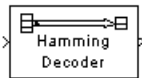**See Also**   OVSF Code Generator, Walsh Code Generator

Purpose   Decode Hamming code to recover binary vector data

Library   Block sublibrary of Channel Coding

Description   The Hamming Decoder block recovers a binary message vector from a
binary Hamming codeword vector. For proper decoding, the parameter
values in this block should match those in the corresponding Hamming
Encoder block.

If the Hamming code has message length K and codeword length N,
then N must have the form 2^M-1 for some integer M greater than or
equal to 3. Also, K must equal N-M.

The input must contain exactly N elements. If it is frame-based, then it
must be a column vector. The output is a vector of length K.

The coding scheme uses elements of the finite field $GF(2^M)$. You can
either specify the primitive polynomial that the algorithm should use,
or you can rely on the default setting:

- To use the default primitive polynomial, simply enter N and K as the
  first and second dialog parameters, respectively. The algorithm uses
  `gfprimdf(M)` as the primitive polynomial for $GF(2^M)$.

- To specify the primitive polynomial, enter N as the first parameter
  and a binary vector as the second parameter. The vector represents
  the primitive polynomial by listing its coefficients in order of
  ascending exponents. You can create primitive polynomials using the
  Communications Toolbox `gfprimfd` function.

This block supports `double` and `boolean` data types.

Hamming Decoder

**Dialog
Box**
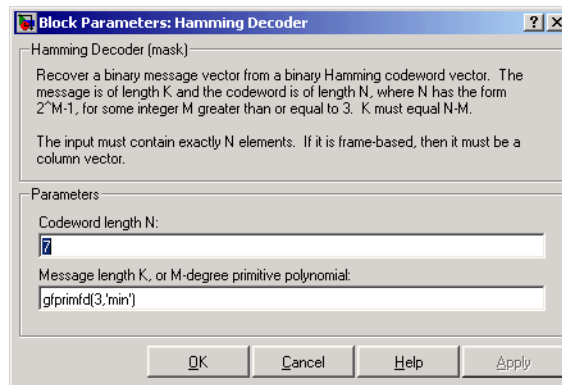
Codeword length N
> The codeword length N, which is also the input vector length.

Message length K, or M-degree primitive polynomial
> Either the message length, which is also the output vector length;
> or a binary vector that represents a primitive polynomial for
> $GF(2^M)$.

Pair Block   Hamming Encoder

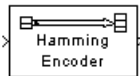**See Also**   hammgen (Communications Toolbox)

Purpose   Create Hamming code from binary vector data

Library   Block sublibrary of Channel Coding

Description   The Hamming Encoder block creates a Hamming code with message length K and codeword length N. The number N must have the form 2^M-1, where M is an integer greater than or equal to 3. Then K equals N-M.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N.

The coding scheme uses elements of the finite field $GF(2^M)$. You can either specify the primitive polynomial that the algorithm should use, or you can rely on the default setting:

- To use the default primitive polynomial, simply enter N and K as the first and second dialog parameters, respectively. The algorithm uses `gfprimdf(M)` as the primitive polynomial for $GF(2^M)$.

- To specify the primitive polynomial, enter N as the first parameter and a binary vector as the second parameter. The vector represents the primitive polynomial by listing its coefficients in order of ascending exponents. You can create primitive polynomials using the Communications Toolbox `gfprimfd` function.

This block supports `double` and `boolean` data types.

Hamming Encoder

Dialog Box
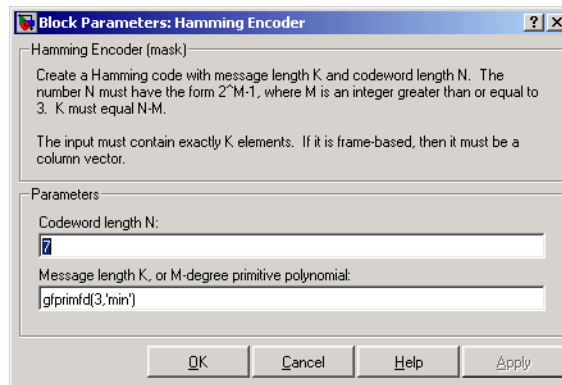
Codeword length N

The codeword length, which is also the output vector length.

Message length K, or M-degree primitive polynomial

Either the message length, which is also the input vector length; or a binary vector that represents a primitive polynomial for $GF(2^M)$.

Pair Block   Hamming Decoder

See Also   hammgen (Communications Toolbox)

Purpose   Restore ordering of symbols permuted by helical interleaver

Library   Convolutional sublibrary of Interleaving

Description   The Helical Deinterleaver block permutes the symbols in the input signal by placing them in an array row by row and then selecting groups in a helical fashion to send to the output port.

> Helical
> Deinterleaver

The block uses the array internally for its computations. If C is the **Number of columns in helical array** parameter, then the array has C columns and unlimited rows. If N is the **Group size** parameter, then the block accepts an input of length C*N at each time step and inserts them into the next N rows of the array. The block also places the **Initial condition** parameter into certain positions in the top few rows of the array (not only to accommodate the helical pattern but also to preserve the vector indices of symbols that pass through the Helical Interleaver and Helical Deinterleaver blocks in turn).

The output consists of consecutive groups of N symbols. Counting from the beginning of the simulation, the block selects the kth output group in the array from column k mod C. The selection is helical because of the reduction modulo C and because the first symbol in the kth group is in row 1+(k-1)*s, where s is the **Helical array step size** parameter.

The number of elements of the input vector must be C times N. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Delay of Interleaver-Deinterleaver Pair

After processing a message with the Helical Interleaver block and the Helical Deinterleaver block, the deinterleaved data lags the original message by

$$CN \left\lceil \frac{s(C-1)}{N} \right\rceil$$

samples. Before this delay elapses, the deinterleaver output is either the **Initial condition** parameter in the Helical Deinterleaver block or the **Initial condition** parameter in the Helical Interleaver block.

If your model incurs an additional delay between the interleaver output and the deinterleaver input, then the restored sequence lags the original sequence by the sum of the additional delay and the amount in the formula above. For proper synchronization, the delay between the interleaver and deinterleaver must be m*C*N for some nonnegative integer m. You can use the Signal Processing Blockset Delay block to adjust delays manually, if necessary.

Dialog Box
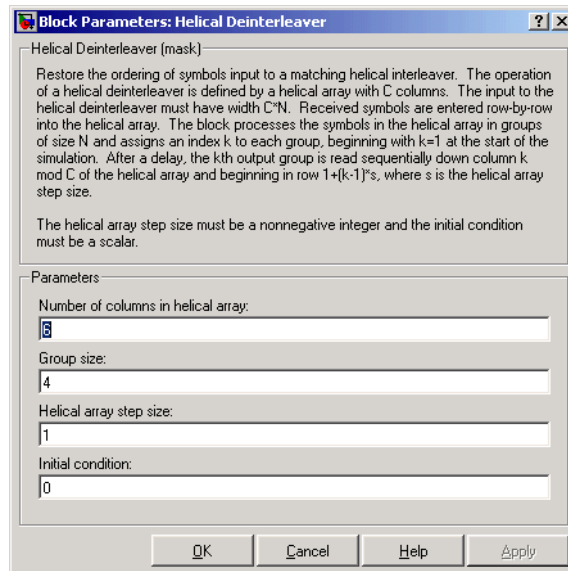
Number of columns in helical array

The number of columns, C, in the helical array.

Group size

The size, N, of each group of symbols. The input width is C times N.

Helical array step size
> The number of rows of separation between consecutive output groups as the block selects them from their respective columns of the helical array.

Initial condition
> A scalar that fills the array before the first input is placed.

Pair Block   Helical Interleaver

See Also   General Multiplexed Deinterleaver

References   [1] Berlekamp, E. R. and P. Tong. "Improved Interleavers for Algebraic Block Codes." U. S. Patent 4559625, Dec. 17, 1985.

Helical Interleaver

Purpose   Permute input symbols using helical array

Library   Convolutional sublibrary of Interleaving

Description   The Helical Interleaver block permutes the symbols in the input signal by placing them in an array in a helical fashion and then sending rows of the array to the output port.

The block uses the array internally for its computations. If C is the **Number of columns in helical array** parameter, then the array has C columns and unlimited rows. If N is the **Group size** parameter, then the block accepts an input of length C*N at each time step and partitions the input into consecutive groups of N symbols. Counting from the beginning of the simulation, the block places the kth group in the array along column k mod C. The placement is helical because of the reduction modulo C and because the first symbol in the kth group is in row 1+(k-1)*s, where s is the **Helical array step size** parameter. Positions in the array that do not contain input symbols have default contents specified by the **Initial condition** parameter.

The block sends C*N symbols from the array to the output port by reading the next N rows sequentially. At a given time step, the output symbols might be the **Initial condition** parameter value, symbols from that time step's input vector, or symbols left in the array from a previous time step.

The number of elements of the input vector must be C times N. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box
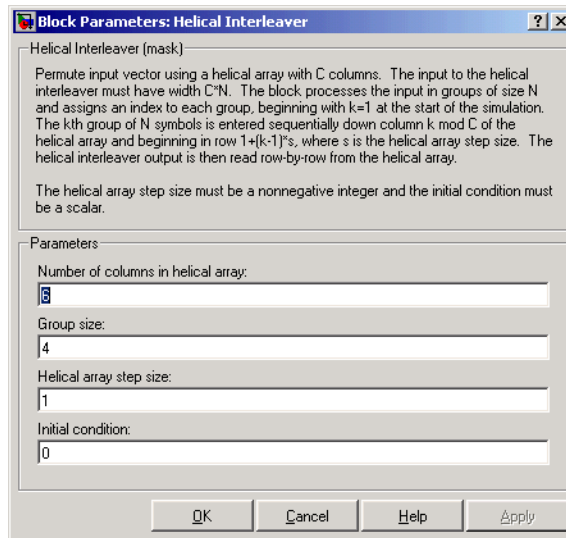
Number of columns in helical array

The number of columns, C, in the helical array.

Group size

The size, N, of each group of input symbols. The input width is C times N.

Helical array step size

The number of rows of separation between consecutive input groups in their respective columns of the helical array.

Initial condition

A scalar that fills the array before the first input is placed.

Examples   Suppose that C = 3, N = 2, the **Helical array step size** parameter is 1, and the **Initial condition** parameter is -1. After receiving inputs of [1:6]', [7:12]', and [13:18]', the block's internal array looks like the schematic below. The coloring of the inputs and the array indicate how the input symbols are placed within the array. The outputs at the first three time steps are [1; -1; -1; 2; 3; -1], [7; 4; 5; 8; 9; 6],

and [13; 10; 11; 14; 15; 12]. (The outputs are not color-coded in the schematic.)

Inputs		**Block's Internal Array**	**Outputs from successive rows of array**
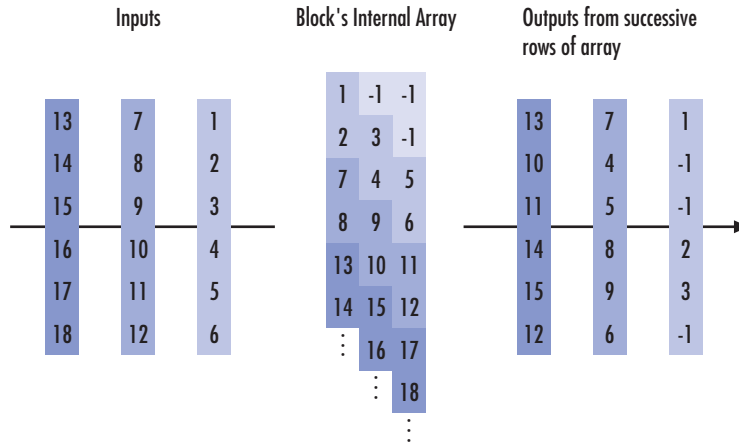
Pair Block   Helical Deinterleaver

See Also   General Multiplexed Interleaver

References   [1] Berlekamp, E. R. and P. Tong. "Improved Interleavers for Algebraic Block Codes." U. S. Patent 4559625, Dec. 17, 1985.
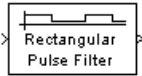
Purpose   Shape input signal using ideal rectangular pulses

Library   Comm Filters

Description   The Ideal Rectangular Pulse Filter block upsamples and shapes the input signal using rectangular pulses. The block replicates each input sample N times, where N is the **Pulse length** parameter. After replicating input samples, the block can also normalize the output signal and/or apply a linear amplitude gain.

If the **Pulse delay** parameter is nonzero, then the block outputs that number of zeros at the beginning of the simulation, before starting to replicate any of the input values.

Inputs and Outputs

The input can be either a scalar or a frame-based column vector. `double`, `single`, and fixed-point data types are supported.

- If the input is sample-based, then the output sample time is 1/N times the input sample time. The output dimensions match the input dimensions. You must set the **Input sampling mode** parameter to `Sample-based`.

- If the input is a frame-based k-by-1 matrix, then the output is a frame-based k*N-by-1 matrix. The output frame period matches the input frame period. You must set the **Input sampling mode** parameter to `Frame-based`.

The vector size (in frame-based mode), the pulse length, and the pulse delay are mutually independent. They do not need to satisfy any conditions with respect to each other.

Normalization Methods

You determine the block's normalization behavior using the **Normalize output signal** and **Linear amplitude gain** parameters.

Ideal Rectangular Pulse Filter

- If you clear the **Normalize output signal** check box, then the block multiplies the set of replicated values by the **Linear amplitude gain** parameter. This parameter must be a scalar.

- If you select **Normalize output signal**, then the **Normalization method** parameter appears. The block scales the set of replicated values so that one of these conditions is true:

  - The sum of the samples in each pulse equals the original input value that the block replicated.

  - The energy in each pulse equals the energy of the original input value that the block replicated. That is, the sum of the squared samples in each pulse equals the square of the input value.

After the block applies the scaling specified in the **Normalization method** parameter, it multiplies the scaled signal by the constant scalar value specified in the **Linear amplitude gain** parameter.

The output is scaled by $\sqrt{N}$. If the output of this block feeds the input to the AWGN Channel block, specify the AWGN signal power parameter to be 1/N.

Dialog Box
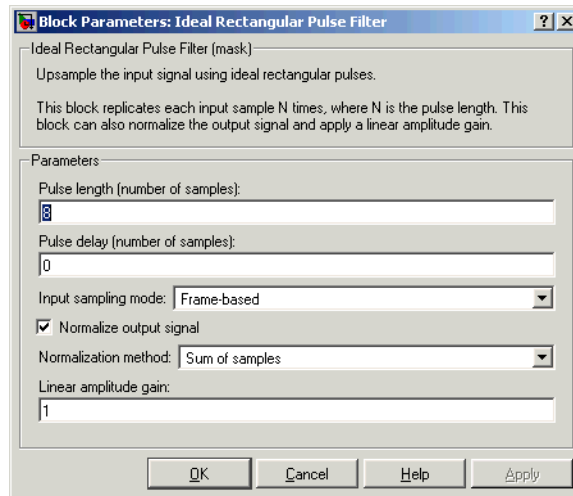
Pulse length

> The number of samples in each output pulse; that is, the number of times the block replicates each input value when creating the output signal.

Pulse delay

> The number of zeros that appear in the output at the beginning of the simulation, before the block replicates any input values.

Input sampling mode

> The type of input signal: `Frame-based` or `Sample-based`.

Normalize output signal

> If you select this, then the block scales the set of replicated values before applying the linear amplitude gain.

Normalization method

> The quantity that the block considers when scaling the set of replicated values. Choices are `Sum of samples` and `Energy per pulse`. This field appears only if you select **Normalize output signal**.

Linear amplitude gain

> A positive scalar used to scale the output signal.

Ideal Rectangular Pulse Filter

Rounding mode

Use this parameter to specify the rounding method to be used when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. The filter coefficients do not obey this parameter; they always round to Nearest.

For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator). See "Filter

Structure Diagrams" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you are able to enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the coefficients, in bits. If applicable, you are able to enter separate fraction lengths for the numerator and denominator coefficients.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the coefficients. If applicable, you are able to enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.

- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool.

Examples   If **Pulse length** is 4 and **Pulse delay** is the scalar 3, then the table below shows how the block treats the beginning of a ramp (1, 2, 3,...) in several situations. (The values shown in the table do not reflect vector sizes but merely indicate numerical values.)

Normalization Method, If Any	Linear Amplitude Gain	First Several Output Values
None (**Normalize output signal** cleared)	1	0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3,...
None (**Normalize output signal** cleared)	10	0, 0, 0, 10, 10, 10, 10, 20, 20, 20, 20, 30, 30, 30, 30,...
Sum of samples	1	0, 0, 0, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.75, 0.75, 0.75, 0.75,..., where 0.25*4=1
Sum of samples	10	0, 0, 0, 2.5, 2.5, 2.5, 2.5, 5, 5, 5, 5, 7.5, 7.5, 7.5, 7.5,...

Ideal Rectangular Pulse Filter

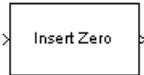| Normalization Method, If Any	Linear Amplitude Gain	First Several Output Values
Energy per pulse	1	0, 0, 0, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.5, 1.5, 1.5, 1.5,..., where (0.5)^2*4=1^2
Energy per pulse	10	0, 0, 0, 5, 5, 5, 5, 10, 10, 10, 10, 15, 15, 15, 15,...

See Also   Upsample, Integrate and Dump

Purpose   Distribute input elements in output vector

Library   Sequence Operations

Description   The Insert Zero block constructs an output vector by inserting zeros among the elements of the input vector. The input can be real or complex. The block determines where to place the zeros by using the **Insert zero vector** parameter. The **Insert zero vector** parameter is a binary vector whose elements are arranged so that:

> Insert Zero

- Each 1 indicates that the block should place the *next* element of the input in the output vector

- Each 0 indicates that the block should place a 0 in the output vector

If the input signal is sample-based, then the input vector length must equal the number of 1s in the **Insert zero vector** parameter.

The block can accept the data types int8, uint8, int16, uint16, int32, uint32, boolean, single, double, and fixed-point. The data type of this output will be the same as that of the input signal.
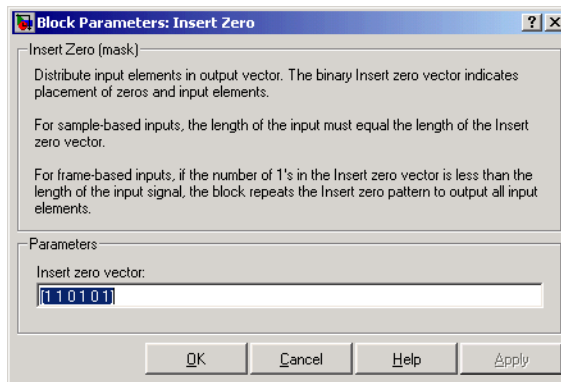
To implement punctured coding using the Puncture and Insert Zero blocks, you should use the same vector for the **Insert zero vector** parameter in this block and for the **Puncture vector** parameter in the Puncture block.

Frame-Based Processing

If the input signal is frame-based, then both it and the **Insert zero vector** parameter must be column vectors. The number of 1s in the **Insert zero vector** parameter must divide the input vector length. If the input vector length is greater than the number of 1s in the **Insert zero vector** parameter, then the block repeats the insertion pattern until it has placed all input elements in the output vector.
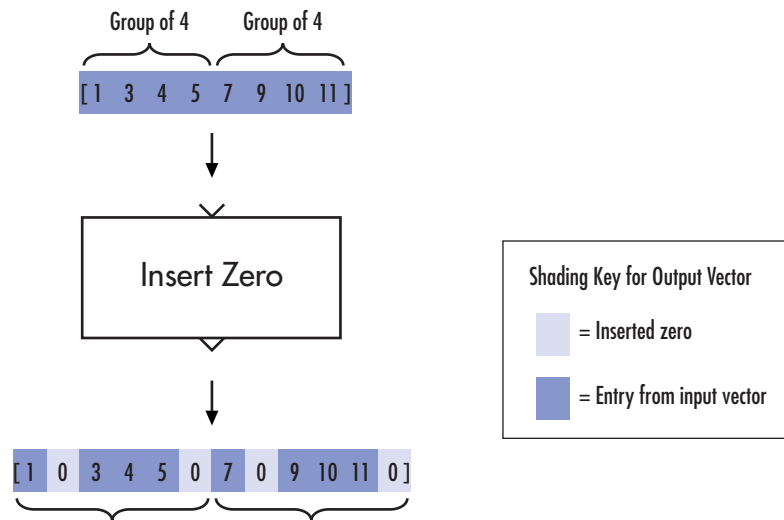
Insert Zero

Dialog Box

Block Parameters: Insert Zero

Insert Zero (mask)

Distribute input elements in output vector. The binary Insert zero vector indicates placement of zeros and input elements.

For sample-based inputs, the length of the input must equal the length of the Insert zero vector.

For frame-based inputs, if the number of 1's in the Insert zero vector is less than the length of the input signal, the block repeats the Insert zero pattern to output all input elements.

Parameters

Insert zero vector:

[1 1 0 1 0 1]

OK   Cancel   Help   Apply

Insert zero vector
> A binary vector whose pattern of 0s and 1s indicates where the block should place either 0s or input vector elements, respectively, in the output vector.

Examples   If the **Insert zero vector** parameter is the six-element vector [1,0,1,1,1,0], then the block inserts zeros after the first and last elements of each consecutive grouping of four input elements. It considers groups of four elements because the **Insert zero vector** parameter has four 1s.

The diagram below depicts the block's operation using this **Insert zero vector** parameter. Notice that the insertion pattern applies twice.

<o=

Group of 4   Group of 4

[1 3 4 5 7 9 10 11]

Insert Zero

Shading Key for Output Vector

= Inserted zero

= Entry from input vector

[1 0 3 4 5 0 7 0 9 10 11 0]

Compare this example with that on the reference page for the Puncture block.
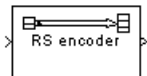
See Also   Puncture

Integer-Input RS Encoder

Purpose   Create Reed-Solomon code from integer vector data

Library   Block sublibrary of Channel Coding

Description This block supports punctures ("Shortening, Puncturing, and Erasures" provides a tutorial).

Note For a puncture vector, a 1 means that the data symbol is passed unaltered, and a 0 means that the data symbol is punctured (i.e., removed) from the data stream. This convention is carried for both the encoder and the decoder.

The Integer-Input RS Encoder block creates a Reed-Solomon code with message length, K, and codeword length, (N - number of punctures). You specify both N and K directly in the block dialog. The symbols for the code are integers between 0 and 2^M-1, which represent elements of the finite field GF(2^M). Restrictions on M and N are described in "Restrictions on M and the Codeword Length N" on page 2-325 below. The difference N - K must be an even integer.

This block can output shortened codewords when N and K are appropriately specified. To specify output codewords that are shortened by a length S, N and K must be specified in the dialog box as $N_{full} - S$ and $K_{full} - S$, where N_{full} and K_{full} are the N and K of an unshortened code. If $S < (N_{full} + 1)/2$, the encoder can automatically determine the value of N_{full} and K_{full}. However, if $S \geq (N_{full} + 1)/2$, **Primitive polynomial** must be specified in order to properly define the extension field for the code.

The input and output are integer-valued signals that represent messages and codewords, respectively. The input must be a frame-based column vector whose length is an integer multiple of K. The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. The output is a frame-based column vector whose length is the same integer multiple of N, and whose data type is

inherited from the input. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon Only)" in *Communications Blockset User's Guide*.

If the encoder is processing multiple codewords per frame, then the same puncture pattern holds for all codewords.

The default value of M is the smallest integer that is greater than or equal to log2(N+1), that is, `ceil(log2(N+1))`. You can change the value of M from the default by specifying the primitive polynomial for GF(2^M), as described in "Specifying the Primitive Polynomial" on page 2-325 below. If N is less than 2^M-1, the block uses a shortened Reed-Solomon code.

An (N, K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword.

Specifying the Primitive Polynomial

You can specify the primitive polynomial that defines the finite field GF(2^M), corresponding to the integers that form messages and codewords. To do so, first select **Specify primitive polynomial**. Then, in the **Primitive polynomial** field, enter a binary row vector that represents a primitive polynomial over GF(2) of degree M, in descending order of powers. For example, to specify the polynomial x^3+x+1, enter the vector [1 0 1 1].

If you do not select **Specify primitive polynomial**, the block uses the default primitive polynomial of degree M = ceil(log2(N+1)). You can display the default polynomial by entering `primpoly(ceil(log2(N+1)))` at the MATLAB prompt.

Restrictions on M and the Codeword Length N

The restrictions on the degree M of the primitive polynomial and the codeword length N are as follows:

- If you do not select **Specify primitive polynomial**, N must lie in the range $3 < N < 2^{16}-1$.

- If you do select **Specify primitive polynomial**, N must lie in the range $3 \leq N < 2^M - 1$ and M must lie in the range $3 \leq M \leq 16$.

Specifying the Generator Polynomial

You can specify the generator polynomial for the Reed-Solomon code. To do so, first select **Specify generator polynomial**. Then, in the **Generator polynomial** field, enter an integer row vector whose entries are between 0 and 2^M-1. The vector represents a polynomial, in descending order of powers, whose coefficients are elements of $GF(2^M)$ represented in integer format. See the section "Integer Format (Reed-Solomon Only)" for more information about integer format. The generator polynomial must be equal to a polynomial with a factored form

$$g(x) = (x+A^b)(x+A^{b+1})(x+A^{b+2})...(x+A^{b+N-K-1})$$

where A is the primitive element of the Galois field over which the input message is defined, and *b* is an integer.

If you do not select **Specify generator polynomial**, the block uses the default generator polynomial, corresponding to *b*=1, for Reed-Solomon encoding. You can display the default generator polynomial by entering `rsgenpoly(N1,K1)`, where `N1 = 2^M-1` and `K1 = K+(N1-N)`, at the MATLAB prompt, if you are using the default primitive polynomial. If the **Specify primitive polynomial** box is selected, and you specify the primitive polynomial specified as `poly`, the default generator polynomial is `rsgenpoly(N1,K1,poly)`.
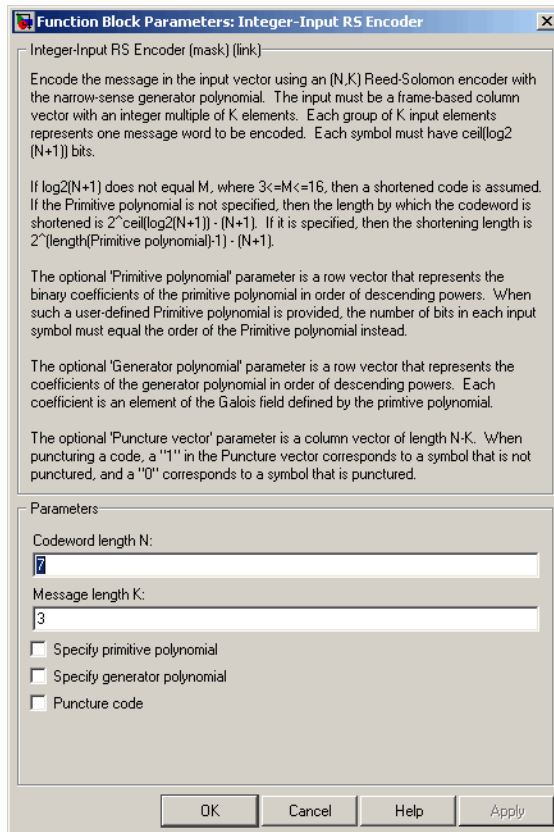
Examples   Suppose M = 3, N = 2^3-1 = 7, and K = 5. Then a message is a vector of length 5 whose entries are integers between 0 and 7. A corresponding codeword is a vector of length 7 whose entries are integers between 0 and 7. The following figure illustrates possible input and output signals to this block when **Codeword length N** is set to 7, **Message length K** is set to 5, and the default primitive and generator polynomials are used.
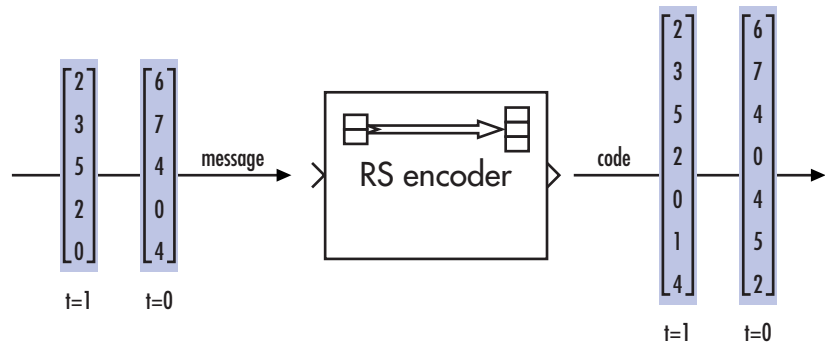
Integer-Input RS Encoder

Dialog Box

Codeword length N
> The codeword length.

Message length K
> The message length.

Specify primitive polynomial
> Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial

This field is available only when **Specify primitive polynomial** is selected.

Binary row vector representing the primitive polynomial in descending order of powers.

Specify generator polynomial

Selecting this check box enables the field **Generator polynomial**.

Generator polynomial

This field is available only when **Specify generator polynomial** is selected.

Integer row vector, whose entries are in the range from 0 to 2^M-1, representing the generator polynomial in descending order of powers.

Puncture code

Selecting this check box enables the field **Puncture vector**.

Puncture vector

This field is available only when **Puncture code** is selected.

A column vector of length *N-K*. A value of 1 in the **Puncture vector** corresponds to a symbol that is not punctured, and a 0 corresponds to a symbol that is punctured.

The default value is [ones(2,1); zeros(2,1)].

Pair Block   Integer-Output RS Decoder

See Also   Binary-Input RS Encoder

Integer-Output RS Decoder

Purpose   Decode Reed-Solomon code to recover integer vector data

Library   Block sublibrary of Channel Coding

Description   This block supports punctures and erasures ("Shortening, Puncturing, and Erasures" provides a tutorial).

Note 1s and 0s have precisely opposite meanings for the puncture and erasure vectors. For a puncture vector, a 1 means that the data symbol is passed unaltered, and a 0 means that the data symbol is punctured (i.e., removed) from the data stream. For an erasure vector, a 1 means that the data symbol is to be replaced with an erasure symbol, and a 0 means that the data symbol is passed unaltered. These conventions are carried for both the encoder and the decoder.

The Integer-Output RS Decoder block recovers a message vector from a Reed-Solomon codeword vector. For proper decoding, the parameter values in this block should match those in the corresponding Integer-Input RS Encoder block.

The Reed-Solomon code has message length, K, and codeword length, (N - number of punctures). You specify both N and K directly in the block dialog. The symbols for the code are integers between 0 and 2^M-1, which represent elements of the finite field GF(2^M). Restrictions on M and N are described in "Restrictions on M and the Codeword Length N" on page 2-325 below. The difference N - K must be an even integer.

This block can output shortened codewords when N and K are appropriately specified. To specify output codewords that are shortened by a length S, N and K must be specified in the dialog box as $N_{full} - S$ and $K_{full} - S$, where N_{full} and K_{full} are the N and K of an unshortened code. If $S < (N_{full} + 1)/2$, the encoder can automatically determine the value of N_{full} and K_{full}. However, if $S \geq (N_{full} + 1)/2$, **Primitive polynomial** must be specified in order to properly define the extension field for the code.

The input and output are integer-valued signals that represent codewords and messages, respectively. The input must be a frame-based column vector whose length is an integer multiple of (N - number of punctures). The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. The output is a frame-based column vector whose length is the same integer multiple of K, and whose data type is inherited from the input. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon Only)" in the *Communications Blockset User's Guide*.

If the decoder is processing multiple codewords per frame, then the same puncture pattern holds for all codewords.

The default value of M is `ceil(log2(N+1))`, that is, the smallest integer greater than or equal to log2(N+1). You can change the value of M from the default by specifying the primitive polynomial for $GF(2^M)$, as described in "Specifying the Primitive Polynomial" on page 2-325 below. If N is less than 2^M-1, the block uses a shortened Reed-Solomon code.

You can also specify the generator polynomial for the Reed-Solomon code, as described in "Specifying the Generator Polynomial" on page 2-326.

An (N, K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword.

The second output is the number of errors detected during decoding of the codeword. A -1 indicates that the block detected more errors than it could correct using the coding scheme. An (N,K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword. The data type of this output is also inherited from the input signal.
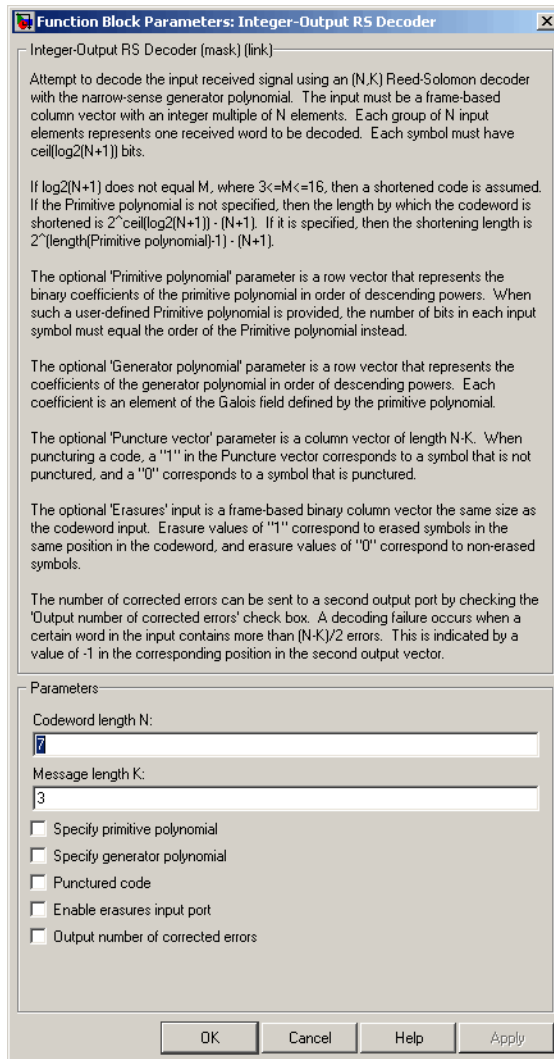
You can disable the second output by deselecting **Output number of corrected errors.** This removes the block's second output port.

In the case of a decoder failure, the message portion of the decoder input is returned unchanged as the decoder output.

The sample times of the input and output signals are equal.

Dialog Box

Codeword length N

The codeword length.

Message length K

The message length.

Specify primitive polynomial

Selecting this check box enables the field **Primitive polynomial**.

Primitive polynomial

This field is available only when **Specify primitive polynomial** is selected.

Binary row vector representing the primitive polynomial in descending order of powers.

Specify generator polynomial

Selecting this check box enables the field **Generator polynomial**.

Generator polynomial

This field is available only when **Specify generator polynomial** is selected.

Integer row vector, whose entries are in the range from 0 to 2^M-1, representing the generator polynomial in descending order of powers.

Puncture code

Selecting this check box enables the field **Puncture vector**.

Puncture vector

This field is available only when **Puncture code** is selected.

A column vector of length *N-K*. A value of 1 in the **Puncture vector** corresponds to a symbol that is not punctured, and a 0 corresponds to a symbol that is punctured.

The default value is `[ones(2,1); zeros(2,1)]`.

Enable erasures input port

Selecting this check box will open the port, `Era`.

Integer-Output RS Decoder

Through the port, you can input a frame-based binary column vector the same size as the codeword input.

Erasure values of 1 correspond to erased symbols in the same position in the codeword, and values of 0 correspond to nonerased symbols.

Output number of corrected errors
When you select this box, the block outputs the number of corrected errors in each word through a second output port.

Algorithm   This block uses the Berlekamp-Massey decoding algorithm. For information about this algorithm, see the references listed below.

Pair Block   Integer-Input RS Encoder

References   [1] Wicker, Stephen B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

[2] Berlekamp, Elwyn R., *Algebraic Coding Theory*, New York, McGraw-Hill, 1968.

[3] Clark, George C., Jr., and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

See Also   Binary-Output RS Decoder

Purpose   Map vector of integers to vector of bits

Library   Utility Blocks
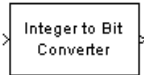
Description   The Integer to Bit Converter block maps each integer or fixed-point value in the input vector to a group of bits in the output vector.

Integer to Bit
Converter

The block maps each integer value (or stored integer when a fixed point input is used) to a group of *M* bits, using the selection for the **Output bit order** to determine the most significant bit. The resulting output vector length is *M* times the input vector length.

For integer-valued inputs, if *M* is the **Number of bits per integer** and **Treat input values as** is set to Unsigned, then the input values must be between 0 and 2^M-1. If *M* is the **Number of bits per integer** and **Treat input values as** is set to Signed, then the block maps each group of *M* bits to an integer between -2^{M-1} and 2^{M-1}-1.

This block is single-rate and single-channel. The input can be either a scalar or a frame-based column vector. For a sample-based scalar input, the output is a 1-D signal with *M* elements. For a frame-based column vector input, the output is a frame-based column vector with a length equal to *M* times the input signal length.

A simulation-only runtime input value check is performed. If any input value is outside of the **Number of bit per integer** value range, the block will issue an error.

The block can accept the data types int8, uint8, int16, uint16, int32, uint32, single, double, and fixed point.

Integer to Bit Converter
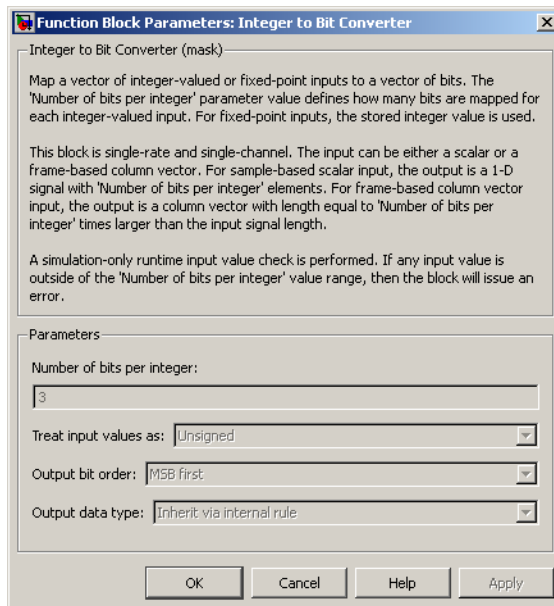
Number of bits per integer
:   The number of bits the block uses to represent each integer of the input. This parameter must be an integer between 1 and 32.

Treat input values as
:   Indicate if the integer value input ranges should be treated as signed or unsigned. The default setting is Unsigned.

Output bit order
:   Define whether the first bit of the output signal is the most significant bit (MSB) or the least significant bit (LSB). The default selection is MSB first.

Output data type
:   You can choose the following **Output data type** options:

    - Inherit via internal rule
    - Smallest integer

- `Same as input`

- `double`

- `single`

- `uint8`

- `uint16`

- `uint32`

The default selection for this parameter is `Inherit via internal rule`.

When the parameter is set to `Inherit via internal rule`, the block determines the output data type based on the input data type.

- If the input signal is floating-point (either `single` or `double`), the output data type is the same as the input data type.

- If the input data type is not floating-point, the output data type is determined as if the parameter is set to `Smallest integer`.

When the parameter is set to `Smallest integer`, the block selects the output data type based on settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box.

- If ASIC/FPRA is selected, the output data type is the ideal one-bit size; ufix1.

- For all other selections, the output data type is an unsigned integer with the smallest available word length, as defined in the Hardware Implementation settings (e.g. uint8)

Integer to Bit Converter

Examples
Fixed-Point Integer To Bit and Bit To Integer Conversion (Audio Scrambling and Descrambling Example)

Overview

This example illustrates how to use the Bit to Integer and Integer to Bit Converter blocks with fixed-point signals.

This example uses a simplified audio scrambler configuration and a 16-bit, fixed-point digital audio source, which is recorded speech. The left-side of the model represents the audio scrambler subsystem and the right-side represents the descrambler subsystem.

Opening the Model

You can open the model by clicking here.

Structure

In the audio scrambler subsystem, the Integer to Bit Converter block unpacks each 16-bit audio sample into a binary, 1-bit signal. The binary signal passes to a linear feedback shift register (LFSR) scrambler, which randomizes the bits in a controllable way, thereby scrambling the signal. The Communications Blockset Scrambler block is used in the LFSR implementation. From the LFSR, the scrambled audio bits pass to the Bit to Integer Converter block. This block packs the scrambled 1-bit samples into 16-bit audio samples. The audio samples pass to the Data Type Conversion block, which converts the integer-based audio samples back into fixed-point samples.

The fixed-point samples pass from the scrambler subsystem to a channel. The channel sends the samples to the descrambler subsystem. For illustrative purposes, this example uses a noiseless channel. In an actual system, a channel may introduce noise. Removing such noise requires a more sophisticated design.

In the audio descrambler subsystem, the Integer to Bit Converter block unpacks each 16-bit audio sample into a binary, 1-bit signal. The binary signal passes to a linear feedback shift register (LFSR) descrambler, which randomizes the bits in a controllable way, reversing the scrambling process. This LFSR descrambler implementation uses the Communications Blockset Descrambler block. From the LFSR, the descrambled audio bits pass to the Bit to Integer Converter block. This

block packs the descrambled 1-bit samples into 16-bit audio samples. The audio samples pass to the Data Type Conversion block, which converts the integer-based audio samples back into fixed-point samples.

In Simulink, the sfix16_En15 data type represents a signed (s) fixed-point (fix) signal with word length 16 and fraction length 15. Therefore, this model represents audio signals using the sfix16_En15 data type, except when converting to and from 1-bit binary signals. All 1-bit signals are represented by ufix1, as seen at the output of the Integer to Bit Converter block. The frame-based audio source has a frame size (or number of samples per frame) of 1024. For more information on fixed-point signals, please refer to Fixed-Point Numbers in the Simulink documentation.

Integer to Bit Converter

Fixed-Point Integer to Bit and Bit to Integer Conversion
(Audio Scrambler and Descrambler Example)
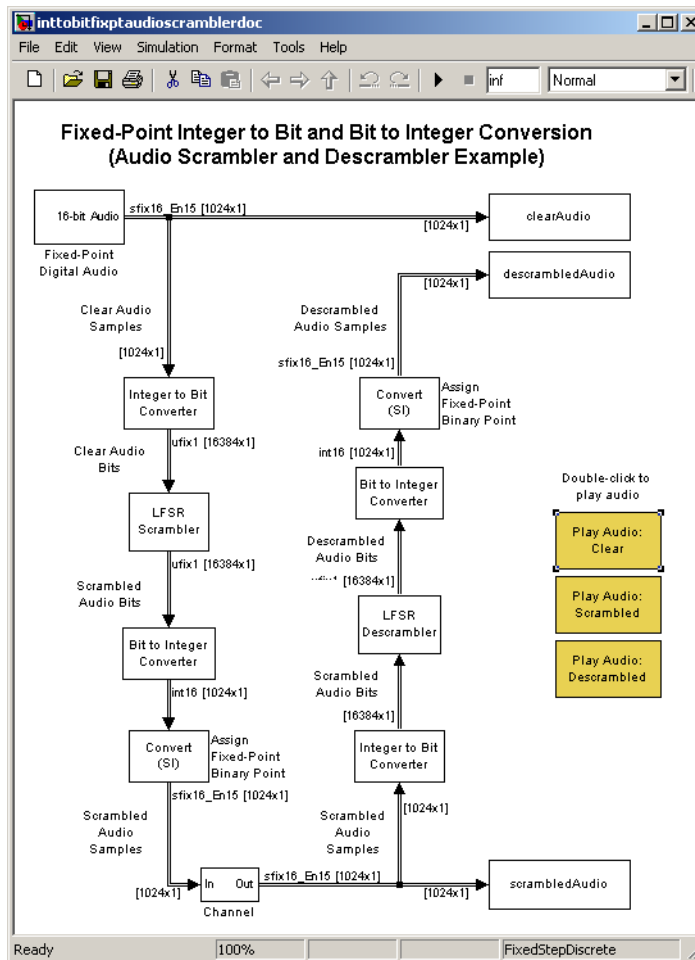
Running the Model

You must run the demo before you can listen to any of the audio signals.

You can run the demo by clicking **Simulation > Start**.

You can hear the audio signals by clicking the model's yellow, audio icons.

Converter Block Settings

In the audio scrambler and descrambler subsystems, the Integer to Bit Converter block settings are:

- **Number of bits per integer:** 16
- **Treat input values as:** Signed
- **Output bit order:** MSB first
- **Output data type:** Inherit via internal rule

In the audio scrambler and descrambler subsystems, the Bit to Integer Converter block settings are:

- **Number of bits per integer:** 16
- **Input bit order:** MSB first
- **After bit packing, treat resulting integer values as:** Signed
- **Output data type:** Inherit via internal rule

Pair Block   Bit to Integer Converter

See Also   de2bi and dec2bin

Integrate and Dump

Purpose   Integrate discrete-time signal, resetting to zero periodically

Library   Comm Filters

Description
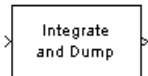
Integrate
and Dump

The Integrate and Dump block creates a cumulative sum of the
discrete-time input signal, while resetting the sum to zero according to
a fixed schedule. When the simulation begins, the block discards the
number of samples specified in the **Offset** parameter. After this initial
period, the block sums the input signal along columns and resets the
sum to zero every N input samples, where N is the **Integration period**
parameter value. The reset occurs after the block produces its output at
that time step.

This block supports `double`, `single`, and `fixed-point` input and output
signals. The port data types are inherited from the signals that drive
the block.

The integrate-and-dump operation is often used in a receiver model
when the system's transmitter uses a simple square-pulse model. It
can also be used in fiber optics and in spread-spectrum communication
systems such as CDMA (code division multiple access) applications.

The input can be either a scalar or a frame-based matrix. If the input is
frame-based, then it must have k*N rows for some positive integer k,
and the block processes each column independently.

The output contents, dimensions, and sample time are affected by the
Output intermediate values check box, as follows:

- If you clear the check box, then the block outputs the cumulative
  sum at each reset time.

  - If the input is sample-based, then the output sample time is N
    times the input sample time and the block experiences a delay
    whose duration is one output sample period. In this case, the
    output dimensions match the input dimensions.

  - If the input is a frame-based (k*N)-by-n matrix, then the output is
    k-by-n. In this case, the block experiences no delay and the output
    frame period matches the input frame period.

- If you select the check box, then the block outputs the cumulative sum at each time step, including the reset times. The output has the same sample time and the same matrix dimensions as the input.

This block will work within a triggered subsystem, as long as it is used in the single-rate mode.
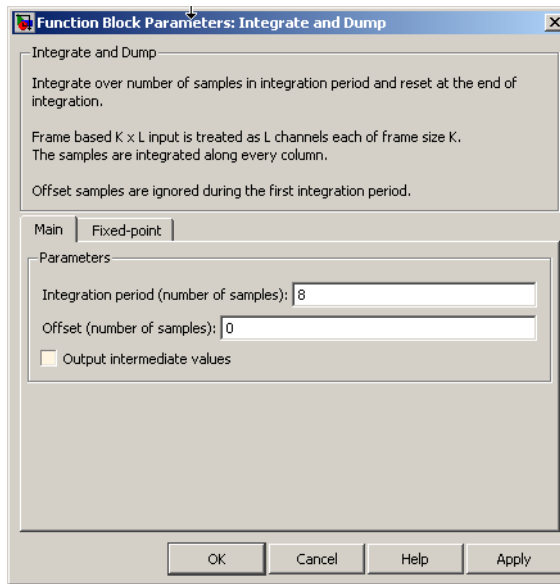
Transients and Delays

A nonzero value in the **Offset** parameter causes the block to output one or more zeros during the initial period while it discards input samples. If the input is a frame-based matrix with n columns and the **Offset** parameter is a length-n vector, then the mth element of the **Offset** vector is the offset for the mth column of data. If **Offset** is a scalar, then the block applies the same offset to each column of data. The output of initial zeros due to a nonzero **Offset** value is a transient effect, not a persistent delay.

When the **Output intermediate values** check box is cleared, the block's output is delayed, relative to its input, throughout the simulation:

- If the input is sample-based, then the output is delayed by one sample after any transient effect is over. That is, after removing transients from the input and output, you can see the result of the mth integration period in the output sample indexed by m+1.

- If the input is frame-based and the **Offset** parameter is nonzero, then after the transient effect is over, the result of each integration period appears in the output frame corresponding to the *last* input sample of that integration period. This is one frame later than the output frame corresponding to the first input sample of that integration period, in cases where an integration period spans two input frames. For an example of this situation, see "Example of Transient and Delay" on page 2-348.

Integrate and Dump

**Dialog
Box**

Function Block Parameters: Integrate and Dump

┌ Integrate and Dump ─────────────────────────────────────

Integrate over number of samples in integration period and reset at the end of integration.

Frame based K x L input is treated as L channels each of frame size K.
The samples are integrated along every column.

Offset samples are ignored during the first integration period.

Main │ Fixed-point

┌ Parameters ─────────────────────────────────────

Integration period (number of samples): 8

Offset (number of samples): 0

☐ Output intermediate values

OK   Cancel   Help   Apply

Integration period
> The number of input samples between resets.

Offset
> A nonnegative integer vector or scalar specifying the number of input samples to discard from each column of input data at the beginning of the simulation.

Output intermediate values
> Determines whether the block suppresses the intermediate cumulative sums between successive resets.

Fixed-Point Signal Flow Diagram
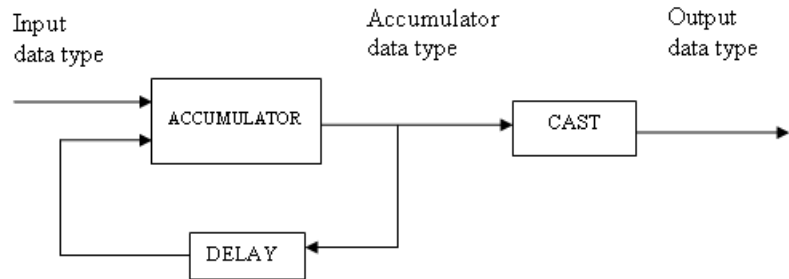
Fixed-Point Attributes
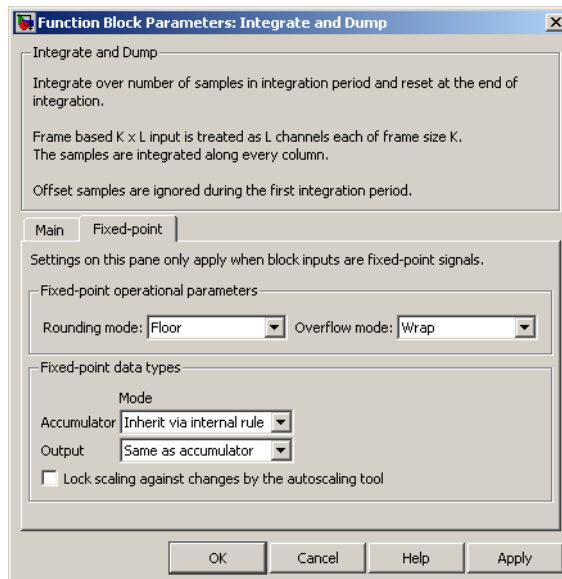
The settings for the following parameters only apply when block inputs are fixed-point signals.

Rounding mode

Use this parameter to specify the rounding method to be used when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result.

For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow mode

Use this parameter to specify the method to be used if the magnitude of a fixed-point calculation result does not fit into the range of the data type and scaling that stores the result:

- `Saturate` represents positive overflows as the largest positive number in the range being used, and negative overflows as the largest negative number in the range being used.

- `Wrap` uses modulo arithmetic to cast an overflow back into the representable range of the data type. See "Modulo Arithmetic" for more information.

Accumulator—Mode

Use the **Accumulator—Mode** parameter to specify how you would like to designate the accumulator word and fraction lengths:

- When you select `Inherit via internal rule`, the block automatically calculates the accumulator output word and fraction lengths.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator.

The bias of all signals in Signal Processing Blockset software is zero.

Output

Use the **Output** parameter to choose how you specify the word length and fraction length of the output of the block:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, enter the word length, in bits, and the slope of the output.

For additional information about the parameters pertaining to fixed-point applications, see Specifying Fixed-Point Attributes.

Examples   If **Integration period** is 4 and **Offset** is the scalar 3, then the table below shows how the block treats the beginning of a ramp (1, 2, 3, 4,...) in several situations. (The values shown in the table do not reflect vector sizes but merely indicate numerical values.)

Output intermediate values Check Box	Input Signal Properties	First Several Output Values
Cleared	Sample-based scalar	0, 0, 4+5+6+7, and 8+9+10+11, where one 0 is an initial transient value and the other 0 is a delay value that results from the cleared check box and sample-based input.

Integrate and Dump

Output intermediate values Check Box	Input Signal Properties	First Several Output Values
Cleared	Frame-based column vector of length 4	0, 4+5+6+7, and 8+9+10+11, where 0 is an initial delay value that results from the nonzero offset. The output is a frame-based scalar.
Selected	Sample-based scalar	0, 0, 0, 4, 4+5, 4+5+6, 4+5+6+7, 8, 8+9, 8+9+10, 8+9+10+11, and 12, where the three 0s are initial transient values.
Selected	Frame-based column vector of length 4	0, 0, 0, 4, 4+5, 4+5+6, 4+5+6+7, 8, 8+9, 8+9+10, 8+9+10+11, and 12, where the three 0s are initial transient values. The output is a frame-based column vector of length 4.

In all cases, the block discards the first three input samples (1, 2, and 3).

Example of Transient and Delay

The figure below illustrates a situation in which the block exhibits both a transient effect for three output samples, as well as a one-sample delay in alternate subsequent output samples for the rest of the simulation. The figure also indicates how the input and output values are organized as frame-based column vectors. In each vector in the figure, the last sample of each integration period is underlined, discarded input samples are white, and transient zeros in the output are white.

Integration period = 5

Offset = 13

Output intermediate values cleared Input signal
= Frame-based column vector of length 10
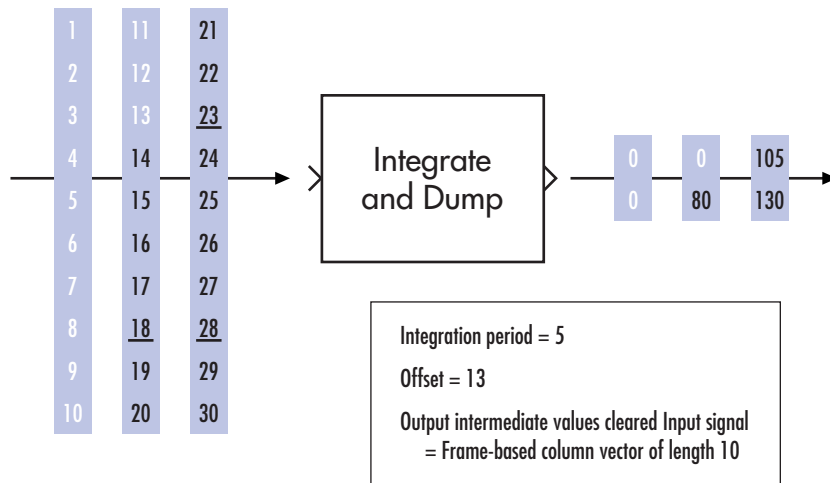
The transient effect lasts for `ceil(13/5)` output samples because the block discards 13 input samples and the integration period is 5. The first output sample after the transient effect is over, 80, corresponds to the sum 14+15+16+17+18 and appears at the time of the input sample 18. The next output sample, 105, corresponds to the sum 19+20+21+22+23 and appears at the time of the input sample 23. Notice that the input sample 23 is one frame later than the input sample 19; that is, this five-sample integration period spans two input frames. As a result, the output of 105 is delayed compared to the first input (19) that contributes to that sum.

See Also   Windowed Integrator, Discrete-Time Integrator (Simulink documentation), Ideal Rectangular Pulse Filter

Interlacer

Purpose   Alternately select elements from two input vectors to generate output vector

Library   Sequence Operations
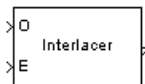
Description   The Interlacer block accepts two inputs that have the same vector size, complexity, and sample time. It produces one output vector by alternating elements from the first input (labeled O for odd) and from the second input (labeled E for even) . As a result, the output vector size is twice that of either input. The output vector has the same complexity and sample time of the inputs.

The inputs can be either scalars or frame-based column vectors. The block can accept the data types int8, uint8, int16, uint16, int32, uint32, boolean, single, double, and fixed-point. The data type of this output will be the same as that of the input signals.

This block can be useful for combining in-phase and quadrature information from separate vectors into a single vector.

Dialog Box
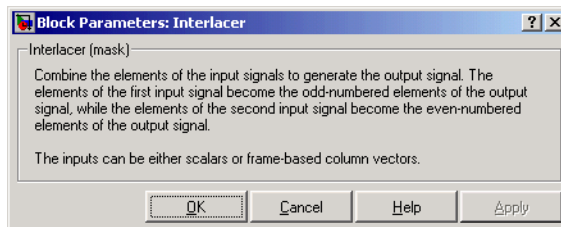
Examples   If the two input vectors are frame-based with values [1; 2; 3; 4] and [5; 6; 7; 8], then the output vector is [1; 5; 2; 6; 3; 7; 4; 8].

Pair Block   Deinterlacer

See Also   General Block Interleaver; Mux (Simulink documentation)

Purpose   Create complex baseband model of signal impairments caused by imbalances between in-phase and quadrature receiver components

Library   RF Impairments
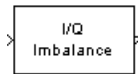
Description   The I/Q Imbalance block creates a complex baseband model of the signal impairments caused by imbalances between in-phase and quadrature receiver components. Typically, these are caused by differences in the physical channels for the two components of the signal.

```
 I/Q
Imbalance
```

The I/Q Imbalance block applies amplitude and phase imbalances to the in-phase and quadrature components of the input signal, and then combines the results into a complex signal. The block

1 Separates the signal into its in-phase and quadrature components.

2 Applies amplitude and phase imbalances, specified by the **I/Q amplitude imbalance (dB)** and **I/Q phase imbalance (deg)** parameters, respectively, to both components.

3 Combines the in-phase and quadrature components into a complex signal.

4 Applies an in-phase dc offset, specified by the **I dc offset** parameter, and a quadrature offset, specified by the **Q dc offset** parameter, to the signal.

The block performs these operations in the subsystem shown in the following diagram, which you can view by right-clicking the block and selecting **Look under mask**:

I/Q Imbalance
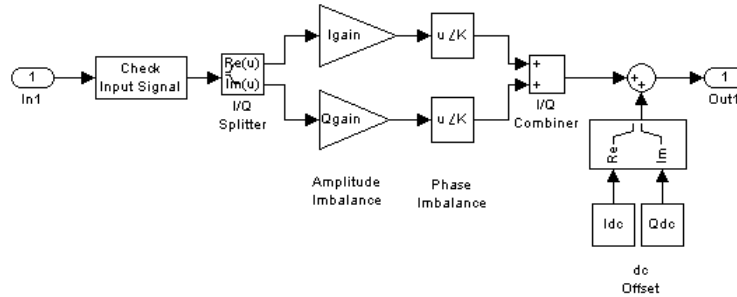
Let

I_a = I/Q amplitude imbalance

I_p = I/Q phase imbalance

I_{DC} = inphase DC offset

Q_{DC} = quadrature DC offset

Also let $x = x_r + x_i$ be the complex input to the block, with x_r and x_i being the real and imaginary parts, respectively, of x. Let y be the complex output of the block.

Then, for an I/Q amplitude imbalance, I_a

$$y_{AmplitudeImbalance} = [10^{(0.5*\frac{I_a}{20})} * x_r] + j[10^{(-0.5*\frac{I_a}{20})} * x_i] \triangleq y_{rAmplitudeImbalance} + j*y_{iAmplitudeImbalance}$$

For an I/Q phase imbalance I_p

$$y_{PhaseImbalance} =$$

$$[\exp(-0.5 * j * \pi * \frac{I_p}{180}) * y_{r_{AmplitudeImbalance}}] + j\{\exp[j(\frac{\pi}{2} + 0.5 * \pi * \frac{I_p}{180})] * y_{i_{AmplitudeImbalance}}\}$$
$$\triangleq y_{rPhaseImbalance} + j * y_{iPhaseImbalance}$$

For DC offsets I_{DC} and Q_{DC}

$$y_{PhaseImbalance} = y_{r\,PhaseImbalance} + j * y_{i\,PhaseImbalance}$$
$$y = y_{r\,PhaseImbalance} + I_{DC} + j * y_{iPhaseImbalance} + Q_{DC}$$

The value of the **I/Q amplitude imbalance (dB)** parameter is divided between the in-phase and quadrature components:

- If you enter a positive value *X* for the **I/Q amplitude imbalance (dB)**, the block applies a gain of +*X/2* dB to the in-phase component and a gain of -*X/2* dB to the quadrature component.

- If you enter a negative value *X* for the **I/Q amplitude imbalance (dB)**, the block applies a gain of -*X/2* dB to the in-phase component and a gain of +*X/2* dB to the quadrature component.

The effects of changing the block's parameters are illustrated by the following scatter plots of a signal modulated by 16-ary quadrature amplitude modulation (QAM) with an average power of 0.01 watts. The usual 16-ary QAM constellation without distortion is shown in the first scatter plot:

I/Q Imbalance
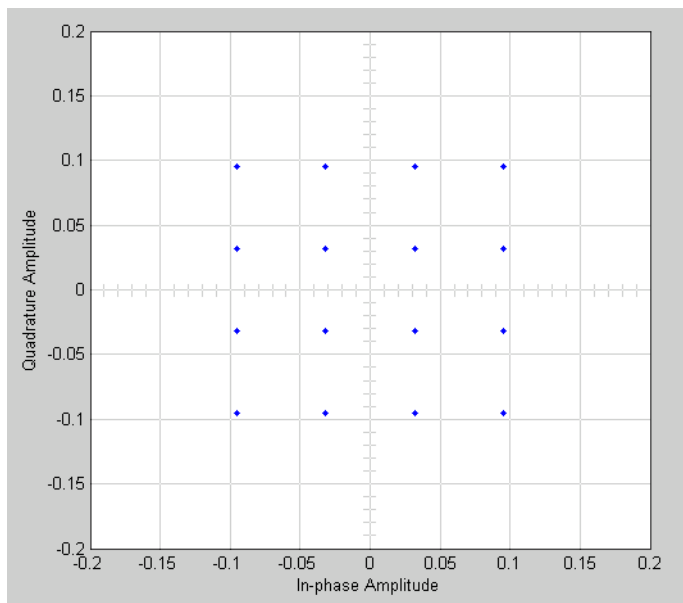
The following figure shows a scatter plot of an output signal, modulated by 16-ary QAM, from the I/Q block with **I/Q amplitude imbalance (dB)** set to 8 and all other parameters set to 0:
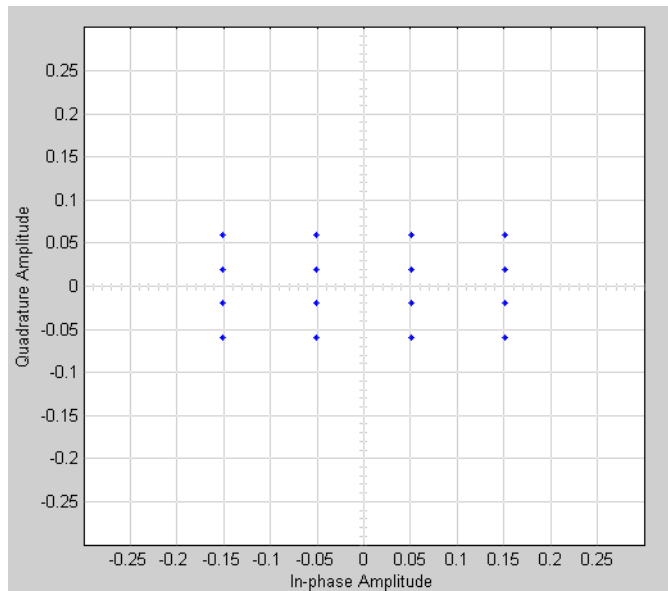
Observe that the scatter plot is stretched horizontally and compressed vertically compared to the undistorted constellation.

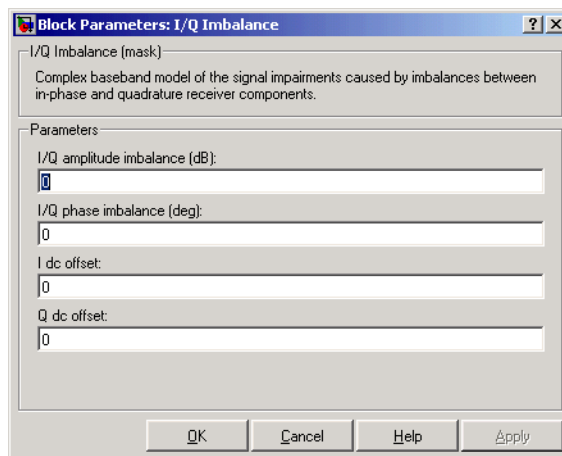If you set **IQ phase imbalance (deg)** to 30 and all other parameters to 0, the scatter plot is skewed clockwise by 30 degrees, as shown below:

I/Q Imbalance
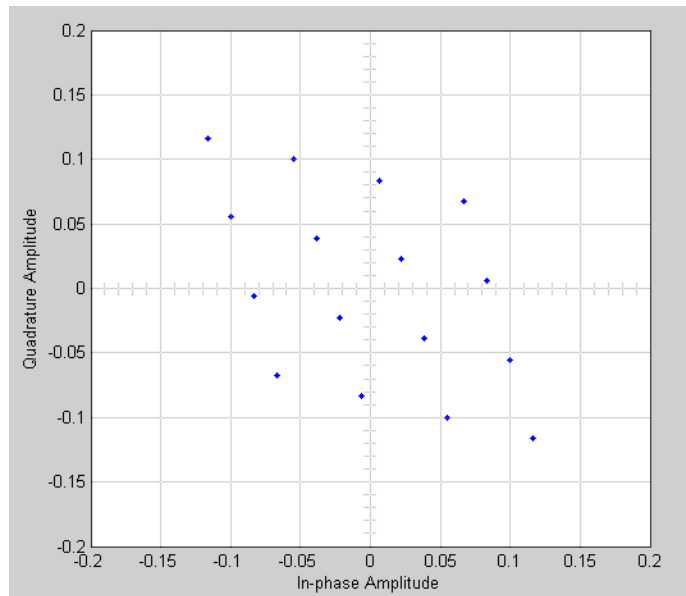
Setting the **I dc offset** to `0.02` and the **Q dc offset** to `0.04` shifts the constellation 0.02 to the right and 0.04 up, as shown below:
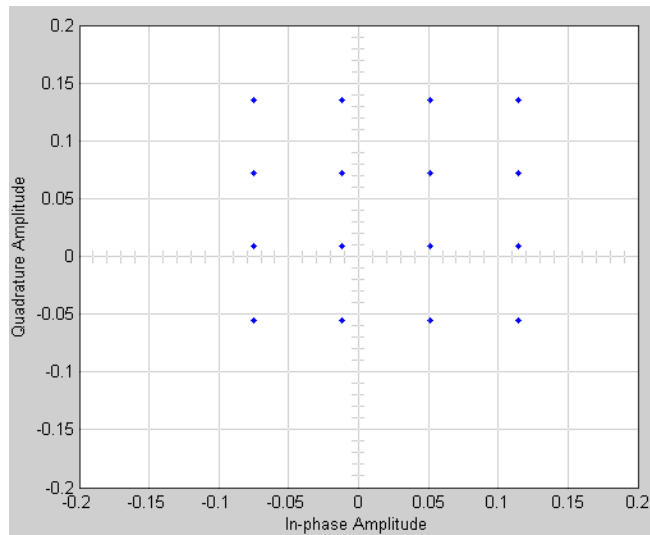
See "Scatter Plot Examples" for a description of the model that generates this plot.

Dialog Box

I/Q Imbalance

I/Q amplitude imbalance (dB)
    Scalar specifying the I/Q amplitude imbalance in decibels.

I/Q phase imbalance (deg)
    Scalar specifying the I/Q phase imbalance in degrees.

I dc offset
    Scalar specifying the in-phase dc offset.

Q dc offset
    Scalar specifying the amplitude dc offset.

See Also   Memoryless Nonlinearity

Purpose   Generate Kasami sequence from set of Kasami sequences

Library   Sequence Generators sublibrary of Comm Sources
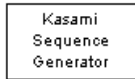
Description   The Kasami Sequence Generator block generates a sequence from the set of Kasami sequences. The Kasami sequences are a set of sequences that have good cross-correlation properties.

Kasami
Sequence
Generator

There are two classes of Kasami sequences: the *small set* and the *large set*. The large set contains all the sequences in the small set. Only the small set is optimal in the sense of matching Welch's lower bound for correlation functions.

Kasami sequences have period $N = 2^n - 1$, where n is a nonnegative, even integer. Let u be a binary sequence of length N, and let w be the sequence obtained by decimating u by $2^{n/2} + 1$. The small set of Kasami sequences is defined by the following formulas, in which T denotes the left shift operator, m is the shift parameter for w, and $\oplus$ denotes addition modulo 2.

$$K_s(u,n,m) = \begin{cases} u & m = -1 \\ u \oplus T^m w & m = 0,...,2^{n/2} - 2 \end{cases}$$

Small Set of Kasami Sequences for n Even

Note that the small set contains $2^{n/2}$ sequences.

For $\mod(n, 4) = 2$, the large set of Kasami sequences is defined as follows. Let v be the sequence formed by decimating the sequence u by $2^{n/2 + 1} + 1$. The large set is defined by the following table, in which k and m are the shift parameters for the sequences v and w, respectively.

Kasami Sequence Generator

$$
K_L(u,n,k,m) = \begin{cases}
u & k=-2;\ m=-1 \\
v & k=-1;\ m=-1 \\
u \oplus T^k v & k=0,...,2^n-2;\ m=-1 \\
u \oplus T^m w & k=-2;\ m=0,...,2^{n/2}-2 \\
v \oplus T^m w & k=-1;\ m=0,...,2^{n/2}-2 \\
u \oplus T^k v \oplus T^m w & k=0,...,2^n-2;\ m=0,...,2^{n/2}-2
\end{cases}
$$

Large Set of Kasami Sequences for mod(n, 4) = 2

The sequences described in the first three rows of the preceding figure correspond to the Gold sequences for mod(n, 4) = 2. See the reference page for the Gold Sequence Generator block for a description of Gold sequences. However, the Kasami sequences form a larger set than the Gold sequences.

The correlation functions for the sequences takes on the values

$\{-t(n),\ -s(n),\ -1,\ s(n)\ -2\ ,\ t(n)\ -\ 2\}$

where

$$
t(n) = 1 + 2^{(n+2)/2},\ n \text{ even}
$$
$$
s(n) = \frac{1}{2}(t(n)+1)
$$

Block Parameters

The **Generator polynomial** parameter specifies the generator polynomial, which determines the connections in the shift register that generates the sequence u. You can specify the **Generator polynomial** parameter using either of these formats:

- A vector that lists the coefficients of the polynomial in descending order of powers. The first and last entries must be 1. Note that the length of this vector is one more than the degree of the generator polynomial.

- A vector containing the exponents of *z* for the nonzero terms of the polynomial in descending order of powers. The last entry must be 0.

For example, [1 0 0 0 0 1 0 1] and [8 2 0] represent the same polynomial, $p(z) = z^8 + z^2 + 1$.

The **Initial states** parameter specifies the initial states of the shift register that generates the sequence u. **Initial States** is a binary scalar or row vector of length equal to the degree of the **Generator polynomial**. If you choose a binary scalar, the block expands the scalar to a row vector of length equal to the degree of the **Generator polynomial**, all of whose entries equal the scalar.

The **Sequence index** parameter specifies the shifts of the sequences *v* and *w* used to generate the output sequence. You can specify the parameter in either of two ways:

- To generate sequences from the small set, for *n* is even, you can specify the **Sequence index** as an integer *m*. The range of *m* is [-1, ..., $2^{n/2} - 2$]. The following table describes the output sequences corresponding to **Sequence index** m:

Sequence Index	Range of Indices	Output Sequence
-1	$m = -1$	u
m	$m = 0, \dots, 2^{n/2} - 2$	$u \oplus T^m w$

- To generate sequences from the large set, for mod (*n*, 4) = 2, where *n* is the degree of the **Generator polynomial**, you can specify **Sequence index** as an integer vector [*k m*]. In this case, the output sequence is from the large set. The range for *k* is [-2, ..., $2^n - 2$], and the range for *m* is [-1, ..., $2^{n/2} - 2$]. The following table describes the output sequences corresponding to **Sequence index** [k m]:

Kasami Sequence Generator

Sequence Index [k m]	Range of Indices	Output Sequence
[-2 -1]	$k = -2$, $m = -1$	u
[-1 -1]	$k = -1$, $m = -1$	v
[k -1]	$k = 0, 1, ... , 2^n - 2$ $m = -1$	$u \oplus T^k v$
[-2 m]	$k = -2$ $m = 0, 1, ..., 2^{n/2} - 2$	$u \oplus T^m w$
[-1 m]	$k = -1$ $m = 0, ... , 2^{n/2} - 2$	$v \oplus T^m w$
[k m]	$k = 0, ... , 2^n - 2$ $m = 0, ... , 2^{n/2} - 2$	$u \oplus T^k v \oplus T^m w$

You can shift the starting point of the Gold sequence with the **Shift** parameter, which is an integer representing the length of the shift.

You can use an external signal to reset the values of the internal shift register to the initial state by selecting the **Reset on nonzero input** check box. This creates an input port for the external signal in the Kasami Sequence Generator block. The way the block resets the internal shift register depends on whether its output signal and the reset signal are sample-based or frame-based. See "Example: Resetting a Signal" on page 2-574 for an example.

Polynomials for Generating Kasami Sequences

The following table lists some of the polynomials that you can use to generate the Kasami set of sequences.

n	N	Polynomial	Set
4	15	[4 1 0]	Small
6	63	[6 1 0]	Large

n	N	Polynomial	Set
8	255	[8 4 3 2 0]	Small
10	1023	[10 3 0]	Large
12	4095	[12 6 4 1 0]	Small

Dialog Box

Source Block Parameters: Kasami Sequence Generator

Kasami Sequence Generator (mask) (link)

Generate a Kasami sequence from the set of Kasami sequences by specifying the generator polynomial.

The generator polynomial parameter value represents the shift register connections. Enter these values as either a binary vector or a descending ordered polynomial to indicate the connection points.

The initial states parameter is a binary vector that represents the starting state of the shift register.

The sequence index parameter denotes the single sequence outputted from the set of Kasami sequences. Specify it as a 2-element integer vector for the Large set of sequences or as a scalar integer for the Small set of sequences.

The shift parameter is a scalar integer that produces an offset in the sequence.

Parameters

Generator polynomial:

[1 0 0 0 0 1 1]

Initial states:

[0 0 0 0 0 1]

Sequence index(es):

0

Shift:

0

Sample time:

1

☐ Frame-based outputs

☐ Reset on nonzero input

Output data type: double

OK   Cancel   Help

Kasami Sequence Generator

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Generator polynomial
Binary vector specifying the generator polynomial for the sequence u.

Initial states
Binary scalar or row vector of length equal to the degree of the **Generator polynomial**, which specifies the initial states of the shift register that generates the sequence u.

Sequence index
Integer or vector specifying the shifts of the sequences v and w used to generate the output sequence.

Shift
Integer scalar that determines the offset of the Kasami sequence from the initial time.

Sample time
Period of each element of the output signal.

Frame-based outputs
Determines whether the output is frame-based or sample-based.

Samples per frame
The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box.

Reset on nonzero input
When selected, you can specify an input signal that resets the internal shift registers to the original values of the **Initial states.**

Output data type
The output type of the block can be specified as a `boolean` or `double`. By default, the block sets this to `double`.

See Also   Gold Sequence Generator, PN Sequence Generator

Reference   [1] Peterson and Weldon, *Error Correcting Codes*, 2nd Ed., MIT Press, Cambridge, MA, 1972.

[2] Proakis, John G., *Digital Communications*, Third edition, New York, McGraw Hill, 1995.

[3] Sarwate, D. V. and Pursley, M.B., "Crosscorrelation Properties of Pseudorandom and Related Sequences," *Proc. IEEE*, Vol. 68, No. 5, May 1980, pp. 583-619.
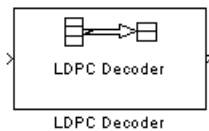
LDPC Decoder

Purpose   Decode binary low-density parity-check code specified by parity-check matrix

Library   Block sublibrary of Error Detection and Correction

Description   This block implements the message-passing algorithm for decoding low-density parity-check (LDPC) codes, which are linear error control codes with sparse parity-check matrices and long block lengths that can attain performance near the Shannon limit.

The LDPC Decoder block is designed to:

- Decode generic binary LDPC codes where no patterns in the parity-check matrix are assumed.

- Execute a user-specified number of iterations or run until all parity-checks are satisfied.

- Output hard decisions or soft decisions (log-likelihood ratios) for decoded bits.

The LDPC Decoder block uses the same algorithm as the `fec.ldpcdec` Communications Toolbox object. See Decoding Algorithm for details.

$(n - k)$ and n are the number of rows and columns, respectively, in the parity-check matrix.

The input must be a frame-based $n \times 1$ vector and a real-valued signal of type double. Each element is the log-likelihood ratio for a received bit (more likely to be 0 if the log-likelihood ratio is positive). The first k elements correspond to the information part of a codeword.

Both the input and the output are discrete-time signals. The ratio of the output sample time to the input sample time is n/k if only the information part is decoded, and 1 if the entire codeword is decoded.

Type `commdvbs2` at the command line to see a demo using this block.

**Dialog
Box**
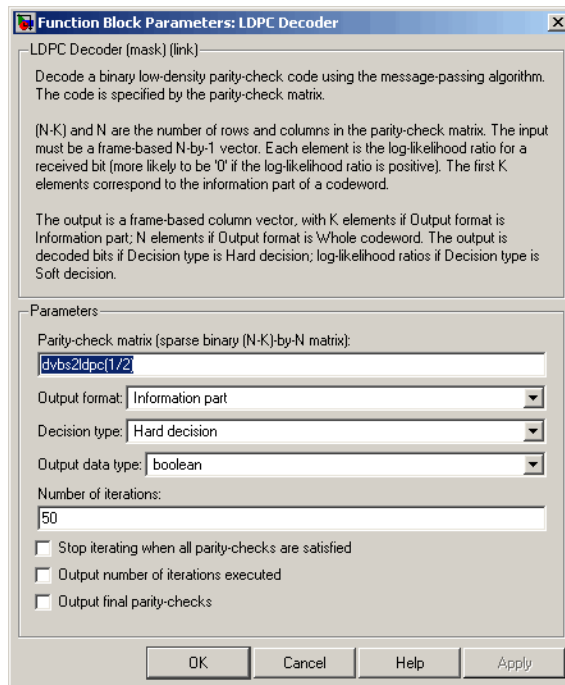
Parity-check matrix

This block can accept a sparse matrix with dimension n -k by n (where $n > k > 0$) of real numbers. All nonzero elements must be equal to 1. The upper bound limit for the value of n is 2^{31}-1

Output format

The options are Information part and Whole codeword.

The output is a real-valued signal in the form of a frame-based column vector. It has k elements if the output format is set to Information part, and n elements if set to whole codeword.

Decision type

The options are Hard decision and Soft decision.

LDPC Decoder

If this is set to `Hard decision`, the output is decoded bits (of type double or boolean). If set to `Soft decision`, the output is log-likelihood ratios (of type double).

Output data type
This parameter appears only when **Decision type** is set to `Hard decision`.

The options are `boolean` and `double`.

Number of iterations
This can be any positive integer.

Stop iterating when all parity checks are satisfied
If checked, the block will determine whether the parity checks are satisfied after each iteration and stop if all are satisfied.

Output number of iterations executed
Creates an output port on the block when selected.

Output final parity checks
Creates an output port on the block when selected.

Pair Block   `dvbs2ldpc` and LDPC Encoder

References   [1] Gallager, Robert G., *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

Purpose   Encode binary low-density parity-check code specified by parity-check matrix

Library   Block sublibrary of Error Detection and Correction
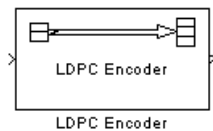
Description   This block supports encoding of low-density parity-check (LDPC) codes, which are linear error control codes with sparse parity-check matrices and long block lengths that can attain performance near the Shannon limit.

The LDPC Encoder block uses the same algorithm as the `fec.ldpcenc` Communications Toolbox object. See LDPC Encoding Method for details.

$(n - k)$ and n are the number of rows and columns, respectively, in the parity-check matrix.

Both the input and the output are discrete-time signals. The ratio of the output sample time to the input sample time is k/n. The input must be a real $k \times 1$ frame-based signal.

The output inherits the data type of the input, and the input must be binary-valued (0 or 1).

Type `commdvbs2` at the command line to see a demo using this block.

Note Model initialization or update may take a long time, because a large matrix may need to be inverted (when the last $(n - k)$ columns of the parity-check matrix is not triangular).

LDPC Encoder

**Dialog
Box**
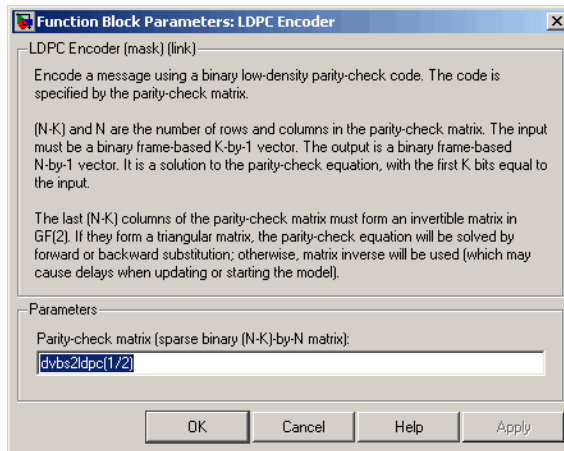
Parity-check matrix

This block can accept a sparse matrix with dimension n -k by n (where $n > k > 0$) of real numbers. All nonzero elements must be equal to 1. The upper bound limit for the value of n is 2^{31}-1

The default value is the parity-check matrix of the half-rate LDPC code from the DVB-S.2 standard.

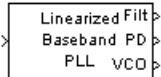**Pair Block**   dvbs2ldpc and LDPC Decoder

Purpose   Implement linearized version of baseband phase-locked loop

Library   Components sublibrary of Synchronization

Description   The Linearized Baseband PLL block is a feedback control system that automatically adjusts the phase of a locally generated signal to match the phase of an input signal. Unlike the Phase-Locked Loop block, this block uses a baseband model method. Unlike the Baseband PLL block, which uses a nonlinear model, this block simplifies the computations by using x to approximate $\sin(x)$. The baseband PLL model depends on the amplitude of the incoming signal but does not depend on a carrier frequency.

```
Linearized Filt
Baseband PD
  PLL   VCO
```

This PLL has these three components:

- An integrator used as a phase detector.

- A filter. You specify the filter's transfer function using the **Lowpass filter numerator** and **Lowpass filter denominator** parameters. Each is a vector that gives the respective polynomial's coefficients in order of descending powers of s.

   To design a filter, you can use functions such as `butter`, `cheby1`, and `cheby2` in Signal Processing Toolbox software. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

   `[num, den] = cheby2(3,40,100,'s')`

- A voltage-controlled oscillator (VCO). You specify the sensitivity of the VCO signal to its input using the **VCO input sensitivity** parameter. This parameter, measured in Hertz per volt, is a scale factor that determines how much the VCO shifts from its quiescent frequency.
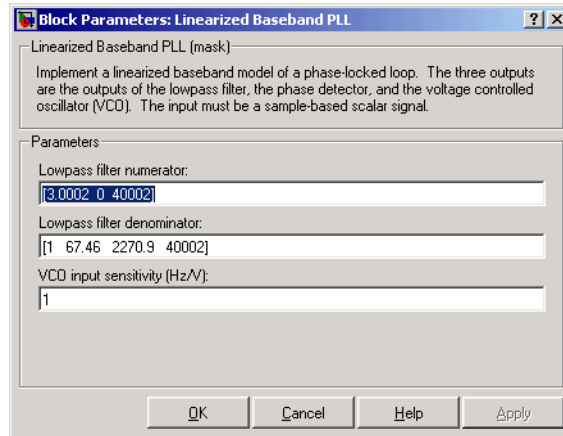
The input signal represents the received signal. The input must be a sample-based scalar signal. The three output ports produce:

- The output of the filter

Linearized Baseband PLL

- The output of the phase detector

- The output of the VCO

Lowpass filter numerator

The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of *s*.

Lowpass filter denominator

The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of *s*.

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the VCO's quiescent frequency.

See Also   Baseband PLL, Phase-Locked Loop

References   For more information about phase-locked loops, see the works listed in "Selected Bibliography for Synchronization" in *Communications Blockset User's Guide*.

Purpose   Equalize using decision feedback equalizer that updates weights with LMS algorithm

Library   Equalizers

Description   The LMS Decision Feedback Equalizer block uses a decision feedback equalizer and the LMS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

LMS Decision Feedback Equalizer

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap of the forward filter.

Dialog Box

Function Block Parameters: LMS Decision Feedback Equalizer

LMS Decision Feedback Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using a decision feedback equalizer and the LMS algorithm.

The block computes filter weights with the LMS algorithm and filters the input signal. The filter weights are updated once for each symbol.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode input port box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, and for decision directed, the mode must be 0. For every frame in which the mode input is 1 or not present, the equalizer trains at the beginning of the frame for the length of the desired signal.

Parameters

Number of forward taps:

6

Number of feedback taps:

6

Number of samples per symbol:

1

Signal constellation:

[-1+j -1-j 1+j 1-j]

Reference tap:

3

Step size:

0.01

Leakage factor:

1

Initial weights:

0

☑ Mode input port

☑ Output error

☑ Output weights

| OK | Cancel | Help | Apply |

LMS Decision Feedback Equalizer

Number of forward taps

> The number of taps in the forward filter of the decision feedback equalizer.

Number of feedback taps

> The number of taps in the feedback filter of the decision feedback equalizer.

Number of samples per symbol

> The number of input samples for each symbol.

Signal constellation

> A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

> A positive integer less than or equal to the number of forward taps in the equalizer.

Step size

> The step size of the LMS algorithm.

Leakage factor

> The leakage factor of the LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

> A vector that concatenates the initial weights for the forward and feedback taps.

Mode input port

> If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

> If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current forward and feedback weights, concatenated into one vector.

References

[1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

[2] Haykin, Simon, *Adaptive Filter Theory*, Third Ed., Upper Saddle River, N.J., Prentice-Hall, 1996.

[3] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

[4] Proakis, John G., *Digital Communications*, Fourth Ed., New York, McGraw-Hill, 2001.

See Also

LMS Linear Equalizer, Normalized LMS Decision Feedback Equalizer, Sign LMS Decision Feedback Equalizer, Variable Step LMS Decision Feedback Equalizer, RLS Decision Feedback Equalizer, CMA Equalizer

LMS Linear Equalizer

Purpose   Equalize using linear equalizer that updates weights with LMS algorithm

Library   Equalizers

Description   The LMS Linear Equalizer block uses a linear equalizer and the LMS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in"Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

$$1+(\textbf{Reference tap}\text{-}1)/(\textbf{Number of samples per symbol})$$

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap.

LMS Linear Equalizer
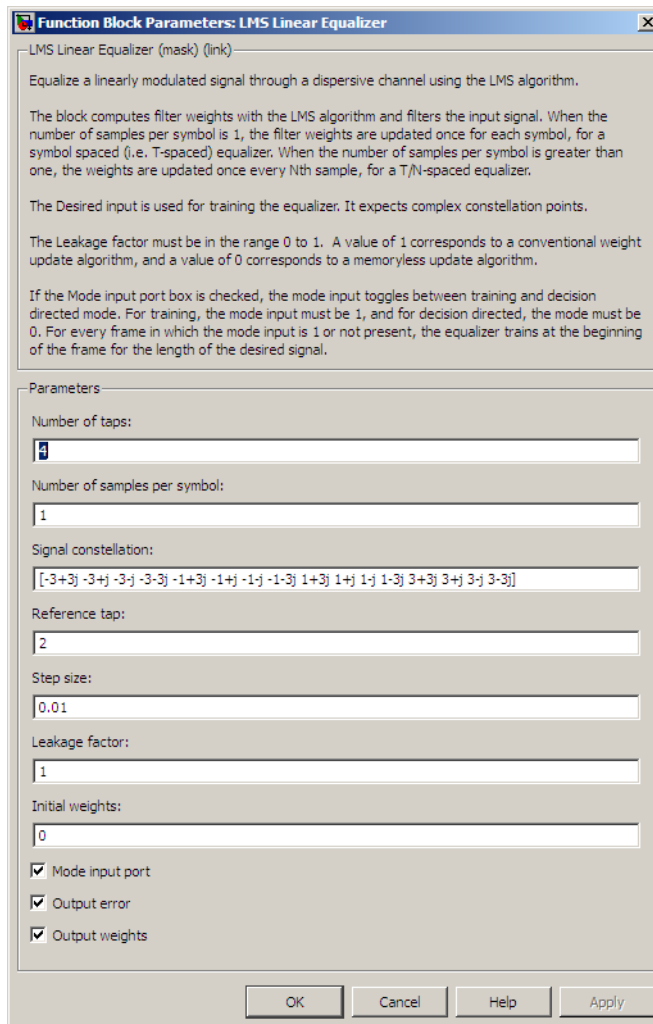
Dialog Box

Number of taps

    The number of taps in the filter of the linear equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulated signal, as determined by the modulator in your model

Reference tap

A positive integer less than or equal to the number of taps in the equalizer.

Step size

The step size of the LMS algorithm.

Leakage factor

The leakage factor of the LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

A vector that lists the initial weights for the taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current weights.

Examples   See "Example: LMS Linear Equalizer" and the Adaptive Equalization demo.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

LMS Linear Equalizer

[2] Haykin, Simon, *Adaptive Filter Theory*, Third Ed., Upper Saddle River, N.J., Prentice-Hall, 1996.

[3] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

[4] Proakis, John G., *Digital Communications*, Fourth Ed., New York, McGraw-Hill, 2001.

See Also   LMS Decision Feedback Equalizer, Normalized LMS Linear Equalizer, Sign LMS Linear Equalizer, Variable Step LMS Linear Equalizer, RLS Linear Equalizer, CMA Equalizer

Purpose   Permute input symbols by filling matrix by columns and emptying it by rows

Library   Block sublibrary of Interleaving

Description

The Matrix Deinterleaver block performs block deinterleaving by filling a matrix with the input symbols column by column and then sending the matrix contents to the output port row by row. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

The length of the input vector must be **Number of rows** times **Number of columns**. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box
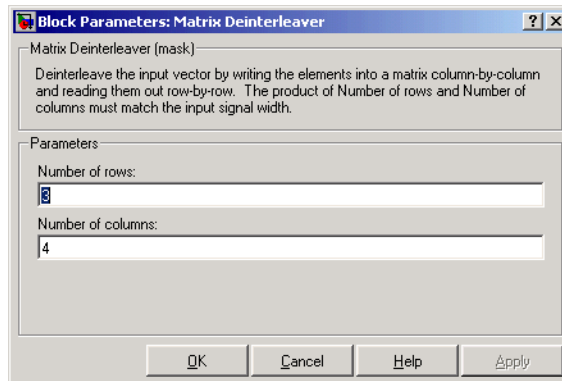
Number of rows
> The number of rows in the matrix that the block uses for its computations.

Matrix Deinterleaver

Number of columns
> The number of columns in the matrix that the block uses for its computations.

Examples   If the **Number of rows** and **Number of columns** parameters are 2 and 3, respectively, then the deinterleaver uses a 2-by-3 matrix for its internal computations. Given an input signal of `[1; 2; 3; 4; 5; 6]`, the block produces an output of `[1; 3; 5; 2; 4; 6]`.

Pair Block   Matrix Interleaver

See Also   General Block Deinterleaver

Purpose   Restore ordering of input symbols by filling matrix along diagonals

Library   Block sublibrary of Interleaving

Description   The Matrix Helical Scan Deinterleaver block performs block deinterleaving by filling a matrix with the input symbols in a helical fashion and then sending the matrix contents to the output port row by row. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

Helical fashion means that the block places input symbols along diagonals of the matrix. The number of elements in each diagonal matches the **Number of columns** parameter, after the block wraps past the edges of the matrix when necessary. The block traverses diagonals so that the row index and column index both increase. Each diagonal after the first one begins one row below the first element of the previous diagonal.

The **Array step size** parameter is the slope of each diagonal, that is, the amount by which the row index increases as the column index increases by one. This parameter must be an integer between zero and the **Number of rows** parameter. If the **Array step size** parameter is zero, then the block does not deinterleave and the output is the same as the input.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Matrix Helical Scan Deinterleaver

Dialog Box
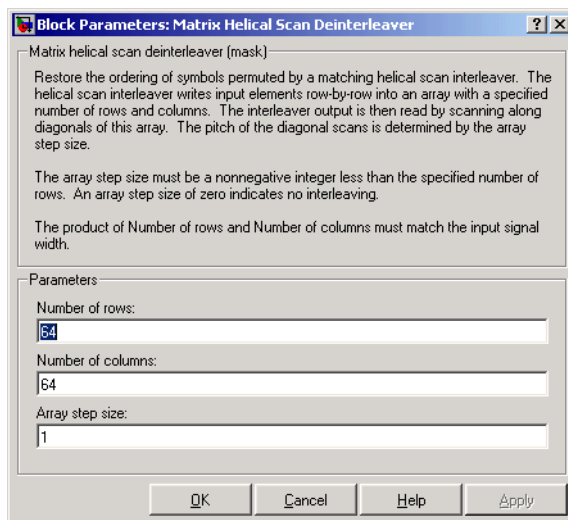
Number of rows
> The number of rows in the matrix that the block uses for its computations.

Number of columns
> The number of columns in the matrix that the block uses for its computations.

Array step size
> The slope of the diagonals that the block writes.

Pair Block   Matrix Helical Scan Interleaver

See Also   General Block Deinterleaver

Purpose   Permute input symbols by selecting matrix elements along diagonals

Library   Block sublibrary of Interleaving

Description The Matrix Helical Scan Interleaver block performs block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port in a helical fashion. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

Helical fashion means that the block selects output symbols by selecting elements along diagonals of the matrix. The number of elements in each diagonal matches the **Number of columns** parameter, after the block wraps past the edges of the matrix when necessary. The block traverses diagonals so that the row index and column index both increase. Each diagonal after the first one begins one row below the first element of the previous diagonal.

The **Array step size** parameter is the slope of each diagonal, that is, the amount by which the row index increases as the column index increases by one. This parameter must be an integer between zero and the **Number of rows** parameter. If the **Array step size** parameter is zero, then the block does not interleave and the output is the same as the input.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Matrix Helical Scan Interleaver

Dialog Box
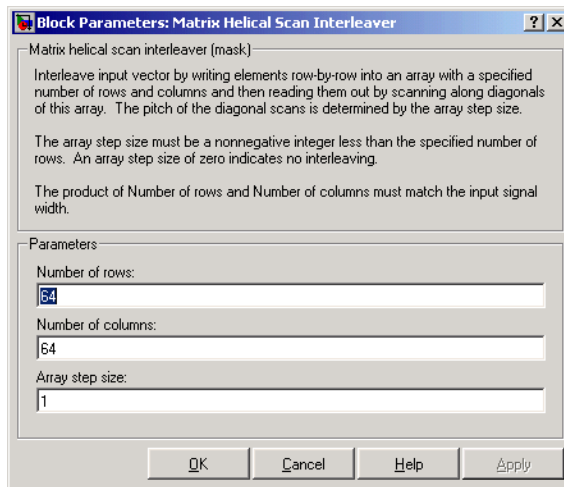
Number of rows
    The number of rows in the matrix that the block uses for its computations.

Number of columns
    The number of columns in the matrix that the block uses for its computations.
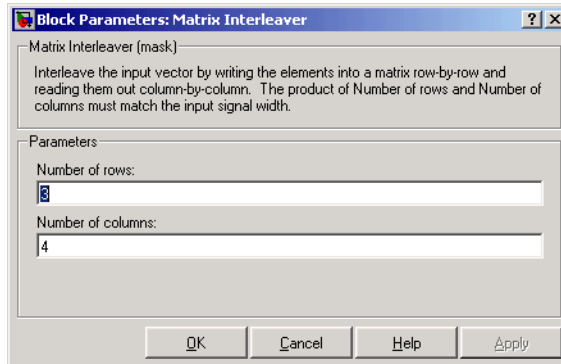
Array step size
    The slope of the diagonals that the block reads.

Examples   If the **Number of rows** and **Number of columns** parameters are 6 and 4, respectively, then the interleaver uses a 6-by-4 matrix for its internal computations. If the **Array step size** parameter is 1, then the diagonals are as shown in the figure below. Positions with the same color form part of the same diagonal, and diagonals with darker colors precede those with lighter colors in the output signal.

Given an input signal of [1:24]', the block produces an output of

Block's Internal Array
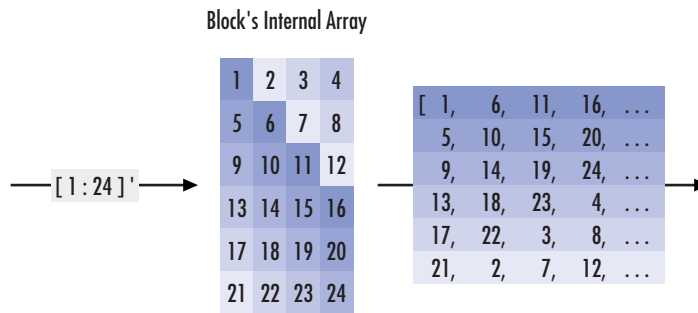

```
[1; 6; 11; 16; 5; 10; 15; 20; 9; 14; 19; 24; 13; 18; 23;...
4; 17; 22; 3; 8; 21; 2; 7; 12]
```

Pair Block   Matrix Helical Scan Deinterleaver

See Also   General Block Interleaver

Matrix Interleaver

Purpose
Permute input symbols by filling matrix by rows and emptying it by columns

Library
Block sublibrary of Interleaving

Description

Matrix
Interleaver

The Matrix Interleaver block performs block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port column by column.

The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box

Number of rows
The number of rows in the matrix that the block uses for its computations.

Number of columns
> The number of columns in the matrix that the block uses for its computations.

Examples If the **Number of rows** and **Number of columns** parameters are 2 and 3, respectively, then the interleaver uses a 2-by-3 matrix for its internal computations. Given an input signal of `[1; 2; 3; 4; 5; 6]`, the block produces an output of `[1; 4; 2; 5; 3; 6]`.
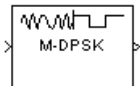
Pair Block Matrix Deinterleaver

See Also General Block Interleaver

M-DPSK Demodulator Baseband

Purpose   Demodulate DPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The M-DPSK Demodulator Baseband block demodulates a signal that
was modulated using the M-ary differential phase shift keying method.
The input is a baseband representation of the modulated signal. The
input and output for this block are discrete-time signals. The input can
be either a scalar or a frame-based column vector. The block accepts the
input data types `single` and `double`.

The **M-ary number** parameter, M, is the number of possible output
symbols that can immediately follow a given output symbol. The block
compares the current symbol to the previous symbol. The block's first
output is the initial condition of zero (or a group of zeros, if the **Output
type** parameter is set to `Bit`) because there is no previous symbol.

Binary or Integer Outputs

If the **Output type** parameter is set to `Integer`, then the block
demodulates a phase difference of

$$\theta + 2\pi k/M$$

to k, where θ is the **Phase rotation** parameter and k is an integer
between 0 and M-1.

If the **Output type** parameter is set to `Bit` and the **M-ary number**
parameter has the form 2^K for some positive integer K, then the block
outputs binary representations of integers between 0 and M-1. It
outputs a group of K bits, called a binary *word*, for each symbol.

In binary output mode, the **Constellation ordering** parameter
indicates how the block maps an integer to a corresponding group of
K output bits. See the reference pages for theM-DPSK Modulator
Baseband andM-PSK Modulator Baseband blocks for details.

Dialog Box
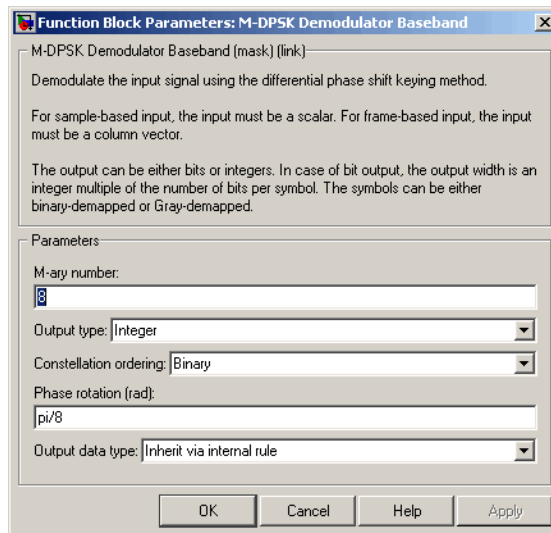
M-ary number
> The number of possible modulated symbols that can immediately follow a given symbol.

Output type
> Determines whether the output consists of integers or groups of bits.

Constellation ordering
> Determines how the block maps each integer to a group of output bits.

Phase rotation (rad)
> The phase difference between the previous and current modulated symbols when the input is zero.

Output data type
> When the parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is of type `single` or `double`.

M-DPSK Demodulator Baseband

For integer outputs, this block can output the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. For bit outputs, output can be `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, or `double`.

Pair Block   M-DPSK Modulator Baseband

See Also   DBPSK Demodulator Baseband, DQPSK Demodulator Baseband, M-PSK Demodulator Baseband

References   [1] Pawula, R. F., "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels," *IEEE Transactions on Communications*, Vol. COM-32, July 1984, 752-761.

Purpose   Modulate using M-ary differential phase shift keying method

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The M-DPSK Modulator Baseband block modulates using the M-ary differential phase shift keying method. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the number of possible output symbols that can immediately follow a given output symbol.

The input must be a discrete-time signal. For integer inputs, the block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. For bit inputs, the block can accept `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`.

Inputs and Constellation Types

If the **Input type** parameter is set to `Integer`, then valid input values are integers between 0 and M-1. In this case, the input can be either a scalar or a frame-based column vector. If the first input is k_1, then the modulated symbol is

$$\exp\left(j\theta + j2\pi\frac{k_1}{m}\right)$$

where θ is the **Phase rotation** parameter. If a successive input is k, then the modulated symbol is

$$\exp\left(j\theta + j2\pi\frac{k}{m}\right)\cdot(\text{previous modulated symbol})$$

If the **Input type** parameter is set to `Bit` and the **M-ary number** parameter has the form 2^K for some positive integer K, then the block accepts binary representations of integers between 0 and M-1. It modulates each group of K bits, called a binary *word*. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K.

M-DPSK Modulator Baseband

In binary input mode, the **Constellation ordering** parameter indicates how the block maps a group of K input bits to a corresponding phase difference. The Binary option uses a natural binary-to-integer mapping, while the Gray option uses a Gray-coded assignment of phase differences. For example, the table below indicates the assignment of phase difference to three-bit inputs, for both the Binary and Gray options. θ is the **Phase rotation** parameter. The phase difference is between the previous symbol and the current symbol.

Current Input	Binary-Coded Phase Difference	Gray-Coded Phase Difference
[0 0 0]	jθ	jθ
[0 0 1]	jθ + jπ/4	jθ + jπ/4
[0 1 0]	jθ + jπ2/4	jθ + jπ3/4
[0 1 1]	jθ + jπ3/4	jθ + jπ2/4
[1 0 0]	jθ + jπ4/4	jθ + jπ7/4
[1 0 1]	jθ + jπ5/4	jθ + jπ6/4
[1 1 0]	jθ + jπ6/4	jθ + jπ4/4
[1 1 1]	jθ + jπ7/4	jθ + jπ5/4

For more details about the Binary and Gray options, see the reference page for theM-PSK Modulator Baseband block. The signal constellation for that block corresponds to the arrangement of phase differences for this block.

**Dialog
Box**
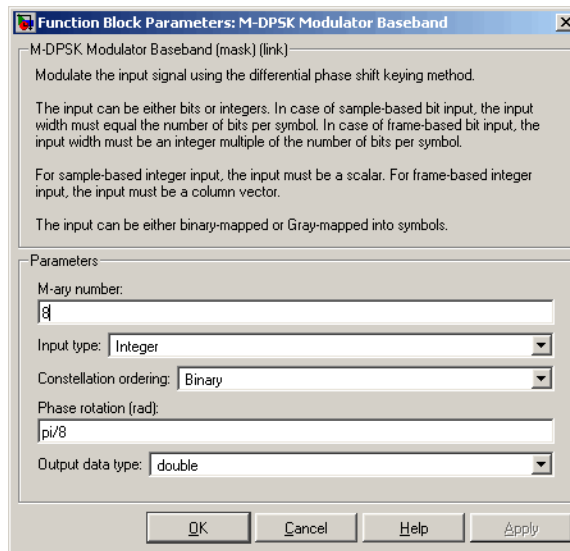
M-ary number
> The number of possible output symbols that can immediately
> follow a given output symbol.

Input type
> Indicates whether the input consists of integers or groups of
> bits. If this parameter is set to Bit, then the **M-ary number**
> parameter must be 2^K for some positive integer K.

Constellation ordering
> Determines how the block maps each group of input bits to a
> corresponding integer.

Phase rotation (rad)
> The phase difference between the previous and current modulated
> symbols when the input is zero.

Output data type
> The output data type can be either single or double. By default,
> the block sets this to double.

M-DPSK Modulator Baseband

Pair Block M-DPSK Demodulator Baseband

See Also DBPSK Modulator Baseband, DQPSK Modulator Baseband, M-PSK Modulator Baseband

References [1] Pawula, R. F., "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels," *IEEE Transactions on Communications*, Vol. COM-32, July 1984, 752-761.

Purpose   Apply memoryless nonlinearity to complex baseband signal

Library   RF Impairments
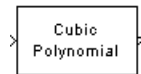
Description   The Memoryless Nonlinearity block applies a memoryless nonlinearity
to a complex, baseband signal. You can use the block to model radio
frequency (RF) impairments to a signal at the receiver.

> **Note** All values of power assume a nominal impedance of 1 ohm.

The Memoryless Nonlinearity block provides five different methods for
modeling the nonlinearity, which you specify by the **Method** parameter.
The options for the **Method** parameter are

- Cubic polynomial

- Hyperbolic tangent

- Saleh model

- Ghorbani model

- Rapp model

The five methods are implemented by subsystems underneath the
block's mask. Each subsystem has the same basic structure, as shown
in the following figure.
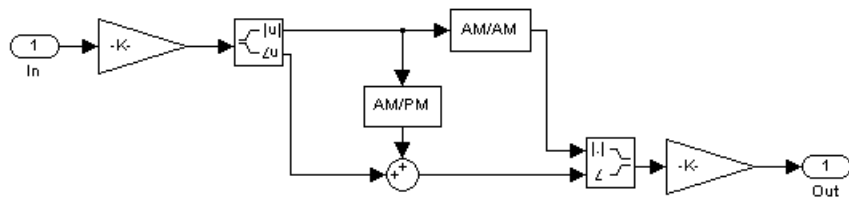
Nonlinearity Subsytem

Memoryless Nonlinearity

All five subsystems apply a nonlinearity to the input signal as follows:

1 Multiply the signal by a gain factor.

2 Split the complex signal into its its magnitude and angle components.

3 Apply an AM/AM conversion to the magnitude of the signal, according to the selected **Method**, to produce the magnitude of the output signal.

4 Apply an AM/PM conversion to the phase of the signal, according to the selected **Method**, and adds the result to the angle of the signal to produce the angle of the output signal.

5 Combine the new magnitude and angle components into a complex signal and multiply the result by a gain factor, which is controlled by the **Linear gain** parameter.

However, the subsystems implement the AM/AM and AM/PM conversions differently, according to the **Method** you specify.

If you want to see exactly how the Memoryless Nonlinearity block implements the conversions for a specific method, you can view the AM/AM and AM/PM subsystems that implement these conversions as follows:

1 Right-click on the Memoryless Nonlinearity block and select **Look under mask**. This displays the block's configuration underneath the mask. The block contains five subsystems corresponding to the five nonlinearity methods.

2 Double-click the subsystem for the method you are interested in. This displays the subsystem shown in the preceding figure, Nonlinearity Subsytem on page 2-399.

3 Double-click on one of the subsystems labeled AM/AM or AM/PM to view how the block implements the conversions.

AM/PM Characteristics of the Cubic Polynomial and Hyperbolic Tangent Methods

The following illustration shows the AM/PM behavior for the Cubic polynomial and Hyperbolic tangent methods:

Input power level (dBm)
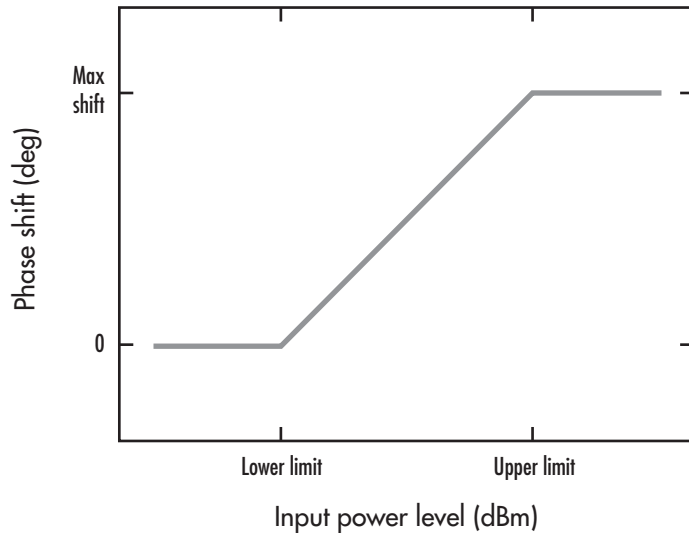
The AM/PM conversion scales linearly with input power value between the lower and upper limits of the input power level (specified by **Lower input power limit for AM/PM conversion (dBm)** and **Upper input power limit for AM/PM conversion (dBm)**). Beyond these values, AM/PM conversion is constant at the values corresponding to the lower and upper input power limits, which are zero and
$$(\text{AM/PM conversion}) \cdot (\text{upper input power limit} - \text{lower input power limit}),$$
respectively.

AM/AM and AM/PM Characteristics of the Saleh Method

The following figure shows, for the Saleh method, plots of

- Output voltage against input voltage for the AM/AM conversion

• Output phase against input voltage for the AM/PM conversion
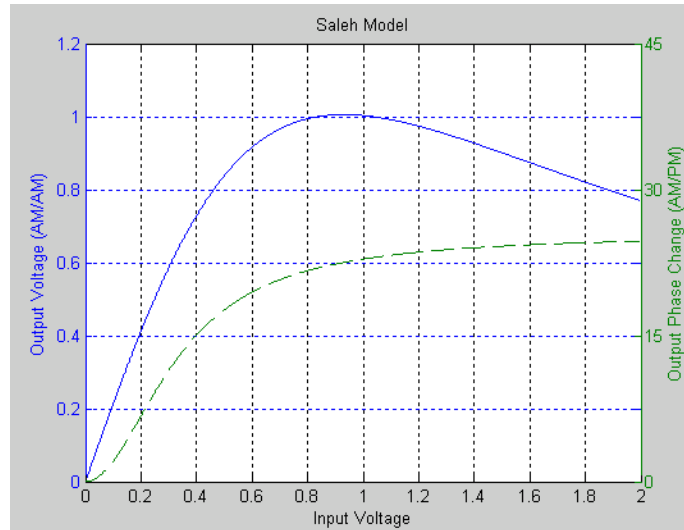
Example with 16-ary QAM

You can see the effect of the Memoryless Nonlinearity block on a signal modulated by 16-ary quadrature amplitude modulation (QAM) in a scatter plot. The constellation for 16-ary QAM without the effect of the Memoryless Nonlinearity block is shown in the following figure:
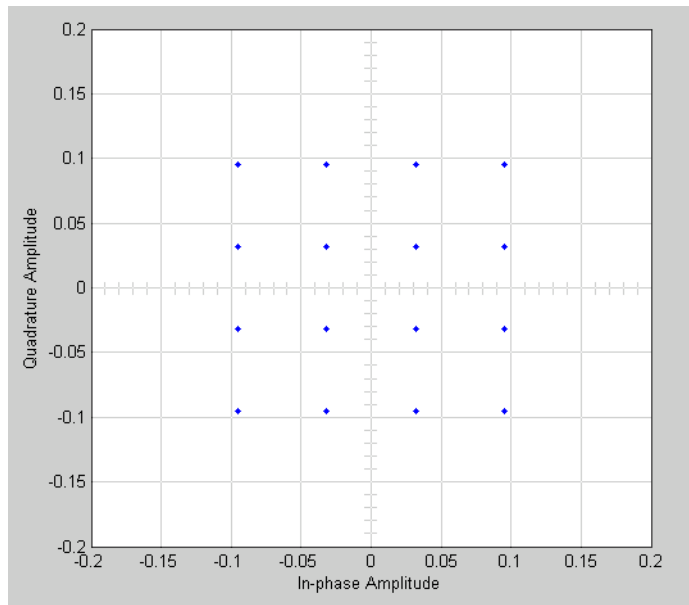
You can generate a scatter plot of the same signal after it passes through the Memoryless Nonlinearity block, with the **Method** parameter set to Saleh Model, as shown in the following figure.

Memoryless Nonlinearity
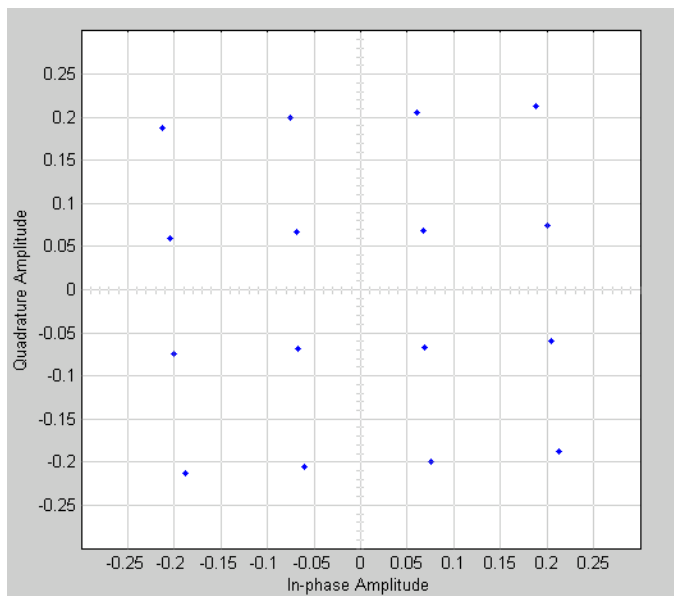
This plot is generated by the model described in "Scatter Plot Examples" with the following parameter settings for the Rectangular QAM Modulator Baseband block:

- **Normalization method** set to `Average Power`
- **Average power (watts)** set to `1e-2`

The following sections discuss parameters specific to the Saleh, Ghorbani, and Rapp models.

Parameters for the Saleh Model

The **Input scaling (dB)** parameter scales the input signal before the nonlinearity is applied. The block multiplies the input signal by the parameter value, converted from decibels to linear units. If you set the parameter to be the inverse of the input signal amplitude, the scaled signal has amplitude normalized to 1.

The AM/AM parameters, alpha and beta, are used to compute the amplitude gain for an input signal using the following function:

$$F_{AM/AM}(u) = \frac{\text{alpha} * u}{1 + \text{beta} * u^2}$$

where u is the magnitude of the scaled signal.

The AM/PM parameters, alpha and beta, are used to compute the phase change for an input signal using the following function:

$$F_{AM/PM}(u) = \frac{\text{alpha} * u^2}{1 + \text{beta} * u^2}$$

where u is the magnitude of the scaled signal. Note that the AM/AM and AM/PM parameters, although similarly named alpha and beta, are distinct.

The **Output scaling (dB)** parameter scales the output signal similarly.

Parameters for the Ghorbani Model

The **Input scaling (dB)** parameter scales the input signal before the nonlinearity is applied. The block multiplies the input signal by the parameter value, converted from decibels to linear units. If you set the parameter to be the inverse of the input signal amplitude, the scaled signal has amplitude normalized to 1.

The AM/AM parameters, $[x_1\ x_2\ x_3\ x_4]$, are used to compute the amplitude gain for an input signal using the following function:

$$F_{AM/AM}(u) = \frac{x_1 u^{x_2}}{1 + x_3 u^{x_2}} + x_4 u$$

where u is the magnitude of the scaled signal.

The AM/PM parameters, $[y_1\ y_2\ y_3\ y_4]$, are used to compute the phase change for an input signal using the following function:

$$F_{AM/PM}(u) = \frac{y_1 u^{y_2}}{1 + y_3 u^{y_2}} + y_4 u$$

where u is the magnitude of the scaled signal.

The **Output scaling (dB)** parameter scales the output signal similarly.

Parameters for the Rapp Model

The **Linear gain (dB)** parameter scales the input signal before the nonlinearity is applied. The block multiplies the input signal by the parameter value, converted from decibels to linear units. If you set the parameter to be the inverse of the input signal amplitude, the scaled signal has amplitude normalized to 1.

The **Smoothness factor** and **Output saturation level** parameters are used to compute the amplitude gain for the input signal:

$$F_{AM/AM}(u) = \frac{u}{\left(1 + \left(\dfrac{u}{O_{sat}}\right)^{2S}\right)^{1/2S}}$$

where u is the magnitude of the scaled signal, S is the **Smoothness factor**, and O_{sat} is the **Output saturation level**.

The Rapp model does not apply a phase change to the input signal.

The **Output saturation level** parameter limits the output signal level.

**Dialog
Box**
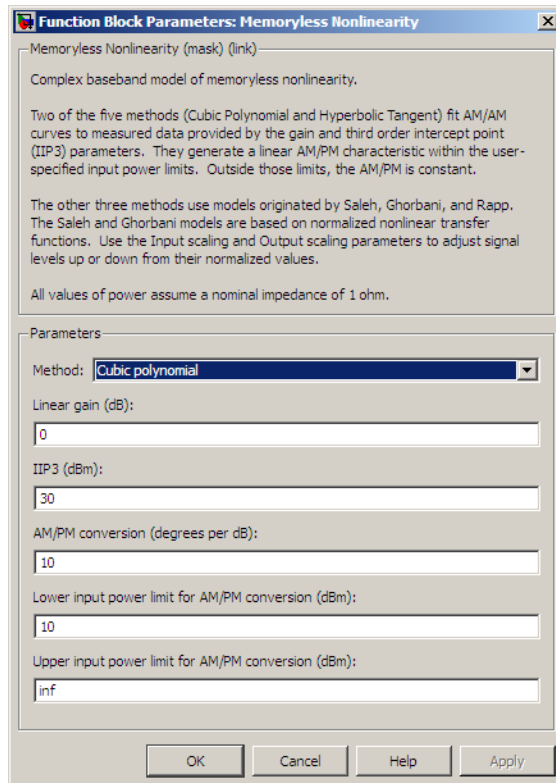
Method

The nonlinearity method.

The following describes specific parameters for each method.

Memoryless Nonlinearity

Linear gain (db)

    Scalar specifying the linear gain for the output function.

IIP3 (dBm)

    Scalar specifying the third order intercept.

AM/PM conversion (degrees per dB)

    Scaler specifying the AM/PM conversion in degrees per decibel.

Lower input power limit (dBm)

    Scalar specifying the minimum input power for which AM/PM conversion scales linearly with input power value. Below this value, the phase shift resulting from AM/PM conversion is zero.

Upper input power limit (dBm)

    Scalar specifying the maximum input power for which AM/PM conversion scales linearly with input power value. Above this value, the phase shift resulting from AM/PM conversion is constant. The value of this maximum shift is given by:

$$(\text{AM/PM conversion}) \cdot (\text{upper input power limit} - \text{lower input power limit})$$

Linear gain (db)

  Scalar specifying the linear gain for the output function.

IIP3 (dBm)

  Scalar specifying the third order intercept.

AM/PM conversion (degrees per dB)

  Scalar specifying the AM/PM conversion in degrees per decibel.

Lower input power limit (dBm)

  Scalar specifying the minimum input power for which AM/PM conversion scales linearly with input power value. Below this value, the phase shift resulting from AM/PM conversion is zero.

Upper input power limit (dBm)

  Scalar specifying the maximum input power for which AM/PM conversion scales linearly with input power value. Above this value, the phase shift resulting from AM/PM conversion is constant. The value of this maximum shift is given by:

$$(\text{AM/PM conversion}) \cdot (\text{upper input power limit} - \text{lower input power limit})$$

Memoryless Nonlinearity

Input scaling (dB)
> Number that scales the input signal level.

AM/AM parameters [alpha beta]
> Vector specifying the AM/AM parameters.

AM/PM parameters [alpha beta]
> Vector specifying the AM/PM parameters.

Output scaling (dB)
> Number that scales the output signal level.

Input scaling (dB)
> Number that scales the input signal level.
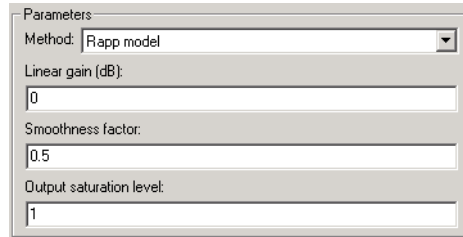
AM/AM parameters [x1 x2 x3 x4]
> Vector specifying the AM/AM parameters.

AM/PM parameters [y1 y2 y3 y4]
> Vector specifying the AM/PM parameters.

Output scaling (dB)
> Number that scales the output signal level.

Linear gain (db)
> Scalar specifying the linear gain for the output function.

Smoothness factor
> Scalar specifying the smoothness factor

Output saturation level
> Scalar specifying the the output saturation level.

See Also   I/Q Imbalance

Reference   [1] Saleh, A.A.M., "Frequency-independent and frequency-dependent nonlinear models of TWT amplifiers," IEEE Trans. Communications, vol. COM-29, pp.1715-1720, November 1981.

[2] A. Ghorbani, and M. Sheikhan, "The effect of Solid State Power Amplifiers (SSPAs) Nonlinearities on MPSK and M-QAM Signal Transmission", Sixth Int'l Conference on Digital Processing of Signals in Comm., 1991, pp. 193-197.

[3] C. Rapp, "Effects of HPA-Nonlinearity on a 4-DPSK/OFDM-Signal for a Digitial Sound Broadcasting System", in Proceedings of the Second

Memoryless Nonlinearity

European Conference on Satellite Communications, Liege, Belgium, Oct. 22-24, 1991, pp. 179-184.

Purpose   Demodulate FSK-modulated data

Library   FM, in Digital Baseband sublibrary of Modulation

Description   The M-FSK Demodulator Baseband block demodulates a signal that
                    was modulated using the M-ary frequency shift keying method. The
                    input is a baseband representation of the modulated signal. The input
                    and output for this block are discrete-time signals. The input can be
                    either a scalar or a frame-based column vector of type `single` or `double`.

                    The **M-ary number** parameter, M, is the number of frequencies in
                    the modulated signal. The **Frequency separation** parameter is the
                    distance, in Hz, between successive frequencies of the modulated signal.

                    The M-FSK Demodulator Baseband block implements a non-coherent
                    energy detector. To obtain the same BER performance as that of
                    coherent FSK demodulation, use the CPFSK Demodulator Baseband
                    block.

                    ### Binary or Integer Outputs

                    If the **Output type** parameter is set to `Integer`, then the block outputs
                    integers between 0 and M-1.

                    If the **Output type** parameter is set to `Bit` and the **M-ary number**
                    parameter has the form 2^K for some positive integer K, then the block
                    outputs binary representations of integers between 0 and M-1. It
                    outputs a group of K bits, called a binary *word*, for each symbol.

                    For `boolean` typed integer outputs, the **M-ary number** parameter
                    must be 2. For `Bit` type outputs, the outputs must be of type `boolean`
                    or `double`.

                    In binary output mode, the **Symbol set ordering** parameter indicates
                    how the block maps an integer to a corresponding group of K output
                    bits. See the reference pages for theM-FSK Modulator Baseband
                    andM-PSK Modulator Baseband blocks for details.

                    Whether the output is an integer or a binary representation of an
                    integer, the block maps the highest frequency to the integer 0 and maps
                    the lowest frequency to the integer M-1. In baseband simulation, the

lowest frequency is the negative frequency with the largest absolute value.

Dialog Box
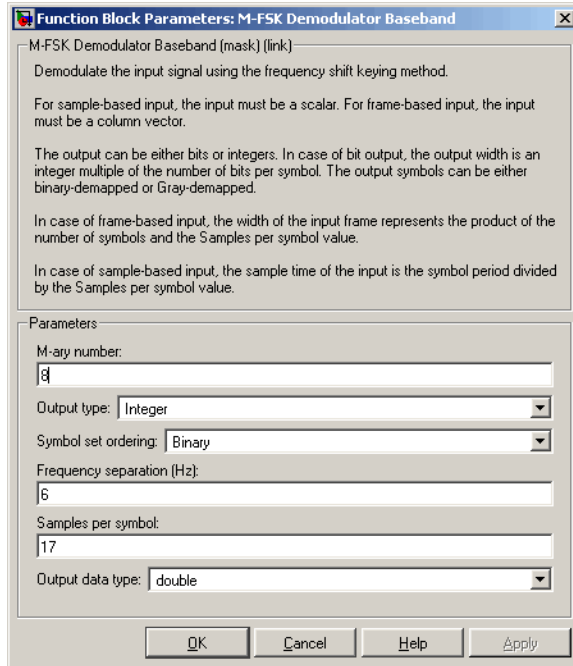
M-ary number

The number of frequencies in the modulated signal.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to Bit, then the **M-ary number** parameter must be 2^K for some positive integer K.

Symbol set ordering

Determines how the block maps each integer to a group of output bits.

Frequency separation (Hz)

    The distance between successive frequencies in the modulated signal.

Samples per symbol

    The number of input samples that represent each modulated symbol.

Output data type

    The output type of the block can be specified here as `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `double`. By default, the block sets this to `double`.

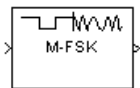**Pair Block**   M-FSK Modulator Baseband

See Also   CPFSK Demodulator Baseband

M-FSK Modulator Baseband

Purpose   Modulate using M-ary frequency shift keying method

Library   FM, in Digital Baseband sublibrary of Modulation

Description   The M-FSK Modulator Baseband block modulates using the M-ary
frequency shift keying method. The output is a baseband representation
of the modulated signal.

The **M-ary number** parameter, M, is the number of frequencies in
the modulated signal. The **Frequency separation** parameter is the
distance, in Hz, between successive frequencies of the modulated signal.

The sampling frequency must be greater than M multiplied by the
Frequency separation or the output signal will be aliased. Sampling
frequency is **Samples per symbol** divided by the input symbol period
(in seconds).

If the **Phase continuity** parameter is set to Continuous, then the
modulated signal maintains its phase even when it changes its
frequency. If the **Phase continuity** parameter is set to Discontinuous,
then the modulated signal comprises portions of M sinusoids of different
frequencies; thus, a change in the input value might cause a change in
the phase of the modulated signal.

Note that a model with the M-FSK Modulator block will not run if it
is used for sample-based signals unless **Tasking mode for periodic
sample times** (in **Simulation > Configuration Parameters >
Solver**) is set to SingleTasking.

Input Signal Values

The input and output for this block are discrete-time signals. The
Input type parameter determines whether the block accepts integers
between 0 and M-1, or binary representations of integers:

- If **Input type** is set to Integer, then the block accepts integers. The
  input can be either a scalar or a frame-based column vector of type
  int8, uint8, int16, uint16, int32, uint32, or a double with an
  integer value. They can also be boolean if the size of the alphabet is
  2 (i.e. M = 2).

- If **Input type** is set to `Bit`, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector (whose length is an integer multiple of K), and must be `boolean` or `double` typed, valued from the set {0, 1}. The **Symbol set ordering** parameter indicates how the block assigns binary words to corresponding integers.

  - If **Symbol set ordering** is set to `Binary`, then the block uses a natural binary-coded ordering.

  - If **Symbol set ordering** is set to `Gray`, then the block uses a Gray-coded ordering. For details about the Gray coding, see the reference page for theM-PSK Modulator Baseband block.

Whether the input is an integer or a binary representation of an integer, the block maps the integer 0 to the highest frequency and maps the integer M-1 to the lowest frequency. In baseband simulation, the lowest frequency is the negative frequency with the largest absolute value.

M-FSK Modulator Baseband

Dialog Box
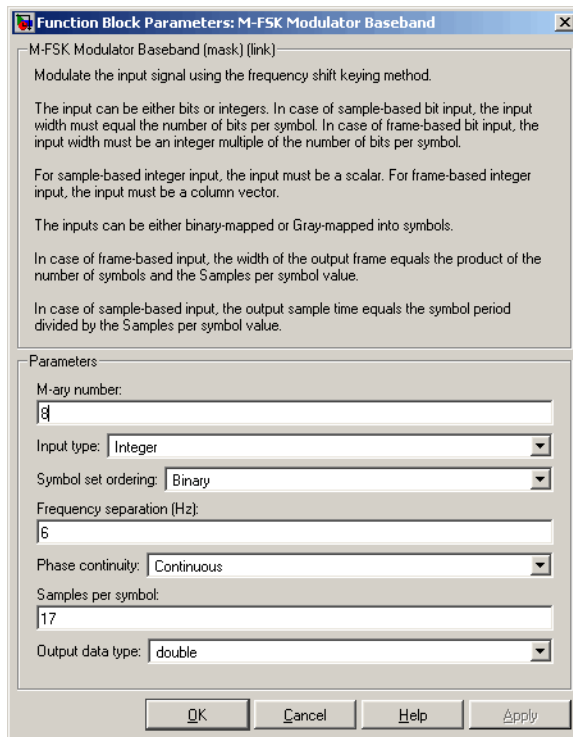
M-ary number

    The number of frequencies in the modulated signal.

Input type

    Indicates whether the input consists of integers or groups of bits. If this parameter is set to Bit, then the **M-ary number** parameter must be 2^K for some positive integer K.

Symbol set ordering

    Determines how the block maps each group of input bits to a corresponding integer.

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Phase continuity

Determines whether the modulated signal changes phases in a continuous or discontinuous way.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Output data type

The output type of the block can be specified as either a `double` or a `single`. By default, the block sets this to `double`.

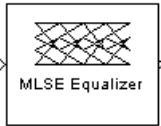**Pair Block**   M-FSK Demodulator Baseband

See Also   CPFSK Modulator Baseband

MLSE Equalizer

Purpose   Equalize using Viterbi algorithm

Library   Equalizers

Description   The MLSE Equalizer block uses the Viterbi algorithm to equalize a
linearly modulated signal through a dispersive channel. The block
receives a frame-based input signal and outputs the maximum
likelihood sequence estimate (MLSE) of the signal, using an estimate of
the channel modeled as a finite input response (FIR) filter.

This block supports single and double data types.

Channel Estimates

The channel estimate takes the form of a column vector containing the
coefficients of an FIR filter in descending order of powers. The length
of this vector is the channel memory, which must be a multiple of the
block's **Samples per input symbol** parameter.

To specify the channel estimate vector, use one of these methods:

- Set **Specify channel via** to Dialog and enter the vector in the
  Channel coefficients field.
- Set **Specify channel via** to Input port. The block displays an
  additional input port, labeled Ch, that receives a frame-based vector.

Signal Constellation

The **Signal constellation** parameter specifies the constellation for
the modulated signal, as determined by the modulator in your model.
Signal constellation is a vector of complex numbers, where the kth
complex number in the vector is the constellation point to which the
modulator maps the integer k-1.

Note The sequence of constellation points must be consistent between
the modulator in your model and the **Signal constellation** parameter
in this block.

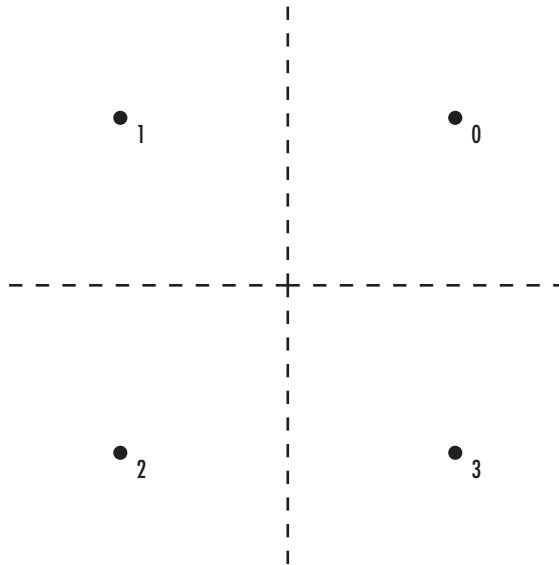For example, to specify the constellation given by the mapping

$$0 \rightarrow +1+i$$
$$1 \rightarrow -1+i$$
$$2 \rightarrow -1-i$$
$$3 \rightarrow +1-i$$

set **Constellation points** to [1+i, -1+i, -1-i, 1-i]. Note that the sequence of numbers in the vector indicates how the modulator maps integers to the set of constellation points. The labeled constellation is shown below.

Preamble and Postamble

If your data is accompanied by a preamble (prefix) or postamble (suffix), then configure the block accordingly:

- If you select **Input contains preamble**, then the **Expected preamble** parameter specifies the preamble that you expect to precede the data in the input signal.

- If you check the **Input contains postamble**, then the **Expected postamble** parameter specifies the postamble that you expect to follow the data in the input signal.

The **Expected preamble** or **Expected postamble** parameter must be a vector of integers between 0 and M-1, where M is the number of constellation points. An integer value of k-1 in the vector corresponds to the kth entry in the **Constellation points** vector and, consequently, to a modulator input of k-1.

The preamble or postamble must already be included at the beginning or end, respectively, of the input signal to this block. If necessary, you can concatenate vectors in Simulink software using the Matrix Concatenation block.

To learn how the block uses the preamble and postamble, see ""Reset Every Frame" Operation Mode" on page 2-422 below.

"Reset Every Frame" Operation Mode

One way that the Viterbi algorithm can transition between successive frames is called `Reset every frame` mode. You can choose this mode using the **Operation mode** parameter.

In `Reset every frame` mode, the block decodes each frame of data independently, resetting the state metric at the end of each frame. The traceback decoding always starts at the state with the minimum state metric.

The initialization of state metrics depends on whether you specify a preamble and/or postamble:

- If you do not specify a preamble, the decoder initializes the metrics of all states to 0 at the beginning of each frame of data.

- If you specify a preamble, the block uses it to initialize the state metrics at the beginning of each frame of data. More specifically, the

block decodes the preamble and assigns a metric of 0 to the decoded state. If the preamble does not decode to a unique state -- that is, if the length of the preamble is less than the channel memory -- the decoder assigns a metric of 0 to all states that can be represented by the preamble. Whenever you specify a preamble, the traceback path ends at one of the states represented by the preamble.

- If you do not specify a postamble, the traceback path starts at the state with the smallest metric.

- If you specify a postamble, the traceback path begins at the state represented by the postamble. If the postamble does not decode to a unique state, the decoder identifies the smallest of all possible decoded states that are represented by the postamble and begins traceback decoding at that state.

Note In Reset every frame mode, the input to the MLSE Equalizer block must contain at least T symbols, not including an optional preamble, where T is the **Traceback depth** parameter.

Continuous Operation Mode

An alternative way that the Viterbi algorithm can transition between successive frames is called Continuous with reset option mode. You can choose this mode using the **Operation mode** parameter.

In Continuous with reset option mode, the block initializes the metrics of all states to 0 at the beginning of the simulation. At the end of each frame, the block saves the internal state metric for use in computing the traceback paths in the next frame.

If you select the **Enable the reset input port** check box, the block displays another input port, labeled Rst. In this case, the block resets the state metrics whenever the scalar value at the Rst port is nonzero.

Decoding Delay

The MLSE Equalizer block introduces an output delay equal to the **Traceback depth** in the `Continuous with reset option` mode, and no delay in the `Reset every frame` mode.

Dialog Box
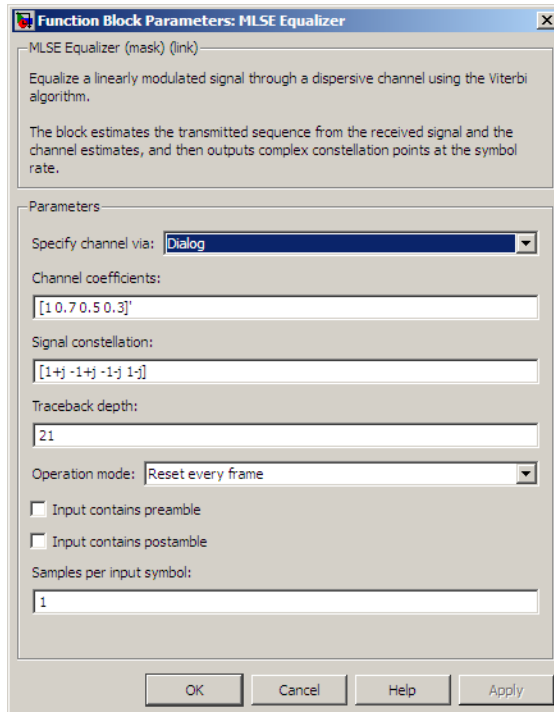
Specify channel via

> The method for specifying the channel estimate. If you select `Input port`, the block displays a second input port that receives the channel estimate. If you select `Dialog`, you can specify the channel estimate as a vector of coefficients for an FIR filter in the **Channel coefficients** field.

Channel coefficients

Vector containing the coefficients of the FIR filter that the block uses for the channel estimate. This field is visible only if you set **Specify channel via** to `Dialog`.

Signal constellation

Vector of complex numbers that specifies the constellation for the modulation.

Traceback depth

The number of trellis branches (equivalently, the number of symbols) the block uses in the Viterbi algorithm to construct each traceback path.

Operation mode

The operation mode of the Viterbi decoder. Choices are `Continuous with reset option` and `Reset every frame`.

Input contains preamble

When checked, you can set the preamble in the **Expected preamble** field. This option appears only if you set **Operation mode** to `Reset every frame`.

Expected preamble

Vector of integers between 0 and M-1 representing the preamble, where M is the size of the constellation. This field is visible and active only if you set **Operation mode** to `Reset every frame` and then select **Input contains preamble**.

Input contains postamble

When checked, you can set the postamble in the **Expected postamble** field. This option appears only if you set **Operation mode** to `Reset every frame`.

Expected postamble

Vector of integers between 0 and M-1 representing the postamble, where M is the size of the constellation. This field is visible and active only if you set **Operation mode** to `Reset every frame` and then select **Input contains postamble**.

Samples per input symbol

The number of input samples for each constellation point.

Enable the reset input port

When you check this box, the block has a second input port labeled Rst. Providing a nonzero input value to this port causes the block to set its internal memory to the initial state before processing the input data. This option appears only if you set **Operation mode** to Continuous with reset option.

See Also   LMS Linear Equalizer, LMS Decision Feedback Equalizer, RLS Linear Equalizer, RLS Decision Feedback Equalizer, CMA Equalizer
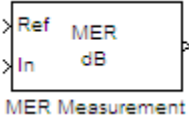
References   [1] Proakis, John G., *Digital Communications*, Fourth edition, New York, McGraw-Hill, 2001.

[2] Steele, Raymond, Ed., *Mobile Radio Communications*, Chichester, England, Wiley, 1996.

Purpose   Measure signal-to-noise ratio (SNR) in digital modulation applications

Library   Utility Blocks

Description

The Modulation Error Ratio (MER) is a measure of the signal-to-noise ratio (SNR) in digital modulation applications. You can use these types of measurements to determine system performance in communications applications. For example, determining if an EDGE system conforms to 3GPP radio transmission standards requires accurate RMS MER, Minimum MER, and 95th percentile for the MER measurements. The block measures all outputs in decibels (dB).

The MER block receives an ideal input signal (at reference port, Ref) and an AWGN corrupted signal (at input port, In) . The MER block then outputs a measure of the modulation accuracy by comparing these inputs. This output corresponds to the SNR of the AWGN channel.

MER output defaults to RMS MER in decibels (dB), with an option of Output minimum MER or Output X-percentile MER values. The minimum MER represents the best-case MER value per burst. For the X-percentile option, you can select to output the number of symbols processed in the percentile computations.

The following table shows the output type, the activation (what selects the output computation), computation units, and the corresponding computation duration.

Output	Activation	Units	Computation Duration
RMS MER	Default	Decibels	Per burst
Min MER	Parameter setting	Decibels	Per burst

Output	Activation	Units	Computation Duration
Percentile MER	Parameter setting	Decibels	Continuous
Number of symbols	Parameter setting if you select **Output X-percentile MER**	None	Continuous

Dimension

The block computes measurements for bursts of data. The data must be of length N symbols, where N is the size of the burst (sample-based or frame-based). The block computes a unique output for each incoming burst; therefore, the computation duration is per burst.

Input Signals

The input signals must be 1-D or 2-D sample-based column vectors or 2-D frame-based column vectors. The input and reference signals must have identical dimensions.

Output Signals

The output is always a scalar value.

Data Type

The block accepts double, single, and fixed-point data types. The output of the block is always double type.

Algorithms

MER is a measure of the SNR in a modulated signal calculated in dB. MER over N symbols is

$$MER = 10 * \log_{10} \left(\frac{\sum_{n=1}^{N} \left(I_k^2 + Q_k^2 \right)}{\sum_{n=1}^{N} \left(e_k \right)} \right) dB$$

The MER for the k_{th} symbol is

$$MER_k = 10 * \log_{10} \left(\frac{\frac{1}{N} \sum_{n=1}^{N} \left(I_k^2 + Q_k^2 \right)}{e_k} \right) db$$

The minimum MER represents the minimum MER value in a burst or

$$MER_{\min} = \min_{k \in [1,...,N]} \left\{ MER_k \right\}$$

where

$$e_k = (I_k - \tilde{I_k})^2 + (Q_k - \tilde{Q}_k)^2$$

I_k = In-phase measurement of the *kth* symbol in the burst

Q_k = Quadrature phase measurement of the *kth* symbol in the burst

I_k and Q_k represent ideal (reference) values. $\tilde{I}_k$ and $\tilde{Q}_k$ represent measured (received) symbols.

The block computes X-percentile MER by creating a histogram of all the incoming MER_k values. The output provides the MER value above which X% of the MER values lay.

MER Measurement

Dialog Box
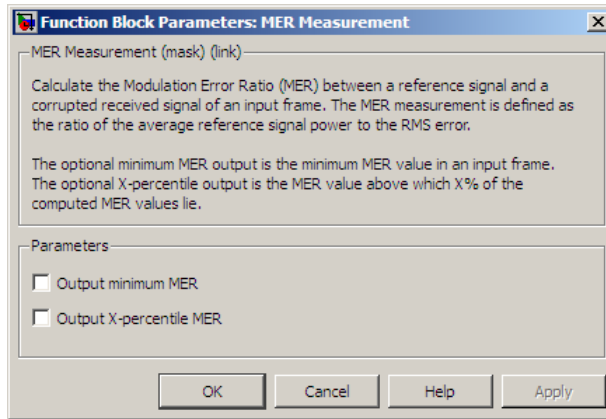
Output Minimum MER
>   Outputs the minimum MER of an input vector or frame.

Output X-percentile MER
>   Enables an output X-percentile MER measurement. When you
>   select this option, specify **X-percentile value (%)**.

X-Percentile value (%)
>   This parameter only appears when you select **Output
>   X-percentile MER**. The Xth percentile is the MER value above
>   which X% of all the computed MER values lie. The parameter
>   defaults to the 95th percentile. Therefore, 95% of all MER values
>   are above this output.

Output the number of symbols processed
>   Outputs the number of symbols that the block uses to compute the
>   **Output X-percentile MER**. This parameter only appears when
>   you select **Output X-percentile MER**.

See Also   EVM Measurement

References [1] *Digital Video Broadcasting (DVB): Measurement guidelines for DVB
systems*, DVB (ETSI) Standard ETR290, May 1997.

Purpose   Demodulate PAM-modulated data

Library   AM, in Digital Baseband sublibrary of Modulation

Description   The M-PAM Demodulator Baseband block demodulates a signal that
was modulated using M-ary pulse amplitude modulation. The input is a
baseband representation of the modulated signal.

The signal constellation has M points, where M is the **M-ary number**
parameter. M must be an even integer. The block scales the signal
constellation based on how you set the **Normalization method**
parameter. For details on the constellation and its scaling, see the
reference page for the M-PAM Modulator Baseband block.

The input can be either a scalar or a frame-based column vector and
must be of data type single, double, or signed fixed-point

Note All values of power assume a nominal impedance of 1 ohm.

Output Signal Values

The **Output type** parameter determines whether the block produces
integers or binary representations of integers. If **Output type** is set
to Integer, then the block produces integers. If **Output type** is set
to Bit, then the block produces a group of K bits, called a *binary word*, for
each symbol. The **Constellation ordering** parameter indicates how
the block assigns binary words to points of the signal constellation.
More details are on the reference page for the M-PAM Modulator
Baseband block.

Algorithm   The demodulator algorithm maps received input signal constellation
values to M-ary integer symbol indices between 0 and M-1 and then
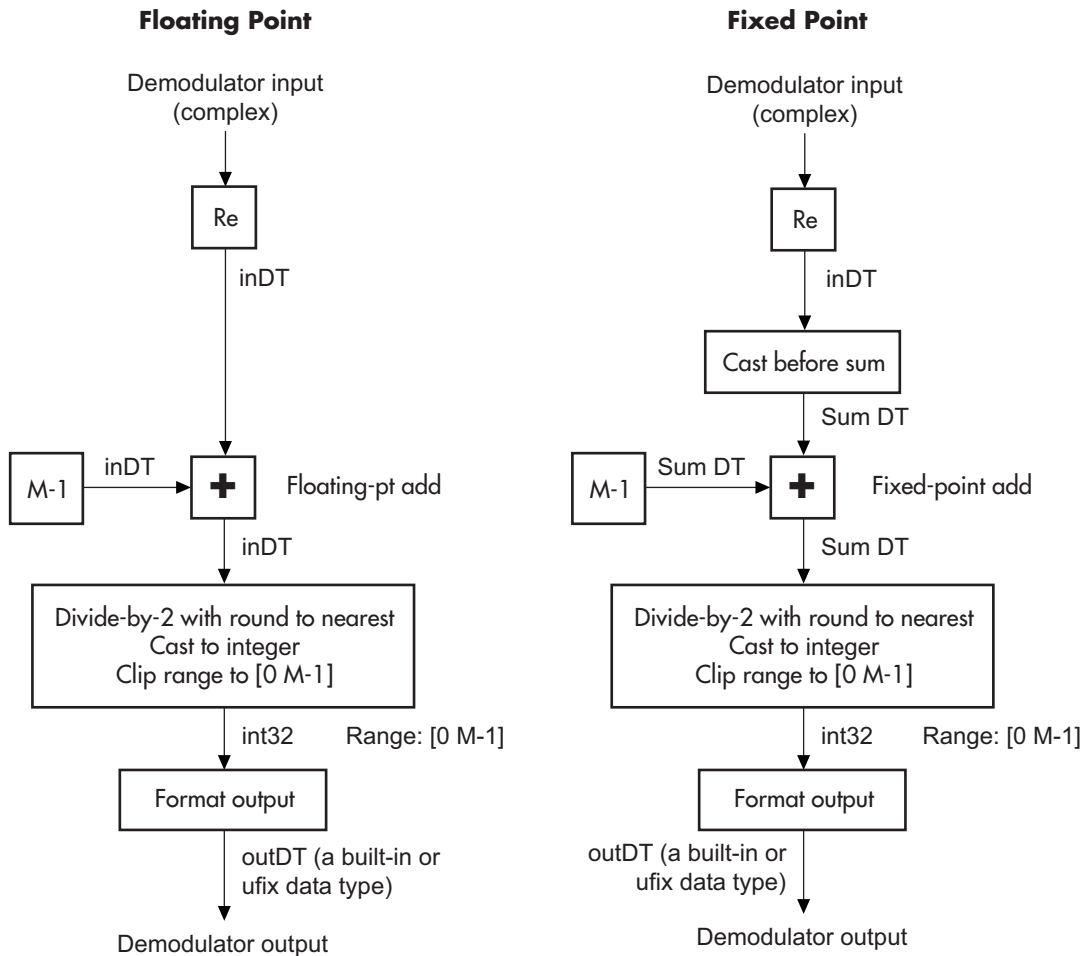maps these demodulated symbol indices to formatted output values.

The integer symbol index computation is performed by first scaling
the real part of the input signal constellation (possibly with noise) by
a denormalization factor derived from the **Normalization method**

M-PAM Demodulator Baseband

and related parameters. This denormalized value is added to M-1 to translate it into an approximate range between 0 and 2 x (M-1) plus noise. The resulting value is then rescaled via a divide-by-two (or, equivalently, a right-shift by one bit for fixed-point operation) to obtain a range approximately between 0 and M-1 (plus noise). The noisy index value is rounded to the nearest integer and clipped, via saturation, to the exact range of [0 M-1]. Finally, based on other block parameters, the integer index is mapped to a symbol value that is formatted and cast to the selected **Output data type**.
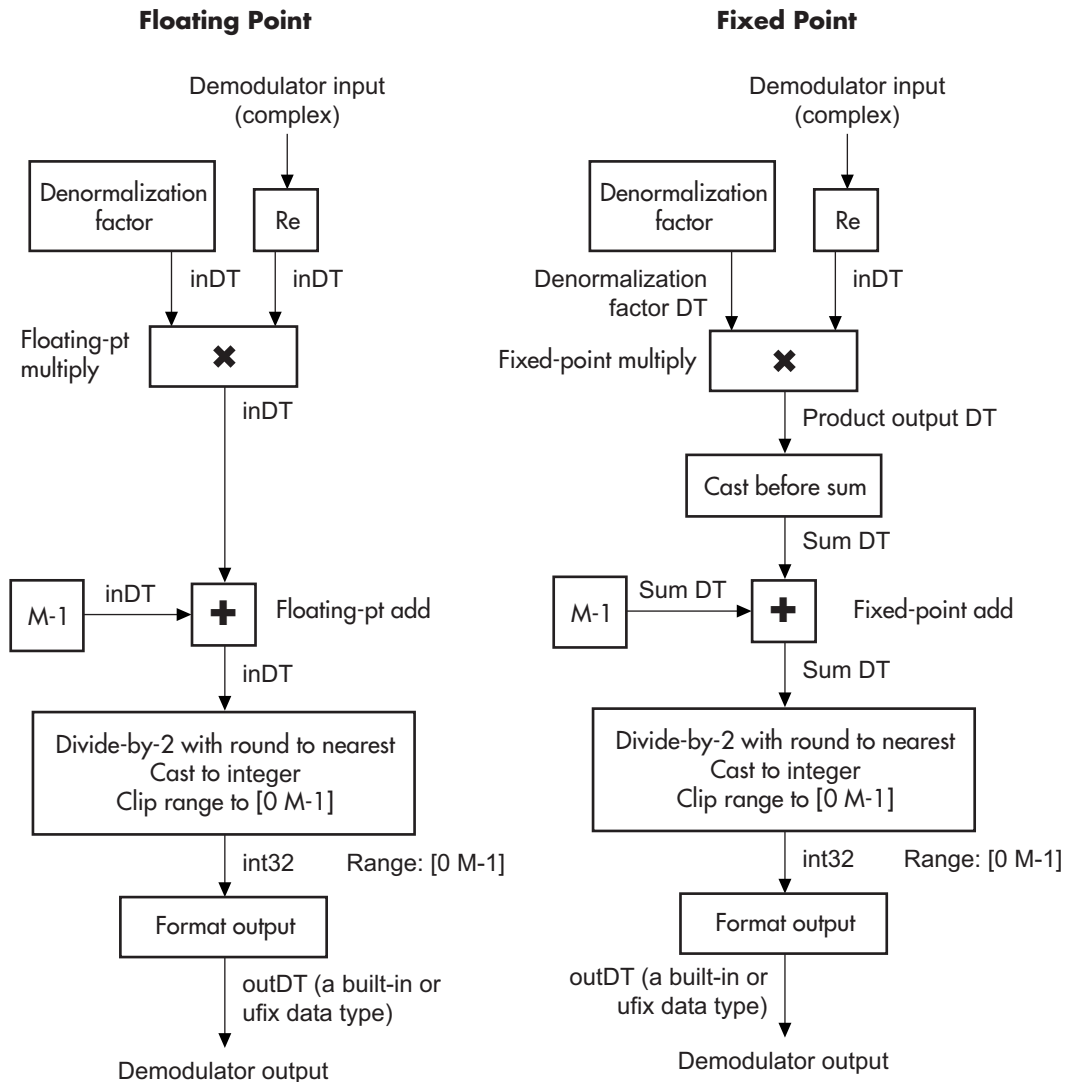
The following figures contains signal flow diagrams for floating-point and fixed-point algorithm operation. The floating-point diagrams apply when the input signal data type is `double` or `single`. The fixed-point diagrams apply when the input signal is a signed fixed-point data type. Note that the diagram is simplified when using normalized constellations (i.e., denormalization factor is 1).
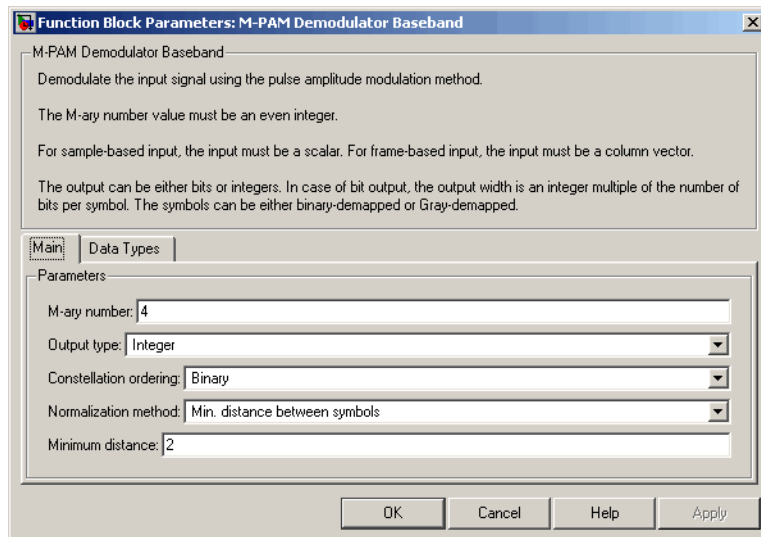
Floating Point

Demodulator input
(complex)

Re

inDT

M-1 → inDT → ➕ Floating-pt add

inDT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 M-1]

int32   Range: [0 M-1]

Format output

outDT (a built-in or
ufix data type)

Demodulator output

Fixed Point

Demodulator input
(complex)

Re

inDT

Cast before sum

Sum DT

M-1 → Sum DT → ➕ Fixed-point add

Sum DT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 M-1]

int32   Range: [0 M-1]

Format output

outDT (a built-in or
ufix data type)

Demodulator output

Signal-Flow Diagrams with Denormalization Factor Equal to 1

M-PAM Demodulator Baseband

Floating Point

Demodulator input
(complex)

Denormalization factor

Re

inDT · inDT

Floating-pt multiply ✖

inDT

M-1 — inDT → ➕ Floating-pt add

inDT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 M-1]

int32   Range: [0 M-1]

Format output

outDT (a built-in or ufix data type)

Demodulator output

Fixed Point

Demodulator input
(complex)

Denormalization factor

Re

Denormalization factor DT · inDT

Fixed-point multiply ✖

Product output DT

Cast before sum

Sum DT

M-1 — Sum DT → ➕ Fixed-point add

Sum DT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 M-1]

int32   Range: [0 M-1]

Format output

outDT (a built-in or ufix data type)

Demodulator output

Signal-Flow Diagrams with Nonunity Denormalization Factor

**Dialog
Box**

M-ary number

The number of points in the signal constellation. It must be an even integer.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to Bit, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to Bit.

Normalization method

Determines how the block scales the signal constellation. Choices are Min. distance between symbols, Average Power, and Peak Power.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to `Min. distance between symbols`.

Average power, referenced to 1 ohm (watts)

The average power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Average Power`.

Peak power, referenced to 1 ohm (watts)

The maximum power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Peak Power`.

Output

> When the parameter is set to `'Inherit via internal rule'`
> (default setting), the block will inherit the output data type from
> the input port. The output data type will be the same as the input
> data type if the input is of type `single` or `double`. Otherwise, the
> output data type will be as if this parameter is set to `'Smallest
> unsigned integer'`.
>
> When the parameter is set to `'Smallest unsigned integer'`,
> the output data type is selected based on the settings used in
> the **Hardware Implementation** pane of the Configuration
> Parameters dialog box of the model. If `ASIC/FPGA` is selected
> in the **Hardware Implementation** pane, the output data
> type is the ideal minimum size, i.e., `ufix(1)` for bit outputs,
> and `ufix(ceil(log2(M)))` for integer outputs. For all other
> selections, it is an unsigned integer with the smallest available
> word length large enough to fit the ideal minimum size, usually
> corresponding to the size of a char (e.g., `uint8`).
>
> For integer outputs, this parameter can be set to `Smallest
> unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`,
> `single`, and `double`. For bit outputs, the options are `Smallest
> unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`,
> `boolean`, `single`, or `double`.

Denormalization factor

> This parameter applies when a fixed-point input is not
> normalized. It can be set to `Same word length as input` or
> `Specify word length`, in which case a field is enabled for user
> input. A best-precision fraction length is always used.

Product output

> This parameter only applies when the input is a fixed-point signal
> and there is a nonunity (not equal to 1) denormalized factor. It
> can be set to `Inherit via internal rule` or `Specify word
> length`, which enables a field for user input.

Setting to `Inherit via internal rule` computes the full-precision product word length and fraction length. Inherit Rule for Product Data Types in *Signal Processing Blockset User's Guide* describes the full-precision Product output internal rule.

Setting to `Specify word length` allows you to define the word length. The block computes a best-precision fraction length based on the word length specified and the pre-computed worst-case (min/max) real world value **Product output** result. The worst-case **Product output** result is precomputed by multiplying the denormalized factor with the worst-case (min/max) input signal range, purely based on the input signal data type.

The block uses the **Rounding** method when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Sum

This parameter only applies when the input is a fixed-point signal. It can be set to `Inherit via internal rule`, `Same as product output`, or `Specify word length`, in which case a field is enabled for user input

Setting `Inherit via internal rule` computes the full-precision sum word length and fraction length, based on the two inputs to the Sum in the fixed-point Hard Decision Algorithm signal flow diagram. The rule is the same as the fixed-point inherit rule of the internal **Accumulator data type** parameter in the Simulink Sum block.

Setting `Specify word length` allows you to define the word length. A best precision fraction length is computed based on the word length specified in the pre-computed maximum range necessary for the demodulated algorithm to produce accurate

results. The signed fixed-point data type that has the best precision fully containing the values in the range 2 * (M-1) for the specified word length.

Setting to `Same as product output` allows the Sum data type to be the same as the **Product output** data type (when **Product output** is used). If the **Product output** is not used, then this setting will be ignored and the `Inherit via internal rule` Sum setting will be used.

Pair Block   M-PAM Modulator Baseband

See Also   General QAM Demodulator Baseband

M-PAM Modulator Baseband

Purpose   Modulate using M-ary pulse amplitude modulation

Library   AM, in Digital Baseband sublibrary of Modulation

Description The M-PAM Modulator Baseband block modulates using M-ary pulse
amplitude modulation. The output is a baseband representation of the
modulated signal. The **M-ary number** parameter, M, is the number of
points in the signal constellation. It must be an even integer.

Note All values of power assume a nominal impedance of 1 ohm.

Constellation Size and Scaling

Baseband M-ary pulse amplitude modulation using the block's default
signal constellation maps an integer m between 0 and M-1 to the
complex value

$$2m - M + 1$$

Note This value is actually a real number. The block's output signal is
a complex data-type signal whose imaginary part is zero.

The block scales the default signal constellation based on how you set
the **Normalization method** parameter. The following table lists the
possible scaling conditions.

Value of Normalization Method Parameter	Scaling Condition
`Min. distance between symbols`	The nearest pair of points in the constellation is separated by the value of the **Minimum distance** parameter

Value of Normalization Method Parameter	Scaling Condition
Average Power	The average power of the symbols in the constellation is the **Average power** parameter
Peak Power	The maximum power of the symbols in the constellation is the **Peak power** parameter

Input Signal Values

The input and output for this block are discrete-time signals. The **Input type** parameter determines whether the block accepts integers between 0 and M-1, or binary representations of integers.

- If **Input type** is set to Integer, then the block accepts integers. The input can be either a scalar or a frame-based column vector of data type int8, uint8, int16, uint16, int32, uint32, single, or double.

- If **Input type** is set to Bit, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K. For bit inputs, the block can accept int8, uint8, int16, uint16, int32, uint32, boolean, single, and double. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation.

  - If **Constellation ordering** is set to Binary, then the block uses a natural binary-coded constellation.

 - If **Constellation ordering** is set to Gray, then the block uses a Gray-coded constellation.

For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block.

Constellation Visualization

The M-PAM Modulator Baseband block provides the capability to visualize a signal constellation from the block mask. This Constellation Visualization feature allows you to visualize a signal constellation for specific block parameters. For more information, see the Constellation Visualization section of the *Communications Blockset User's Guide*.

Dialog Box
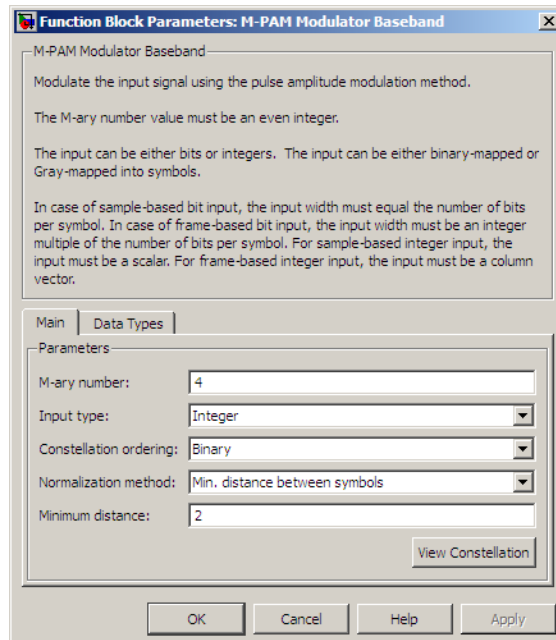
M-ary number

The number of points in the signal constellation. It must be an even integer.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to `Bit`, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer.

Normalization method

Determines how the block scales the signal constellation. Choices are `Min. distance between symbols`, `Average Power`, and `Peak Power`.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to `Min. distance between symbols`.

Average power, referenced to 1 ohm (watts)

The average power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Average Power`.

Peak power, referenced to 1 ohm (watts)

The maximum power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Peak Power`.

Output data type

The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Setting this parameter to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this parameter to `Inherit via back propagation`, sets the output data type and scaling to match the following block.

Output word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

M-PAM Modulator Baseband

User-defined data type

Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point software. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set output fraction length to

Specify the scaling of the fixed-point output by either of the following methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter or when you select `User-defined` and the specified output data type is a fixed-point data type.

Output fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   M-PAM Demodulator Baseband

See Also   General QAM Modulator Baseband

Purpose   Demodulate PSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description The M-PSK Demodulator Baseband block demodulates a signal that
was modulated using the M-ary phase shift keying method. The input
is a baseband representation of the modulated signal. The input and
output for this block are discrete-time signals. The **M-ary number**
parameter, M, is the number of points in the signal constellation. The
input can be either a scalar or a frame-based column vector of data
types double, single, or signed fixed-point (when M <= 8, and in
hard-decision modes only).

Binary or Integer Outputs

If the **Output type** parameter is set to Integer, the block outputs
integers between 0 and M-1, representing M-ary symbols.

If the **Output type** parameter is set to Bit and **Decision type** is set to
Hard decision, the block outputs binary representations of integers
between 0 and M-1. It outputs a group of K bits, called a *binary word*,
for each symbol.

Algorithm   For M=2, refer to the BPSK Demodulator Baseband block reference
page.

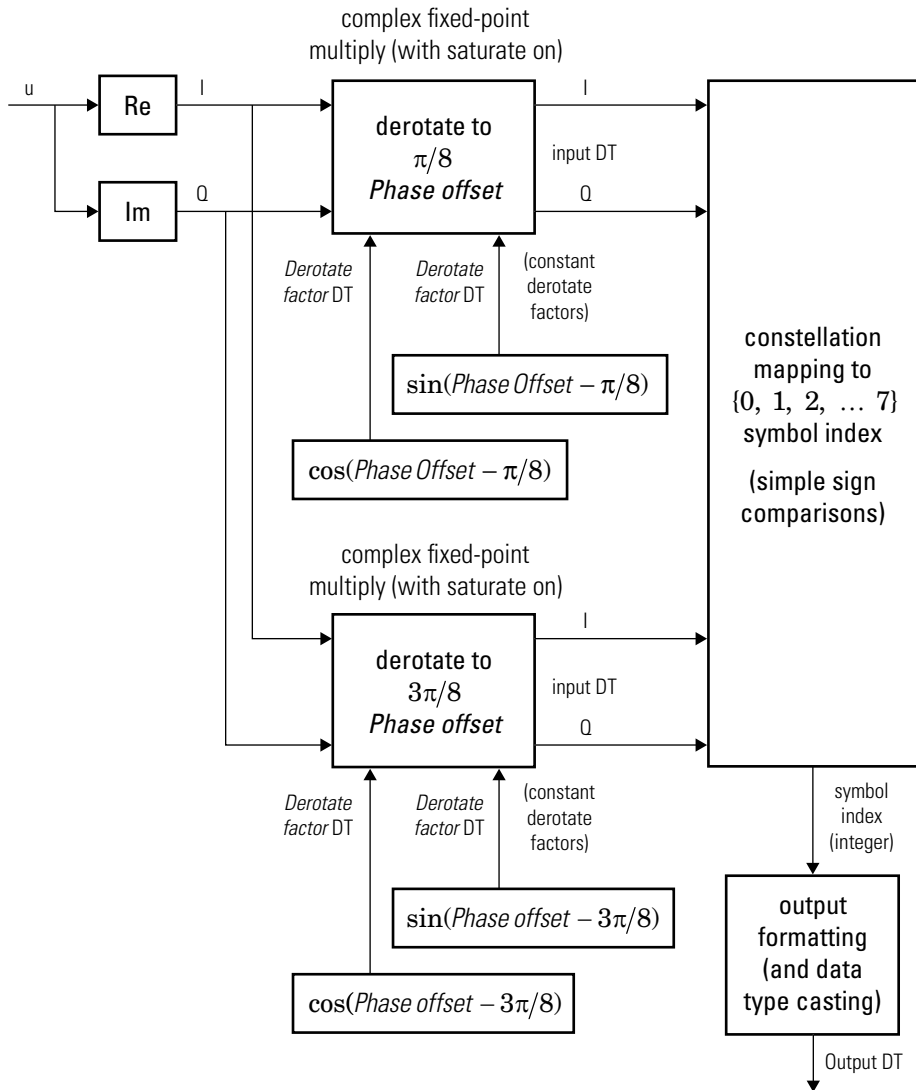For M=4, refer to the QPSK Demodulator Baseband block reference
page.

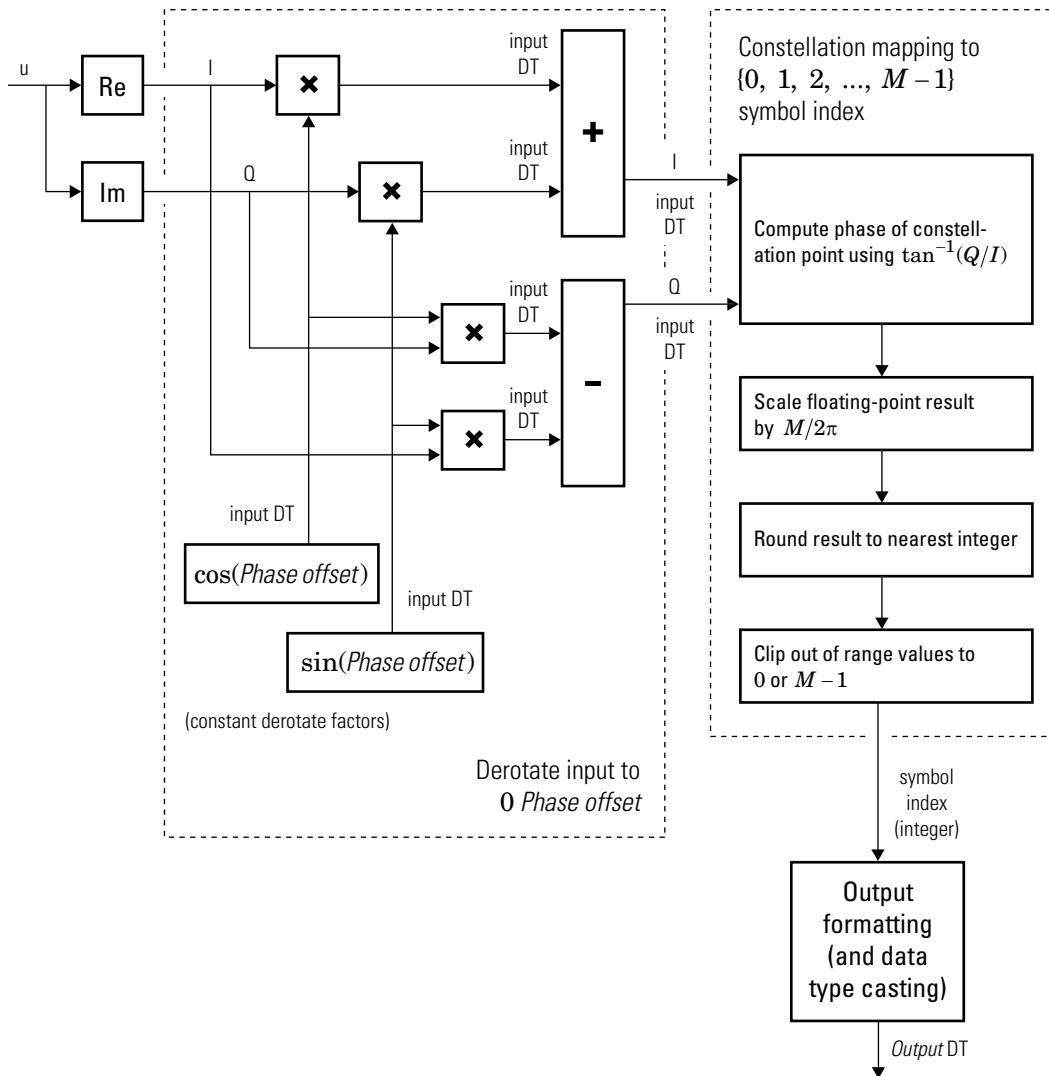For M=8 and greater, see the following signal diagrams.

M-PSK Demodulator Baseband

Hard-Decision 8-PSK Demodulator Floating-Point Signal Diagram

complex fixed-point
multiply (with saturate on)

u → Re I

Im Q

derotate to
$\pi/8$
Phase offset

I

input DT

Q

*Derotate
factor* DT

*Derotate
factor* DT

(constant
derotate
factors)

$\sin(Phase\ Offset - \pi/8)$

$\cos(Phase\ Offset - \pi/8)$

constellation
mapping to
$\{0,\ 1,\ 2,\ \dots\ 7\}$
symbol index

(simple sign
comparisons)

complex fixed-point
multiply (with saturate on)

derotate to
$3\pi/8$
Phase offset

I

input DT

Q

*Derotate
factor* DT

*Derotate
factor* DT

(constant
derotate
factors)

$\sin(Phase\ offset - 3\pi/8)$

$\cos(Phase\ offset - 3\pi/8)$

symbol
index
(integer)

output
formatting
(and data
type casting)

Output DT

Hard-Decision 8-PSK Demodulator Fixed-Point Signal Diagram

M-PSK Demodulator Baseband

Hard-Decision M-PSK Demodulator (M > 8) Floating-Point Signal Diagram for Nontrivial Phase Offset

For *M* > 8, in order to improve speed and implementation costs, no derotation arithmetic is performed when **Phase offset** is 0, $\pi/2$, π, or $3\pi/2$ (i.e., when it is trivial).

Also, for *M* > 8, this block will only support inputs of type double and single.

The exact LLR and approximate LLR cases (soft-decision) are described in "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide*.

Dialog Box
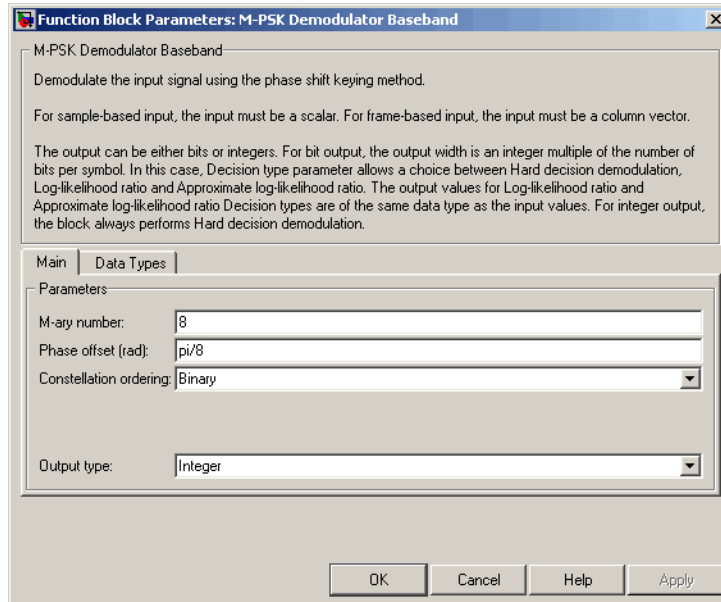
M-ary number
> The number of points in the signal constellation.

Phase offset
> The phase of the zeroth point of the signal constellation.

M-PSK Demodulator Baseband

Constellation ordering

> Determines how the block maps a symbol to the corresponding K output bits or integer. See the reference page for the M-PSK Modulator Baseband block for details. Selecting `User-defined` displays the field **Constellation mapping**, allowing for user-specified mapping.

Constellation mapping

> This field appears when `User-defined` is selected in the drop-down list **Constellation ordering**.
>
> This parameter is a row or column vector of size M and must have unique integer values in the range [0, M-1]. The values must be of data type `double`.
>
> The first element of this vector corresponds to the constellation point at 0 angle, with subsequent elements running counterclockwise. The last element corresponds to the -pi/M constellation point.

Output type

> Determines whether the output consists of integers or groups of bits. If this parameter is set to `Bit`, the **M-ary number** parameter must be 2^K for some positive integer K.

Decision type

> This field appears when `Bit` is selected in the drop-down list **Output type**.
>
> Specifies the output to be bitwise hard decision, LLR, or approximate LLR. See "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide* for algorithm details.

Noise variance source

> This field appears when `Approximate log-likelihood ratio` or `Log-likelihood ratio` is selected for **Decision type**.

When set to `Dialog`, the noise variance can be specified in the **Noise variance** field. When set to `Port`, a port appears on the block through which the noise variance can be input.

Noise variance

This parameter appears when the **Noise variance source** is set to `Dialog` and specifies the noise variance in the input signal. This parameter is tunable in normal mode, Accelerator mode and Rapid Accelerator mode.
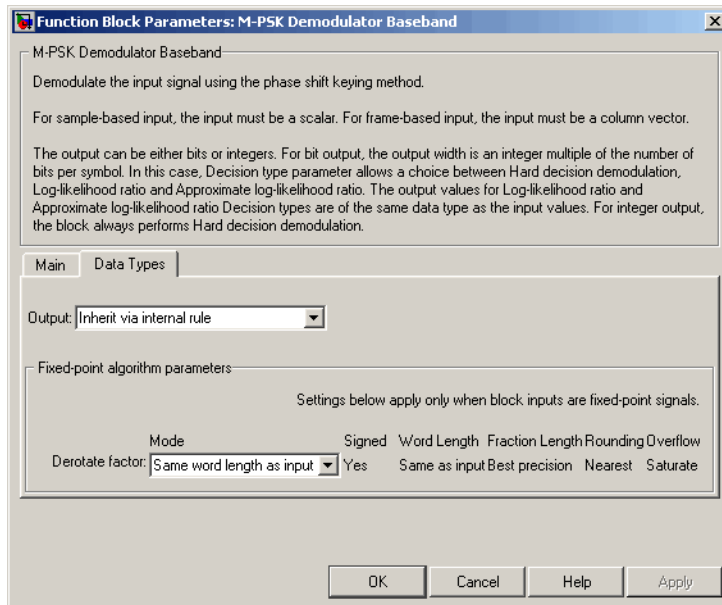
If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameter without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

The LLR algorithm involves computing exponentials of very large or very small numbers using finite precision arithmetic and would yield:

- `Inf` to `-Inf` if **Noise variance** is very high
- `NaN` if **Noise variance** and signal power are both very small

In such cases, use approximate LLR, as its algorithm does not involve computing exponentials.

M-PSK Demodulator Baseband

Data Types Pane for Hard-Decision

Output

> For bit outputs, when **Decision type** is set to `Hard decision`, the output data type can be set to `'Inherit via internal rule'`, `'Smallest unsigned integer'`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`.
>
> For integer outputs, the output data type can be set to `'Inherit via internal rule'`, `'Smallest unsigned integer'`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.
>
> When this parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is a floating-point type (`single` or `double`). If the input data type is fixed-point (supported only when **M-ary**

number is 2, 4, or 8), the output data type will work as if this parameter is set to `'Smallest unsigned integer'`.

When this parameter is set to `'Smallest unsigned integer'`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model.

If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, and **Output type** is `Bit`, the output data type is the ideal minimum one-bit size, i.e., `ufix(1)`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., `uint8`).

If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, and **Output type** is `Integer`, the output data type is the ideal minimum integer size, i.e., `ufix(ceil(log2(M)))`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit the ideal minimum size, usually corresponding to the size of a char (e.g., `uint8`).

Derotate factor

This parameter only applies when **M-ary number** is 2, 4, or 8, the input is fixed-point, and **Phase offset** is nontrivial (multiple of $\frac{\pi}{2}$ for M=2, odd multiple of $\frac{\pi}{4}$ for M=4, or any value for M=8).

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input.

M-PSK Demodulator Baseband
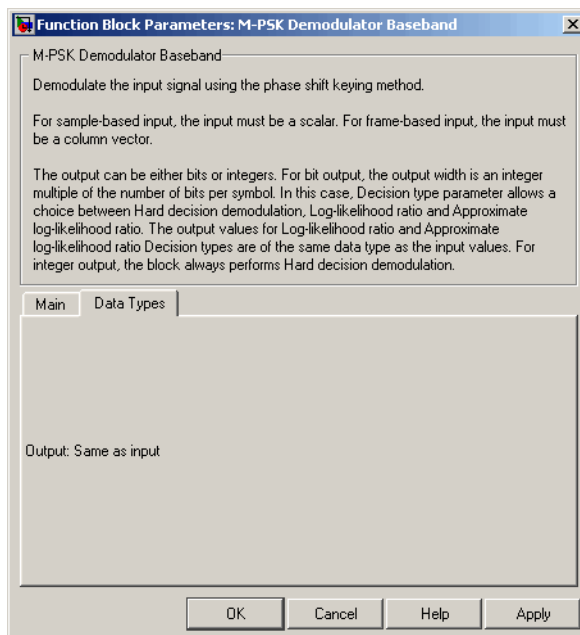
Data Types Pane for Soft-Decision

For bit outputs, when **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, the output data type is inherited from the input (e.g., if the input is of data type `double`, the output is also of data type `double`).
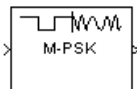
Pair Block   M-PSK Modulator Baseband

See Also   BPSK Demodulator Baseband, QPSK Demodulator Baseband, M-DPSK Demodulator Baseband

M-PSK Modulator Baseband

Purpose   Modulate using M-ary phase shift keying method

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The M-PSK Modulator Baseband block modulates using the M-ary
phase shift keying method. The output is a baseband representation
of the modulated signal. The **M-ary number** parameter, M, is the
number of points in the signal constellation.

The input and output for this block are discrete-time signals. For
integer inputs, the block can accept the data types `int8`, `uint8`, `int16`,
`uint16`, `int32`, `uint32`, `single`, and `double`. For bit inputs, the block
can accept `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`,
`single`, and `double`.

Alternative configurations of the block determine how the block
interprets its input and arranges its output, as explained in the
following sections.

Binary Inputs

If the **Input type** parameter is set to `Bit` and the **M-ary number**
parameter has the form 2^K for some positive integer K, the block accepts
binary representations of integers between 0 and M-1. It modulates
each group of K bits, called a *binary word*. The input can be either a
vector of length K or a frame-based column vector whose length is an
integer multiple of K. For more information, see "Binary-Valued and
Integer-Valued Signals" in *Communications BlocksetUser's Guide*.

Frame-Based Inputs

If the input is a frame-based column vector, the block processes several
integers or several binary words in each time step. (If the **Input type**
parameter is set to `Bit`, a binary word consists of $\log_2(M)$ bits.)

For example, the following schematics illustrate how the block processes
two 8-ary integers or binary words in one time step. The signals
involved are all frame-based column vectors. In both cases, the **Phase
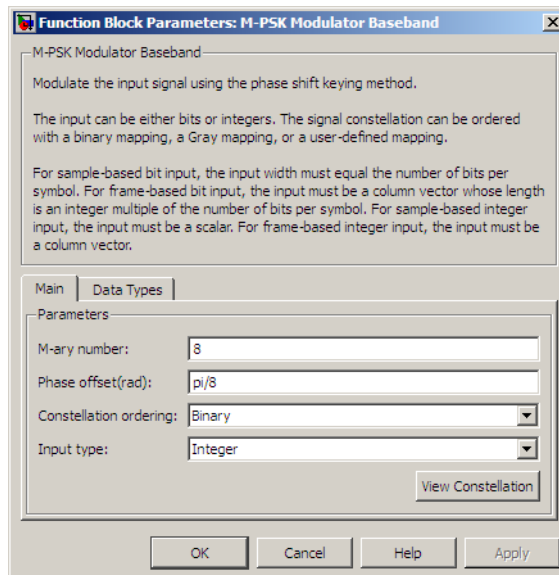offset** parameter is 0.

M-PSK Modulator Baseband

Input type parameter is **Integer** and **Constellation ordering** parameter is **Binary**.

Input type parameter is **Bit** and **Constellation ordering** parameter is **Binary**.
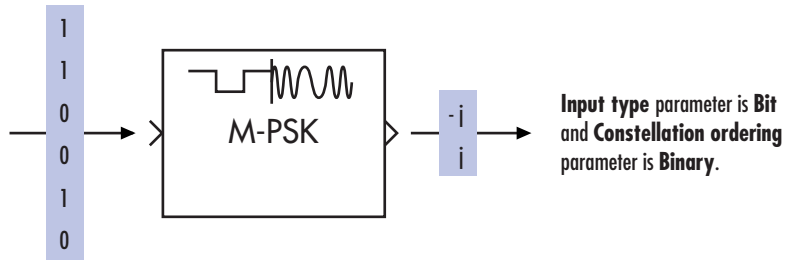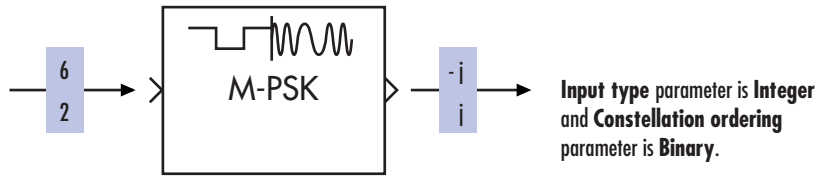
Constellation Visualization

The M-PSK Modulator Baseband block provides the capability to visualize a signal constellation from the block mask. This Constellation Visualization feature allows you to visualize a signal constellation for specific block parameters. For more information, see the Constellation Visualization section of the *Communications Blockset User's Guide*.

Dialog Box

M-PSK Modulator Baseband

Modulate the input signal using the phase shift keying method.

The input can be either bits or integers. The signal constellation can be ordered with a binary mapping, a Gray mapping, or a user-defined mapping.

For sample-based bit input, the input width must equal the number of bits per symbol. For frame-based bit input, the input must be a column vector whose length is an integer multiple of the number of bits per symbol. For sample-based integer input, the input must be a scalar. For frame-based integer input, the input must be a column vector.

Main | Data Types

Parameters

M-ary number: 8

Phase offset(rad): pi/8

Constellation ordering: Binary

Input type: Integer

View Constellation

OK | Cancel | Help | Apply

M-ary number
> The number of points in the signal constellation.

Phase offset
> The phase of the zeroth point of the signal constellation.

Constellation ordering
> Determines how the block maps an integer or group of K input bits to the corresponding symbol.
>
> If set to Binary, baseband M-ary phase shift keying modulation with a phase offset of θ maps an integer m between 0 and M-1 to the complex value
>
> $$\exp(j\theta + j2\pi m/M)$$
>
> If set to Gray, the block uses a Gray-coded signal constellation; as a result, binary representations that differ in more than one

bit cannot map to consecutive integers modulo M. The explicit mapping is described in "Algorithm" on page 2-459.

Selecting `User-defined` displays the field **Constellation mapping**, which allows for user-specified mapping.

Constellation mapping
This field appears when `User-defined` is selected in the drop-down list **Constellation ordering**.

This parameter is a row or column vector of size M and must have unique integer values in the range [0, M-1]. The values must be of data type `double`.

The first element of this vector corresponds to the constellation point at 0 + **Phase offset** angle, with subsequent elements running counterclockwise. The last element corresponds to the -pi/M + **Phase offset** constellation point.

Input type
Indicates whether the input consists of integers or groups of bits.

If this parameter is set to `Bit`, the **M-ary number** parameter must be 2^K for some positive integer K. K consecutive elements in each channel or column represent a symbol, where $K = \log2(M)$. The number of elements in each channel must be an integer multiple of K, and each element must be 0 or 1.

To use integers between 0 and M-1 as input values, set this to `Integer`. In this case, the input can be either a scalar or a frame-based column vector.

Output data type
The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Setting this to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this to `Inherit`

via back propagation, sets the output data type and scaling to match the following block.

Output word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select Fixed-point for the **Output data type** parameter.

User-defined data type

Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the sfix, sint, sfrac, and fixdt functions from Simulink Fixed Point software. This parameter is only visible when you select User-defined for the **Output data type** parameter.

Output fraction length

Specify the scaling of the fixed-point output by either of the following methods:

- Choose Best precision to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose User-defined to specify the output scaling in the **Output fraction length** parameter.

This parameter is only visible when you select Fixed-point for the **Output data type** parameter or when you select User-defined and the specified output data type is a fixed-point data type.

Output fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select Fixed-point or User-defined for the **Output data type** parameter and User-defined for the **Set output fraction length to** parameter.

Algorithm   If the **Constellation ordering** parameter is set to Gray, the block internally assigns the binary inputs to points of a predefined Gray-coded

signal constellation. The block's predefined M-ary Gray-coded signal constellation assigns the binary representation

```
de2bi(bitxor(M,floor(M/2)), log2(M),'left-msb')
```

to the Mth phase. The zeroth phase in the constellation is the **Phase offset** parameter, and successive phases are counted in a counterclockwise direction.

Note This transformation might seem counterintuitive because it constitutes a Gray-to-binary mapping. However, the block must use it to impose a Gray ordering on the signal constellation, which has a natural binary ordering.

In other words, if the block input is the natural binary representation, *u*, of the integer *U*, the block output has phase

$$j\theta + j2\pi m/M$$

where θ is the **Phase offset** parameter and *m* is an integer between 0 and M-1 that satisfies

$$m \text{ XOR } \lfloor m/2 \rfloor = U$$

For example, if M = 8, the binary representations that correspond to the zeroth through seventh phases are as follows.

```
M = 8; m = [0:M-1]';
de2bi(bitxor(m,floor(m/2)), log2(M),'left-msb')

ans =

     0     0     0
     0     0     1
     0     1     1
     0     1     0
```

```
1      1      0
1      1      1
1      0      1
1      0      0
```

The following diagram shows the 8-ary Gray-coded constellation that the block uses if the **Phase offset** parameter is π/8.
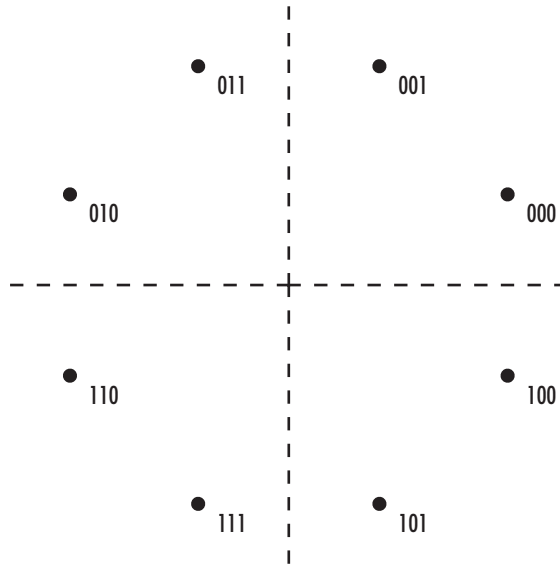
Pair Block   M-PSK Demodulator Baseband

See Also   BPSK Modulator Baseband, QPSK Modulator Baseband, M-DPSK Modulator Baseband

M-PSK Phase Recovery

Purpose   Recover carrier phase using M-Power method

Library   Carrier Phase Recovery sublibrary of Synchronization
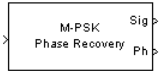
Description The M-PSK Phase Recovery block recovers the carrier phase of the input signal using the M-Power method. This feedforward, non-data-aided, clock-aided method is suitable for systems that use baseband phase shift keying (PSK) modulation. It is also suitable for systems that use baseband quadrature amplitude modulation (QAM), although the results are less accurate than those for comparable PSK systems. The alphabet size for the modulation must be an even integer.

For PSK signals, the **M-ary number** parameter is the alphabet size. For QAM signals, the **M-ary number** should be 4 regardless of the alphabet size because the 4-power method is the most appropriate for QAM signals.

The M-Power method assumes that the carrier phase is constant over a series of consecutive symbols, and returns an estimate of the carrier phase for the series. The **Observation interval** parameter is the number of symbols for which the carrier phase is assumed constant. This number must be an integer multiple of the input signal's vector length.

Input and Outputs

The input signal must be a frame-based column vector or a sample-based scalar of type `double` or `single`. The input signal represents a baseband signal at the symbol rate, so it must be complex-valued and must contain one sample per symbol.

The outputs are as follows:

- The output port labeled `Sig` gives the result of rotating the input signal counterclockwise, where the amount of rotation equals the carrier phase estimate. The `Sig` output is thus a corrected version of the input signal, and has the same sample time and vector size as the input signal.

- The output port labeled Ph outputs the carrier phase estimate, in
  degrees, for all symbols in the observation interval. The Ph output is
  a scalar signal.

Note Because the block internally computes the argument of
a complex number, the carrier phase estimate has an inherent
ambiguity. The carrier phase estimate is between -180/M and 180/M
degrees and might differ from the actual carrier phase by an integer
multiple of 360/M degrees.

Delays and Latency

The block's algorithm requires it to collect symbols during a period of
length **Observation interval** before computing a single estimate of the
carrier phase. Therefore, each estimate is delayed by **Observation
interval** symbols and the corrected signal has a latency of **Observation
interval** symbols, relative to the input signal.

**Dialog
Box**
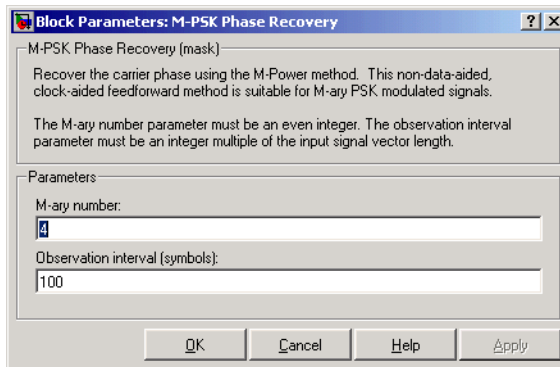
M-ary number

> The number of points in the signal constellation of the transmitted
> PSK signal, or 4 for a QAM signal. This must be an even integer.

M-PSK Phase Recovery

Observation interval
> The number of symbols for which the carrier phase is assumed constant.

Examples   See "Carrier Phase Recovery Example" in *Communications Blockset User's Guide*.

Algorithm   If the symbols occurring during the observation interval are x(1), x(2), x(3),..., x(L), then the resulting carrier phase estimate is

$$\frac{1}{M}\arg\left\{\sum_{k=1}^{L}(x(k))^M\right\}$$

where the arg function returns values between -180 degrees and 180 degrees.

References   [1] Mengali, Umberto, and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

[2] Moeneclaey, Marc, and Geert de Jonghe, "ML-Oriented NDA Carrier Synchronization for General Rotationally Symmetric Signal Constellations," *IEEE Transactions on Communications*, Vol. 42, No. 8, Aug. 1994, pp. 2531-2533.

See Also   CPM Phase Recovery, M-PSK Modulator Baseband

Purpose   Decode trellis-coded modulation data, modulated using PSK method

Library   Trellis-Coded Modulation
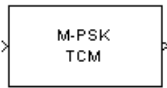
Description   The M-PSK TCM Decoder block uses the Viterbi algorithm to decode a trellis-coded modulation (TCM) signal that was previously modulated using a PSK signal constellation.

The **M-ary number** parameter is the number of points in the signal constellation, which also equals the number of possible output symbols from the convolutional encoder. (That is, $\log_2$(**M-ary number**) is the number of output bit streams from the convolutional encoder.)

The **Trellis structure** and **M-ary number** parameters in this block should match those in theM-PSK TCM Encoder block, to ensure proper decoding.

Input and Output Signals

The input signal must be a frame-based column vector containing complex numbers.

If the convolutional encoder described by the trellis structure represents a rate k/n code, then the M-PSK TCM Decoder block's output is a frame-based binary column vector whose length is k times the vector length of the input signal.

The input signal must be double or single. The reset port accepts double or boolean.

Operation Modes

The block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses. This parameter also affects the range of possible values for the **Traceback depth** parameter, D.

- In Continuous mode, the block initializes all state metrics to zero at the beginning of the simulation, waits until it accumulates D symbols, and then uses a sequence of D symbols to compute each of the traceback paths. D can be any positive integer. At the end of

2-465

each frame, the block saves its internal state metric for use with the next frame.

If you select the **Enable the reset input** check box, the block displays another input port, labeled Rst. This port receives an integer scalar signal. Whenever the value at the Rst port is nonzero, the block resets all state metrics to zero and sets the traceback memory to zero.
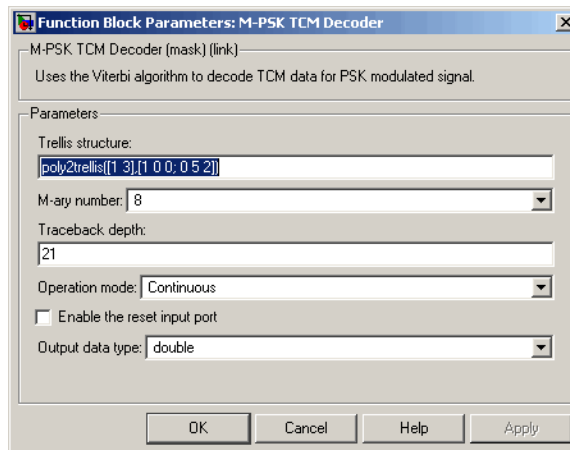
- In Truncated mode, the block treats each frame independently. The traceback path starts at the state with the lowest metric. D must be less than or equal to the vector length of the input.

- In Terminated mode, the block treats each frame independently. The traceback path always starts at the all-zeros state. D must be less than or equal to the vector length of the input. If you know that each frame of data typically ends at the all-zeros state, then this mode is an appropriate choice.

Decoding Delay

If you set **Operation mode** to Continuous, then this block introduces a decoding delay equal to **Traceback depth***k bits, for a rate k/n convolutional code. The decoding delay is the number of zeros that precede the first decoded bit in the output.

The block incurs no delay for other values of **Operation mode**.

**Dialog
Box**

Function Block Parameters: M-PSK TCM Decoder

M-PSK TCM Decoder (mask) (link)

Uses the Viterbi algorithm to decode TCM data for PSK modulated signal.

Parameters

Trellis structure:

poly2trellis([1 3],[1 0 0; 0 5 2])

M-ary number: 8

Traceback depth:

21

Operation mode: Continuous

☐ Enable the reset input port

Output data type: double

OK   Cancel   Help   Apply

Trellis structure
 MATLAB structure that contains the trellis description of the
 convolutional encoder.

M-ary number
 The number of points in the signal constellation.

Traceback depth
 The number of trellis branches (equivalently, the number of
 symbols) the block uses in the Viterbi algorithm to construct each
 traceback path.

Operation mode
 The operation mode of the Viterbi decoder. Choices are
 Continuous, Truncated, and Terminated.

Enable the reset input port
 When you check this box, the block has a second input port labeled
 Rst. Providing a nonzero input value to this port causes the block
 to set its internal memory to the initial state before processing the
 input data. This option appears only if you set **Operation mode**
 to Continuous.

Output data type
    The output type of the block can be specified as a `boolean` or `double`. By default, the block sets this to `double`.

Pair Block   M-PSK TCM Encoder

See Also   General TCM Decoder, `poly2trellis`

References   [1] Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

[2] Proakis, John G., *Digital Communications*, Fourth edition, New York, McGraw-Hill, 2001.

Purpose	Convolutionally encode binary data and modulate using PSK method
Library	Trellis-Coded Modulation
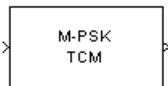
Description

The M-PSK TCM Encoder block implements trellis-coded modulation (TCM) by convolutionally encoding the binary input signal and mapping the result to a PSK signal constellation.

The **M-ary number** parameter is the number of points in the signal constellation, which also equals the number of possible output symbols from the convolutional encoder. (That is, $\log_2$(**M-ary number**) is equal to n for a rate k/n convolutional code.)

Input and Output Signals

If the convolutional encoder described by the trellis structure represents a rate k/n code, then the M-PSK TCM Encoder block's input must be a frame-based binary column vector whose length is L*k for some positive integer L.

The output from the M-PSK TCM Encoder block is a frame-based complex column vector of length L.

The input signal must be a binary value.

Specifying the Encoder

To define the convolutional encoder, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in "Trellis Description of a Convolutional Encoder" in the Communications Toolbox documentation. You can use this parameter field in two ways:

- If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, then use apoly2trellis command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7,[171 133],171)
```

- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is faster because it causes Simulink software to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the previous bulleted item.

The encoder registers begin in the all-zeros state. You can configure the encoder so that it resets its registers to the all-zeros state during the course of the simulation. To do this, set the Operation mode to **Reset on nonzero input via port**. The block then opens a second input port, labeled Rst. The signal at the Rst port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.

Signal Constellations

The trellis-coded modulation technique partitions the constellation into subsets called cosets, so as to maximize the minimum distance between pairs of points in each coset. This block internally forms a valid partition based on the value you choose for the **M-ary number** parameter.

The figure below shows the labeled set-partitioned signal constellation that the block uses when **M-ary number** is 8. For constellations of other sizes, see [1].
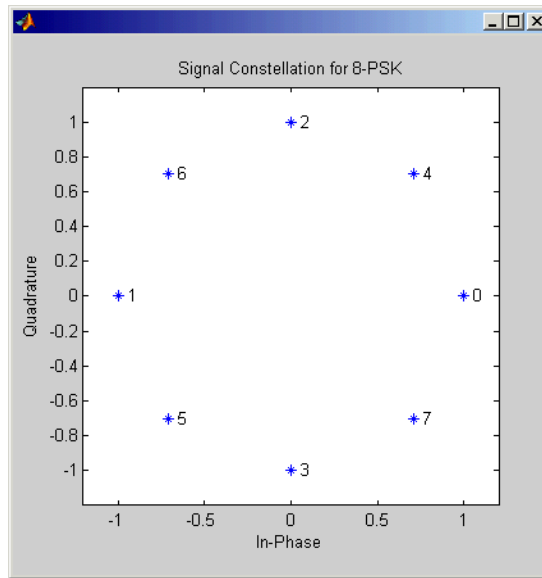
Coding Gains

Coding gains of 3 to 6 decibels, relative to the uncoded case can be achieved in the presence of AWGN with multiphase trellis codes [3].

M-PSK TCM Encoder

Dialog Box

Trellis structure

> MATLAB structure that contains the trellis description of the convolutional encoder.

Operation mode

> In `Continuous` mode (default setting), the block retains the encoder states at the end of each frame, for use with the next frame.
>
> In `Truncated (reset every frame)` mode, the block treats each frame independently. I.e., the encoder states are reset to all-zeros state at the start of each frame.
>
> In `Terminate trellis by appending bits` mode, the block treats each frame independently. For each input frame, extra bits are used to set the encoder states to all-zeros state at the end of the frame. The output length is given by
>
> $y = n \cdot (x + s)/k$, where x is the number of input bits, and
>
> $s = \text{constraint length} - 1$ (or, in the case of multiple constraint

lengths, s =sum(ConstraintLength(i)-1)). This mode is supported for frame based inputs only.

In Reset on nonzero input via port mode, the block has an additional input port, labeled Rst. When the Rst input is nonzero, the encoder resets to the all-zeros state.

M-ary number
The number of points in the signal constellation.

Output data type
The output type of the block can be specified as a single or double. By default, the block sets this to double.

Pair Block M-PSK TCM Decoder

See Also General TCM Encoder, poly2trellis

References [1] Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

[2] Proakis, John G., Digital Communications, Fourth edition, New York, McGraw-Hill, 2001

[3] Ungerboeck, G., "Channel Coding with Multilevel/Phase Signals", *IEEE Trans. on Information Theory*, Vol IT28, Jan. 1982, pp. 55–67.

MSK Demodulator Baseband

Purpose   Demodulate MSK-modulated data

Library   CPM, in Digital Baseband sublibrary of Modulation

Description The MSK Demodulator Baseband block demodulates a signal that was modulated using the minimum shift keying method. The input is a baseband representation of the modulated signal. The **Phase offset** parameter is the initial phase of the modulated waveform.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D, in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of D+1 zero symbols.

- If the input signal is frame-based, then the delay consists of D zero symbols.

The optimal **Traceback length** parameter value choice is dependent on minimum squared Euclidean distance calculations. Alternatively, a typical value can be chosen using the "five-times-the-constraint-length" rule which depends on the number of states given by:

$$numStates = \begin{cases} p * m^{(L-1)} < 0 \\ 2p * m^{(L-1)} > 0 \end{cases}$$

- h = m/p is the modulation index proper rational form

  - m = numerator of modulation index

  - p = denominator of modulation index

- *L* is the Pulse length

- *M* is M-ary number

Inputs and Outputs

The input can be either a scalar or a frame-based column vector and must be of type single or double. If the **Output type** parameter is set to Integer, then the block produces values of 1 and -1. If the **Output type** parameter is set to Bit, then the block produces values of 0 and 1.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

Dialog Box

MSK Demodulator Baseband

Output type
> Determines whether the output consists of bipolar or binary values.

Phase offset (rad)
> The initial phase of the modulated waveform.

Samples per symbol
> The number of input samples that represent each modulated symbol.

Traceback length
> The number of trellis branches that the MSK Demodulator Baseband block uses to construct each traceback path.

Output data type
> The output data type can be `boolean`, `int8`, `int16`, `int32`, or `double`.

Pair Block   MSK Modulator Baseband

See Also   CPM Demodulator Baseband, Viterbi Decoder

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg, *Digital Phase Modulation*, New York, Plenum Press, 1986.

Purpose   Modulate using minimum shift keying method

Library   CPM, in Digital Baseband sublibrary of Modulation

Description   The MSK Modulator Baseband block modulates using the minimum shift keying method. The output is a baseband representation of the modulated signal.

The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

Input Attributes

The input can be either a scalar or a frame-based column vector. If the **Input type** parameter is set to Integer, then the block accepts values of 1 and -1. If the **Input type** parameter is set to Bit, then the block accepts values of 0 and 1.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see "Upsampled Signals and Rate Changes" in *Communications Blockset User's Guide*.

MSK Modulator Baseband
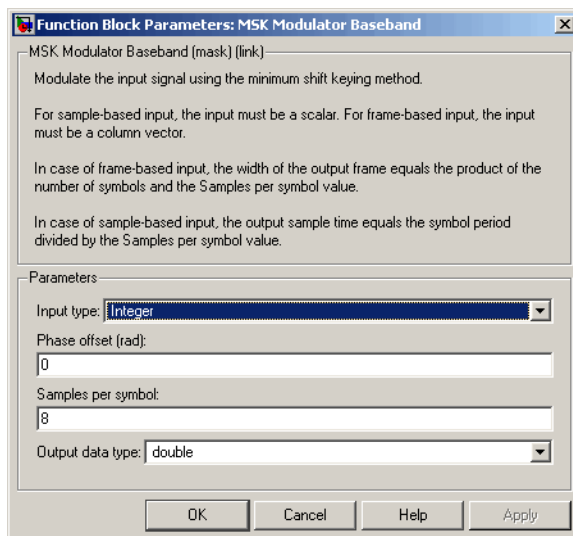
Dialog Box

Input type

Indicates whether the input consists of bipolar or binary values.

Phase offset (rad)

The initial phase of the output waveform.

Samples per symbol

The number of output samples that the block produces for each integer or bit in the input.

Output data type

The output type of the block can be specified as a single or double. By default, the block sets this to double.

Pair Block   MSK Demodulator Baseband

See Also   CPM Modulator Baseband

References   [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg, *Digital Phase Modulation*, New York, Plenum Press, 1986.

Purpose   Recover symbol timing phase using fourth-order nonlinearity method

Library   Timing Phase Recovery sublibrary of Synchronization

Description   The MSK-Type Signal Timing Recovery block recovers the symbol
timing phase of the input signal using a fourth-order nonlinearity
method. This block implements a general non-data-aided feedback
method that is independent of carrier phase recovery but requires prior
compensation for the carrier frequency offset. This block is suitable for
systems that use baseband minimum shift keying (MSK) modulation or
Gaussian minimum shift keying (GMSK) modulation.

Inputs

By default, the block has one input port. The input signal could be (but
is not required to be) the output of a receive filter that is matched to the
transmitting pulse shape, or the output of a lowpass filter that limits
the amount of noise entering this block.

The input must be a scalar or a frame-based column vector. The input
uses N samples to represent each symbol, where N > 1 is the **Samples
per symbol** parameter. If the input is frame-based, then its vector
length is N*R, where R is a positive integer that indicates the number
of symbols per frame. If the input is sample-based, then its sample time
is 1/N times the underlying symbol period.

If the **Reset** parameter is set to On nonzero input via port, then the
block has a second input port, labeled Rst. The Rst input determines
when the timing estimation process restarts, and must be a scalar
signal. The sample time of the Rst input equals the symbol period if the
input signal is sample-based, and the frame period if the input signal
is frame-based.

Outputs

The block has two output ports, labeled Sym and Ph:

- The Sym output is the result of applying the estimated phase
  correction to the input signal. This output is the signal value for each

symbol, which can be used for decision purposes. The values in the Sym output occur at the symbol rate:

- If the input signal is a frame-based column vector of length N*R, then the Sym output is a frame-based column vector of length R having the same frame period.

- If the input signal is a sample-based scalar with sample time T/N, then the Sym output is a sample-based scalar with sample time T.

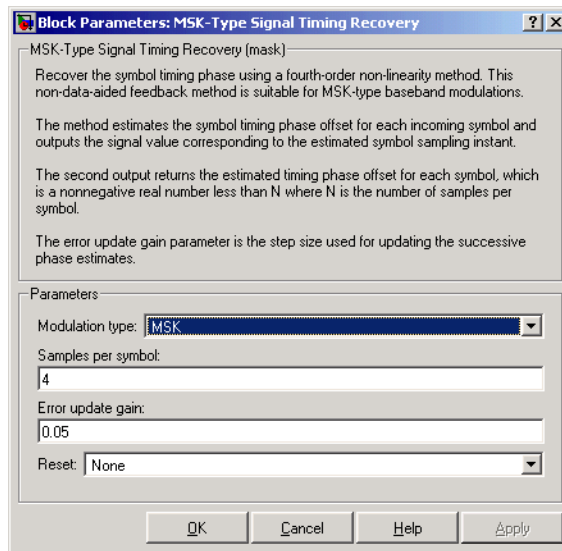- The Ph output gives the phase estimate for each symbol in the input signal.

  The Ph output contains nonnegative real numbers less than N. Noninteger values for the phase estimate correspond to interpolated values that lie between two values of the input signal. The sample time or frame period of the Ph output is the same as that of the Sym output.

Note If the Ph output is very close to either zero or **Samples per symbol**, or if the actual timing phase offset in your input signal is very close to zero, then the block's accuracy might be compromised by small amounts of noise or jitter. The block works well when the timing phase offset is significant rather than very close to zero.

Delays

This block incurs a delay of two symbols when the input signal is frame-based and three symbols when the input signal is sample-based.

Dialog Box

Modulation type

> The type of modulation in the system. Choices are `MSK` and `GMSK`.

Samples per symbol

> The number of samples, N, that represent each symbol in the input signal. This must be greater than 1.

Error update gain

> A positive real number representing the step size that the block uses for updating successive phase estimates. Typically, this number is less than 1/N, which corresponds to a slowly varying phase.

Reset

> Determines whether and under what circumstances the block restarts the phase estimation process. Choices are `None`, `Every frame`, and `On nonzero input via port`. The last option causes the block to have a second input port, labeled `Rst`.

MSK-Type Signal Timing Recovery

Algorithm

This block's algorithm extracts timing information by passing the sampled baseband signal through a fourth-order nonlinearity followed by a digital differentiator whose output is smoothed to yield an error signal. The algorithm then uses the error signal to make the sampling adjustments.

More specifically, this block uses a timing error detector whose result for the kth symbol is e(k), given in [2] by

$$
\begin{aligned}
e(k) = {} & (-1)^{D+1} \operatorname{Re}\{r^2(kT - T_s + d_{k-1})r^{*2}((k-1)T - T_s + d_{k-2})\} \\
& -(-1)^{D+1} \operatorname{Re}\{r^2(kT + T_s + d_{k-1})r^{*2}((k-1)T + T_s + d_{k-1})\}
\end{aligned}
$$

where

- r is the block's input signal
- T is the symbol period
- T_s is the sampling period
- * means complex conjugate
- d_k is the phase estimate for the kth symbol
- D is 1 for MSK and 2 for Gaussian MSK modulation

For more information about the role that e(k) plays in this block's algorithm, see "Feedback Methods for Timing Phase Recovery" in *Communications Blockset User's Guide*.

References

[1] D'Andrea, A. N., U. Mengali, and R. Reggiannini, "A Digital Approach to Clock Recovery in Generalized Minimum Shift Keying," *IEEE Transactions on Vehicular Technology*, Vol. 39, No. 3, August 1990, pp. 227-234.

[2] Mengali, Umberto and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

See Also

Early-Late Gate Timing Recovery, Squaring Timing Recovery

Purpose   Recover symbol timing phase using Mueller-Muller method

Library   Timing Phase Recovery sublibrary of Synchronization
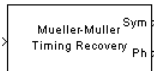
Description   The Mueller-Muller Timing Recovery block recovers the symbol timing phase of the input signal using the Mueller-Muller method. This block implements a decision-directed, data-aided feedback method that requires prior recovery of the carrier phase.

Inputs

By default, the block has one input port. Typically, the input signal is the output of a receive filter that is matched to the transmitting pulse shape. The input must be a scalar or a frame-based column vector. The input uses N samples to represent each symbol, where N > 1 is the **Samples per symbol** parameter. If the input is frame-based, then its vector length is N*R, where R is a positive integer that indicates the number of symbols per frame. If the input is sample-based, then its sample time is 1/N times the underlying symbol period.

If the **Reset** parameter is set to On nonzero input via port, then the block has a second input port, labeled Rst. The Rst input determines when the timing estimation process restarts, and must be a scalar. The sample time of the Rst input equals the symbol period if the input signal is sample-based, and the frame period if the input signal is frame-based.

Outputs

The block has two output ports, labeled Sym and Ph:

- The Sym output is the result of applying the estimated phase correction to the input signal. This output is the signal value for each symbol, which can be used for decision purposes. The values in the Sym output occur at the symbol rate:

  - If the input signal is a frame-based column vector of length N*R, then the Sym output is a frame-based column vector of length R having the same frame period.

- If the input signal is a sample-based scalar with sample time T/N, then the `Sym` output is a sample-based scalar with sample time T.

- The `Ph` output gives the phase estimate for each symbol in the input signal.

  The `Ph` output contains nonnegative real numbers less than N. Noninteger values for the phase estimate correspond to interpolated values that lie between two values of the input signal. The sample time or frame period of the `Ph` output is the same as that of the `Sym` output.

  Note If the `Ph` output is very close to either zero or **Samples per symbol**, or if the actual timing phase offset in your input signal is very close to zero, then the block's accuracy might be compromised by small amounts of noise or jitter. The block works well when the timing phase offset is significant rather than very close to zero.

Delays

This block incurs a delay of two symbols when the input signal is frame-based and three symbols when the input signal is sample-based.

Dialog Box

Samples per symbol

The number of samples, N, that represent each symbol in the input signal. This must be greater than 1.

Error update gain

A positive real number representing the step size that the block uses for updating successive phase estimates. Typically, this number is less than 1/N, which corresponds to a slowly varying phase.

Reset

Determines whether and under what circumstances the block restarts the phase estimation process. Choices are None, Every frame, and On nonzero input via port. The last option causes the block to have a second input port, labeled Rst.

Algorithm   This block uses a timing error detector whose result for the kth symbol is e(k), given by

Mueller-Muller Timing Recovery

$$e(k) = \text{Re}\{c_{k-1}^{*} y(kT + d_k) - c_k^{*} y((k-1)T + d_{k-1})\}$$

where

- y is the block's input signal
- c_k is the decision based on the sample value $y(kT+d_k)$
- T is the symbol period
- d_k is the phase estimate for the kth symbol

For more information about the role that e(k) plays in this block's algorithm, see "Feedback Methods for Timing Phase Recovery" in *Communications BlocksetUser's Guide*.

References

[1] Mengali, Umberto and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

[2] Meyr, Heinrich, Marc Moeneclaey, and Stefan A. Fechtel, *Digital Communication Receivers*, Vol 2, New York, Wiley, 1998.

[3] Mueller, K. H., and M. S. Muller, "Timing Recovery in Digital Synchronous Data Receivers," *IEEE Transactions on Communications*, Vol. COM-24, May 1976, pp. 516-531.

See Also

Early-Late Gate Timing Recovery, Squaring Timing Recovery

Purpose   Implement μ-law compressor for source coding

Library   Source Coding

Description The Mu-Law Compressor block implements a μ-law compressor for the input signal. The formula for the μ-law compressor is
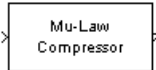
$$y = \frac{V \log(1 + \mu |x| / V)}{\log(1 + \mu)} \operatorname{sgn}(x)$$

where μ is the μ-law parameter of the compressor, *V* is the peak magnitude of *x*, log is the natural logarithm, and sgn is the signum function (`sign` in MATLAB).

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box
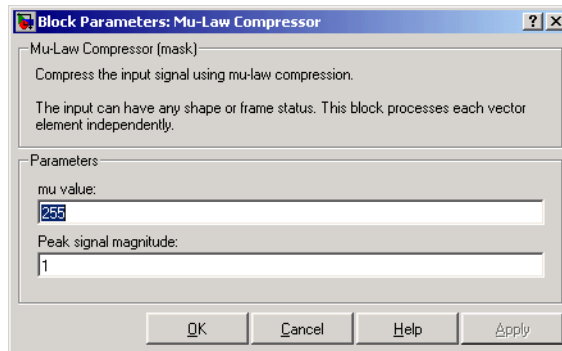
mu value
    The μ-law parameter of the compressor.

Peak signal magnitude
    The peak value of the input signal. This is also the peak value of the output.

Pair Block   Mu-Law Expander

Mu-Law Compressor

See Also   A-Law Compressor

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Purpose   Implement μ-law expander for source coding

Library   Source Coding
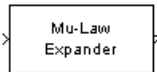
Description   The Mu-Law Expander block recovers data that the Mu-Law
                     Compressor block compressed. The formula for the μ-law expander,
                     shown below, is the inverse of the compressor function.

$$x = \frac{V}{\mu}\left(e^{|y|\log(1+\mu)/V} - 1\right)\mathrm{sgn}(y)$$

The input can have any shape or frame status. This block processes
each vector element independently.

**Dialog
Box**
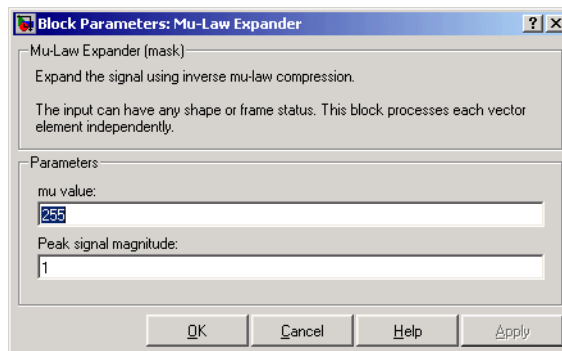
mu value
  The μ-law parameter of the compressor.

Peak signal magnitude
  The peak value of the input signal. This is also the peak value
  of the output.

Pair Block   Mu-Law Compressor

See Also   A-Law Expander

Mu-Law Expander

References   [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Purpose   Simulate multipath Rayleigh fading propagation channel

Library   Channels

Description   The Multipath Rayleigh Fading Channel block implements a baseband
simulation of a multipath Rayleigh fading propagation channel. You
can use this block to model mobile wireless communication systems.
For details about fading channels, see the references listed below.

This block accepts only frame-based complex signals at its input. To
work with sample-based inputs, use the `Frame conversion` block of
Signal Processing blockset to reformat the signal.

The block inherits sample time from the input signal. The input signal
must have a discrete sample time greater than 0.

Relative motion between the transmitter and receiver causes Doppler
shifts in the signal frequency. You can specify the Doppler spectrum of
the Rayleigh process using the **Doppler spectrum type** parameter.
For channels with multiple paths, you can assign each path a different
Doppler spectrum, by entering a vector of doppler objects in the
Doppler spectrum field.

Because a multipath channel reflects signals at multiple places, a
transmitted signal travels to the receiver along several paths, each of
which may have differing lengths and associated time delays. In the
block's parameter dialog box, the **Discrete path delay vector** specifies
the time delay for each path. If you do not check **Normalize gain
vector to 0 dB overall gain**, then the **Average path gain vector**
specifies the gain for each path. When you check the box, the block uses
a multiple of **Average path gain vector** instead of the **Average path
gain vector** itself, choosing the scaling factor so that the channel's
effective gain, considering all paths, is 0 dB.

The number of paths indicates the length of **Discrete path delay
vector** or **Average path gain vector**, whichever is larger. If both of
these parameters are vectors, then they must have the same length; if
exactly one of these parameters contains a scalar value, then the block

expands it into a vector whose size matches that of the other vector parameter.

The block multiplies the input signal by samples of a Rayleigh-distributed complex random process. The scalar **Initial seed** parameter seeds the random number generator. Then, the block generates random numbers using the Ziggurat method. This method is the same one used by the MATLAB `randn` function.

Double-clicking this block during simulation or checking the block dialog's check-box labeled **Open channel visualization at start of simulation** plots the channel characteristics using the channel visualization tool. See "Using the Channel Visualization Tool" in *Communications Toolbox User's Guide* for details.

Dialog Box
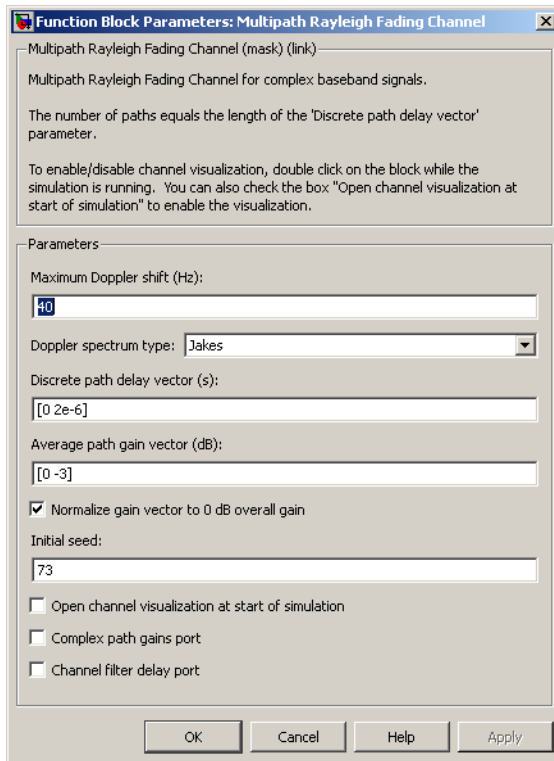
Maximum Doppler shift (Hz)

A positive scalar value that indicates the maximum Doppler shift.

Doppler spectrum type

Specifies the Doppler spectrum of the Rayleigh process.

This parameter defaults to Jakes Doppler spectrum.
Alternatively, you can also choose any of the following types:

- Flat

- Gaussian

- Rounded

- Restricted Jakes

- Asymmetrical Jakes

- Bi-Gaussian

- Bell

For all Doppler spectrum types except `Jakes` and `Flat`, you can choose one or more parameters to control the shape of the spectrum.

You can also select `Specify as dialog parameter` for the **Doppler spectrum type**. Specify the Doppler spectrum by entering an object in the **Doppler spectrum** field. See the `doppler` function reference in *Communications Toolbox User's Guide* for details on how to construct Doppler objects, and also for the meaning of the parameters associated with the various Doppler spectrum types.

Discrete path delay vector (s)
A vector that specifies the propagation delay for each path.

Average path gain vector (dB)
A vector that specifies the gain for each path.

Normalize gain vector to 0 dB overall gain
Checking this box causes the block to scale the **Gain vector** parameter so that the channel's effective gain (considering all paths) is 0 dB.

Initial seed
The scalar seed for the Gaussian noise generator.

Open channel visualization at start of simulation
Checking this box opens the channel visualization tool when a simulation begins.

Complex path gains port
Checking this box creates a port that outputs the values of the complex path gains for each path. In this N-by-M multichannel

frame, N represents the number of samples per frame and M represents the number of discrete paths (number of delays).

Channel filter delay port

Checking this box creates a port that outputs the value of the delay (in samples) that results from the filtering operation of this block. This delay is zero if only one path is simulated, but can be greater than zero if more than one path is present. See "Simulation of Multipath Fading Channels: Methodology" in *Communications Toolbox User's Guide* for a definition of this delay, where it is denoted as N_1.

Algorithm

This implementation is based on the direct-form simulator described in Reference [1]. A detailed explanation of the implementation, including a review of the different Doppler spectra, can be found in [4].

Some wireless applications, such as standard GSM (Global System for Mobile Communication) systems, prefer to specify Doppler shifts in terms of the speed of the mobile. If the mobile moves at speed v making an angle of θ with the direction of wave motion, then the Doppler shift is

$$f_d = (vf/c)\cos\ \theta$$

where f is the transmission carrier frequency and c is the speed of light. The Doppler frequency represents the maximum Doppler shift arising from motion of the mobile.

Example

Generating Ideal Theoretical BER Results for a Rayleigh Fading Channel

This example illustrates how to generate ideal theoretical BER results for a flat Rayleigh fading channel. The model uses reproduces known theoretical results and shows the correct BER performance for a flat Rayleigh fading channel. In this example, you will run the model and compare the simulation results to the BERTool theoretical results for verification purposes. Note that the EbNo value for the model's AWGN block is 5 dB. You can change the noise power by double-clicking

Multipath Rayleigh Fading Channel

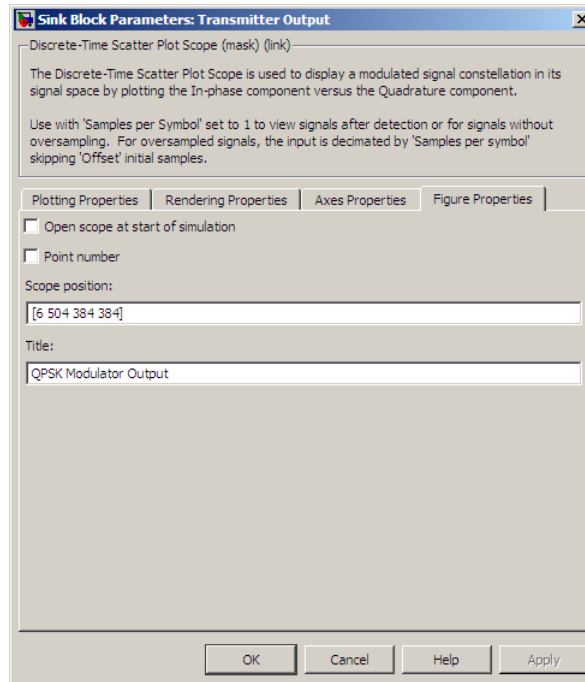the AWGN block and entering another numeric value in the EbNo parameter.

Opening the Model

You can open the model by clicking here in the MATLAB Help browser. Alternatively, you can type `doc_qpsk_rayleigh_derotated` at the MATLAB command line.

Running the Model and Comparing Results

1 You can run the demo by clicking **Simulation > Start**.

2 After the model collects more than 5000 errors, click the stop button.

3 Close the three scopes.

4 In the Simulink model window, double-click the Transmitter Output block. In the mask window, click the **Figure Properties** tab, uncheck **Open scope at start of Simulation**, then click **OK**.

Sink Block Parameters: Transmitter Output

Discrete-Time Scatter Plot Scope (mask) (link)

The Discrete-Time Scatter Plot Scope is used to display a modulated signal constellation in its signal space by plotting the In-phase component versus the Quadrature component.

Use with 'Samples per Symbol' set to 1 to view signals after detection or for signals without oversampling. For oversampled signals, the input is decimated by 'Samples per symbol' skipping 'Offset' initial samples.

| Plotting Properties | Rendering Properties | Axes Properties | Figure Properties |

☐ Open scope at start of simulation

☐ Point number

Scope position:

[6 504 384 384]

Title:

QPSK Modulator Output

| OK | Cancel | Help | Apply |

5 In the Simulink model window, double-click the Rayleigh Channel Output block. In the mask window, click the **Figure Properties** tab, uncheck **Open scope at start of Simulation**, then click **OK**.

6 In the Simulink model window, double-click the Noisy Rayleigh Channel Output block. In the mask window, click the **Figure Properties** tab, uncheck **Open scope at start of Simulation**, then clock **OK**.

7 In the Simulink model window, double-click the Error Rate Calculation block, check **Stop simulation**, enter 5000 for **Target number of error**, then click **OK**.

8 Click the play button to rerun the demo.

9 Open BERTool by typing `bertool` at the MATLAB command line.

10 In BERTool, click the **Theoretical** tab and make the following selections:
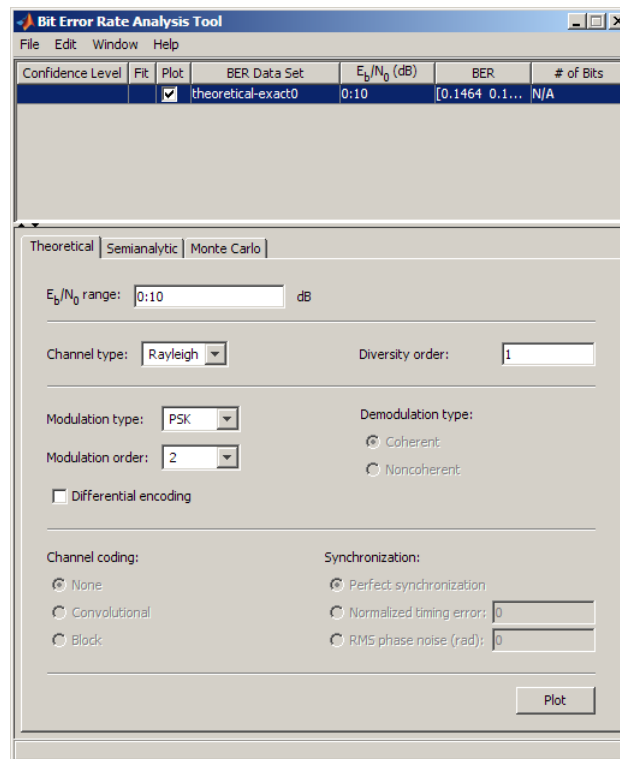
- For **Eb/No range** enter `0:10`

- For **Channel type**, select `Rayleigh`

- For **Diversity Order** enter `1`

- For **Modulation Type**, select `PSK`

- For **Modulation order**, select `4`

11 Click **Plot**.

12 Since the Simulink model uses an EbNo value of 5 dB, verify the probability of error on the BERTool curve at 5 dB. The two values should be approximately equal.
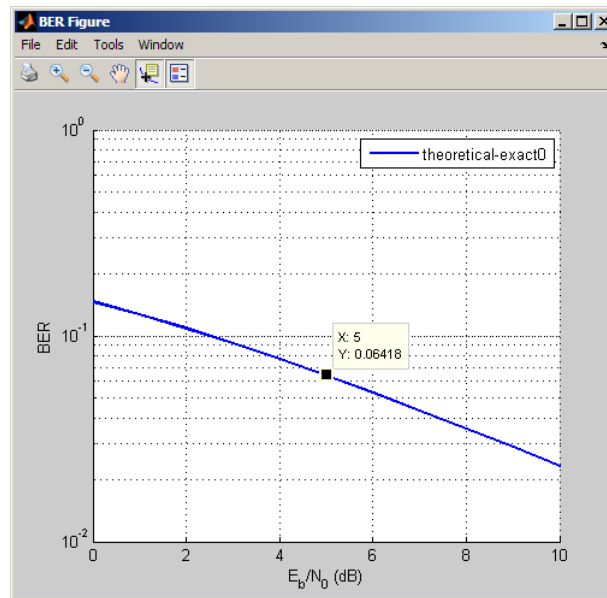
Click the Data Cursor button ⌷ (second from right) and click on the BERTool curve at 5dB.

See Also   Rayleigh Noise Generator, Multipath Rician Fading Channel, doppler

References   [1] Jeruchim, Michel C., Balaban, Philip, and Shanmugan, K. Sam, *Simulation of Communication Systems*, Second edition, New York, Kluwer Academic/Plenum, 2000.

[2] Jakes, William C., ed. *Microwave Mobile Communications*, New York, IEEE Press, 1974.

[3] Lee, William C. Y., *Mobile Communications Design Fundamentals*, 2nd Ed. New York, Wiley, 1993.

[4] Iskander, Cyril-Daniel, *A MATLAB-based Object-Oriented Approach to Multipath Fading Channel Simulation*, a MATLAB Central submission available from www.mathworks.com.

Purpose　　　Simulate multipath Rician fading propagation channel

Library　　　Channels

Description
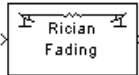
The Multipath Rician Fading Channel block implements a baseband simulation of a multipath Rician fading propagation channel. You can use this block to model mobile wireless communication systems when the transmitted signal can travel to the receiver along a dominant line-of-sight or direct path. For details about fading channels, see Channels in the *Communications Blockset User's Guide*.

This block accepts only frame-based, complex signals at its input. To work with sample-based inputs, use the Frame Conversion block of Signal Processing Blockset to reformat the signal. The sample time is inherited from the input signal.

The input signal must have a discrete sample time greater than 0.

Relative motion between the transmitter and receiver causes Doppler shifts in the signal frequency. You can specify the Doppler spectrum of the Rician process using the **Doppler spectrum type** pop-up menu. For channels with multiple paths, you can assign each path a different Doppler spectrum, by entering a vector of doppler objects in the **Doppler spectrum** field.

Because a multipath channel reflects signals at multiple places, a transmitted signal travels to the receiver along several paths, each of which may have differing lengths and associated time delays. In the block's parameter dialog box, the **Discrete path delay vector** specifies the time delay for each path. If you do not check the **Normalize gain vector to 0 dB overall gain** box, then the **Average path gain vector** specifies the gain for each path. When you check the box, the block uses a multiple of **Average path gain vector** instead of the **Average path gain vector** itself, choosing the scaling factor so that the channel's effective gain considering all paths is 0 dB.

The number of paths indicates the length of **Discrete path delay vector** or **Average path gain vector**, whichever is larger. If both of these parameters are vectors, they must have the same length; if exactly

one of these parameters contains a scalar value, the block expands it into a vector whose size matches that of the other vector parameter.

Fading causes the signal to become diffuse. The **K-factor** parameter, which is part of the statistical description of the Rician distribution, represents the ratio between the power in the line-of-sight component and the power in the diffuse component. The ratio is expressed linearly, not in decibels. While the Average path gain vector parameter controls the overall gain through the channel, the **K-factor** parameter controls the gain's partition into line-of-sight and diffuse components.

You can specify the **K-factor** parameter as a scalar or a vector. If the **K-factor** parameter is a scalar, then the first discrete path of the channel is a Rician fading process (it contains a line-of-sight component) with the specified **K-factor**, while the remaining discrete paths indicate independent Rayleigh fading processes (with no line-of-sight component). If the **K-factor** parameter is a vector of the same size as **Discrete path delay vector**, then each discrete path is a Rician fading process with a **K-factor** given by the corresponding element of the vector. You can attribute the line-of-sight component a Doppler shift, through the **Doppler shift(s) of line-of-sight component(s)** parameter, and an initial phase, through the **Initial phase(s) of line-of-sight component(s)**. The **Doppler shift(s) of line-of-sight component(s)** and **Initial phase(s) of line-of-sight component(s)** parameters must be of the same size as the K-factor parameter.

The block multiplies the input signal by samples of a Rician-distributed complex random process. The scalar **Initial seed** parameter seeds the random number generator. The block then generates random numbers using the Ziggurat method, which is the same method used by the MATLAB randn function.

Double-clicking this block during simulation or selecting the block dialog's check box labeled **Open channel visualization at start of simulation** plots the channel characteristics using the channel visualization tool. See "Using the Channel Visualization Tool" in *Communications Toolbox User's Guide* for details.

**Dialog
Box**

Function Block Parameters: Multipath Rician Fading Channel

Multipath Rician Fading Channel (mask) (link)

Multipath Rician Fading Channel for complex baseband signals.

The number of paths equals the length of the 'Discrete path delay vector' parameter. If the K-factor is a non-zero scalar, then only the first path has a line-of-sight component. If the K-factor is a vector of the same length as the 'Discrete path delay vector', then all paths have line-of-sight components, except for those whose corresponding element of the K-factor is zero.

To enable/disable channel visualization, double click on the block while the simulation is running. You can also check the box "Open channel visualization at start of simulation" to enable the visualization.

Parameters

K-factor (scalar or vector):

[1]

Doppler shift(s) of line-of-sight component(s) (Hz):

[0]

Initial phase(s) of line-of-sight component(s) (rad):

[0]

Maximum diffuse Doppler shift (Hz):

[40]

Doppler spectrum type: [Jakes ▼]

Discrete path delay vector (s):

[0]

Average path gain vector (dB):

[0]

☑ Normalize average path gain vector to 0 dB overall gain

Initial seed:

[73]

☐ Open channel visualization at start of simulation

☐ Complex path gains port

☐ Channel filter delay port

[OK] [Cancel] [Help] [Apply]

K-factor

> The ratio of power in the line-of-sight component to the power in the diffuse component. The ratio is expressed linearly, not in decibels. If **K-factor** is a scalar value, then the first discrete path

is a Rician fading process (it contains a line-of-sight component) with the specified K-factor, while the remaining discrete paths are independent Rayleigh fading processes (with no line-of-sight component). If **K-factor** is a vector of the same size as **Discrete path delay vector**, then each discrete path is a Rician fading process with a **K-factor** given by the corresponding element of the vector.

Doppler shift(s) of line-of-sight components(s) (Hz)

The Doppler shift of the line-of-sight component. It must be a scalar (if **K-factor** is a scalar) or a vector of the same size as **K-factor**. If this parameter contains a scalar value, then the line-of-sight component of the first discrete path has the specified Doppler shift, while the remaining discrete paths become independent Rayleigh fading processes. If the parameter contains a vector, then the line-of-sight component of each discrete path has a Doppler shift given by the corresponding element of the vector.

Initial phase(s) of line-of-sight component(s) (rad)

The initial phase of the line-of-sight component. It must be either a scalar (if **K-factor** is a scalar value) or a vector of the same size as **K-factor**.

Maximum diffuse Doppler shift (Hz)

A positive scalar value that indicates the maximum diffuse Doppler shift.

Doppler spectrum type

Specifies the Doppler spectrum of the Rician process.

This parameter defaults to `Jakes` Doppler spectrum. Alternately, you can choose any of the following types:

- Flat
- Gaussian
- Rounded
- Restricted Jakes

- Asymmetrical Jakes

- Bi-Gaussian

- Bell

For all Doppler spectrum types except `Jakes` and `Flat`, You can use one or more parameters to control the shape of the spectrum.

You can also select `Specify as dialog parameter` for the **Doppler spectrum type**. Specify the Doppler spectrum by entering an object in the **Doppler spectrum** field. See the `doppler` function reference in *Communications Toolbox User's Guide* for details on how to construct doppler objects, and for the meaning of the parameters associated with the various Doppler spectrum types.

Discrete delay vector(s)
A vector that specifies the propagation delay for each path.

Average path gain vector (dB)
A vector that specifies the gain for each path.

Initial seed
The scalar seed for the Gaussian noise generator.

Open channel visualization at start of simulation
Checking this box opens the channel visualization tool when a simulation begins.

Complex path gains port
Checking this box creates a port that outputs the values of the complex path gains for each path. In this N-by-M multichannel frame, N represents the number of samples per frame and M represents the number of discrete paths (number of delays).

Channel filter delay port
Checking this box creates a port that outputs the value of the delay (in samples) that results from the filtering operation of this block. This delay is zero if only one path is simulated, but can be greater than zero if more than one path is present. See

Multipath Rician Fading Channel

"Simulation of Multipath Fading Channels: Methodology" in *Communications Toolbox User's Guide* for a definition of this delay, where it is denoted as N_1.

Algorithm

This implementation is based on the direct form simulator described in Reference [1]. A detailed explanation of the implementation, including a review of the different Doppler spectra, can be found in [4].

Some wireless applications, such as standard GSM (Global System for Mobile Communication) systems, prefer to specify Doppler shifts in terms of the speed of the mobile. If the mobile moves at speed v making an angle of θ with the direction of wave motion, the Doppler shift is

$$f_d = (vf/c)\cos θ$$

where f is the transmission carrier frequency and c is the speed of light. The Doppler frequency is the maximum Doppler shift arising from the motion of the mobile.

See Also

Rician Noise Generator, Multipath Rayleigh Fading Channel, `doppler`

References

[1] Jeruchim, Michel C., Balaban, P., and Shanmugan, K. Sam, *Simulation of Communication Systems*, Second edition, New York, Kluwer Academic/Plenum, 2000.

[2] Jakes, William C., ed., *Microwave Mobile Communications*, New York, IEEE Press, 1974.

[3] Lee, William C. Y., *Mobile Communications Design Fundamentals*, 2nd ed., New York, John Wiley & Sons, Inc., 1993.

[4] Iskander, Cyril-Daniel, *A MATLAB-based Object-Oriented Approach to Multipath Fading Channel Simulation*, a MATLAB Central submission available from www.mathworks.com.

Purpose   Equalize using decision feedback equalizer that updates weights with normalized LMS algorithm

Library   Equalizers

Description   The Normalized LMS Decision Feedback Equalizer block uses a decision feedback equalizer and the normalized LMS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the normalized LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

2-507

- `Weights` output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap of the forward filter.

Dialog Box

Function Block Parameters: Normalized LMS Decision Feedback Equalizer

─ Normalized LMS Decision Feedback Equalizer (mask) (link) ─

Equalize a linearly modulated signal through a dispersive channel using a decision feedback equalizer and the normalized LMS algorithm.

The block computes filter weights with the normalized LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. The equalizer will train for the length of the Desired signal. If the mode input is not present, the equalizer will train at the beginning of every frame for the length of the Desired signal.

─ Parameters ─

Number of forward taps:

6

Number of feedback taps:

6

Number of samples per symbol:

1

Signal constellation:

[-1+j -1-j 1+j 1-j]

Reference tap:

3

Step size:

0.01

Leakage factor:

1

Bias:

0

Initial weights:

0

☑ Mode input port

☑ Output error

☑ Output weights

[OK]   [Cancel]   [Help]   [Apply]

Number of forward taps

The number of taps in the forward filter of the decision feedback equalizer.

Number of feedback taps

The number of taps in the feedback filter of the decision feedback equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of forward taps in the equalizer.

Step size

The step size of the normalized LMS algorithm.

Leakage factor

The leakage factor of the normalized LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Bias

The bias parameter of the normalized LMS algorithm, a nonnegative real number. This parameter is used to overcome difficulties when the algorithm's input signal is small.

Initial weights

A vector that concatenates the initial weights for the forward and feedback taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current forward and feedback weights, concatenated into one vector.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

See Also   Normalized LMS Linear Equalizer, LMS Decision Feedback Equalizer

Normalized LMS Linear Equalizer

Purpose   Equalize using linear equalizer that updates weights with normalized LMS algorithm

Library   Equalizers

Description The Normalized LMS Linear Equalizer block uses a linear equalizer and the normalized LMS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the normalized LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap.

Normalized LMS Linear Equalizer

Dialog Box

Function Block Parameters: Normalized LMS Linear Equalizer

Normalized LMS Linear Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using the normalized LMS algorithm.

The block computes filter weights with the normalized LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. For every frame in which the mode input is 1 or not present, the equalizer trains at the beginning of the frame for the length of the desired signal.

Parameters

Number of taps:

4

Number of samples per symbol:

1

Signal constellation:

[-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j]

Reference tap:

2

Step size:

0.01

Leakage factor:

1

Bias:

0

Initial weights:

0

☑ Mode input port

☑ Output error

☑ Output weights

OK   Cancel   Help   Apply

Number of taps

The number of taps in the filter of the linear equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of taps in the equalizer.

Step size

The step size of the normalized LMS algorithm.

Leakage factor

The leakage factor of the normalized LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Bias

The bias parameter of the normalized LMS algorithm, a nonnegative real number. This parameter is used to overcome difficulties when the algorithm's input signal is small.

Initial weights

A vector that lists the initial weights for the taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current weights.

Normalized LMS Linear Equalizer

Examples   See the Adaptive Equalization demo.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

See Also   Normalized LMS Decision Feedback Equalizer, LMS Linear Equalizer

Purpose   Demodulate OQPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation
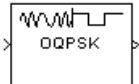
Description   The OQPSK Demodulator Baseband block demodulates a signal that

was modulated using the offset quadrature phase shift keying method.
The input is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The input can be
either a scalar or a frame-based column vector. The block accepts the
data types double, single, and signed fixed-point.

If the **Output type** parameter is set to Integer, then the block outputs
integer symbol values between 0 and 3. If the **Output type** parameter
is set to Bit, then the block outputs 2–bit binary representations of such
integers, in a binary-valued vector whose length is an even number.

The block produces one output symbol for each pair of input symbols.
The input symbol period is half the period of each output integer or
bit pair. The constellation used to map bit pairs to symbols is on the
reference page for the OQPSK Modulator Baseband block.

Sample-Based Inputs

If the input is a sample-based scalar, then the block outputs a single
integer symbol (or one pair of bits) for each output time step. The input
sample period is half the output symbol period, in this case.
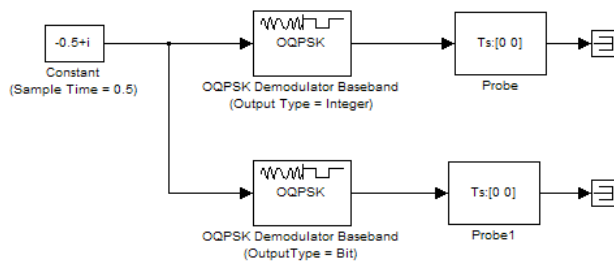
To open the completed model, click here in the MATLAB Help browser

OQPSK Demodulator Baseband

Frame-Based Inputs

If the input is a frame-based column vector, then the block processes several integer symbols (or several pairs of bits) for each time step. In this case, the frame-based output port frame time equals that of the input.

The output frame length is adjusted based on the input frame length and the setting for the **Output type** parameter. When you set the **Output type** parameter to Integer, the output frame length is half that of the input frame length. When you set the **Output type** parameter to Bit, the output frame length equals the input frame length (because each symbol is two bits).

Note The input frame length must be even or else the block will error out.
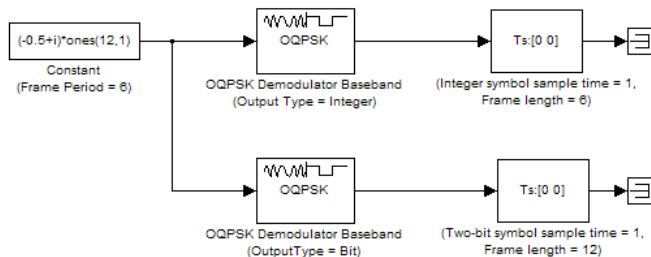
To open the completed model, click here in the MATLAB Help browser

Delays

The modulator-demodulator pair incurs a delay, as described in "Delays in Digital Modulation".

Signal Flow Diagram
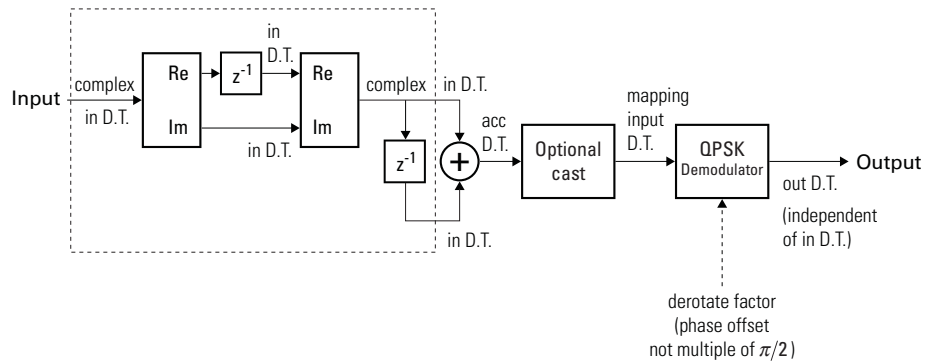
OQPSK Fixed-Point Signal Flow Diagram

Note Every two input samples produce one output symbol. In the preceding figure, the dotted line represents the region comprised of input sample processing.

OQPSK Demodulator Baseband
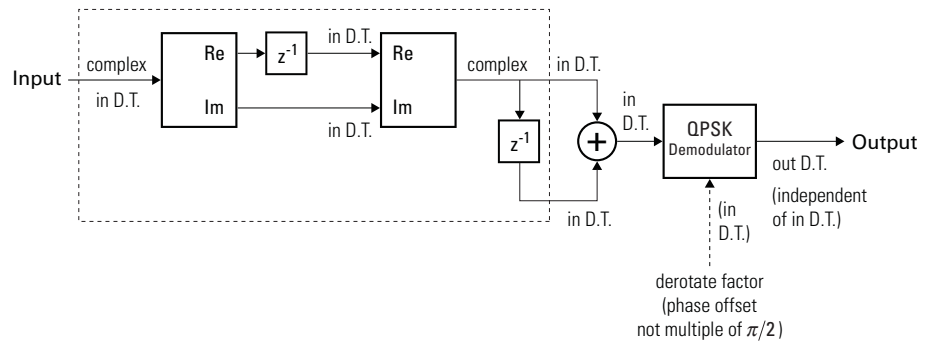
OQPSK Floating Point Signal Flow Diagram

Note Every two input samples produce one output symbol. In the preceding figure, the dotted line represents the region comprised of input sample processing.

**Dialog
Box**
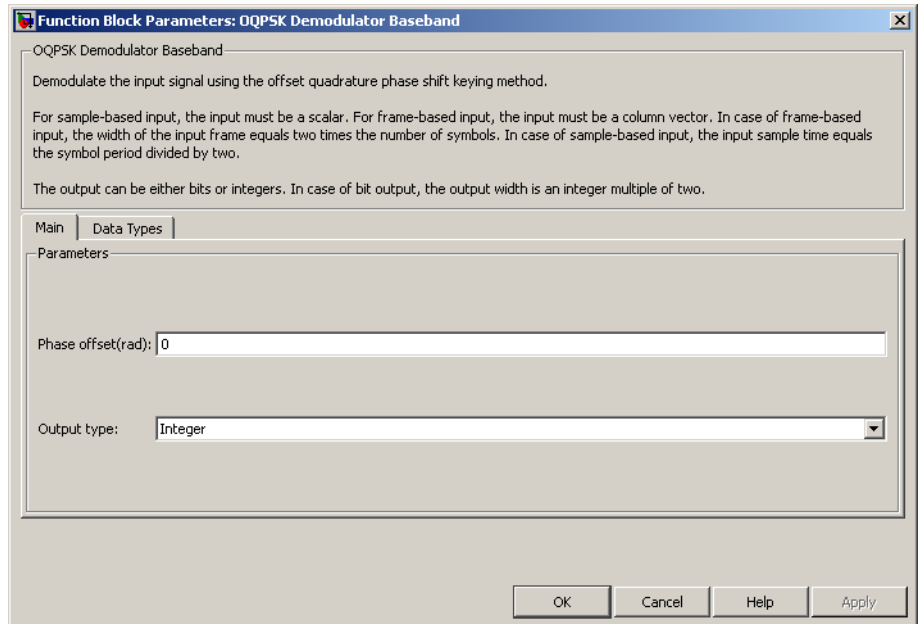
Phase offset (rad)
> The amount by which the phase of the zeroth point of the signal constellation is shifted from π/4.

Output type
> Determines whether the output consists of integers or pairs of bits.

OQPSK Demodulator Baseband
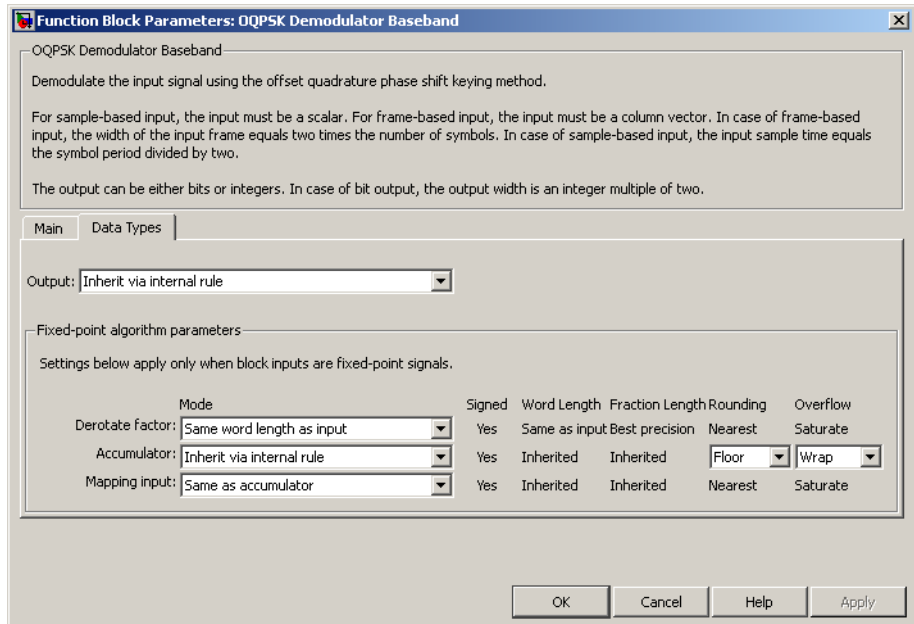
Data Type Pane

Output

For bit outputs, the output data type can be set to `'Inherit via internal rule'`, `'Smallest unsigned integer'`, double, single, int8, uint8, int16, uint16, int32, uint32, or boolean.

For integer outputs, the output data type can be set to `'Inherit via internal rule'`, `'Smallest unsigned integer'`, double, single, int8, uint8, int16, uint16, int32, or uint32.

When this parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is a floating-point type (single or double). If the input data type is fixed-point, the output data type will work as if this parameter is set to `'Smallest unsigned integer'`.

When this parameter is set to `'Smallest unsigned integer'`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model.

If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, and **Output type** is `Bit`, the output data type is the ideal minimum one-bit size, i.e., `ufix1`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., `uint8`).

If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, and **Output type** is `Integer`, the output data type is the ideal minimum two-bit size, i.e., `ufix2`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit two bits, usually corresponding to the size of a char (e.g., `uint8`).

Derotate factor

This parameter only applies when the input is fixed-point and

Phase offset is not a multiple of $\dfrac{\Pi}{2}$.

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input.

Accumulator

Specify the data type for the Accumulator. You can set this parameter to `Inherit via internal rule`, `Same as input` or `Binary point scaling`.

The **Accumulator** parameter only applies for fixed-point inputs. The selections you make for the **Rounding** and **Overflow** parameters affect the **Accumulator**.

Fixed-point Communications Blockset blocks that must hold summation results for further calculation usually allow you to

specify the data type and scaling of the accumulator. Most such blocks cast to the accumulator data type prior to summation:

The result of each addition remains in the accumulator data type.

See the reference page of a specific block in the Block Reference for details on the accumulator data type of a specific block.

Use the **Accumulator—Mode** parameter to specify how you would like to designate the accumulator word and fraction lengths:

- When you select `Inherit via internal rule`, the accumulator output word and fraction lengths are automatically calculated for you. Refer to Inherit via Internal Rule for more information.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator.

The bias of all signals in Signal Processing Blockset software is zero.

Rounding

The block uses the **Rounding** method when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow

Specify the method of storing the result when the magnitude of a fixed-point calculation result that does not does not fit within the range of the data type selected. You can select either `Wrap` or `Saturate` for this parameter.

For more information refer to Overflow in the Precision and Range subsection of the Signal Processing Blockset Getting Started Guide.

Mapping input

This can be set to `Same as accumulator` or `Binary point scaling`. This parameter only applies for fixed-point inputs.

Pair Block   OQPSK Modulator Baseband

See Also   QPSK Demodulator Baseband

OQPSK Modulator Baseband

Purpose　　　　Modulate using offset quadrature phase shift keying method

Library　　　　PM, in Digital Baseband sublibrary of Modulation

Description　　The OQPSK Modulator Baseband block modulates using the offset
quadrature phase shift keying method. The output is a baseband
representation of the modulated signal.

Note The OQPSK modulator block upsamples by a factor of 2.

If the **Input type** parameter is set to Integer, then valid input values
are 0, 1, 2, and 3. In this case, the input can be either a scalar or a
frame-based column vector.

If the **Input type** parameter is set to Bit, then the input must be a
binary-valued vector. In this case, the input can be either a vector of
length two or a frame-based column vector whose length is an even
integer.

For integer inputs, the block can accept the data types int8, uint8,
int16, uint16, int32, uint32, single, and double. For bit inputs, the
block can accept int8, uint8, int16, uint16, int32, uint32, boolean,
single, and double.

The symbol period is half the input period. The first output symbol is
an initial condition of zero that is unrelated to the input values.

The constellation used to map bit pairs to symbols is in the following
figure. If the block's **Phase offset** parameter is nonzero, then this
constellation is rotated by that parameter value.
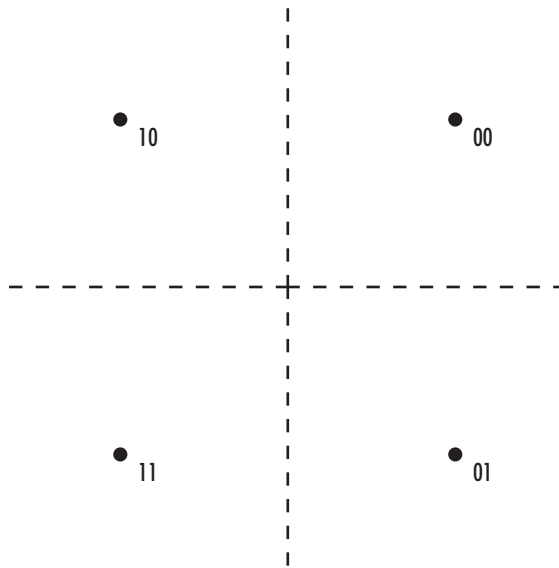
Sample-Based Inputs

For sample-based inputs, the output sample time equals half of the input symbol period.

- In bit input mode, the input must be of length two.

- In integer input mode, the input must be a scalar value.

If the input is a sample-based scalar, then the block outputs a single integer symbol for each time step. In this case, the sample-based input symbol period is half the output symbol period.

OQPSK Modulator Baseband
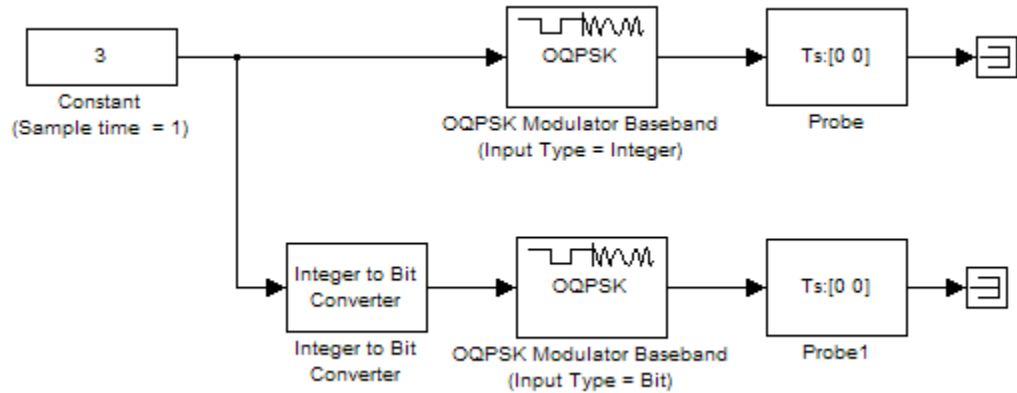
To open the completed model, click here in the MATLAB Help browser

Frame-Based Inputs

If the input is a frame-based column vector, then the block processes several integers or several pairs of bits in each time step. In this case, the output sample time equals the input sample time, even though the period of each output sample is half the period of each integer or bit pair in the input.
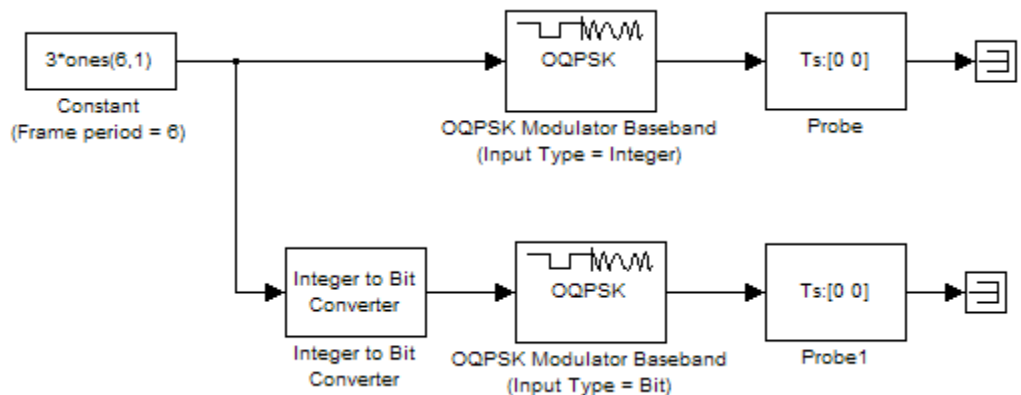
To open the completed model, click here in the MATLAB Help browser

Delays

The modulator-demodulator pair incurs a delay, as described in "Delays in Digital Modulation".

Dialog Box
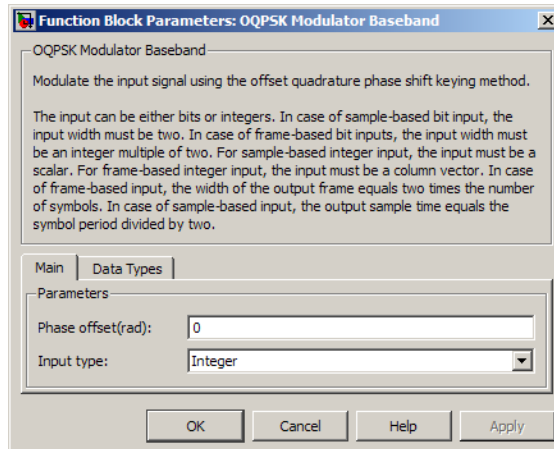
Phase offset (rad)

The amount by which the phase of the zeroth point of the signal constellation is shifted from π/4.

Input type

Indicates whether the input consists of integers or pairs of bits.

Output data type

The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Setting this parameter to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this parameter to `Inherit via back propagation`, sets the output data type and scaling to match the following block.

OQPSK Modulator Baseband

Output word length
>Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type
>Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Output fraction length
>Specify the scaling of the fixed-point output by either of the following methods:

>- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

>- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

>This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter or when you select `User-defined` and the specified output data type is a fixed-point data type.

Output fraction length
>For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   OQPSK Demodulator Baseband

See Also   QPSK Modulator Baseband
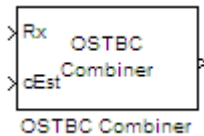
Purpose   Combine inputs for received signals and channel estimate according to orthogonal space-time block code (OSTBC)

**Library**   MIMO

Description   The OSTBC Combiner block combines the input signal (from all of the receive antennas) and the channel estimate signal to extract the soft information of the symbols encoded by an OSTBC. The input channel estimate may not be constant during each codeword block transmission and the combining algorithm uses only the estimate for the first symbol period per codeword block. A symbol demodulator or decoder would follow the Combiner block in a MIMO communications system.

The block conducts the combining operation for each symbol independently. The combining algorithm depends on the structure of the OSTBC. For more information, see the OSTBC Combining Algorithms section of this help page.

Dimension

Along with the time and spatial domains for OSTBC transmission, the block supports an optional dimension, over which the combining calculation is independent. This dimension can be thought of as the frequency domain for OFDM-based applications. The following illustration indicates the supported dimensions for inputs and output of the OSTBC Combiner block.
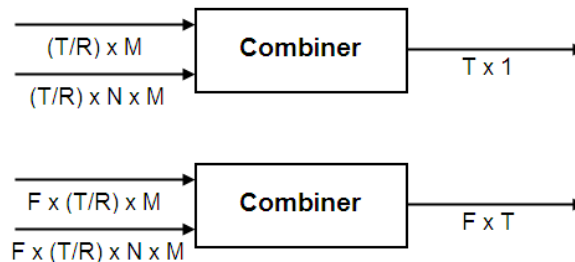
The following table describes each variable for the block.

OSTBC Combiner

Variable	Description
F	The additional dimension; typically the frequency dimension. The combining calculation is independent of this dimension.
N	Number of transmit antennas.
M	Number of receive antennas.
T	Output symbol sequence length in time domain.
R	Symbol rate of the code.

Note On the two inputs, T/R is the symbol sequence length in the time domain.

F and *M* can be any positive integers. *N* can be 2, 3 or 4, indicated by **Number of transmit antennas** parameter. The time domain length T/R must be a multiple of the codeword block length (2 for Alamouti; 4 for all other OSTBC). For *N* = 2, T/R must be a multiple of 2. When *N* > 2, T/R must be a multiple of 4. *R* defaults to 1 for 2 antennas. *R* can

be either $\frac{3}{4}$ or $\frac{1}{2}$ for more than 2 antennas.

The supported dimensions for the block depend upon the values of *F* and *M*. For one receive antenna (*M* = 1), the received signal input must be a column vector or a full 2–D matrix, depending on the value for *F*. The corresponding channel estimate input must be a full 2-D or 3-D matrix.

For more than one receive antenna (*M* > 1), the received signal input must be a full 2-D or 3-D matrix, depending on the value for *F*. Correspondingly, the channel estimate input must be a 3-D or 4-D matrix, depending on the value for *F*.

To understand the block's dimension propagation, refer to the following table.

	Input 1 (Received Signal)	Input 2 (Channel Estimate)	Output
F = 1 and M = 1	Column vector	2-D	Column vector
F = 1 and M > 1	2-D	3-D	Column vector
F > 1 and M = 1	2-D	3-D	2-D
F > 1 and M > 1	3-D	4-D	2-D

Data Type

The two input signals must have the same data types: double, single or signed fixed-point. The output signal inherits the data type from the inputs. The block supports different fixed-point properties for the two inputs. For fixed-point signals, the output word length and fractional length depend on the block's mask parameter settings. See Fixed-Point Signals for more information about fixed-point data propagation of this block.

Frames

The output inherits the frameness of the received signal input. For either column vector or full 2-D matrix input signal, the input can be either frame-based or sample-based. A 3–D or 4–D matrix input signal must have sample-based input.

OSTBC Combining Algorithms

The OSTBC Combiner block supports five different OSTBC combining computation algorithms. Depending on the selection for **Rate** and **Number of transmit antennas**, you can select one of the algorithms shown in the following table.

OSTBC Combiner

Transmit Antenna	Rate	Computational Algorithm per Codeword Block Length
2	1	$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \end{pmatrix} = \frac{1}{\|H\|^2} \sum_{j=1}^{M} \begin{pmatrix} h_{1,j}^* \, r_{1,j} + h_{2,j} \, r_{2,j}^* \\ h_{2,j}^* \, r_{1,j} - h_{1,j} \, r_{2,j}^* \end{pmatrix}.$$
3	1/2	$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \end{pmatrix} = \frac{1}{\|H\|^2} \sum_{j=1}^{M} \begin{pmatrix} h_{1,j}^* \, r_{1,j} + h_{2,j} \, r_{2,j}^* + h_{3,j} \, r_{3,j}^* \\ h_{2,j}^* \, r_{1,j} - h_{1,j} \, r_{2,j}^* - h_{3,j} \, r_{4,j}^* \end{pmatrix}.$$
3	3/4	$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}_3 \end{pmatrix} = \frac{1}{\|H\|^2} \sum_{j=1}^{M} \begin{pmatrix} h_{1,j}^* \, r_{1,j} + h_{2,j} \, r_{2,j}^* - h_{3,j} \, r_{3,j}^* \\ h_{2,j}^* \, r_{1,j} - h_{1,j} \, r_{2,j}^* - h_{3,j} \, r_{4,j}^* \\ h_{3,j}^* \, r_{1,j} + h_{1,j} \, r_{3,j}^* + h_{2,j} \, r_{4,j}^* \end{pmatrix}.$$
4	1/2	$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \end{pmatrix} = \frac{1}{\|H\|^2} \sum_{j=1}^{M} \begin{pmatrix} h_{1,j}^* \, r_{1,j} + h_{2,j} \, r_{2,j}^* + h_{3,j}^* 3,j \, r_{3,j} + h_{4,j} \, r_{4,j}^* \\ h_{2,j}^* \, r_{1,j} - h_{1,j} \, r_{2,j}^* + h_{4,j}^* \, r_{3,j} - h_{3,j} \, r_{4,j}^* \end{pmatrix}.$$
4	3/4	$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}_3 \end{pmatrix} = \frac{1}{\|H\|^2} \sum_{j=1}^{M} \begin{pmatrix} h_{1,j}^* \, r_{1,j} + h_{2,j} \, r_{2,j}^* - h_{3,j} \, r_{3,j}^* - h_{4,j}^* \, r_{4j} \\ h_{2,j}^* \, r_{1j} - h_{1,j} \, r_{2,j}^* + h_{4,j}^* \, r_{3,j} - h_{3,j} \, r_{4,j}^* \\ h_{3,j}^* \, r_{1j} + h_{4,j}^* \, r_{2,j} + h_{1,j} \, r_{3,j}^* + h_{2,j} \, r_{4,j}^* \end{pmatrix}.$$

$\hat{s}_k$ represents the estimated kth symbol in the OSTBC codeword matrix. h_{ij} represents the estimate for the channel from the ith transmit antenna and the jth receive antenna. The values of i and j can range from 1 to N (the number of transmit antennas) and to M (the number of receive antennas) respectively. r_{lj} represents the lth symbol at the jth receive antenna per codeword block. The value of l can range from 1 to

the codeword block length. $\|H\|^2$ represents the summation of channel

power per link, i.e., $\|H\|^2 = \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{M} \left\|h_{ij}\right\|^2$

Fixed-Point Signals

Use the following formula for $\hat{s}_1$ for Alamouti code with 1 receive antenna to highlight the data types used for fixed-point signals.

$$\hat{s}_1 = \frac{h_{1,1}^* \, r_{1,1} + h_{2,1} r_{2,1}^*}{\|H\|^2} = \frac{h_{1,1}^* \, r_{1,1} + h_{2,1} r_{2,1}^*}{h_{1,1} h_{1,1}^* + h_{2,1} h_{2,1}^*}$$

In this equation, the data types for **Product output** and **Accumulator** correspond to the product and summation in the numerator. Similarly, the types for **Energy product output** and **Energy accumulator** correspond to the product and summation in the denominator.
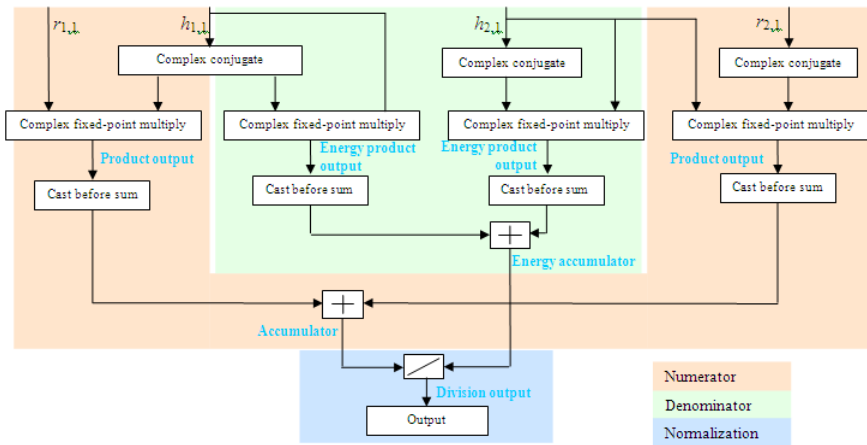
Signal Flow Diagram for s_1 Combining Calculation of Alamouti Code with One Receive Antenna

OSTBC Combiner

The following formula shows the data types used within the OSTBC Combiner block for fixed-point signals for more than one receive antenna for Alamouti code, where M represents the number of receive antennas.

$$\hat{s}_1 = \frac{h_{1,1}^*,\, r_{1,1} + h_{2,1}r_{2,1}^* + h_{1,2}^*\, r_{1,2} + h_{2,2}r_{2,2}^* + \ldots + h_{1,M}^*,\, r_{1,M} + h_{2,M}\, r_{2,M}^*}{h_{1.1}h_{1.1}^* + h_{2.1}h_{2.1}^* + h_{1.2}h_{1.2}^* + h_{2.2}h_{2.2}^* + \ldots + h_{1.M}h_{1.M}^* + h_{2.M}.h_{2.M}^*}$$
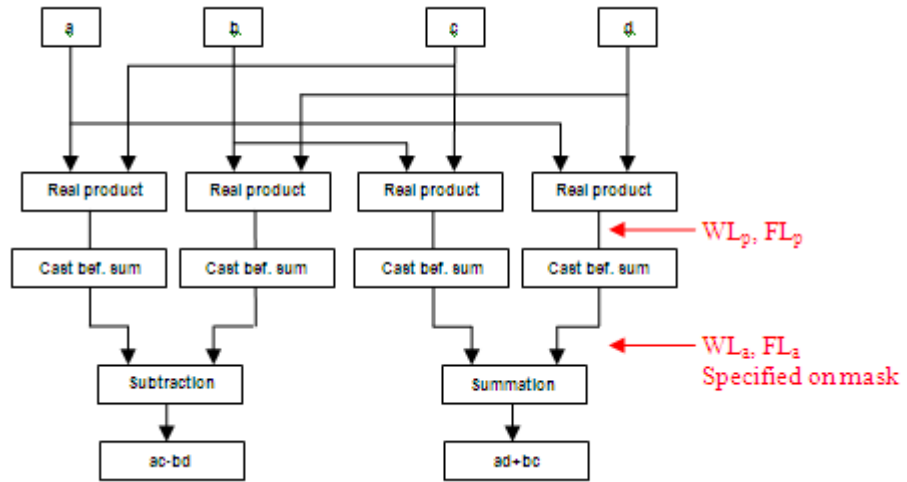
Signal Flow Diagram for Complex Multiply of *a + ib* and *c + id*

For Binary point scaling, you can not specify WL_p and FL_p. Instead, the blocks determine these values implicitly from WL_a and FL_a

The Internal Rule for **Product output** and **Energy product output** are:

- When you select `Inherit via internal rule`, the internal rule determines WL_p and FL_p. Therefore, $WL_a = WL_p + 1$ and $FL_a = FL_p$

- For `Binary point scaling`, you specify WL_a and FL_a. Therefore, $WL_p = WL_a - 1$ and $FL_a = FL_p$.

For information on how the Internal Rule applies to the **Accumulator** and **Energy Accumulator**, see Inherit via Internal Rule in the *Signal Processing Blockset User's Guide*.

Dialog Box
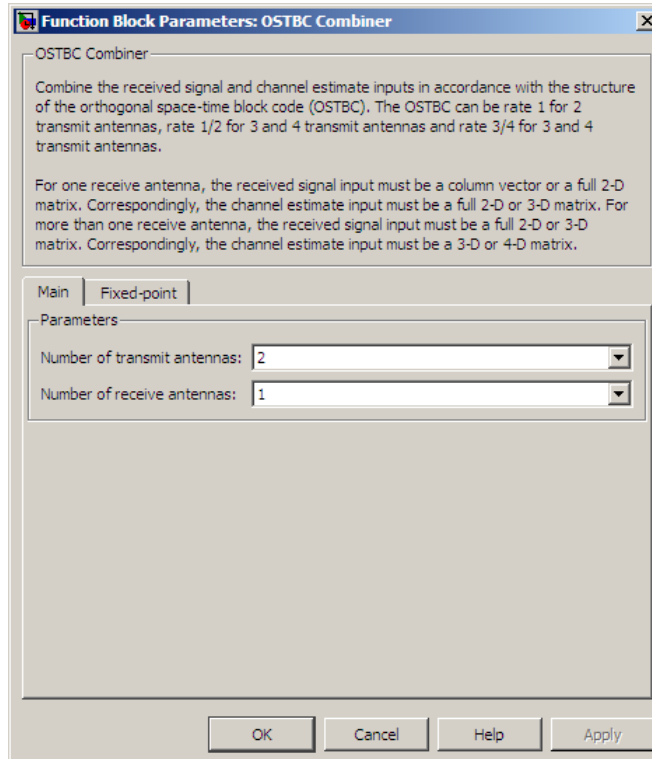
Block Parameters

Number of transmit antennas

Sets the number of transmit antennas. The block supports 2, 3, or 4 transmit antennas. This value defaults to 2.

Rate

Sets the symbol rate of the code. You can specify either 3/4 or 1/2. This field only appears when you use more than 2 transmit

antennas. This field defaults to $\dfrac{3}{4}$ for more than 2 transmit antennas. For 2 transmit antennas, there is no rate option and the implicit (default) rate defaults to 1.

Number of receive antennas

The number of antennas the block uses to receive signal streams. The block supports from 1 to 8 receive antennas. This value defaults to 1.
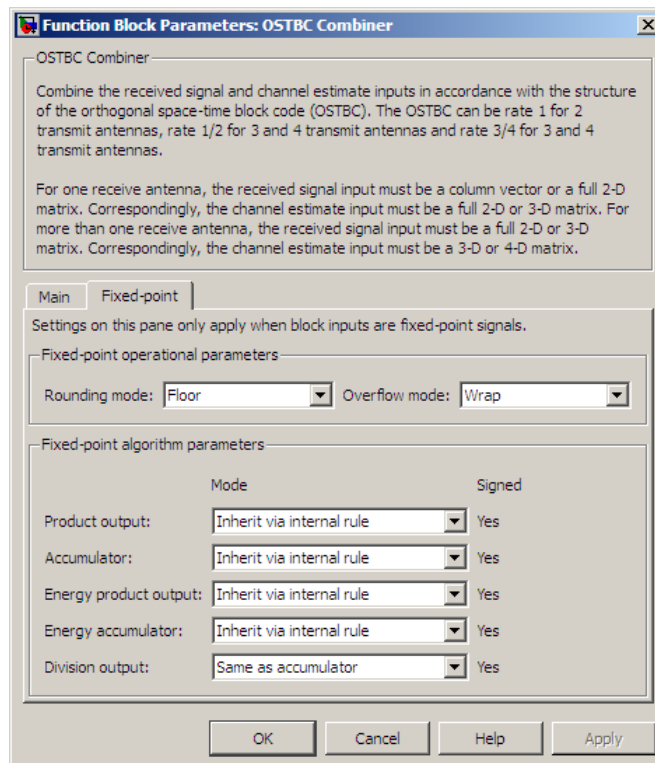
Rounding mode

Sets the rounding mode for fixed-point calculations. The block uses the rounding mode if a value cannot be represented exactly by the specified data type and scaling. When this occurs, the value is rounded to a representable number. For more information refer to Rounding Mode Parameter in the Specifying Fixed-Point Attributes section of the *Signal Processing Blockset User's Guide*.

Overflow mode

Sets the overflow mode for fixed-point calculations. Use this parameter to specify the method to be used if the magnitude of a fixed-point calculation result does not fit into the range of the data type and scaling that stores the result. For more information refer to Overflow Mode Parameter in the Specifying Fixed-Point Attributes section of the *Signal Processing Blockset User's Guide*.

Product Output

Complex product in the numerator for the diversity combining. For more information refer to the Fixed-Point Signals section of this help page.

Accumulator

Summation in the numerator for the diversity combining.

Fixed-point Communications Blockset blocks that must hold summation results for further calculation usually allow you to specify the data type and scaling of the accumulator. Most such blocks cast to the accumulator data type prior to summation:

The result of each addition remains in the accumulator data type.

See the reference page of a specific block in the Block Reference for details on the accumulator data type of a specific block.

Use the **Accumulator—Mode** parameter to specify how you would like to designate the accumulator word and fraction lengths:

- When you select Inherit via internal rule, the accumulator output word and fraction lengths are automatically calculated for you. Refer to Inherit via Internal Rule for more information.

- When you select Same as product output, these characteristics match those of the product output.

- When you select Same as input, these characteristics match those of the first input to the block.

- When you select Binary point scaling, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select Slope and bias scaling, you are able to enter the word length, in bits, and the slope of the accumulator. The bias of all signals in Signal Processing Blockset software is zero.

Energy product output

Complex product in the denominator for calculating total energy in the MIMO channel .

Energy accumulator

Summation in the denominator for calculating total energy in the MIMO channel.

Division output

Normalized diversity combining by total energy in the MIMO channel.

Examples   For an example of this block in use, see OSTBC Over 3x2 Rayleigh Fading Channel in the Communications blockset *User's Guide.* The model shows the use of a rate ¾ OSTBC for 3 transmit and 2 receive antennas with BPSK modulation using independent fading links and AWGN.

You can also see the block in the Concatenated OSTBC with TCM demo by typing `commtcmostbc` or the IEEE 802.16–2004 OFDM PHY Link, Including Space-Time Block Coding demo by typing `commwman80216dstbc` at the MATLAB command line.

See Also   OSTBC Encoder

OSTBC Encoder

Purpose　　　Encode input message using orthogonal space-time block code (OSTBC)

Library　　　MIMO

Description

The OSTBC Encoder block encodes an input symbol sequence using orthogonal space-time block code (OSTBC). The block maps the input symbols block-wise and concatenates the output codeword matrices in the time domain. For more information, see the OSTBC Encoding Algorithms section of this help page.
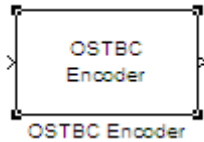
Dimension

The block supports time and spatial domains for OSTBC transmission. It also supports an optional dimension, over which the encoding calculation is independent. This dimension can be thought of as the frequency domain. The following illustration indicates the supported dimensions for the inputs and output of the OSTBC Encoder block.
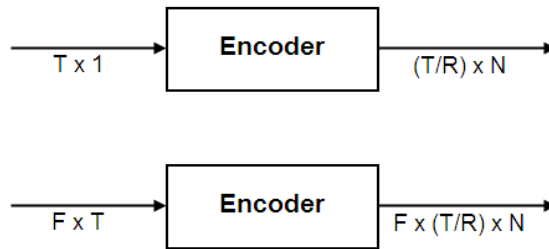
The following table describes the variables.

Variable	Description
F	The additional dimension; typically the frequency domain. The encoding does not depend on this dimension.
T	Input symbol sequence length for the time domain.

Variable	Description
R	Symbol rate of the code.
N	Number of transmit antennas.

Note On the output, T/R is the symbol sequence length in time domain.

F can be any positive integer. N can be 2, 3 or 4, indicated by **Number of transmit antennas**. For $N = 2$, R must be 1. For $N = 3$ or 4, R can be 3/4 or 1/2, indicated by **Rate**. The time domain length T must be a multiple of the number of symbols in each codeword matrix. Specifically, for $N = 2$ or $R = 1/2$, T must be a multiple of 2 and when $R = 3/4$, T must be a multiple of 3.

To understand the block's dimension propagation, refer to the following table.

Dimension	Input	Output
F = 1	Column vector	2-D
F > 1	2-D	3-D

Data Type

The input signal must be double, single, or signed fixed-point. The output signal inherits the data type from the input signal. For fixed-point signals, the complex conjugation may cause overflows which the fixed-point parameter **Overflow mode** must handle.

Frames

The output signal inherits frame type from the input signal. A column vector input requires either frame-based or sample-based input; otherwise, the input must be sample-based.

OSTBC Encoder

OSTBC Encoding Algorithms

The OSTBC Encoder block supports five different OSTBC encoding algorithms. Depending on the selection for **Rate** and **Number of transmit antennas**, the block implements one of the algorithms in the following table:

Transmit Antenna	Rate	OSTBC Codeword Matrix
2	1	$\begin{pmatrix} s_1 & s_2 \\ -s_2^* & s_1^* \end{pmatrix}$
3	1/2	$\begin{pmatrix} s_1 & s_2 & 0 \\ -s_2^* & s_1^* & 0 \\ 0 & 0 & s_1 \\ 0 & 0 & -s_2^* \end{pmatrix}$
3	3/4	$\begin{pmatrix} s_1 & s_2 & s_3 \\ -s_2^* & s_1^* & 0 \\ s_3^* & 0 & -s_1^* \\ 0 & s_3^* & -s_2^* \end{pmatrix}$

Transmit Antenna	Rate	OSTBC Codeword Matrix
4	1/2	$$\begin{pmatrix} s_1 & s_2 & 0 & 0 \\ -s_2^* & s_1^* & 0 & 0 \\ 0 & 0 & s_1 & s_2 \\ 0 & 0 & -s_2^* & s_1^* \end{pmatrix}$$
4	3/4	$$\begin{pmatrix} s_1 & s_2 & s_3 & 0 \\ -s_2^* & s_1^* & 0 & s_3 \\ s_3^* & 0 & -s_1^* & s_2 \\ 0 & s_3^* & -s_2^* & -s_1 \end{pmatrix}$$

In each matrix, its (l, i) entry indicates the symbol transmitted from the ith antenna in the lth time slot of the block. The value of i can range from 1 to N (the number of transmit antennas). The value of l can range from 1 to the codeword block length.

OSTBC Encoder

Dialog Box
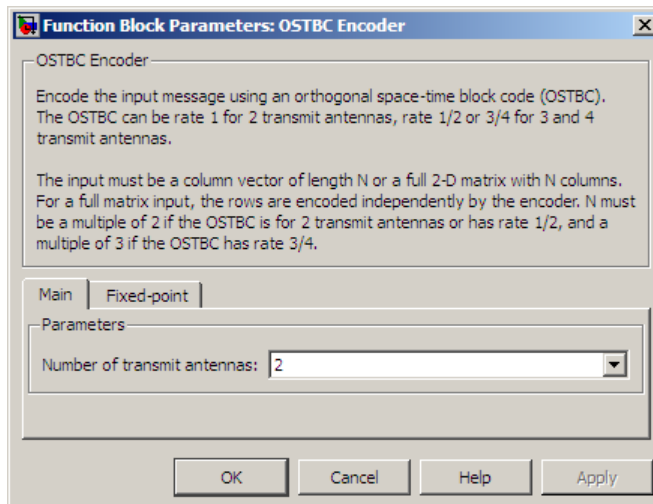
Block Parameters

Number of transmit antennas

Sets the number of antennas at the transmitter side. The block supports 2, 3, or 4 transmit antennas. The value defaults to 2.

Rate

Sets the symbol rate of the code. You can specify either 3/4 or 1/2. This field only appears when using more than 2 transmit antennas. This field defaults to $\frac{3}{4}$ for more than 2 transmit antennas. For 2 transmit antennas, there is no rate option and the rate defaults to 1.
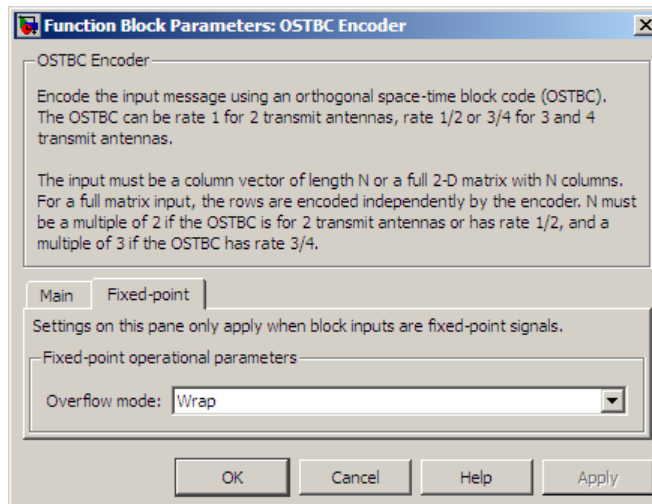
Encode the input message using an orthogonal space-time block code (OSTBC). The OSTBC can be rate 1 for 2 transmit antennas, rate 1/2 or 3/4 for 3 and 4 transmit antennas.

The input must be a column vector of length N or a full 2-D matrix with N columns. For a full matrix input, the rows are encoded independently by the encoder. N must be a multiple of 2 if the OSTBC is for 2 transmit antennas or has rate 1/2, and a multiple of 3 if the OSTBC has rate 3/4.

Main | Fixed-point

Settings on this pane only apply when block inputs are fixed-point signals.

Fixed-point operational parameters

Overflow mode: Wrap

Overflow mode

> Sets the overflow mode for fixed-point calculations. Use this parameter to specify the method to be used if the magnitude of a fixed-point calculation result does not fit into the range of the data type and scaling that stores the result. For more information refer to Overflow Mode Parameter in the Specifying Fixed-Point Attributes section of the *Signal Processing Blockset User's Guide*.

Examples   For an example of this block in use, see OSTBC Over 3x2 Rayleigh Fading Channel in the Communications blockset *User's Guide*. The model shows the use of a rate ¾ OSTBC for 3 transmit and 2 receive antennas with BPSK modulation using independent fading links and AWGN

You can also see the block in the Concatenated OSTBC with TCM demo by typing commtcmostbc. View the IEEE 802.16–2004 OFDM PHY Link, Including Space-Time Block Coding demo by typing commwman80216dstbc at the MATLAB command line.

See Also   OSTBC Combiner

OVSF Code Generator

Purpose   Generate orthogonal variable spreading factor (OVSF) code from set of orthogonal codes

Library   Spreading Codes
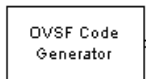
Description   The OVSF Code Generator block generates an OVSF code from a set of orthogonal codes. OVSF codes were first introduced for 3G communication systems. OVSF codes are primarily used to preserve orthogonality between different channels in a communication system.

OVSF codes are defined as the rows of an N-by-N matrix, C_N, which is defined recursively as follows. First, define $C_1 = [1]$. Next, assume that C_N is defined and let $C_N(k)$ denote the kth row of C_N. Define C_{2N} by

$$C_{2N} = \begin{bmatrix} C_N(0) & C_N(0) \\ C_N(0) & -C_N(0) \\ C_N(1) & C_N(1) \\ C_N(1) & -C_N(1) \\ ... & ... \\ C_N(N-1) & C_N(N-1) \\ C_N(N-1) & -C_N(N-1) \end{bmatrix}$$

Note that C_N is only defined for N a power of 2. It follows by induction that the rows of C_N are orthogonal.

The OVSF codes can also be defined recursively by a tree structure, as shown in the following figure.

$$C_{8,0} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$$

$$C_{4,0} = [1\ 1\ 1\ 1]$$

$$C_{8,1} = [1\ 1\ 1\ 1\ -1\ -1\ -1\ -1]$$

$$C_{2,0} = [1\ 1]$$

$$C_{8,2} = [1\ 1\ -1\ -1\ 1\ 1\ -1\ -1]$$

$$C_{4,1} = [1\ 1\ -1\ -1]$$

$$C_{8,3} = [1\ 1\ -1\ -1\ -1\ -1\ 1\ 1]$$

$$C_{1,0} = [1]$$

$$C_{8,4} = [1\ -1\ 1\ -1\ 1\ -1\ 1\ -1]$$

$$C_{4,2} = [1\ -1\ 1\ -1]$$

$$C_{8,5} = [1\ -1\ 1\ -1\ -1\ 1\ -1\ 1]$$

$$C_{2,1} = [1\ -1]$$

$$C_{8,6} = [1\ -1\ -1\ 1\ 1\ -1\ -1\ 1]$$

$$C_{4,3} = [1\ -1\ -1\ 1]$$

$$C_{8,7} = [1\ -1\ -1\ 1\ -1\ 1\ 1\ -1]$$

SF = 1   SF = 2   SF = 4   SF = 8
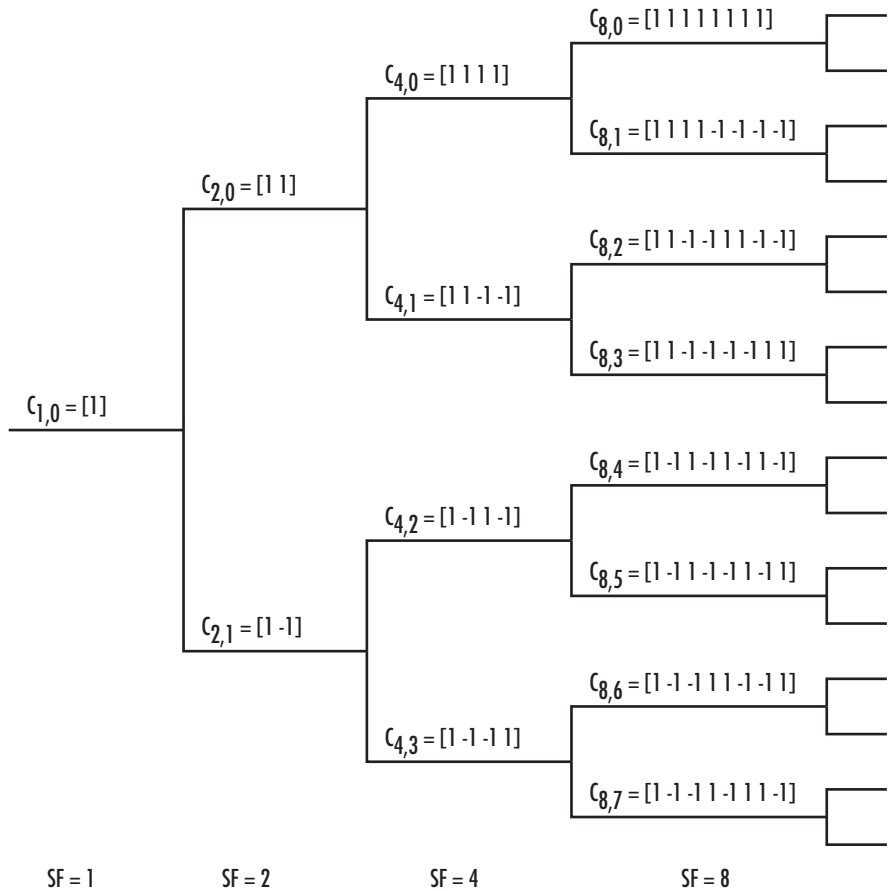
If [C] is a code length 2^r at depth r in the tree, where the root has depth 0, the two branches leading out of C are labeled by the sequences [C C] and [C -C], which have length 2^{r+1}. The codes at depth r in the tree are the rows of the matrix C_N, where $N = 2^r$.

Note that two OVSF codes are orthogonal if and only if neither code lies on the path from the other code to the root. Since codes assigned to different users in the same cell must be orthogonal, this restricts the number of available codes for a given cell. For example, if the code C_{41}

in the tree is assigned to a user, the codes C_{10}, C_{20}, C_{82}, C_{83}, and so on, cannot be assigned to any other user in the same cell.

Block Parameters

You specify the code the OVSF Code Generator block outputs by two parameters in the block's dialog: the **Spreading factor**, which is the length of the code, and the **Code index**, which must be an integer in the range $[0, 1, \ldots, N - 1]$, where N is the spreading factor. If the code appears at depth r in the preceding tree, the **Spreading factor** is 2^r. The **Code index** specifies how far down the column of the tree at depth r the code appears, counting from 0 to $N - 1$. For $C_{N, k}$ in the preceding diagram, N is the **Spreading factor** and k is the **Code index**.

You can recover the code from the **Spreading factor** and the **Code index** as follows. Convert the **Code index** to the corresponding binary number, and then add 0s to the left, if necessary, so that the resulting binary sequence $x_1 \, x_2 \ldots x_r$ has length r, where r is the logarithm base 2 of the **Spreading factor**. This sequence describes the path from the root to the code. The path takes the upper branch from the code at depth i if $x_i = 0$, and the lower branch if $x_i = 1$.

To reconstruct the code, recursively define a sequence of codes C_i for as follows. Let C_0 be the root [1]. Assuming that C_i has been defined, for $i < r$, define C_{i+1} by

$$C_{i+1} = \begin{cases} C_i C_i & \text{if } x_i = 0 \\ C_i(-C_i) & \text{if } x_i = 1 \end{cases}$$

The code C_N has the specified **Spreading factor** and **Code index**.

For example, to find the code with **Spreading factor** 16 and **Code index** 6, do the following:

1 Convert 6 to the binary number 110.

2 Add one 0 to the left to obtain 0110, which has length $4 = \log_2 16$.

3 Construct the sequences C_i according to the following table.

i	x_i	C_i
0		$C_0 = [1]$
1	0	$C_1 = C_0\ C_0 = [1]\ [1]$
2	1	$C_2 = C_1\ \text{-}C_1 = [1\ 1]\ [\text{-}1\ \text{-}1]$
3	1	$C_3 = C_2\ \text{-}C_2 = [1\ 1\ \text{-}1\ \text{-}1]\ [\text{-}1\ \text{-}1\ 1\ 1]$
4	0	$C_4 = C_3\ C_3 = [1\ 1\ \text{-}1\ \text{-}1\ \text{-}1\ \text{-}1\ 1\ 1]\ [1\ 1\ \text{-}1\ \text{-}1\ \text{-}1\ \text{-}1\ 1\ 1]$

The code C_4 has **Spreading factor** 16 and **Code index** 6.

Dialog Box
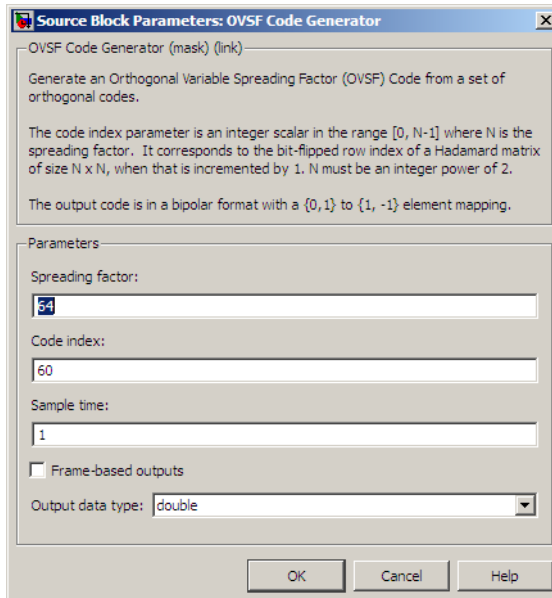
Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

OVSF Code Generator

Spreading factor

Positive integer that is a power of 2, specifying the length of the code.

Code index

Integer in the range [0, 1, ... , N - 1] specifying the code, where N is the **Spreading factor**.

Sample time

A positive real scalar specifying the sample time of the output signal.

Frame-based outputs

Determines whether the output is frame-based or sample-based.

Samples per frame

The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box.

Output data type

The output type of the block can be specified as an `int8` or `double`. By default, the block sets this to `double`.

See Also   Hadamard Code Generator, Walsh Code Generator

Purpose   Apply phase and frequency offsets to complex baseband signal

Library   RF Impairments
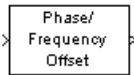
Description   The Phase/Frequency Offset block applies phase and frequency offsets to an incoming signal.

This block operates without any delay, and accepts real and complex inputs of data type `double` or `single`. Its output data type matches that of its input.

Phase Offset

The block applies a phase offset to the input signal, specified by the **Phase offset** parameter.

Frequency Offset

The block applies a frequency offset to the signal that is specified by the **Frequency offset** parameter. Alternatively, this information may be provided to the block through an input port that is enabled when the **Frequency offset from port** checkbox is selected.

The block supports signals of single-samples, multiple-sample single-channel frames, single-sample multi-channel frames, and multi-sample multi-channel frames. The two block parameters can be scalars, vectors, or matrices, as appropriate with the input signal's dimensions. If the input and frequency signals are both frame-based, they must both have the same frame size. If the frequency signal is multi-channel, it must have either one channel or the same number of channels as the input signal. If it is not a scalar, the number of samples in the channel must match the input signal.

The effects of changing the block's parameters are illustrated by the following scatter plots of a signal modulated by 16-ary quadrature amplitude modulation (QAM). The usual 16-ary QAM constellation without the effect of the Phase/Frequency Offset block is shown in the first scatter plot:

Phase/Frequency Offset
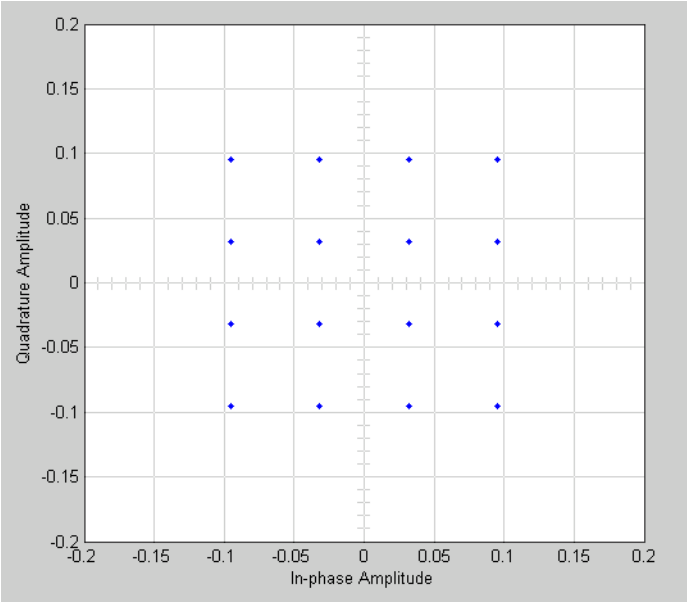
The following figure shows a scatter plot of an output signal, modulated by 16-ary QAM, from the Phase/Frequency Offset block with **Phase offset** set to 20 and **Frequency offset** set to 0:
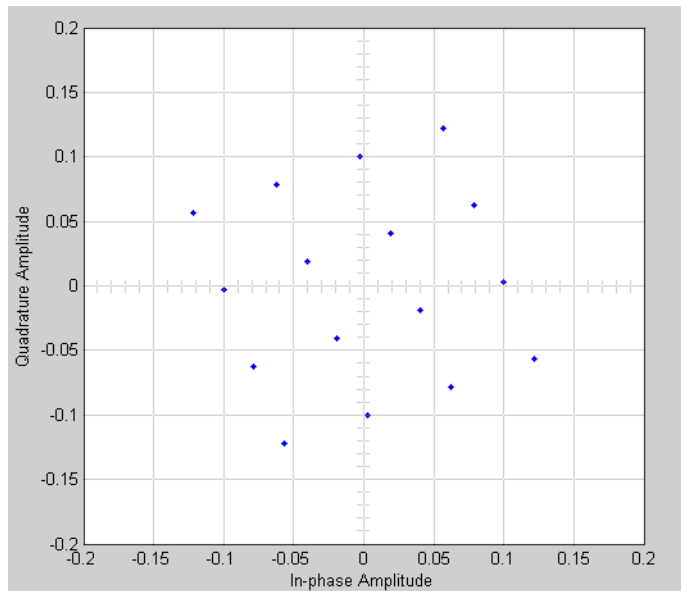
Observe that each point in the constellation is rotated by a 20 degree angle counterclockwise.

If you set **Frequency offset** to 2 and **Phase offset** to 0, the angles of points in the constellation change linearly over time. This causes points in the scatter plot to shift radially, as shown in the following figure:

Phase/Frequency Offset
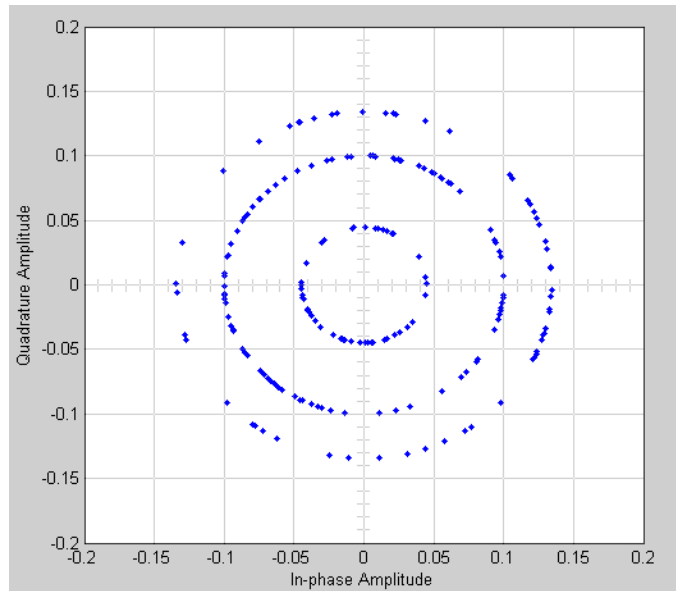
Note that every point in the scatter plot has magnitude equal to a point in the original constellation.

See "Scatter Plot Examples" for a description of the model that generates this plot.

I/O Ports

Only 1-D (unoriented) and 2-D ports are supported, as well as 2-D frame-based (single or multichannel) ports.

Scalar (length-one) sample-based data ports are also supported, but nonscalar sample-based ports are not.

Interdependent Parameter-Port Dimensions Table

Data I/O Frmstat	No. of Dim.	Data I/O Dim.	Frame Size	No. of Channels	Freq. or Phase Offset Parameter Dim.
Any	Any	Scalar	1	1	Scalar

Data I/O Frmstat	No. of Dim.	Data I/O Dim.	Frame Size	No. of Channels	Freq. or Phase Offset Parameter Dim.
Frame-based	2	M x 1	M	1	M, M x 1, 1 x M, 1, 1 x 1
Frame-based	2	1 x N	1	N	N, N x 1, 1 x N, 1, 1 x 1
Frame-based	2	M x N	M	N	M x N, N, N x 1, 1 x N, 1, 1 x 1
Sample-based	2	M x 1 (Errors — not currently supported)			
Sample-based	2	1 x N (Errors — not currently supported)			
Sample-based	2	M x N (Errors — not currently supported)			
Sample-based	1	L (Errors — not currently supported)			

Phase/Frequency Offset

**Dialog
Box**

Function Block Parameters: Phase/ Frequency Offset

Phase/Frequency Offset (mask)

Apply a frequency and phase offset to the input signal.

Parameters

☐ Frequency offset from port

Frequency offset (Hz):

0

Phase offset (deg):

0

OK   Cancel   Help   Apply
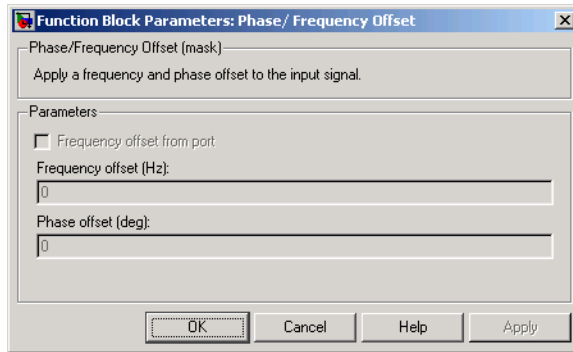
Frequency offset from port
> Selecting this option opens a port on the block through which you
> can input the frequency offset information.

Frequency offset
> Specifies the frequency offset in hertz.

Phase offset
> Specifies the phase offset in degrees.

If **Frequency offset** and **Phase offset** are both vectors or both
matrices, their dimensions (vector lengths, or number of rows and
columns) must be the same.

See Also   Phase Noise

Purpose   Implement phase-locked loop to recover phase of input signal

Library   Components sublibrary of Synchronization
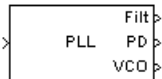
Description   The Phase-Locked Loop (PLL) block is a feedback control system that
automatically adjusts the phase of a locally generated signal to match
the phase of an input signal. This block is most appropriate when the
input is a narrowband signal.

This PLL has these three components:

- A multiplier used as a phase detector.

- A filter. You specify the filter's transfer function using the **Lowpass
filter numerator** and **Lowpass filter denominator** parameters.
Each is a vector that gives the respective polynomial's coefficients in
order of descending powers of *s*.

  To design a filter, you can use functions such as `butter`, `cheby1`, and
`cheby2` in Signal Processing Toolbox software. The default filter is
a Chebyshev type II filter whose transfer function arises from the
command below.

  `[num, den] = cheby2(3,40,100,'s')`

- A voltage-controlled oscillator (VCO). You specify characteristics of
the VCO using the **VCO quiescent frequency**, **VCO initial phase**,
and **VCO output amplitude** parameters.

The input signal represents the received signal. The input must be a
sample-based scalar signal. The three output ports produce:

- The output of the filter

- The output of the phase detector

- The output of the VCO

Phase-Locked Loop

**Dialog
Box**

Lowpass filter numerator

The numerator of the lowpass filter's transfer function,
represented as a vector that lists the coefficients in order of
descending powers of *s*.

Lowpass filter denominator

The denominator of the lowpass filter's transfer function,
represented as a vector that lists the coefficients in order of
descending powers of *s*.

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the
shift from the **VCO quiescent frequency** value. The units of
VCO input sensitivity are Hertz per volt.

VCO quiescent frequency (Hz)

The frequency of the VCO signal when the voltage applied to it is
zero. This should match the carrier frequency of the input signal.

VCO initial phase (rad)
The initial phase of the VCO signal.

VCO output amplitude
The amplitude of the VCO signal.

See Also   Baseband PLL, Linearized Baseband PLL, Charge Pump PLL

References   For more information about phase-locked loops, see the works listed in "Selected Bibliography for Synchronization" in *Communications Blockset User's Guide*.

Phase Noise

Purpose Apply receiver phase noise to complex baseband signal

Library RF Impairments

Description
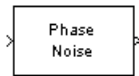
Phase
Noise

The Phase Noise block applies phase noise to a complex, baseband signal. The block applies the phase noise as follows:

1 Generates additive white Gaussian noise (AWGN) and filters it with a digital filter.

2 Adds the resulting noise to the angle component of the input signal.

The Phase Noise block generates phase noise over the entire spectral observation window, from 0 Hz (or as close as possible to 0 Hz) to $\pm\frac{F_S}{2}$, where F_s represents the sampling frequency. The noise is scaled so that it is at the block-specific phase noise level at the specified frequency offset. The block generates a phase noise with $\frac{1}{f}$ characteristic over the entire frequency range.

You can view the block's implementation of phase noise by right-clicking on the block and selecting **Look under mask**. This displays the following figure:
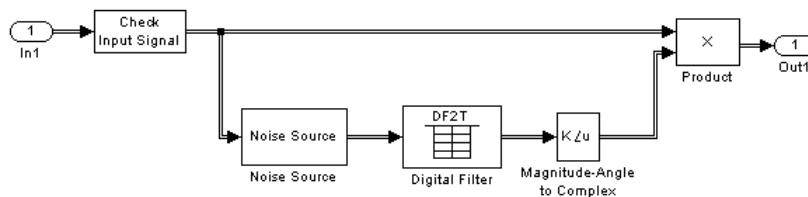
You can view the construction of the Noise Source subsystem by double-clicking it.

The effects of changing the block's parameters are illustrated by the following scatter plots of a signal modulated by 16-ary quadrature amplitude modulation (QAM). The usual 16-ary QAM constellation without distortion is shown in the first scatter plot:
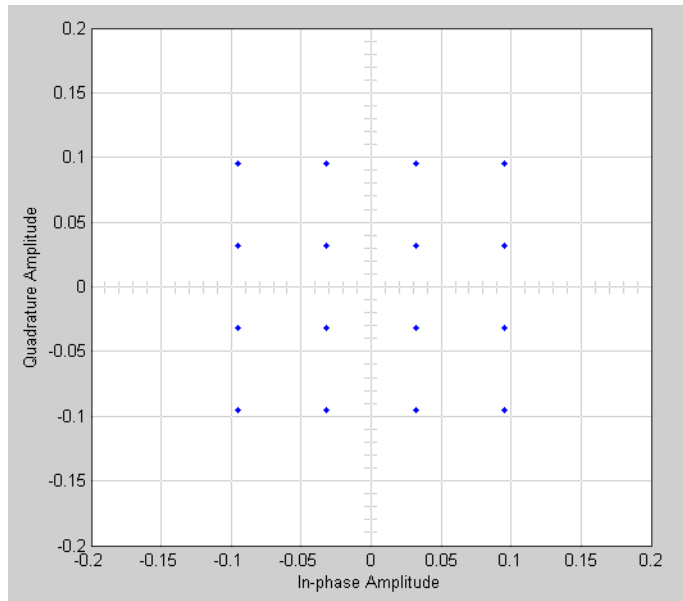
The following figure shows a scatter plot of an output signal, modulated by 16-ary QAM, from the Phase Noise block with **Phase noise level (dBc/Hz)** set to -70 and **Frequency offset (Hz)** set to 100:

Phase Noise
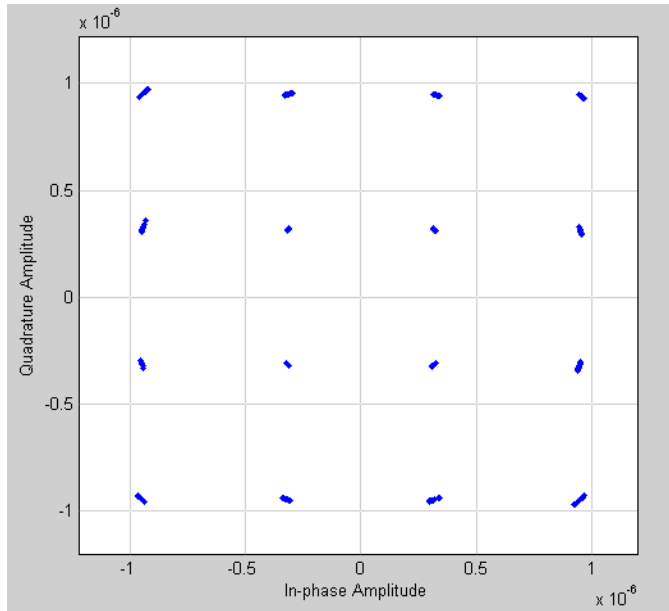
This plot is generated by the model described in "Scatter Plot Examples" with the following parameter settings for the Rectangular QAM Modulator Baseband block:

- **Normalization method** set to Average Power
- **Average power (watts)** set to 1e-12

Dialog Box
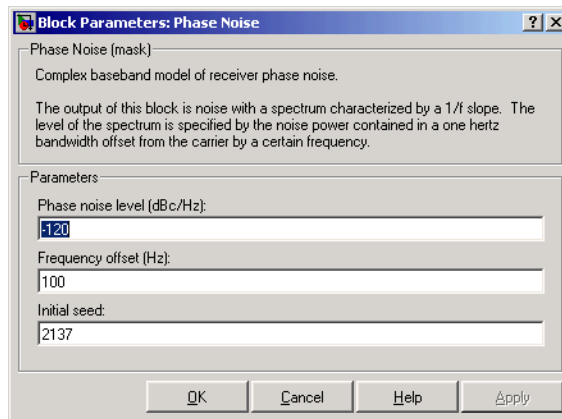
Phase noise level (dBc/Hz)
  Scalar specifying the phase noise level.

Frequency offset (Hz)
  Scalar specifying the frequency offset in Hertz.

Initial seed
  Nonnegative integer specifying the initial seed for the random number generator the block uses to generate noise.

See Also   Phase/Frequency Offset

References   [1] Kasdin, N.J., "Discrete Simulation of Colored Noise and Stochastic Processes and 1/(f^alpha); Power Law Noise Generation," The Proceedings of the IEEE, May, 1995, Vol. 83, No. 5

PM Demodulator Passband

Purpose   Demodulate PM-modulated data

Library   Analog Passband Modulation, in Modulation
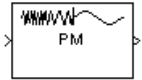
Description   The PM Demodulator Passband block demodulates a signal that was modulated using phase modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

For best results, use a carrier frequency which is estimated to be larger than 10% of your input signal's sample time. This is due to the implementation of the Hilbert transform by means of a filter.

In the following example, we sample a 10Hz input signal at 8000 samples per second. We then designate a Hilbert Transform filter of order 100. Below is the response of the Hilbert Transform filter as returned by `fvtool`.

Note the bandwidth of the filter's magnitude response. By choosing a carrier frequency larger than 10% (but less than 90%) of the input signal's sample time (8000 samples per second, in this example) or equivalently, a carrier frequency larger than 400Hz, we ensure that the Hilbert Transform Filter will be operating in the flat section of the filter's magnitude response (shown in blue), and that our modulated signal will have the desired magnitude and form.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

PM Demodulator Passband

This block works only with real inputs of type double. This block is not suited to be placed inside a triggered subsystem.

Dialog Box
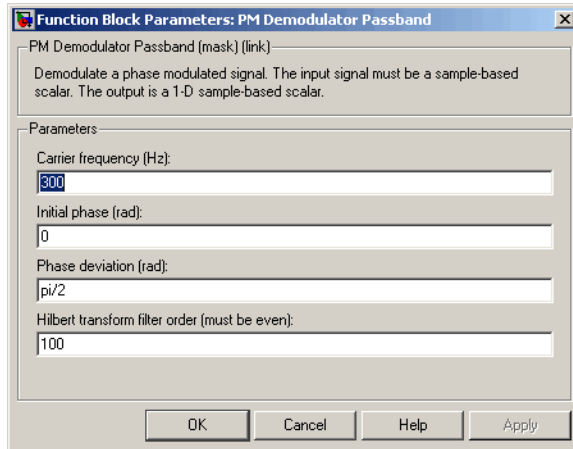
Carrier frequency (Hz)
    The frequency of the carrier.

Initial phase (rad)
    The initial phase of the carrier in radians.

Frequency deviation (Hz)
    The phase deviation of the carrier frequency in radians.
    Sometimes it is referred to as the "variation" in the phase.

Hilbert transform filter order
    The length of the FIR filter used to compute the Hilbert transform.

Pair Block   PM Modulator Passband

Purpose   Modulate using phase modulation

Library   Analog Passband Modulation, in Modulation
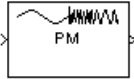
Description   The PM Modulator Passband block modulates using phase modulation. The output is a passband representation of the modulated signal. The output signal's frequency varies with the input signal's amplitude. Both the input and output signals are real sample-based scalar signals.

If the input is $u(t)$ as a function of time t, then the output is

$$\cos(2\pi f_c t + K_c u(t) + \theta)$$

where f_c is the **Carrier frequency** parameter, θ is the **Initial phase** parameter, and K_c is the **Modulation constant** parameter.

An appropriate **Carrier frequency** value is generally much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

This block works only with real inputs of type `double`. This block is not suited to be placed inside a triggered subsystem.

Dialog Box

Function Block Parameters: PM Modulator Passband

PM Modulator Passband (mask) (link)

Modulate the input signal using the phase modulation method. The input signal must be a sample-based scalar. The output is a 1-D sample-based scalar.

Parameters

Carrier frequency (Hz):
800

Initial phase (rad):
0

Phase deviation (rad):
pi/2

| OK | Cancel | Help | Apply |
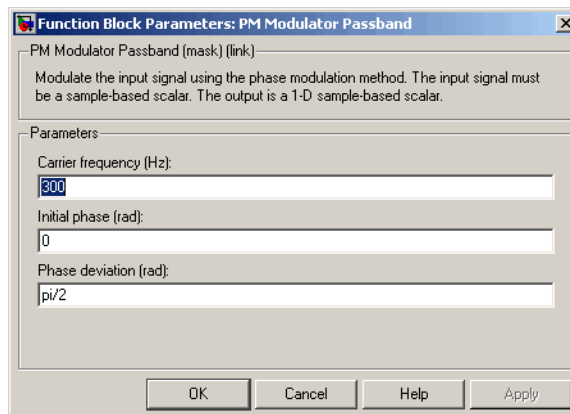
PM Modulator Passband

Carrier frequency (Hz)
> The frequency of the carrier.

Initial phase (rad)
> The initial phase of the carrier in radians.

Phase deviation
> The phase deviation of the carrier frequency in radians. This is sometimes referred to as the "variation" in the phase.
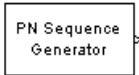
Pair Block   PM Demodulator Passband

Purpose   Generate pseudonoise sequence

**Library**   Sequence Generators sublibrary of Comm Sources

Description   The PN Sequence Generator block generates a sequence of
pseudorandom binary numbers using a linear-feedback shift register
(LFSR). The LFSR is implemented using a simple shift register
generator (SSRG, or Fibonacci) configuration. A pseudonoise sequence
can be used in a pseudorandom scrambler and descrambler. It can also
be used in a direct-sequence spread-spectrum system.

PN Sequence
Generator

The PN Sequence Generator block uses a shift register to generate
sequences, as shown below.
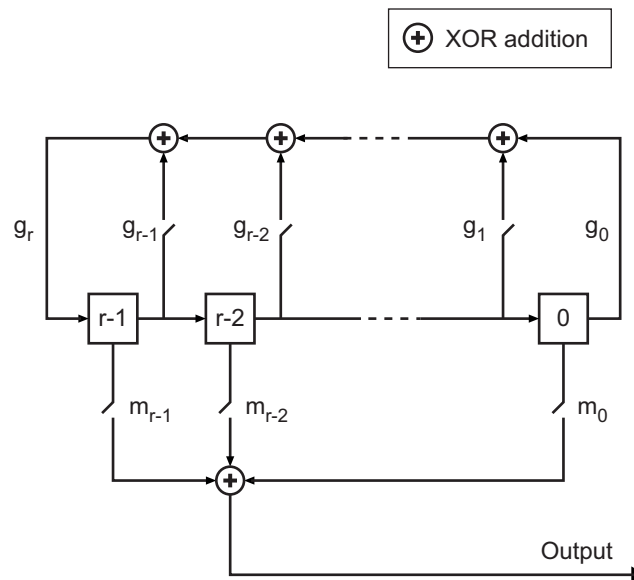
All *r* registers in the generator update their values at each time step
according to the value of the incoming arrow to the shift register. The
adders perform addition modulo 2. The shift register is described by
the **Generator Polynomial** parameter, which is a primitive binary

polynomial in z, $g_r z^r + g_{r-1} z^{r-1} + g_{r-2} z^{r-2} + \ldots + g_0$. The coefficient g_k is 1 if there is a connection from the kth register, as labeled in the preceding diagram, to the adder. The leading term g_r and the constant term g_0 of the **Generator Polynomial** parameter must be 1 because the polynomial must be primitive.

You can specify the **Generator polynomial** parameter using either of these formats:

- A vector that lists the coefficients of the polynomial in descending order of powers. The first and last entries must be 1. Note that the length of this vector is one more than the degree of the generator polynomial.

- A vector containing the exponents of z for the nonzero terms of the polynomial in descending order of powers. The last entry must be 0.

For example, [1 0 0 0 0 1 0 1] and [8 2 0] represent the same polynomial, $p(z) = z^8 + z^2 + 1$.

The **Initial states** parameter is a vector specifying the initial values of the registers. The **Initial states** parameter must satisfy these criteria:

- All elements of the **Initial states** vector must be binary numbers.

- The length of the **Initial states** vector must equal the degree of the generator polynomial.

Note At least one element of the **Initial states** vector must be nonzero in order for the block to generate a nonzero sequence. That is, the initial state of at least one of the registers must be nonzero.

For example, the following table indicates two sets of parameter values that correspond to a generator polynomial of $p(z) = z^8 + z^2 + 1$.

Quantity	Example 1	Example 2
Generator polynomial	g1 = [1 0 0 0 0 0 1 0 1]	g2 = [8 2 0]
Degree of generator polynomial	8, which is `length(g1)-1`	8
Initial states	[1 0 0 0 0 0 1 0]	[1 0 0 0 0 0 1 0]

Output mask vector (or scalar shift value) shifts the starting point of the output sequence. With the default setting for this parameter, the only connection is along the arrow labeled m_0, which corresponds to a shift of 0. The parameter is described in greater detail below.

You can shift the starting point of the PN sequence with **Output mask vector (or scalar shift value)**. You can specify the parameter in either of two ways:

- An integer representing the length of the shift

- A binary vector, called the *mask vector*, whose length is equal to the degree of the generator polynomial

The difference between the block's output when you set **Output mask vector (or scalar shift value)** to 0, versus a positive integer d, is shown in the following table.

	T = 0	**T = 1**	**T = 2**	**...**	**T = d**	**T = d+1**
Shift = 0	x_0	x_1	x_2	...	x_d	x_{d+1}
Shift = d	x_d	x_{d+1}	x_{d+2}	...	x_{2d}	x_{2d+1}

Alternatively, you can set **Output mask vector (or scalar shift value)** to a binary vector, corresponding to a polynomial in *z*, $m_{r-1}z^{r-1} + m_{r-2}z^{r-2} + ... + m_1z + m_0$, of degree at most r-1. The mask vector

corresponding to a shift of d is the vector that represents $m(z) = z^d$ modulo $g(z)$, where $g(z)$ is the generator polynomial. For example, if the degree of the generator polynomial is 4, then the mask vector corresponding to $d = 2$ is [0 1 0 0], which represents the polynomial $m(z) = z^2$. The preceding schematic diagram shows how **Output mask vector (or scalar shift value)** is implemented when you specify it as a mask vector. The default setting for **Output mask vector (or scalar shift value)** is 0. You can calculate the mask vector using the Communications Toolbox function shift2mask.

You can use an external signal to reset the values of the internal shift register to the initial state by selecting the **Reset on nonzero input** check box. This creates an input port for the external signal in the PN Sequence Generator block. The way the block resets the internal shift register depends on whether its output signal and the reset signal are sample-based or frame-based. The following example demonstrates the possible alternatives.

Example: Resetting a Signal

Suppose that the PN Sequence Generator block outputs [1 0 0 1 1 0 1 1] when there is no reset. You then select the **Reset on nonzero input** check box and input a reset signal [0 0 0 1]. The following table shows three possibilities for the properties of the reset signal and the PN Sequence Generator block.

Reset Signal Properties	PN Sequence Generator block	Reset Signal, Output Signal
Sample-based **Sample time** = 1	Sample-based **Sample time** = 1	Reset 0 0 0 [1] 1 0 0 [1] 0 0 1 1 0 1 1

Reset Signal Properties	PN Sequence Generator block	Reset Signal, Output Signal
Frame-based **Sample time** =1 **Samples per frame** = 2	Frame-based **Sample time** = 1 **Samples per frame** = 2	Reset 0 0 0 |1| 1 0 0 |1| 0 0 1 1 0 1 1
Sample-based **Sample time** = 2 **Samples per frame** = 1	Frame-based **Sample time** = 1 **Samples per frame** = 2	Reset 0 0 0 |1| 1 0 0 1 0 0 |1| 0 0 1 1 0 1 1

In the first two cases, the PN sequence is reset at the fourth bit, because the fourth bit of the reset signal is a 1 and the **Sample time** is 1. Note that in the second case, the frame sizes are 2, and the reset occurs at the end of the second frame.

In the third case, the PN sequence is reset at the seventh bit. This is because the reset signal has **Sample time** 2, so the reset bit is first sampled at the seventh bit. With these settings, the reset always occurs at the beginning of a frame.

Attributes of Output Signal

If the **Frame-based outputs** box is selected, the output signal is a frame-based column vector whose length is the **Samples per frame** parameter. Otherwise, the output signal is a one-dimensional scalar.

Sequences of Maximum Length

If you want to generate a sequence of the maximum possible length for a fixed degree, r, of the generator polynomial, you can set **Generator polynomial** to a value from the following table. See [1] for more information about the shift-register configurations that these polynomials represent.

PN Sequence Generator

r	Generator Polynomial	r	Generator Polynomial
2	[2 1 0]	21	[21 19 0]
3	[3 2 0]	22	[22 21 0]
4	[4 3 0]	23	[23 18 0]
5	[5 3 0]	24	[24 23 22 17 0]
6	[6 5 0]	25	[25 22 0]
7	[7 6 0]	26	[26 25 24 20 0]
8	[8 6 5 4 0]	27	[27 26 25 22 0]
9	[9 5 0]	28	[28 25 0]
10	[10 7 0]	29	[29 27 0]
11	[11 9 0]	30	[30 29 28 7 0]
12	[12 11 8 6 0]	31	[31 28 0]
13	[13 12 10 9 0]	32	[32 31 30 10 0]
14	[14 13 8 4 0]	33	[33 20 0]
15	[15 14 0]	34	[34 15 14 1 0]
16	[16 15 13 4 0]	35	[35 2 0]
17	[17 14 0]	36	[36 11 0]
18	[18 11 0]	37	[37 12 10 2 0]
19	[19 18 17 14 0]	38	[38 6 5 1 0]
20	[20 17 0]	39	[39 8 0]
40	[40 5 4 3 0]	47	[47 14 0]
41	[41 3 0]	48	[48 28 27 1 0]
42	[42 23 22 1 0]	49	[49 9 0]
43	[43 6 4 3 0]	50	[50 4 3 2 0]
44	[44 6 5 2 0]	51	[51 6 3 1 0]

r	Generator Polynomial	r	Generator Polynomial
45	[45 4 3 1 0]	52	[52 3 0]
46	[46 21 10 1 0]	53	[53 6 2 1 0]

Example of PN Sequence Generation

This example clarifies the operation of the PN Sequence Generator block by comparing the output sequence from the library block with that generated from primitive Simulink blocks.

This model is located at: *matlabroot*/help/toolbox/commblks/examples/doc_pnseq2. If you are reading this reference page in the MATLAB Help Browser, type doc_pnseq2 to open the model.
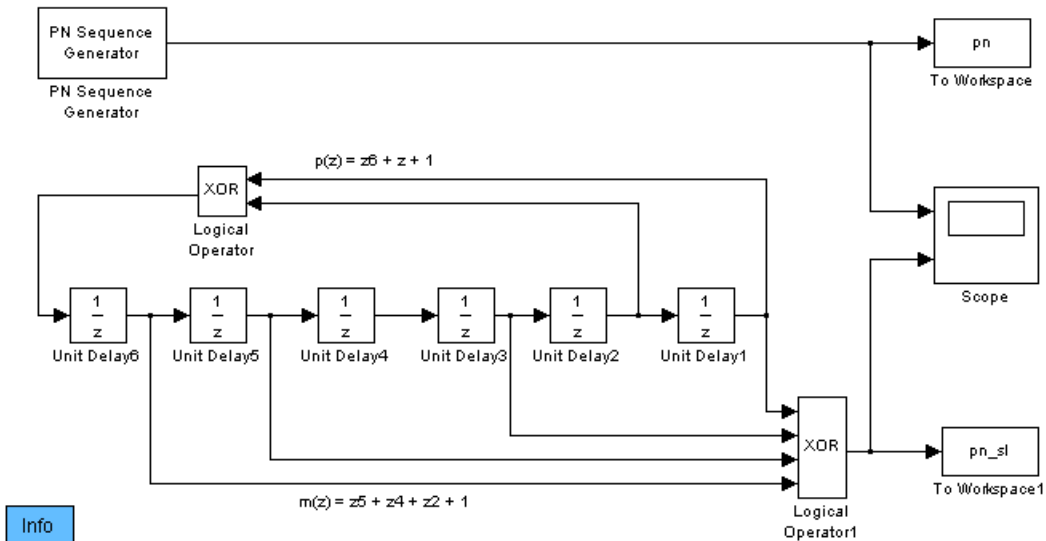
PN Sequence Generator

For the chosen generator polynomial, $p(z) = z^6 + z + 1$, the model generates a PN sequence of period 63, using both the library block and corresponding Simulink blocks. It shows how the two parameters, **Initial states** and **Output mask vector (or scalar shift value)**, are interpreted in the latter schematic.

You can experiment with different initial states, by changing the value of **Initial states** prior to running the simulation. For all values, the two generated sequences are the same.

Using the PN Sequence Generator block allows you to easily generate PN sequences of large periods.
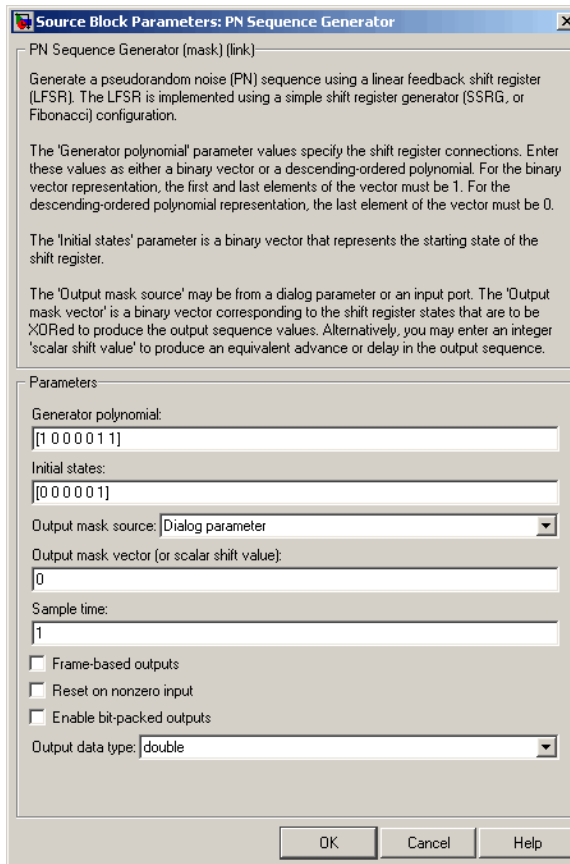
**Dialog
Box**

Opening this dialog box causes a running simulation to pause.
See "Changing Source Block Parameters" in the online Simulink
documentation for details.

Generator polynomial
Polynomial that determines the shift register's feedback
connections.

Initial states
Vector of initial states of the shift registers.

Output mask source

Specifies how output mask information is given to the block.

When set to `Dialog parameter`, the field **Output mask vector (or scalar shift value)** is enabled for user input. When set to `Input port`, a port appears on the block icon for output mask vector (or scalar shift value) input.

Output mask vector (or scalar shift value)

This field is available only when **Output mask source** is set to `Dialog parameter`.

Integer scalar or binary vector that determines the delay of the PN sequence from the initial time. If you specify the shift as a binary vector, the vector's length must equal the degree of the generator polynomial.

Sample time

Period of each element of the output signal.

Frame-based outputs

Determines whether the output is frame-based or sample-based.

Samples per frame

The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box.

Reset on nonzero input

When selected, you can specify an input signal that resets the internal shift registers to the original values of the **Initial states** parameter.

Enable bit-packed outputs

When selected, the field **Number of packed bits** and the option **Interpret bit-packed values as signed** is enabled.

Number of packed bits

Indicates how many bits to pack into each output data word (allowable range is 1 to 32).

Interpret bit-packed values as signed

Indicates whether packed bits are treated as signed or unsigned integer data values. When selected, a 1 in the most significant bit (sign bit) indicates a negative value.

Output data type

By default, this is set to `double`.

When **Enable bit-packed outputs** is not selected, the output data type can be specified as a `double`, `boolean`, or `Smallest unsigned integer`. When the parameter is set to `Smallest unsigned integer`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model. If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, the output data type is the ideal minimum one-bit size, i.e., `ufix(1)`. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., `uint8`).

When **Enable bit-packed outputs** is selected, the output data type can be specified as `double` or `Smallest integer`. When the parameter is set to `Smallest integer`, the output data type is selected based on **Interpret bit-packed values as signed**, **Number of packed bits**, and the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model. If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, the output data type is the ideal minimum n-bit size, i.e., `sfix(n)` or `ufix(n)`, based on **Interpret bit-packed values as signed**. For all other selections, it is a signed or unsigned integer with the smallest available word length large enough to fit `n` bits.

See Also   Kasami Sequence Generator, Scrambler

References   [1] Proakis, John G., *Digital Communications*, Third edition, New York, McGraw Hill, 1995.

[2] Lee, J. S., and L. E. Miller, *CDMA Systems Engineering Handbook*, Artech House, 1998.

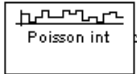[3] Golomb, S.W., *Shift Register Sequences*, Aegean Park Press, 1967.

Purpose   Generate Poisson-distributed random integers

Library   Random Data Sources sublibrary of Comm Sources

Description   The Poisson Integer Generator block generates random integers using
a Poisson distribution. The probability of generating a nonnegative
integer k is

$$\lambda^k \exp(-\lambda)/(k!)$$

where λ is a positive number known as the Poisson parameter.

You can use the Poisson Integer Generator to generate noise in a binary
transmission channel. In this case, the Poisson parameter **Lambda**
should be less than 1, usually much less.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row
or column vector, or a sample-based one-dimensional array. These
attributes are controlled by the **Frame-based outputs**, **Samples per
frame**, and **Interpret vector parameters as 1-D** parameters. See
"Signal Attribute Parameters for Random Sources" in *Communications
Blockset User's Guide* for more details.

The number of elements in the **Initial seed** parameter becomes the
number of columns in a frame-based output or the number of elements
in a sample-based vector output. Also, the shape (row or column) of
the **Initial seed** parameter becomes the shape of a sample-based
two-dimensional output signal.

Poisson Integer Generator
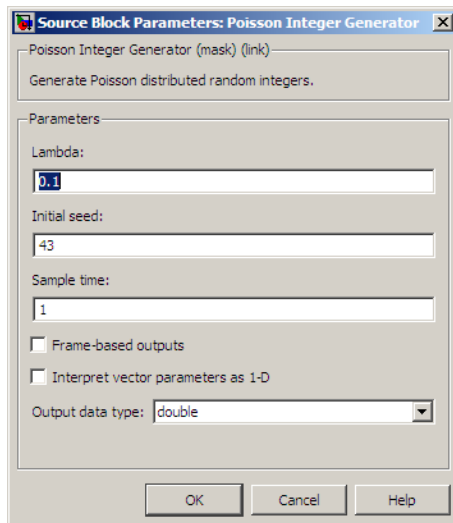
Source Block Parameters: Poisson Integer Generator

Poisson Integer Generator (mask) (link)

Generate Poisson distributed random integers.

Parameters

Lambda:

0.1

Initial seed:

43

Sample time:

1

☐ Frame-based outputs

☐ Interpret vector parameters as 1-D

Output data type: double

OK   Cancel   Help

Opening this dialog box causes a running simulation to pause.
See "Changing Source Block Parameters" in the online Simulink
documentation for details.

Lambda

The Poisson parameter λ. If it is a scalar, then every element in
the output vector shares the same Poisson parameter.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a
frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based.
This box is active only if **Interpret vector parameters as 1-D**
is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

The output type of the block can be specified as a `double`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. By default, the block sets this to `double`.

See Also   Random Integer Generator; `poissrnd` (Statistics Toolbox)

Puncture

Purpose   Output elements which correspond to 1s in binary Puncture vector

Library   Sequence Operations
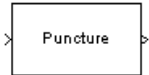
Description   The Puncture block creates an output vector by removing selected elements of the input vector and preserving others. The input can be a real or complex vector of length K. The block determines which elements to remove or preserve by using the binary **Puncture vector** parameter:

- If **Puncture vector**(k) = 0, then the kth element of the input vector does not become part of the output vector.

- If **Puncture vector**(k) = 1, then the kth element of the input vector is preserved in the output vector.

Here, k is between 1 and K. The preserved elements appear in the output vector in the same order in which they appear in the input vector.

The block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, `double`, and fixed-point. The data type of the output will be the same as that of the input signal.

Frame-Based Processing

If the input is frame-based, then both it and the **Puncture vector** parameter must be column vectors. The length of the **Puncture vector** parameter must divide K. The block repeats the puncturing pattern, if necessary, to cover all input elements. That is, in the bulleted items above you can replace **Puncture vector**(k) by **Puncture vector**(n), where

  n = mod(k,length(**Puncture vector**))

and mod is the modulus function (`mod` in MATLAB).

Dialog Box
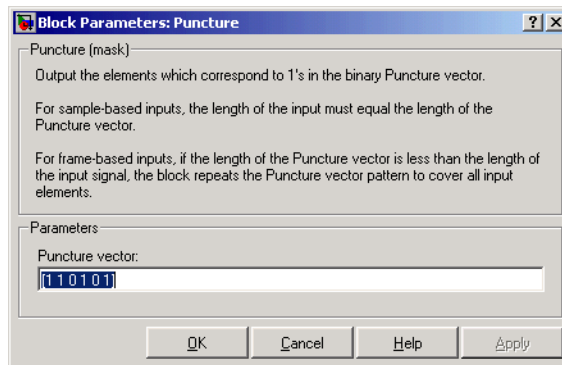
Puncture vector
> A binary vector whose pattern of 0s (1s) indicates which elements of the input the block should remove (preserve).
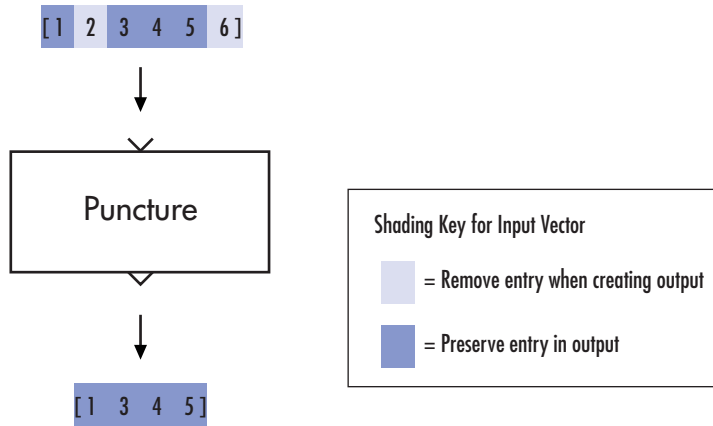
Examples   If the **Puncture vector** parameter is the six-element vector [1;0;1;1;1;0], then the block:

• Removes the second and sixth elements from the group of six input elements.

• Sends the first, third, fourth, and fifth elements to the output vector.

The diagram below depicts the block's operation on an input vector of [1:6], using this **Puncture vector** parameter.

Puncture

See Also   Insert Zero

Purpose   Demodulate QPSK-modulated data

Library   PM, in Digital Baseband sublibrary of Modulation

Description   The QPSK Demodulator Baseband block demodulates a signal that was
modulated using the quaternary phase shift keying method. The input
is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The input can be
either a scalar or a frame-based column vector. The block accepts
the data types double, single, and signed fixed-point (for both
hard-decision modes and soft decision modes).

Algorithm
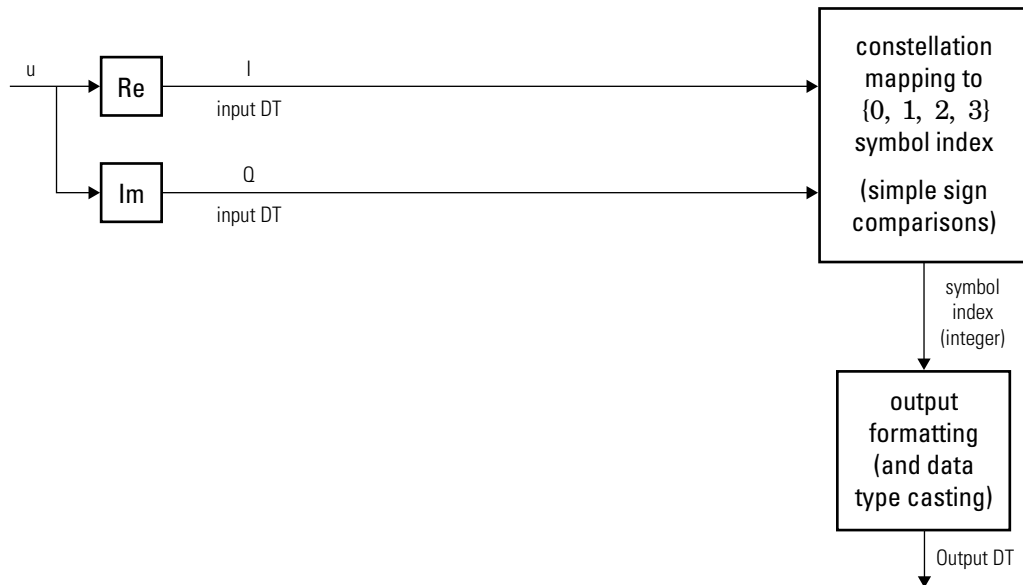
Hard-Decision QPSK Demodulator Signal Diagram for Trivial Phase

Offset (odd multiple of $\frac{\pi}{4}$)

QPSK Demodulator Baseband
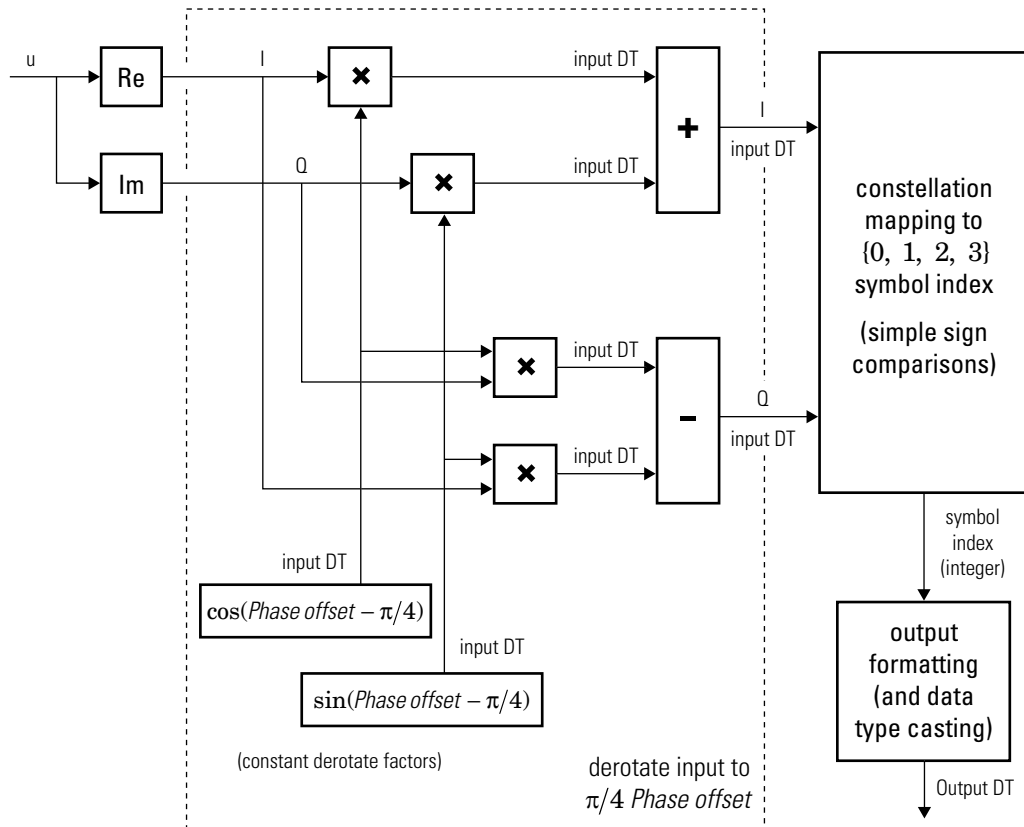
Hard-Decision QPSK Demodulator Floating-Point Signal Diagram for Nontrivial Phase Offset
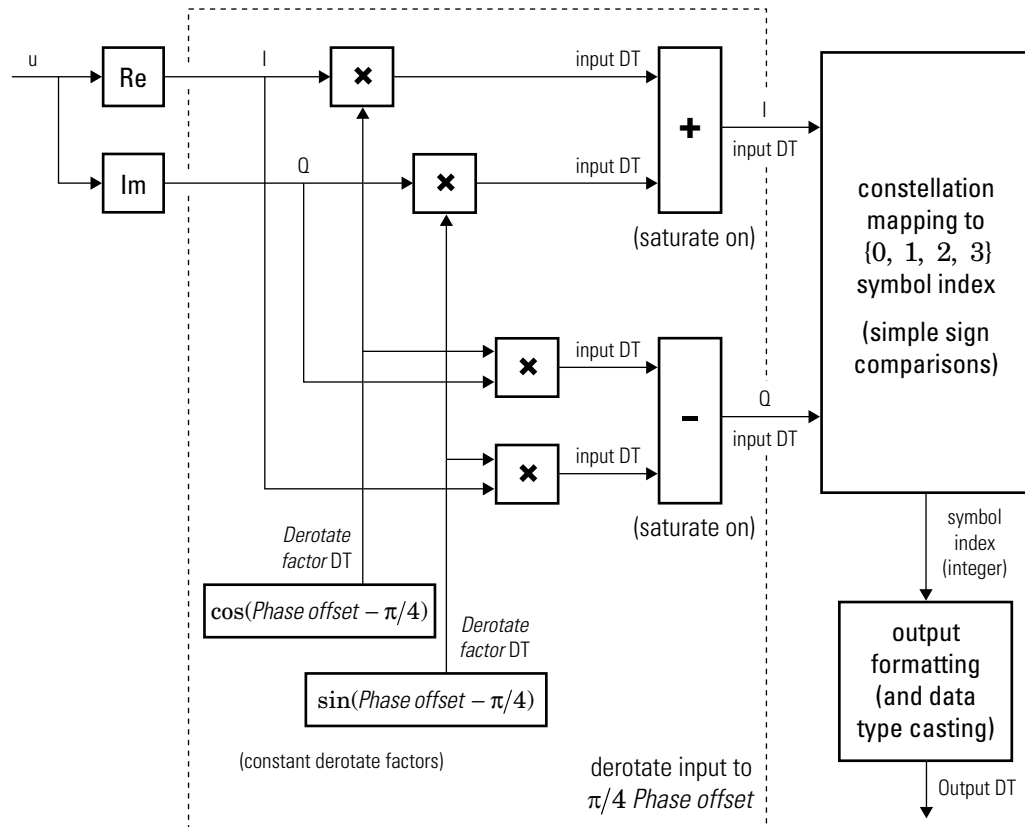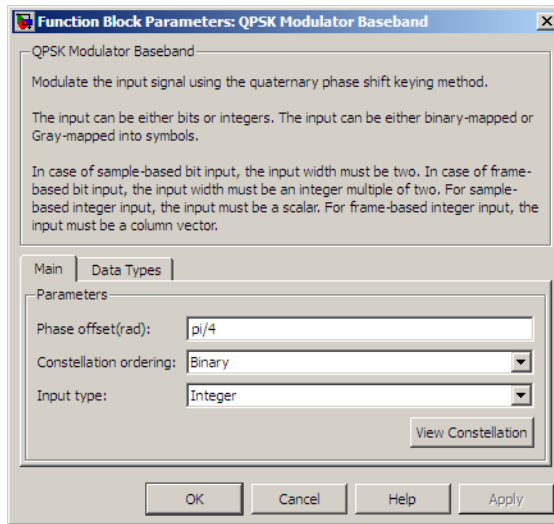
Hard-Decision QPSK Demodulator Fixed-Point Signal Diagram for Nontrivial Phase Offset

The exact LLR and approximate LLR cases (soft-decision) are described in "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide*.

QPSK Demodulator Baseband

Dialog Box

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Constellation ordering

Determines how the block maps each integer to a pair of output bits.

Output type

Determines whether the output consists of integers or bits.

If the **Output type** parameter is set to Integer and **Constellation ordering** is set to Binary, then the block maps the point

$$\exp(j\theta + j\pi m/2)$$

to m, where θ is the **Phase offset** parameter and m is 0, 1, 2, or 3.

The reference page for the QPSK Modulator Baseband block shows the signal constellations for the cases when **Constellation ordering** is set to either `Binary` or `Gray`.

If the **Output type** is set to `Bit`, then the output contains pairs of binary values if **Decision type** is set to `Hard decision`.

If the **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, then the output contains bitwise LLR or approximate LLR values, respectively.

Decision type

This field appears when `Bit` is selected in the drop-down list **Output type**.

Specifies the use of hard decision, LLR, or approximate LLR during demodulation. See "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide* for algorithm details.

Noise variance source

This field appears when `Approximate log-likelihood ratio` or `Log-likelihood ratio` is selected for **Decision type**.

When set to `Dialog`, the noise variance can be specified in the **Noise variance** field. When set to `Port`, a port appears on the block through which the noise variance can be input.

Noise variance

This parameter appears when the **Noise variance source** is set to `Dialog` and specifies the noise variance in the input signal. This parameter is tunable in normal mode, Accelerator mode and Rapid Accelerator mode.

If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameter without recompiling the model. This is useful for Monte
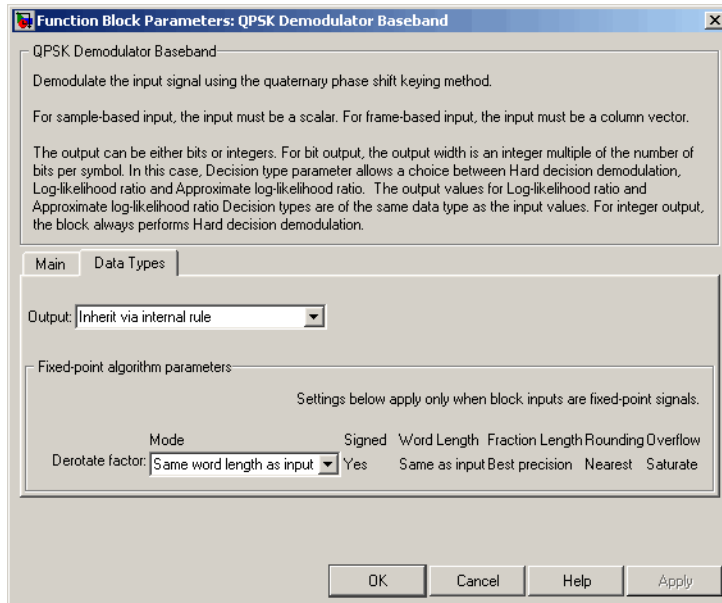
QPSK Demodulator Baseband

Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

The LLR algorithm involves computing exponentials of very large or very small numbers using finite precision arithmetic and would yield:

- `Inf` to `-Inf` if **Noise variance** is very high
- `NaN` if **Noise variance** and signal power are both very small

In such cases, use approximate LLR, as its algorithm does not involve computing exponentials.

Data Types Pane for Hard-Decision

Output

For bit outputs, when **Decision type** is set to `Hard decision`, the output data type can be set to `'Inherit via internal rule'`,

'Smallest unsigned integer', double, single, int8, uint8, int16, uint16, int32, uint32, or boolean.

For integer outputs, the output data type can be set to 'Inherit via internal rule', 'Smallest unsigned integer', double, single, int8, uint8, int16, uint16, int32, or uint32.

When this parameter is set to 'Inherit via internal rule' (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is a floating-point type (single or double). If the input data type is fixed-point, the output data type will work as if this parameter is set to 'Smallest unsigned integer'.

When this parameter is set to 'Smallest unsigned integer', the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model.

If ASIC/FPGA is selected in the **Hardware Implementation** pane, and **Output type** is Bit, the output data type is the ideal minimum one-bit size, i.e., ufix(1). For all other selections, it is an unsigned integer with the smallest available word length large enough to fit one bit, usually corresponding to the size of a char (e.g., uint8).

If ASIC/FPGA is selected in the **Hardware Implementation** pane, and **Output type** is Integer, the output data type is the ideal minimum two-bit size, i.e., ufix(2). For all other selections, it is an unsigned integer with the smallest available word length large enough to fit two bits, usually corresponding to the size of a char (e.g., uint8).

Derotate factor

This parameter only applies when the input is fixed-point and

Phase offset is not an odd multiple of $\frac{\pi}{4}$.

QPSK Demodulator Baseband

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input.
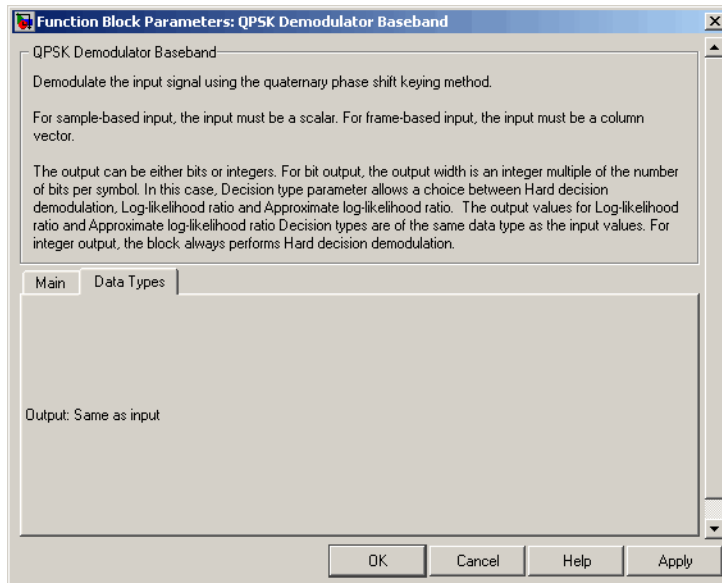
Data Types Pane for Soft-Decision

For bit outputs, when **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, the output data type is inherited from the input (e.g., if the input is of data type `double`, the output is also of data type `double`).

Pair Block   QPSK Modulator Baseband

See Also   M-PSK Demodulator Baseband, BPSK Demodulator Baseband, DQPSK Demodulator Baseband

Purpose   Modulate using quaternary phase shift keying method

Library   PM in Digital Baseband sublibrary of Modulation

Description   The QPSK Modulator Baseband block modulates using the quaternary phase shift keying method. The output is a baseband representation of the modulated signal.

Inputs and Constellation Types

If the **Input type** parameter is set to Integer, then valid input values are 0, 1, 2, and 3. If **Constellation ordering** is set to Binary, for input m the output symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** parameter (see the following figure for Gray constellation ordering). In this case, the input can be either a scalar or a frame-based column vector.

For integer inputs, the block can accept the data types int8, uint8, int16, uint16, int32, uint32, single, and double. For bit inputs, the block can accept int8, uint8, int16, uint16, int32, uint32, boolean, single, and double.

If the **Input type** parameter is set to Bit, then the input contains pairs of binary values. The input can be either a vector of length 2 or a frame-based column vector whose length is an even integer. If the **Phase offset** parameter is set to pi/4, then the block uses one of the signal constellations in the following figure, depending on whether the **Constellation ordering** parameter is set to Binary or Gray.

# QPSK Modulator Baseband

```
          Binary                              Gray

            |                                   |
            |                                   |
     •      |      •                     •      |      •
      01    |       00                    01    |       00
            |                                   |
    _ _ _ _ + _ _ _ _                 _ _ _ _ + _ _ _ _
            |                                   |
            |                                   |
     •      |      •                     •      |      •
      10    |       11                    11    |       10
            |                                   |
```
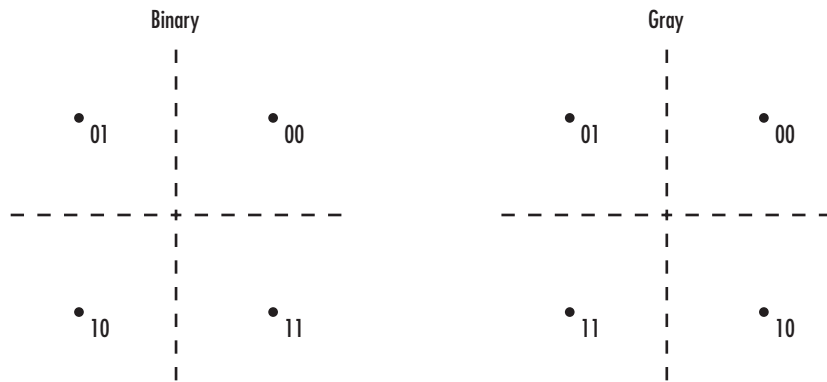
Constellation Visualization

The QPSK Modulator Baseband block provides the capability to visualize a signal constellation from the block mask. This Constellation Visualization feature allows you to visualize a signal constellation for specific block parameters. For more information, see the Constellation Visualization section of the *Communications Blockset User's Guide*.

Dialog Box
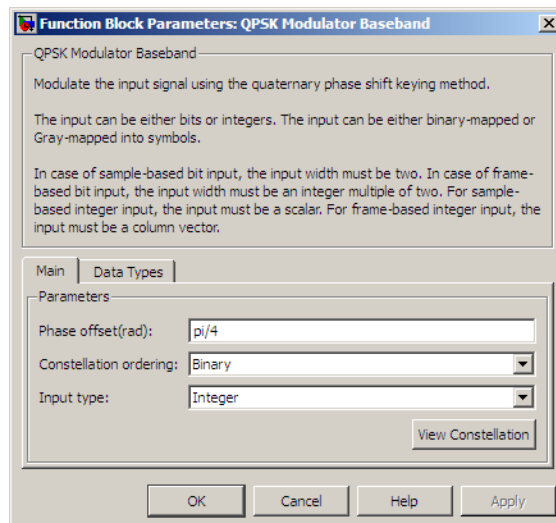
Phase offset (rad)

    The phase of the zeroth point of the signal constellation.

Constellation ordering

    Determines how the block maps each pair of input bits or input integers to constellation symbols.

Input type

    Indicates whether the input consists of integers or pairs of bits.

Output data type

    The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

    Setting this parameter to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this parameter to `Inherit via back propagation`, sets the output data type and scaling to match the following block.

Output word length

    Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Set output fraction length to

    Specify the scaling of the fixed-point output by either of the following methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

    This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter or when you select `User-defined` and the specified output data type is a fixed-point data type.

QPSK Modulator Baseband

User-defined data type

Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Output fraction length

For fixed-point output data types, specify the number of fractional bits or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   QPSK Demodulator Baseband

See Also   M-PSK Modulator Baseband, BPSK Modulator Baseband, DQPSK Modulator Baseband

Purpose   Decode quantization index according to codebook

Library   Source Coding
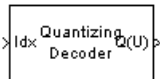
Description   The Quantizing Decoder block converts quantization indices to the corresponding codebook values. The **Quantization codebook** parameter, a vector of length N, prescribes the possible output values. If the input is an integer k between 0 and N-1, then the output is the (k+1)st element of **Quantization codebook**.

The input can be either a scalar, a vector, or a matrix. The input must be a discrete-time signal. This block processes each vector element independently.

Note The Quantizing Encoder block also uses a **Quantization codebook** parameter. The first output of that block corresponds to the input of Quantizing Decoder, while the second output of that block corresponds to the output of Quantizing Decoder.

Dialog Box
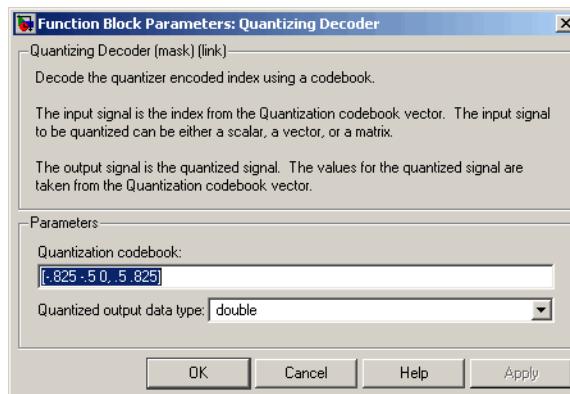
Quantizing Decoder

Quantization codebook
A real vector that prescribes the output value corresponding to each nonnegative integer of the input.
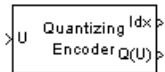
Pair Block   Quantizing Encoder

See Also   Scalar Quantizer (Obsolete) (Signal Processing Blockset documentation)

Purpose   Quantize signal using partition and codebook

Library   Source Coding

Description   The Quantizing Encoder block quantizes the input signal according to the **Partition** vector and encodes the input signal according to the **Codebook** vector. The input signal can be either a scalar, a vector, or a matrix. This block processes each vector element independently.

The first output is the quantization index. The second output is the quantized signal. The values for the quantized signal are taken from the **Codebook** vector.

The **Quantization partition** parameter, P, is a real vector of length n whose entries are in strictly ascending order. The quantization index (second output signal value) corresponding to an input value of x is

- 0 if x ≤ P(1)
- m if P(m) < x ≤ P(m+1)
- n if P(n) < x

The **Quantization codebook** parameter, whose length is n+1, prescribes a value for each partition in the quantization. The first element of **Quantization codebook** is the value for the interval between negative infinity and the first element of P. The second output signal from this block contains the quantization of the input signal based on the quantization indices and prescribed values.

You can use the function `lloyds` in Communications Toolbox with a representative sample of your data as training data, to obtain appropriate partition and codebook parameters.

Quantizing Encoder

Dialog Box
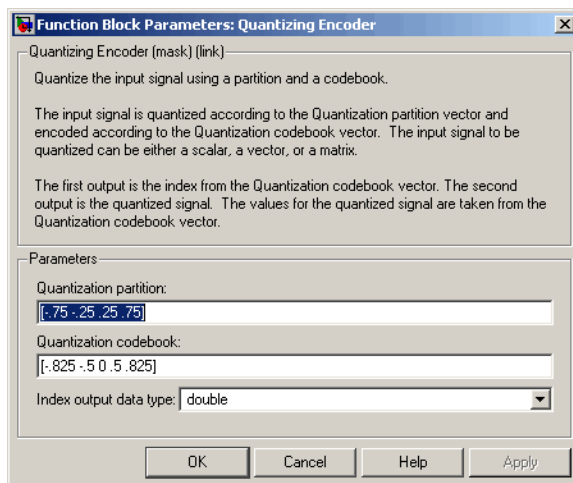
Quantization partition
The vector of endpoints of the partition intervals.

Quantization codebook
The vector of output values assigned to each partition.

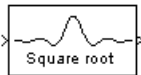**Pair Block**   Quantizing Decoder

See Also   Scalar Quantizer (Obsolete) (Signal Processing Blockset documentation), lloyds (Communications Toolbox documentation)

Purpose   Filter input signal, possibly downsampling, using raised cosine FIR filter

Library   Comm Filters

Description   The Raised Cosine Receive Filter block filters the input signal using a normal raised cosine FIR filter or a square root raised cosine FIR filter. It also downsamples the filtered signal if you set the **Output mode** parameter to Downsampling. The FIR Decimation block implements this functionality. The Raised Cosine Receive Filter block's icon shows the filter's impulse response.

Characteristics of the Filter

Characteristics of the raised cosine filter are the same as in the Raised Cosine Transmit Filter block, except that the length of the filter's input response has a slightly different expression: 2 * N * **Group delay** + 1, where N is the value of the **Input samples per symbol** parameter (not the **Upsampling factor** parameter, as in the case of the Raised Cosine Transmit Filter block).

If the **Filter gain** parameter is chosen to be User-specified, then the passband gain of the filter is:

- $20 \log_{10}(\text{Input samples per symbol}(N) \times \text{Linear amplitude filter gain})$ for a normal filter.

- $20 \log_{10}(\text{Input samples per symbol}(N) \times \text{Linear amplitude filter gain})$ for a square root filter.

Downsampling the Filtered Signal

To have the block downsample the filtered signal, set the **Output mode** parameter to Downsampling. By default, downsampling is on. If L is the **Downsampling factor** parameter value, then the block retains 1/L of the samples, choosing them as follows:

- If the **Sample offset** parameter is zero, then the block selects the samples of the filtered signal indexed by 1, L+1, 2*L+1, 3*L+1, etc.

- If the **Sample offset** parameter is a positive integer less than L, then the block initially discards that number of samples from the filtered signal and downsamples the remaining data as in the previous case.

To preserve the entire filtered signal and avoid downsampling, set **Output mode** to None. This setting is appropriate, for example, when the output from the filter block forms the input to a timing phase recovery block such as Squaring Timing Recovery. The timing phase recovery block performs the downsampling in that case.

The underlying implementation varies as follows:

- If **Sample offset** is set to zero offset or **Framing** is set to Maintain input frame size, the block uses FIR Decimation under the mask. This polyphase decimation improves the filter efficiency in terms of simulation speed and quality of code generation.

- If **Framing** is set to Maintain input frame rate and **Sample offset** is set to a nonzero number, the block uses Digital filter under the mask.

Input and Output Signals

The input signal must be a scalar value or a frame-based column vector. The block supports double, single, and fixed-point data types. You can select either Maintain input frame size or Maintain input frame rate for the **Framing** parameter.

If you set **Output mode** to None, then the input and output signals share the same sampling mode, sample time, and vector length.

If you set **Output mode** to Downsampling and **Downsampling factor** is L, then L and the input sampling mode determine characteristics of the output signal:

- If the input is sample-based, then the output is sample-based and the output sample time is 1/L times the input sample time.

- If the input is frame-based, then the output is a frame-based vector whose length is 1/L times the length of the input vector. The output frame period equals the input frame period.

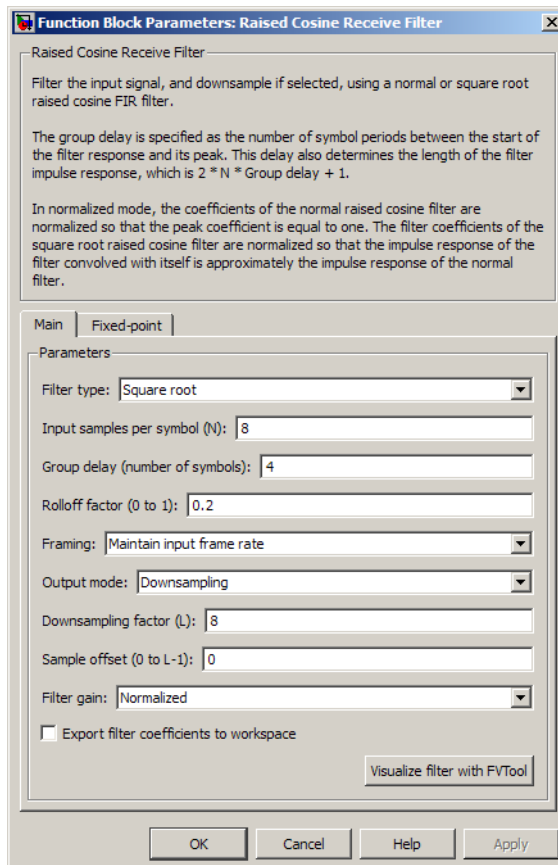### Exporting Filter Coefficients to the MATLAB Workspace

To examine or manipulate the coefficients of the filter that this block designs, select **Export filter coefficients to workspace**. Then set the **Coefficient variable name** parameter to the name of a variable that you want the block to create in the MATLAB workspace. Running the simulation causes the block to create the variable, overwriting any previous contents in case the variable already exists.

Latency

For information pertaining to the latency of the block, see details in FIR Decimation.

Raised Cosine Receive Filter

Function Block Parameters: Raised Cosine Receive Filter

Raised Cosine Receive Filter

Filter the input signal, and downsample if selected, using a normal or square root raised cosine FIR filter.

The group delay is specified as the number of symbol periods between the start of the filter response and its peak. This delay also determines the length of the filter impulse response, which is 2 * N * Group delay + 1.

In normalized mode, the coefficients of the normal raised cosine filter are normalized so that the peak coefficient is equal to one. The filter coefficients of the square root raised cosine filter are normalized so that the impulse response of the filter convolved with itself is approximately the impulse response of the normal filter.

Main | Fixed-point

Parameters

Filter type: `Square root`

Input samples per symbol (N): `8`

Group delay (number of symbols): `4`

Rolloff factor (0 to 1): `0.2`

Framing: `Maintain input frame rate`

Output mode: `Downsampling`

Downsampling factor (L): `8`

Sample offset (0 to L-1): `0`

Filter gain: `Normalized`

☐ Export filter coefficients to workspace

[Visualize filter with FVTool]

[OK] [Cancel] [Help] [Apply]

Filter type

The type of raised cosine filter: `Square root` or `Normal`.

Input samples per symbol

An integer greater than 1 representing the number of samples per symbol in the input signal.

Group delay

A positive integer that represents the number of symbol periods between the start of the filter response and its peak.

Rolloff factor

The rolloff factor for the filter, a real number between 0 and 1.

Framing

Selects the framing method. The framing method choices are: `Maintain input frame size` or `Maintain input frame rate`.

Output mode

Determines whether or not the block downsamples the signal after filtering. Choices are `Downsampling` and `None`.

Downsampling factor

The factor by which the block downsamples the signal after filtering. This field appears only if **Output mode** is set to `Downsampling`.

Sample offset

The number of filtered samples the block discards before downsampling. This field appears only if **Output mode** is set to `Downsampling`.

Filter gain

Determines how the block scales the filter coefficients. Choices are `Normalized` and `User-specified`.

Linear amplitude filter gain

A positive scalar used to scale the filter coefficients. This field appears only if **Filter gain** is set to `User-specified`.

Export filter coefficients to workspace

If you check this box, then the block creates a variable in the MATLAB workspace that contains the filter coefficients.

Coefficient variable name

The name of the variable to create in the MATLAB workspace. This field appears only if **Export filter coefficients to workspace** is selected.

Visualize filter with FVTool

If you click this button, then MATLAB launches the Filter Visualization Tool, `fvtool`, to analyze the raised cosine filter

Raised Cosine Receive Filter

whenever you apply any changes to the block's parameters. If you launch `fvtool` for the filter, and subsequently change parameters in the mask, `fvtool` will not update. You will need to launch a new `fvtool` in order to see the new filter characteristics. Also note that if you have launched `fvtool`, then it will remain open even after the model is closed.

Rounding mode

Select the rounding mode for fixed-point operations. The block uses the **Rounding mode** when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. The filter coefficients

do not obey this parameter; they always round to `Nearest`. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator).

See the Coefficients section of the FIR Decimation help page and "Filter Structure Diagrams" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the coefficient data types in this block:

See the Coefficients subsection of the Digital Filter help page for descriptions of parameter settings.

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you are able to enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the coefficients, in bits. If applicable, you are able to enter separate fraction lengths for the numerator and denominator coefficients.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the coefficients. If applicable, you are able to enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.

- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output
Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator
Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool

Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool.

Pair Block   Raised Cosine Transmit Filter

See Also   Gaussian Filter, `rcosine`, `rcosflt`

Raised Cosine Transmit Filter

Purpose   Upsample and filter input signal using raised cosine FIR filter

Library   Comm Filters
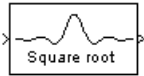
Description   The Raised Cosine Transmit Filter block upsamples and filters the input signal using a normal raised cosine FIR filter or a square root raised cosine FIR filter. The block's icon shows the filter's impulse response.

Characteristics of the Filter

The **Filter type** parameter determines which type of filter the block uses; choices are Normal and Square root.

The impulse response of a normal raised cosine filter with rolloff factor R and symbol period T is

$$h(t) = \frac{\sin(\pi t / T)}{(\pi t / T)} \cdot \frac{\cos(\pi R t / T)}{(1 - 4R^2 t^2 / T^2)}$$

The impulse response of a square root raised cosine filter with rolloff factor R is

$$h(t) = 4R \frac{\cos\big((1 + R)\pi t / T\big) + \dfrac{\sin\big((1 - R)\pi t / T\big)}{(4R t / T)}}{\pi\sqrt{T}\left(1 - (4R t / T)^2\right)}$$

The impulse response of a square root raised cosine filter convolved with itself is approximately equal to the impulse response of a normal raised cosine filter.

The **Group delay** parameter is the number of symbol periods between the start of the filter's response and the peak of the filter's response. The group delay and the upsampling factor, N, determine the length of the filter's impulse response, which is 2 * N * **Group delay** + 1.

The **Rolloff factor** parameter is the filter's rolloff factor. It must be a real number between 0 and 1. The rolloff factor determines the excess

bandwidth of the filter. For example, a rolloff factor of .5 means that the bandwidth of the filter is 1.5 times the input sampling frequency.

The **Filter gain** parameter indicates how the block normalizes the filter coefficients. If you choose `Normalized`, then the block uses an automatic scaling:

- If **Filter type** is `Normal`, then the block normalizes the filter coefficients so that the peak coefficient equals 1.

- If **Filter type** is `Square root`, then the block normalizes the filter coefficients so that the convolution of the filter with itself produces a normal raised cosine filter whose peak coefficient equals 1.

As a result, the output is scaled by $\sqrt{N}$. If the output of this block feeds the input to the AWGN Channel block, specify the AWGN signal power parameter to be 1/N.

If the **Filter gain** parameter is chosen to be `User-specified`, then the passband gain of the filter is:

- $20 \log_{10}(\text{Upsampling factor}(N) \times \text{Linear amplitude filter gain})$ for a normal filter.

- $20 \log_{10}(\sqrt{\text{Upsampling factor}(N)} \times \text{Linear amplitude filter gain})$ for a square root filter.

Input and Output Signals

The input signal must be a scalar or a frame-based column vector. The block supports `double`, `single`, and fixed-point data types. You can select either `Maintain input frame size` or `Maintain input frame rate` for the **Framing** parameter.

The Framing method and the value of the **Upsampling factor (N)** parameter determine the characteristics of the output signal:

- If the input is a sample-based scalar, then the output is a sample-based scalar and the output sample time is N times the input sample time.

- If the input is frame-based, then the output is a frame-based vector whose length is N times the length of the input vector. The output frame period equals the input frame period.

Exporting Filter Coefficients to the MATLAB Workspace

To examine or manipulate the coefficients of the filter that this block designs, select **Export filter coefficients to workspace**. Then set the **Coefficient variable name** parameter to the name of a variable that you want the block to create in the MATLAB workspace. Running the simulation causes the block to create the variable, overwriting any previous contents in case the variable already exists.

**Dialog
Box**

Function Block Parameters: Raised Cosine Transmit Filter

Raised Cosine Transmit Filter

Upsample and filter the input signal using a normal or square root raised cosine FIR filter.

The group delay is specified as the number of symbol periods between the start of the filter response and its peak. This delay also determines the length of the filter impulse response, which is 2 * N * Group delay + 1.

In normalized mode, the coefficients of the normal raised cosine filter are normalized so that the peak coefficient is equal to one. The filter coefficients of the square root raised cosine filter are normalized so that the impulse response of the filter convolved with itself is approximately the impulse response of the normal filter.

| Main | Fixed-point |

Parameters

Filter type: Square root

Group delay (number of symbols): 4

Rolloff factor (0 to 1): 0.2

Framing: Maintain input frame rate

Upsampling factor (N): 8

Filter gain: Normalized

☐ Export filter coefficients to workspace

Visualize filter with FVTool

OK   Cancel   Help   Apply

Filter type
> The type of raised cosine filter: Square root or Normal.

Group delay
> A positive integer that represents the number of symbol periods between the start of the filter response and its peak.

Rolloff factor
> The rolloff factor for the filter, a real number between 0 and 1.

Framing
> Selects the framing method. The framing method choices are: Maintain input frame size or Maintain input frame rate.

Raised Cosine Transmit Filter

Upsampling factor

An integer greater than 1 representing the number of samples per symbol in the filtered output signal.

Filter gain

Determines how the block scales the filter coefficients. Choices are `Normalized` and `User-specified`.

Linear amplitude filter gain

A positive scalar used to scale the filter coefficients. This field appears only if **Filter gain** is set to `User-specified`.

Export filter coefficients to workspace

If you check this box, then the block creates a variable in the MATLAB workspace that contains the filter coefficients.

Coefficient variable name

The name of the variable to create in the MATLAB workspace. This field appears only if **Export filter coefficients to workspace** is selected.

Visualize filter with FVTool

If you click this button, then MATLAB launches the Filter Visualization Tool, `fvtool`, to analyze the raised cosine filter whenever you apply any changes to the block's parameters. If you launch `fvtool` for the filter, and subsequently change parameters in the mask, `fvtool` will not update. You will need to launch a new `fvtool` in order to see the new filter characteristics. Also note that if you have launched `fvtool`, then it will remain open even after the model is closed.

Rounding mode

> Select the rounding mode for fixed-point operations. The block uses the **Rounding mode** when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. The filter coefficients do not obey this parameter; they always round to Nearest. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow mode

> Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator). See "Filter Structure Diagrams" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you are able to enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the coefficients, in bits. If applicable, you are able to enter separate fraction lengths for the numerator and denominator coefficients.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the coefficients. If applicable, you are able to enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.

- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" in *Signal Processing*

Blockset Reference Guide for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool.

Pair Block   Raised Cosine Receive Filter

See Also   Gaussian Filter, `rcosine`, `rcosflt`

Purpose   Restore ordering of input symbols using random permutation

Library   Block sublibrary of Interleaving

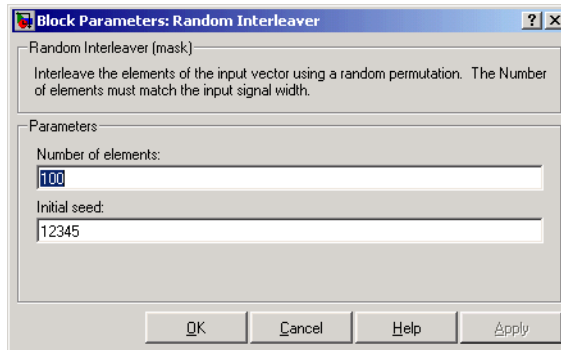**Description**   The Random Deinterleaver block rearranges the elements of its input vector using a random permutation. The **Initial seed** parameter initializes the random number generator that the block uses to determine the permutation. If this block and theRandom Interleaver block have the same value for **Initial seed**, then the two blocks are inverses of each other.

The **Number of elements** parameter indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

The block can accept the data types int8, uint8, int16, uint16, int32, uint32, boolean, single, double, and fixed-point. The data type of this output will be the same as that of the input signal.

Dialog Box
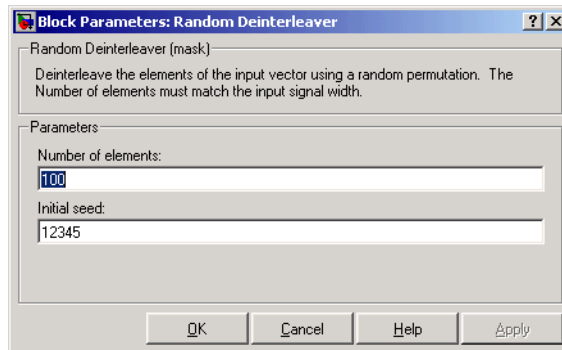
Number of elements
    The number of elements in the input vector.

Initial seed
    The initial seed value for the random number generator.

Pair Block   Random Interleaver

Random Deinterleaver

See Also   General Block Deinterleaver

Purpose   Generate integers randomly distributed in range [0, M-1]

Library   Random Data Sources sublibrary of Comm Sources
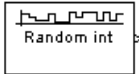
Description   The Random Integer Generator block generates uniformly distributed
random integers in the range [0, *M*-1], where *M* is the **M-ary number**
defined in the dialog box.

The **M-ary number** can be either a scalar or a vector. If it is a scalar,
then all output random variables are independent and identically
distributed (i.i.d.). If the **M-ary number** is a vector, then its length
must equal the length of the **Initial seed**; in this case each output
has its own output range.

If the **Initial seed** parameter is a constant, then the resulting noise is
repeatable.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row
or column vector, or a sample-based one-dimensional array. These
attributes are controlled by the **Frame-based outputs**, **Samples per
frame**, and **Interpret vector parameters as 1-D** parameters. See
"Signal Attribute Parameters for Random Sources" in *Communications
BlocksetUser's Guide* for more details.

The number of elements in the **Initial seed** parameter becomes the
number of columns in a frame-based output or the number of elements
in a sample-based vector output. Also, the shape (row or column) of
the **Initial seed** parameter becomes the shape of a sample-based
two-dimensional output signal.

Random Integer Generator

Dialog Box
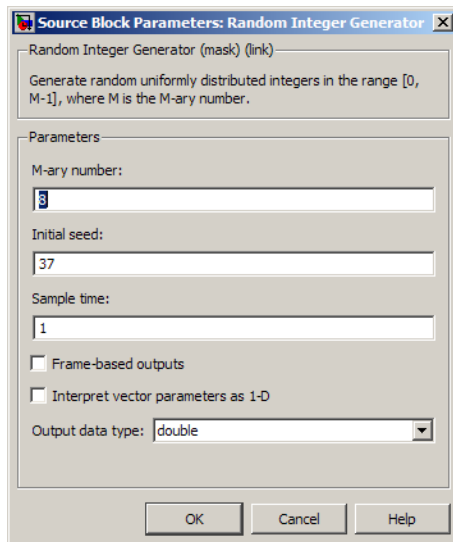
Opening this dialog box causes a running simulation to pause.
See "Changing Source Block Parameters" in the online Simulink
documentation for details.

M-ary number
> The positive integer, or vector of positive integers, that indicates
> the range of output values.

Initial seed
> The initial seed value for the random number generator. The
> vector length of the seed determines the length of the output
> vector.

Sample time
> The period of each sample-based vector or each row of a
> frame-based matrix.

Frame-based outputs
> Determines whether the output is frame-based or sample-based.
> This box is active only if **Interpret vector parameters as 1-D**
> is unchecked.

Samples per frame

> The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

> If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

> The output type of the block can be specified as a `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, or `double`. By default, the block sets this to `double`. Single outputs may lead to different results when compared with double outputs for the same set of parameters. For Boolean typed outputs, the **M-ary number** must be `2`.

See Also   `randint` (Communications Toolbox)

Random Interleaver

Purpose   Reorder input symbols using random permutation

Library   Block sublibrary of Interleaving

Description   The Random Interleaver block rearranges the elements of its input
                     vector using a random permutation. The **Number of elements**
                     parameter indicates how many numbers are in the input vector. If the
                     input is frame-based, then it must be a column vector.

Random
Interleaver

The block can accept the data types int8, uint8, int16, uint16, int32,
uint32, boolean, single, double, and fixed-point. The data type of this
output will be the same as that of the input signal.

The **Initial seed** parameter initializes the random number generator
that the block uses to determine the permutation. The block is
predictable for a given seed, but different seeds produce different
permutations.

**Dialog
Box**

Number of elements
The number of elements in the input vector.

Initial seed
The initial seed value for the random number generator.

Pair Block   Random Deinterleaver

See Also General Block Interleaver

Rayleigh Noise Generator

Purpose   Generate Rayleigh distributed noise

Library   Noise Generators sublibrary of Comm Sources

Description   The Rayleigh Noise Generator block generates Rayleigh distributed
noise. The Rayleigh probability density function is given by

$$f(x) = \begin{cases} \dfrac{x}{\sigma^2}\exp\left(-\dfrac{x^2}{2\sigma^2}\right) & x \ge 0 \\ 0 & x < 0 \end{cases}$$

where σ^2 is known as the *fading envelope* of the Rayleigh distribution.

The block requires you to specify the **Initial seed** for the random
number generator. If it is a constant, then the resulting noise is
repeatable. The **sigma** parameter can be either a vector of the same
length as the **Initial seed**, or a scalar. When **sigma** is a scalar, every
element of the output signal shares that same value.

Initial Seed

The **Initial seed** parameter initializes the random number generator
that the Rayleigh Noise Generator block uses to add noise to the input
signal. For best results, the **Initial seed** should be a prime number
greater than 30. Also, if there are other blocks in a model that have
an **Initial seed** parameter, you should choose different initial seeds
for all such blocks.

You can choose seeds for the Rayleigh Noise Generator block using the
Communications Blockset randseed function. At the MATLAB prompt,
enter

```
randseed
```

This returns a random prime number greater than 30. Entering
randseed again produces a different prime number. If you supply an
integer argument, randseed always returns the same prime for that
integer. For example, randseed(5) always returns the same answer.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See "Signal Attribute Parameters for Random Sources" in *Communications Blockset User's Guide* for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Rayleigh Noise Generator

Sigma

Specify σ as defined in the Rayleigh probability density function.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

The output can be set to `double` or `single` data types.

See Also   Multipath Rayleigh Fading Channel; `raylrnd` (Statistics Toolbox)

References   [1] Proakis, John G., *Digital Communications*, Third edition, New York, McGraw Hill, 1995.

Purpose   Apply receiver thermal noise to complex baseband signal

Library   RF Impairments
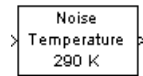
Description   The Receiver Thermal Noise block simulates the effects of thermal noise on a complex, baseband signal. You can specify the amount of thermal noise in three ways, according to which **Specification method** you select:

- Noise temperature specifies the noise in degrees kelvin.

- Noise factor specifies the noise as 1+(Noise temperature / 290).

- Noise figure specifies the noise as 10*log10(1+(Noise temperature / 290)). This is the decibel equivalent of Noise factor.

The following scatter plot shows the effect of the Receiver Thermal Noise block, with **Specification method** set to Noise figure and **Noise figure (dB)** set to 3.01, on a signal modulated by 16-QAM.
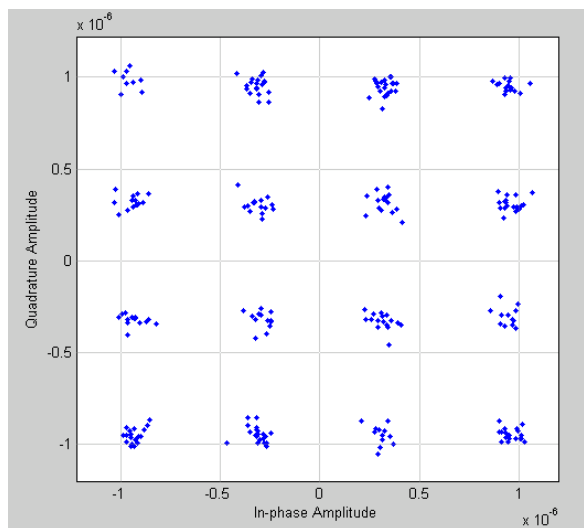
Receiver Thermal Noise

This plot is generated by the model described in "Scatter Plot Examples" with the following parameter settings:

- Rectangular QAM Modulator Baseband
    - **Normalization method** set to `Average Power`
    - **Average power (watts)** set to `1e-12`
- Receiver Thermal Noise
    - **Specification method** set to `Noise figure`
    - **Noise figure (dB)** set to `3.01`

Dialog Box
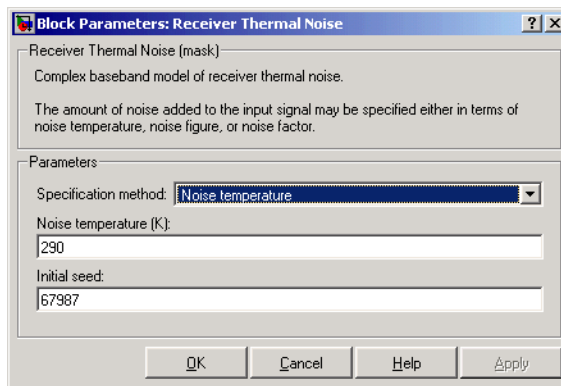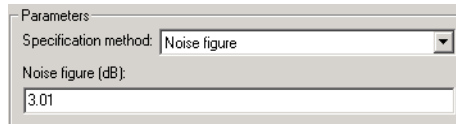
Specification method
>   The method by which you specify the amount of noise. The choices are `Noise temperature`, `Noise figure`, and `Noise factor`.
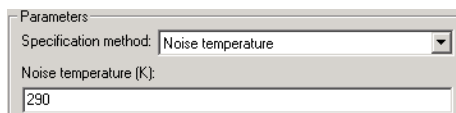
Noise temperature (K)
>   Scalar specifying the amount of noise in degrees kelvin.
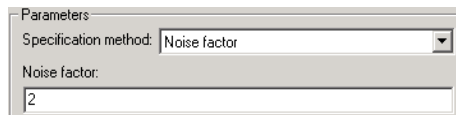
Noise figure

Scalar specifying the amount of noise in decibels relative to a noise temperature of 290 degrees kelvin. A **Noise figure** setting of 0 dB indicates a noiseless system.

Noise factor

Scalar specifying the amount of noise relative to a noise temperature of 290 degrees kelvin.

Initial seed

The initial seed value for the random number generator that generates the noise.

See Also   Free Space Path Loss

Rectangular QAM Demodulator Baseband

Purpose   Demodulate rectangular-QAM-modulated data

Library   AM, in Digital Baseband sublibrary of Modulation

Description   The Rectangular QAM Demodulator Baseband block demodulates a
signal that was modulated using quadrature amplitude modulation
with a constellation on a rectangular lattice.

Note All values of power assume a nominal impedance of 1 ohm.

The signal constellation has M points, where M is the **M-ary number**
parameter. M must have the form 2^K for some positive integer K.
The block scales the signal constellation based on how you set the
Normalization method parameter. For details, see the reference page
for the Rectangular QAM Modulator Baseband block.

The input can be either a scalar or a frame-based column vector of data
types `single` or `double` in all modes, as well as signed fixed-point
(binary-point only scaling) in hard-decision modes.

**Hard
Decision
Algorithm**

The demodulator algorithm maps received input signal constellation
values to M-ary integer I and Q symbol indices between 0 and $\sqrt{M}-1$
and then maps these demodulated symbol indices to formatted output
values.

The integer symbol index computation is performed by first derotating
and scaling the complex input signal constellation (possibly with noise)
by a derotate factor and denormalization factor, respectively. These
factors are derived from the **Phase offset**, **Normalization method**,
and related parameters. These derotated and denormalized values
are added to $\sqrt{M}-1$ to translate them into an approximate range
between 0 and $2\times(\sqrt{M}-1)$ (plus noise). The resulting values are then
rescaled via a divide-by-two (or, equivalently, a right-shift by one bit
for fixed-point operation) to obtain a range approximately between 0

and $\sqrt{M}-1$ (plus noise) for I and Q. The noisy index values are rounded to the nearest integer and clipped, via saturation, and mapped to integer symbol values in the range [0 M-1]. Finally, based on other block parameters, the integer index is mapped to a symbol value that is formatted and cast to the selected **Output data type**.

The following figures contains signal flow diagrams for floating-point and fixed-point algorithm operation. The floating-point diagrams apply when the input signal data type is double or single. The fixed-point diagrams apply when the input signal is a signed fixed-point data type. Note that the diagram is simplified when **Phase offset** is a multiple of $\dfrac{\pi}{2}$, and/or the derived denormalization factor is 1.

Rectangular QAM Demodulator Baseband
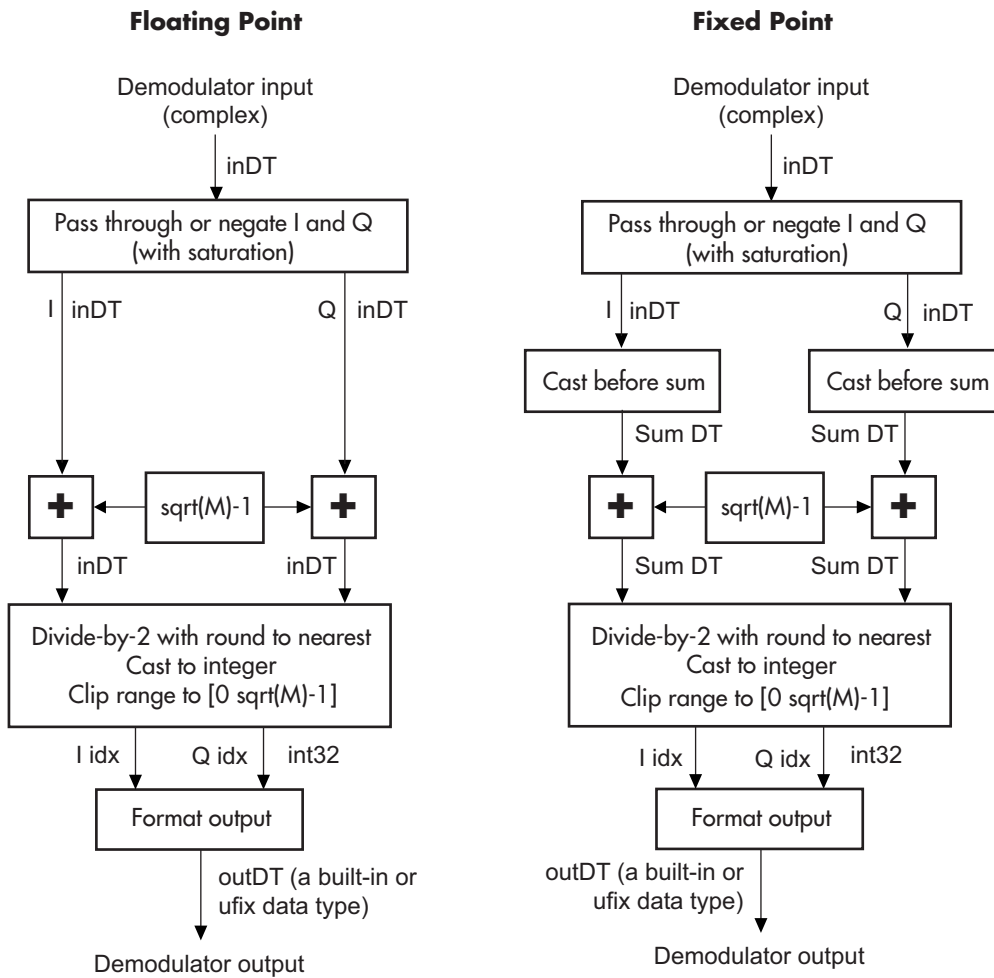
Signal-Flow Diagrams with Trivial Phase Offset and Denormalization Factor Equal to 1

Floating Point

Demodulator input
(complex)

Derotate factors
(sin and cos)

inDT   inDT   inDT

Complex floating-point multiply

I | inDT   Q | inDT

✖ ← Denorm factor → ✖

inDT   inDT

➕ ← sqrt(M)-1 → ➕

inDT   inDT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 sqrt(M)-1]

I idx   Q idx   int32

Format output

outDT (a built-in or
ufix data type)

Demodulator output

Fixed Point

Demodulator input
(complex)

Derotate factors
(sin and cos)

Derotate
factor DT   inDT

Complex fixed-point multiply

I | inDT   Q | inDT

Denorm   Denorm
DT   DT

✖ ← Denorm factor → ✖

Product output DT

Cast before sum   Cast before sum

Sum DT   Sum DT

➕ ← sqrt(M)-1 → ➕

Sum DT   Sum DT

Divide-by-2 with round to nearest
Cast to integer
Clip range to [0 sqrt(M)-1]

I idx   Q idx   int32

Format output

outDT (a built-in or
ufix data type)

Demodulator output

Signal-Flow Diagrams with Nontrivial Phase Offset and Nonunity Denormalization Factor

Rectangular QAM Demodulator Baseband

Function Block Parameters: Rectangular QAM Demodulator Baseband

Rectangular QAM Demodulator Baseband

Demodulate the input signal using the rectangular quadrature amplitude modulation method.

The M-ary number must be an integer power of two.

For sample-based input, the input must be a scalar. For frame-based input, the input must be a column vector.

The output can be either bits or integers. For bit output, the output width is an integer multiple of the number of bits per symbol. In this case, Decision type parameter allows to choose between Hard decision demodulation, Log-likelihood ratio and Approximate log-likelihood ratio. The output values for Log-likelihood ratio and Approximate log-likelihood ratio Decision types are of the same data type as the input values. For integer output, the block always performs Hard decision demodulation.

| Main | Data Types |

Parameters

M-ary number: 16

Normalization method: Min. distance between symbols

Minimum distance: 2

Phase offset (rad): 0

Constellation ordering: Binary

Output type: Integer

[OK] [Cancel] [Help] [Apply]

M-ary number
> The number of points in the signal constellation. It must have the form 2^K for some positive integer K.

Normalization method
> Determines how the block scales the signal constellation. Choices are Min. distance between symbols, Average Power, and Peak Power.

Minimum distance
> This parameter appears when **Normalization method** is set to Min. distance between symbols.
>
> The distance between two nearest constellation points.

Average power, referenced to 1 ohm (watts)
>The average power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to Average Power.

Peak power, referenced to 1 ohm (watts)
>The maximum power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to Peak Power.

Phase offset (rad)
>The rotation of the signal constellation, in radians.

Constellation ordering
>Determines how the block assigns binary words to points of the signal constellation. More details are on the reference page for the Rectangular QAM Modulator Baseband block.
>
>Selecting User-defined displays the field **Constellation mapping**, allowing for user-specified mapping.

Constellation mapping
>This parameter appears when User-defined is selected in the pull-down list **Constellation ordering**.
>
>This is a row or column vector of size M and must have unique integer values in the range [0, M-1]. The values must be of data type double.
>
>The first element of this vector corresponds to the top-leftmost point of the constellation, with subsequent elements running down column-wise, from left to right. The last element corresponds to the bottom-rightmost point.

Output type
>Determines whether the block produces integers or binary representations of integers.
>
>If set to Integer, the block produces integers.

Rectangular QAM Demodulator Baseband

If set to `Bit`, the block produces a group of K bits, called a *binary word*, for each symbol, when **Decision type** is set to `Hard decision`. If **Decision type** is set to `Log-likelihood ratio` or `Approximate log-likelihood ratio`, the block outputs bitwise LLR and approximate LLR, respectively.

Decision type

This parameter appears when `Bit` is selected in the pull-down list **Output type**.

Specifies the use of hard decision, LLR, or approximate LLR during demodulation. See "Exact LLR Algorithm" and "Approximate LLR Algorithm" in the *Communications Toolbox User's Guide* for algorithm details.

Noise variance source

This parameter appears when `Approximate log-likelihood ratio` or `Log-likelihood ratio` is selected for **Decision type**.

When set to `Dialog`, the noise variance can be specified in the **Noise variance** field. When set to `Port`, a port appears on the block through which the noise variance can be input.

Noise variance

This parameter appears when the **Noise variance source** is set to `Dialog` and specifies the noise variance in the input signal. This parameter is tunable in normal mode, Accelerator mode and Rapid Accelerator mode.

If you use the Real-Time Workshop® rapid simulation (RSIM) target to build an RSIM executable, then you can tune the parameter without recompiling the model. This is useful for Monte Carlo simulations in which you run the simulation multiple times (perhaps on multiple computers) with different amounts of noise.

The LLR algorithm involves computing exponentials of very large or very small numbers using finite precision arithmetic and would yield:

- `Inf` to `-Inf` if **Noise variance** is very high

- `NaN` if **Noise variance** and signal power are both very small

In such cases, use approximate LLR, as its algorithm does not involve computing exponentials.

Output

When the parameter is set to `'Inherit via internal rule'` (default setting), the block will inherit the output data type from the input port. The output data type will be the same as the input data type if the input is of type `single` or `double`. Otherwise, the output data type will be as if this parameter is set to `'Smallest unsigned integer'`.

Rectangular QAM Demodulator Baseband

When the parameter is set to `'Smallest unsigned integer'`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model. If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, the output data type is the ideal minimum size, i.e., `ufix(1)` for bit outputs, and `ufix(ceil(log2(M)))` for integer outputs. For all other selections, it is an unsigned integer with the smallest available word length large enough to fit the ideal minimum size, usually corresponding to the size of a char (e.g., `uint8`).

For integer outputs, this parameter can be set to `Smallest unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`. For bit outputs, the options are `Smallest unsigned integer`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, or `double`.

Derotate factor

This parameter only applies when the input is fixed-point and

Phase offset is not a multiple of $\frac{\pi}{2}$.

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input.

Denormalization factor

This parameter only applies when the input is fixed-point and the derived denormalization factor is nonunity (not equal to 1). This scaling factor is derived from **Normalization method** and other parameter values in the block dialog.

This can be set to `Same word length as input` or `Specify word length`, in which case a field is enabled for user input. A best-precision fraction length is always used.

Product output

This parameter only applies when the input is a fixed-point signal and there is a nonunity (not equal to 1) denormalized factor. It

can be set to `Inherit via internal rule` or `Specify word length`, which enables a field for user input.

Setting to `Inherit via internal rule` computes the full-precision product word length and fraction length. Inherit Rule for Product Data Types in *Signal Processing Blockset User's Guide* describes the full-precision **Product output** internal rule.

Setting to `Specify word length` allows you to define the word length. The block computes a best-precision fraction length based on the word length specified and the pre-computed worst-case (min/max) real world value **Product output** result. The worst-case **Product output** result is precomputed by multiplying the denormalized factor with the worst-case (min/max) input signal range, purely based on the input signal data type.

The block uses the **Rounding mode** when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Sum

This parameter only applies when the input is a fixed-point signal. It can be set to `Inherit via internal rule`, `Same as product output`, or `Specify word length`, in which case a field is enabled for user input

Setting to `Inherit via internal rule` computes the full-precision sum word length and fraction length, based on the two inputs to the Sum in the fixed-point Hard Decision Algorithm signal flow diagram. The rule is the same as the fixed-point inherit rule of the internal **Accumulator data type** parameter in the SimulinkSimulinkSimulink Sum block.

Rectangular QAM Demodulator Baseband

Setting to `Specify word length` allows you to define the word length. A best precision fraction length is computed based on the word length specified in the pre-computed maximum range necessary for the demodulated algorithm to produce accurate results. The signed fixed-point data type that has the best

precision fully containing the values in the range $2*(\sqrt{M}-1)$ for the specified word length.

Setting to `Same as product output` allows the Sum data type to be the same as the **Product output** data type (when **Product output** is used). If the **Product output** is not used, then this setting will be ignored and the `Inherit via internal rule` Sum setting will be used.

Pair Block   Rectangular QAM Modulator Baseband

See Also   General QAM Demodulator Baseband

References   [1] Smith, Joel G., "Odd-Bit Quadrature Amplitude-Shift Keying," *IEEE Transactions on Communications*, Vol. COM-23, March 1975, 385–389.

Purpose   Modulate using rectangular quadrature amplitude modulation

Library   AM, in Digital Baseband sublibrary of Modulation

Description   The Rectangular QAM Modulator Baseband block modulates using
                  M-ary quadrature amplitude modulation with a constellation on a
                  rectangular lattice. The output is a baseband representation of the
                  modulated signal.

Note All values of power assume a nominal impedance of 1 ohm.

Constellation Size and Scaling

The signal constellation has M points, where M is the **M-ary number**
parameter. M must have the form 2^K for some positive integer K.
The block scales the signal constellation based on how you set the
Normalization method parameter. The following table lists the
possible scaling conditions.

Value of Normalization Method Parameter	Scaling Condition
Min. distance between symbols	The nearest pair of points in the constellation is separated by the value of the **Minimum distance** parameter
Average Power	The average power of the symbols in the constellation is the **Average power** parameter
Peak Power	The maximum power of the symbols in the constellation is the **Peak power** parameter

Rectangular QAM Modulator Baseband

Input Signal Values

The input and output for this block are discrete-time signals. The **Input type** parameter determines whether the block accepts integers between 0 and M-1, or binary representations of integers:

- If **Input type** is set to `Integer`, the block accepts integers. The input can be either a scalar or a frame-based column vector, and the block can accept the data types `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, and `double`.

- If **Input type** is set to `Bit`, the block accepts groups of K bits, called *binary words*. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K. For bit inputs, the block can accept `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, `single`, and `double`. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation. Such assignments apply independently to the in-phase and quadrature components of the input:

  - If **Constellation ordering** is set to `Binary`, the block uses a natural binary-coded constellation.

  - If **Constellation ordering** is set to `Gray` and K is even, the block uses a Gray-coded constellation.

  - If **Constellation ordering** is set to `Gray` and K is odd, the block codes the constellation so that pairs of nearest points differ in one or two bits. The constellation is cross-shaped, and the schematic below indicates which pairs of points differ in two bits. The schematic uses M = 128, but suggests the general case.
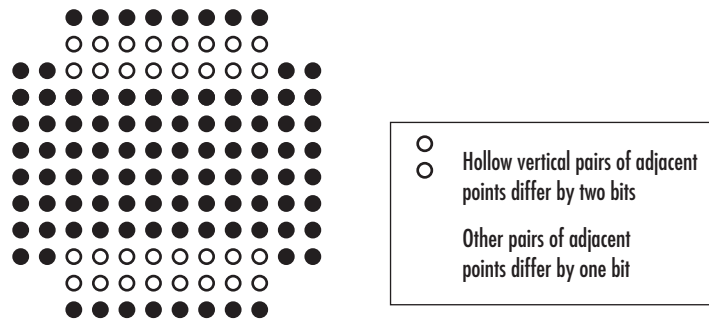
Hollow vertical pairs of adjacent points differ by two bits

Other pairs of adjacent points differ by one bit

For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block and the paper listed in References. Because the in-phase and quadrature components are assigned independently, the Gray and binary orderings coincide when M = 4.

Constellation Visualization

The Rectangular QAM Modulator Baseband block provides the capability to visualize a signal constellation from the block mask. This Constellation Visualization feature allows you to visualize a signal constellation for specific block parameters. For more information, see the Constellation Visualization section of the *Communications Blockset User's Guide*.

Rectangular QAM Modulator Baseband
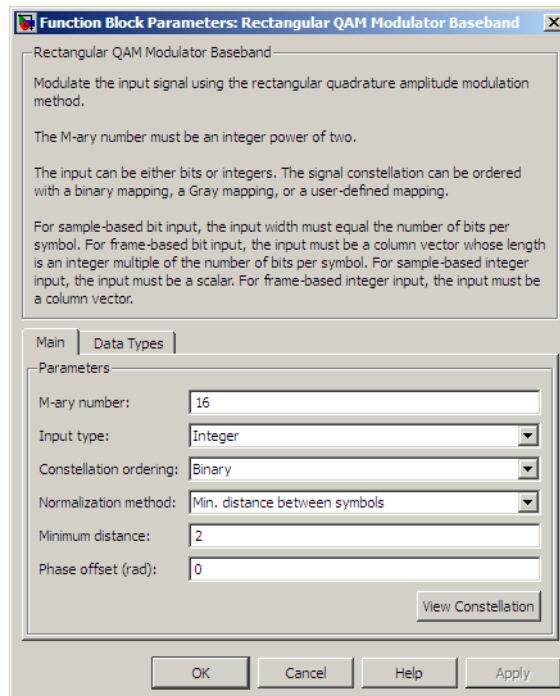
**Dialog
Box**

M-ary number

The number of points in the signal constellation. It must have the form 2^K for some positive integer K.

Input type

Indicates whether the input consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each symbol to a group of output bits or integer.

Selecting User-defined displays the field **Constellation mapping**, which allows for user-specified mapping.

Constellation mapping

This parameter is a row or column vector of size M and must have unique integer values in the range [0, M-1]. The values must be of data type `double`.

The first element of this vector corresponds to the top-leftmost point of the constellation, with subsequent elements running down column-wise, from left to right. The last element corresponds to the bottom-rightmost point.

This field appears when `User-defined` is selected in the drop-down list **Constellation ordering**.

Normalization method

Determines how the block scales the signal constellation. Choices are `Min. distance between symbols`, `Average Power`, and `Peak Power`.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to `Min. distance between symbols`.

Average power, referenced to 1 ohm (watts)

The average power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Average Power`.

Peak power, referenced to 1 ohm (watts)

The maximum power of the symbols in the constellation, referenced to 1 ohm. This field appears only when **Normalization method** is set to `Peak Power`.

Phase offset (rad)

The rotation of the signal constellation, in radians.

Output data type

The output data type can be set to `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Rectangular QAM Modulator Baseband

Setting this parameter to `Fixed-point` or `User-defined` enables fields in which you can further specify details. Setting this parameter to `Inherit via back propagation`, sets the output data type and scaling to match the following block.

Output word length
Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type
Specify any signed built-in or signed fixed-point data type. You can specify fixed-point data types using the `sfix`, `sint`, `sfrac`, and `fixdt` functions from Simulink Fixed Point software. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set output fraction length to
Specify the scaling of the fixed-point output by either of the following methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Output fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter or when you select `User-defined` and the specified output data type is a fixed-point data type.

Output fraction length
For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set output fraction length to** parameter.

Pair Block   Rectangular QAM Demodulator Baseband

See Also   General QAM Modulator Baseband

References   [1] Smith, Joel G., "Odd-Bit Quadrature Amplitude-Shift Keying," *IEEE Transactions on Communications*, Vol. COM-23, March 1975, 385–389.

Rectangular QAM TCM Decoder

Purpose

Decode trellis-coded modulation data, modulated using QAM method

Library

Trellis-Coded Modulation

Description

The Rectangular QAM TCM Decoder block uses the Viterbi algorithm to decode a trellis-coded modulation (TCM) signal that was previously modulated using a QAM signal constellation.
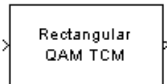
The **M-ary number** parameter is the number of points in the signal constellation, which also equals the number of possible output symbols from the convolutional encoder. (That is, $\log_2$(**M-ary number**) is the number of output bit streams from the convolutional encoder.)

The **Trellis structure** and **M-ary number** parameters in this block should match those in theRectangular QAM TCM Encoder block, to ensure proper decoding.

Input and Output Signals

The input signal must be a frame-based column vector containing complex numbers.

If the convolutional encoder described by the trellis structure represents a rate k/n code, then the Rectangular QAM TCM Decoder block's output is a frame-based binary column vector whose length is k times the vector length of the input signal.

Operation Modes

The block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses. This parameter also affects the range of possible values for the **Traceback depth** parameter, D.

- In Continuous mode, the block initializes all state metrics to zero at the beginning of the simulation, waits until it accumulates D symbols, and then uses a sequence of D symbols to compute each of the traceback paths. D can be any positive integer. At the end of each frame, the block saves its internal state metric for use with the next frame.

If you select the **Enable the reset input** check box, the block displays another input port, labeled Rst. This port receives an integer scalar signal. Whenever the value at the Rst port is nonzero, the block resets all state metrics to zero and sets the traceback memory to zero.

- In Truncated mode, the block treats each frame independently. The traceback path starts at the state with the lowest metric. D must be less than or equal to the vector length of the input.

- In Terminated mode, the block treats each frame independently. The traceback path always starts at the all-zeros state. D must be less than or equal to the vector length of the input. If you know that each frame of data typically ends at the all-zeros state, then this mode is an appropriate choice.

Decoding Delay

If you set **Operation mode** to Continuous, then this block introduces a decoding delay equal to **Traceback depth***k bits, for a rate k/n convolutional code. The decoding delay is the number of zeros that precede the first decoded bit in the output.

The block incurs no delay for other values of **Operation mode**.

Dialog Box

Rectangular QAM TCM Decoder

Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

M-ary number

The number of points in the signal constellation.

Traceback depth

The number of trellis branches (equivalently, the number of symbols) the block uses in the Viterbi algorithm to construct each traceback path.

Operation mode

The operation mode of the Viterbi decoder. Choices are `Continuous`, `Truncated`, and `Terminated`.

Enable the reset input port

When you check this box, the block has a second input port labeled `Rst`. Providing a nonzero input value to this port causes the block to set its internal memory to the initial state before processing the input data. This option appears only if you set **Operation mode** to `Continuous`.

Pair Block   Rectangular QAM TCM Encoder

See Also   General TCM Decoder, `poly2trellis`

References   [1] Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.
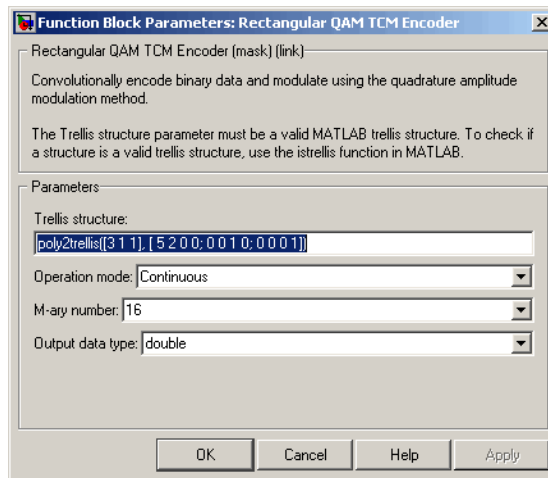
[2] Proakis, John G., *Digital Communications*, Fourth edition, New York, McGraw-Hill, 2001.

Purpose   Convolutionally encode binary data and modulate using QAM method

Library   Trellis-Coded Modulation
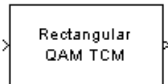
Description   The Rectangular QAM TCM Encoder block implements trellis-coded modulation (TCM) by convolutionally encoding the binary input signal and mapping the result to a QAM signal constellation.

Rectangular
QAM TCM

The **M-ary number** parameter is the number of points in the signal constellation, which also equals the number of possible output symbols from the convolutional encoder. (That is, $\log_2$(**M-ary number**) is equal to n for a rate k/n convolutional code.)

Input and Output Signals

If the convolutional encoder described by the trellis structure represents a rate k/n code, then the Rectangular QAM TCM Encoder block's input must be a frame-based binary column vector whose length is L*k for some positive integer L.

The output from the Rectangular QAM TCM Encoder block is a frame-based complex column vector of length L.

Specifying the Encoder

To define the convolutional encoder, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in "Trellis Description of a Convolutional Encoder" in the Communications Toolbox documentation. You can use this parameter field in two ways:

• If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, then use a `poly2trellis` command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7,[171 133],171)
```

Rectangular QAM TCM Encoder

- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is faster because it causes Simulink to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the previous bulleted item.

The encoder registers begin in the all-zeros state. You can configure the encoder so that it resets its registers to the all-zeros state during the course of the simulation. To do this, set the `Operation mode` to **Reset on nonzero input via port**. The block then opens a second input port, labeled `Rst`. The signal at the `Rst` port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.

Signal Constellations

The trellis-coded modulation technique partitions the constellation into subsets called cosets, so as to maximize the minimum distance between pairs of points in each coset. This block internally forms a valid partition based on the value you choose for the **M-ary number** parameter.

The figures below show the labeled set-partitioned signal constellations that the block uses when **M-ary number** is 16, 32, and 64. For constellations of other sizes, see Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

Rectangular QAM TCM Encoder
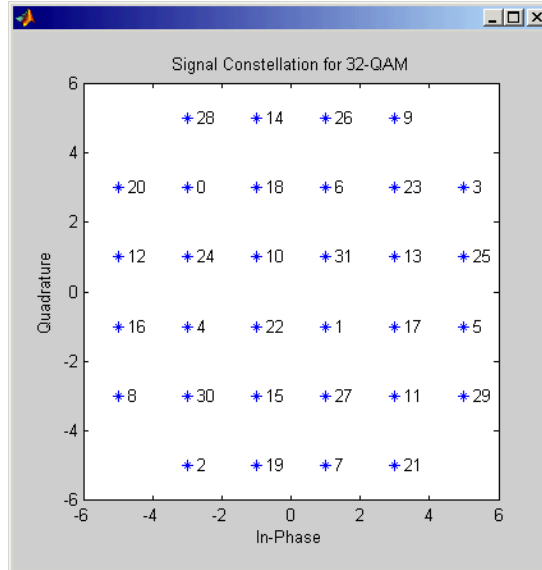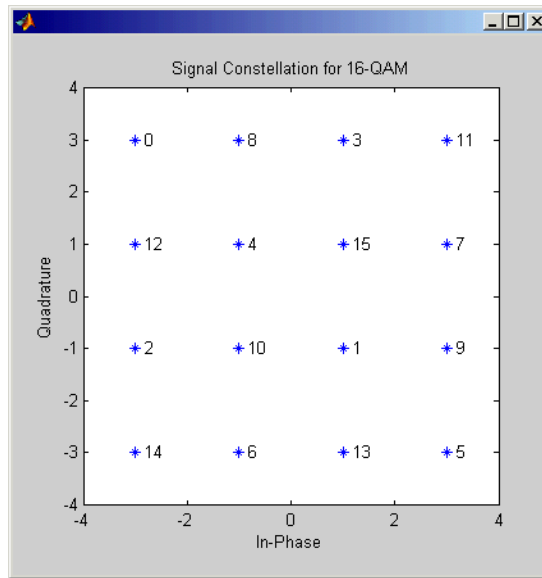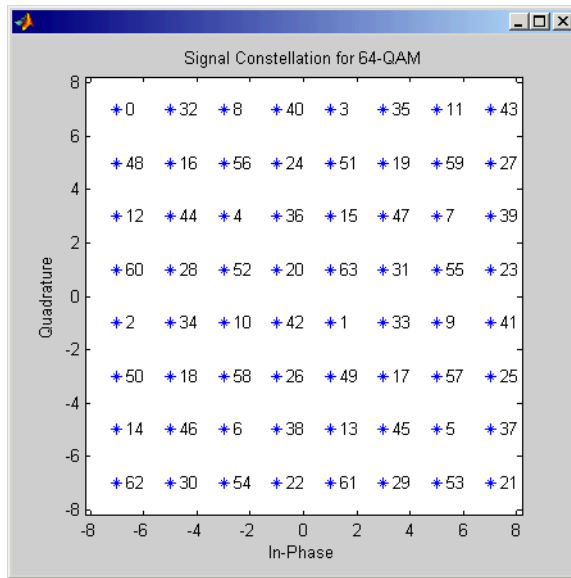
Signal Constellation for 64-QAM

Coding Gains

Coding gains of 3 to 6 decibels, relative to the uncoded case can be achieved in the presence of AWGN with multiphase trellis codes. For more information, see Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

Dialog Box

Trellis structure
>   MATLAB structure that contains the trellis description of the convolutional encoder.

Operation mode
>   In Continuous mode (default setting), the block retains the encoder states at the end of each frame, for use with the next frame.
>
>   In Truncated (reset every frame) mode, the block treats each frame independently. I.e., the encoder states are reset to all-zeros state at the start of each frame.
>
>   In Terminate trellis by appending bits mode, the block treats each frame independently. For each input frame, extra bits are used to set the encoder states to all-zeros state at the end of the frame. The output length is given by
>
> $y = n \cdot (x + s)/k$, where x is the number of input bits, and
>
> $s = \text{constraint length} - 1$ (or, in the case of multiple constraint

lengths, s =sum(ConstraintLength(i)-1)). This mode is supported for frame based inputs only.

In Reset on nonzero input via port mode, the block has an additional input port, labeled Rst. When the Rst input is nonzero, the encoder resets to the all-zeros state.

M-ary number
    The number of points in the signal constellation.

Output data type
    The output type of the block can be specified as a single or double. By default, the block sets this to double.

Pair Block   Rectangular QAM TCM Decoder

See Also   General TCM Encoder, poly2trellis

References   [1] Biglieri, E., D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, New York, Macmillan, 1991.

[2] Proakis, John G., Digital Communications, Fourth edition, New York, McGraw-Hill, 2001

[3] Ungerboeck, G., "Channel Coding with Multilevel/Phase Signals", *IEEE Trans. on Information Theory*, Vol IT28, Jan. 1982, pp. 55–67.

Purpose   Resample input at higher rate by repeating values

Library   Signal Operations

Description   The Filter block is a Signal Processing Blockset block. For more information, see the Repeat block reference page in the Signal Processing Blockset documentation.

Rician Noise Generator

Purpose   Generate Rician distributed noise

Library   Noise Generators sublibrary of Comm Sources

Description   The Rician Noise Generator block generates Rician distributed noise. The Rician probability density function is given by
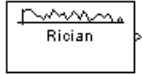
$$f(x) = \begin{cases} \dfrac{x}{\sigma^2} I_0\left(\dfrac{mx}{\sigma^2}\right) \exp\left(-\dfrac{x^2 + m^2}{2\sigma^2}\right) & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where:

- σ is the standard deviation of the Gaussian distribution that underlies the Rician distribution noise

- $m^2 = m_I^2 + m_Q^2$, where m_I and m_Q are the mean values of two independent Gaussian components

- I_0 is the modified 0th-order Bessel function of the first kind given by

$$I_0(y) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{y\cos t} dt$$

Note that *m* and σ are *not* the mean value and standard deviation for the Rician noise.

You must specify the **Initial seed** for the random number generator. When it is a constant, the resulting noise is repeatable. The vector length of the Initial seed parameter should equal the number of columns in a frame-based output or the number of elements in a sample-based output. The set of numerical parameters above the **Initial seed** parameter in the dialog box can consist of vectors having the same length as the **Initial seed**, or scalars.

Initial Seed

The scalar **Initial seed** parameter initializes the random number generator that the block uses to generate its Rician-distributed complex random process. For best results, the **Initial seed** should be a prime number greater than 30. Also, if there are other blocks in a model that have an **Initial seed** parameter, you should choose different initial seeds for all such blocks.

You can choose seeds for the Rician Noise Generator block using the Communications Blockset randseed function. At the MATLAB prompt, enter

```
randseed
```

This returns a random prime number greater than 30. Entering randseed again produces a different prime number. If you supply an integer argument, randseed always returns the same prime for that integer. For example, randseed(5) always returns the same answer.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See "Signal Attribute Parameters for Random Sources" in *Communications Blockset User's Guide* for more details.

The number of elements in the **Initial seed** and **Sigma** parameters becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** and **Sigma** parameters becomes the shape of a sample-based two-dimensional output signal.

Rician Noise Generator
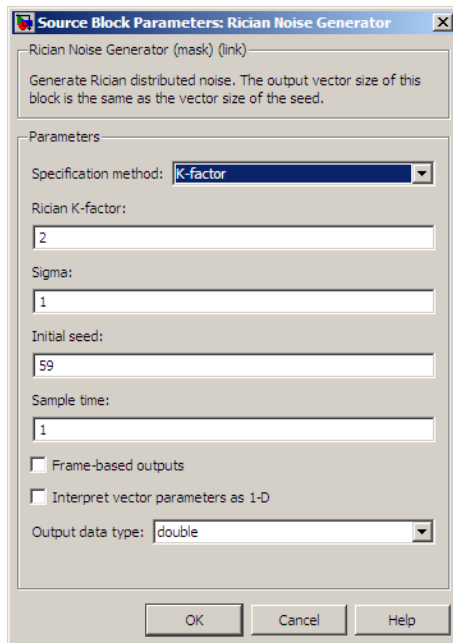
Dialog Box

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Specification method

> Either K-factor or Quadrature components.

Rician K-factor

> $K = m^2/(2\sigma^2)$, where m is as in the Rician probability density function. This field appears only if **Specification method** is K-factor.

In-phase component (mean), Quadrature component (mean)

> The mean values m_I and m_Q, respectively, of the Gaussian components. These fields appear only if **Specification method** is Quadrature components.

Sigma

The variable σ in the Rician probability density function.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type

The output can be set to double or single data types.

See Also   Multipath Rician Fading Channel

References   [1] Proakis, John G., *Digital Communications*, Third edition, New York, McGraw Hill, 1995.

RLS Decision Feedback Equalizer

Purpose   Equalize using decision feedback equalizer that updates weights with RLS algorithm

Library   Equalizers

Description   The RLS Decision Feedback Equalizer block uses a decision feedback equalizer and the RLS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the RLS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications BlocksetUser's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications BlocksetUser's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap of the forward filter.

RLS Decision Feedback Equalizer

Dialog Box

Number of forward taps

The number of taps in the forward filter of the decision feedback equalizer.

Number of feedback taps

The number of taps in the feedback filter of the decision feedback equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of forward taps in the equalizer.

Forgetting factor

The forgetting factor of the RLS algorithm, a number between 0 and 1.

Inverse correlation matrix

The initial value for the inverse correlation matrix. The matrix must be N-by-N, where N is the total number of forward and feedback taps.

Initial weights

A vector that concatenates the initial weights for the forward and feedback taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights
> If you check this box, the block outputs the current forward and feedback weights, concatenated into one vector.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

[2] Haykin, Simon, *Adaptive Filter Theory*, Third Ed., Upper Saddle River, N.J., Prentice-Hall, 1996.

[3] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

[4] Proakis, John G., *Digital Communications*, Fourth Ed., New York, McGraw-Hill, 2001.

See Also   RLS Linear Equalizer, LMS Decision Feedback Equalizer, CMA Equalizer

Purpose   Equalize using linear equalizer that updates weights using RLS algorithm

Library   Equalizers

Description   The RLS Linear Equalizer block uses a linear equalizer and the RLS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the RLS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

RLS Linear Equalizer

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap.

Dialog Box

```
Function Block Parameters: RLS Linear Equalizer                    ×

─ RLS Linear Equalizer (mask) (link) ─────────────────────────────

Equalize a linearly modulated signal through a dispersive channel using the RLS algorithm.

The block computes filter weights with the RLS algorithm and filters the input signal. When
the number of samples per symbol is 1, the filter weights are updated once for each symbol,
for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is
greater than one, the weights are updated once every Nth sample, for a fractionally spaced
(i.e. T/N-spaced) equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

If the Mode input port box is checked, the mode input toggles between training and decision
directed mode. For training, the mode input must be 1, and for decision directed, the mode
must be 0. For every frame in which the mode input is 1 or not present, the equalizer trains
at the beginning of the frame for the length of the desired signal.

─ Parameters ─────────────────────────────────────────────────────

Number of taps:
┌──────────────────────────────────────────────────────────────┐
│ 4                                                              │
└──────────────────────────────────────────────────────────────┘

Number of samples per symbol:
┌──────────────────────────────────────────────────────────────┐
│ 1                                                              │
└──────────────────────────────────────────────────────────────┘

Signal constellation:
┌──────────────────────────────────────────────────────────────┐
│ [-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j] │
└──────────────────────────────────────────────────────────────┘

Reference tap:
┌──────────────────────────────────────────────────────────────┐
│ 2                                                              │
└──────────────────────────────────────────────────────────────┘

Forgetting factor:
┌──────────────────────────────────────────────────────────────┐
│ .99                                                            │
└──────────────────────────────────────────────────────────────┘

Inverse correlation matrix
┌──────────────────────────────────────────────────────────────┐
│ .1*eye(4)                                                      │
└──────────────────────────────────────────────────────────────┘

Initial weights:
┌──────────────────────────────────────────────────────────────┐
│ 0                                                              │
└──────────────────────────────────────────────────────────────┘

☑ Mode input port
☑ Output error
☑ Output weights

        OK        Cancel        Help        Apply
```

Number of taps

> The number of taps in the filter of the linear equalizer.

Number of samples per symbol

> The number of input samples for each symbol.

RLS Linear Equalizer

Signal constellation
A vector of complex numbers that specifies the constellation for the modulation.

Reference tap
A positive integer less than or equal to the number of taps in the equalizer.

Forgetting factor
The forgetting factor of the RLS algorithm, a number between 0 and 1.

Inverse correlation matrix
The initial value for the inverse correlation matrix. The matrix must be N-by-N, where N is the number of taps.

Initial weights
A vector that lists the initial weights for the taps.

Mode input port
If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error
If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights
If you check this box, the block outputs the current weights.

Examples   See the Adaptive Equalization demo.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

[2] Haykin, Simon, *Adaptive Filter Theory*, Third Ed., Upper Saddle River, N.J., Prentice-Hall, 1996.

[3] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

[4] Proakis, John G., *Digital Communications*, Fourth Ed., New York, McGraw-Hill, 2001.

See Also   RLS Decision Feedback Equalizer, LMS Linear Equalizer, CMA Equalizer

Scrambler

Purpose　　　Scramble input signal

Library　　　Sequence Operations

Description
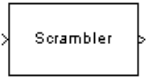
Scrambler

The Scrambler block scrambles the input signal, which must be a scalar or a frame-based column vector. If the **Calculation base** parameter is N, then the input values must be integers between 0 and N-1.

One purpose of scrambling is to reduce the length of strings of 0s or 1s in a transmitted signal, since a long string of 0s or 1s may cause transmission synchronization problems. Below is a schematic of the scrambler. All adders perform addition modulo N.
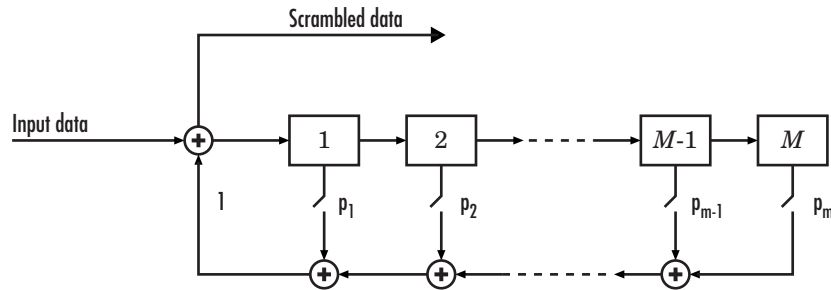
At each time step, the input causes the contents of the registers to shift sequentially. Each switch in the scrambler is on or off as defined by the **Scramble polynomial** parameter. You can specify the polynomial by listing its coefficients in order of ascending powers of z^{-1}, where $p(z^{-1}) = 1 + p_1 z^{-1} + p_2 z^{-2} + ...$, or by listing the powers of z that appear in the polynomial with a coefficient of 1. For example $p = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1]$ and $p = [0\ -6\ -8]$ both represent the polynomial $p(z^{-1}) = 1 + z^{-6} + z^{-8}$.

The **Initial states** parameter lists the states of the scrambler's registers when the simulation starts. The elements of this vector must be integers between 0 and N-1. The vector length of this parameter must equal the order of the scramble polynomial. (If the **Scramble polynomial** parameter is a vector that lists the coefficients in order, then the order of the scramble polynomial is one less than the vector length.)
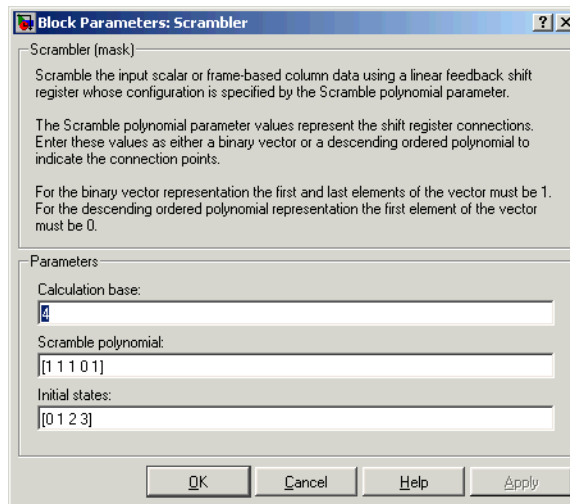
**Dialog
Box**

Calculation base

The calculation base N. The input and output of this block are integers in the range [0, N-1].

Scramble polynomial

A polynomial that defines the connections in the scrambler.

Initial states

The states of the scrambler's registers when the simulation starts.

Pair Block   Descrambler

See Also   PN Sequence Generator

Sign LMS Decision Feedback Equalizer

Purpose   Equalize using decision feedback equalizer that updates weights with
               signed LMS algorithm

Library   Equalizers

Description The Sign LMS Decision Feedback Equalizer block uses a decision
               feedback equalizer and an algorithm from the family of signed LMS
               algorithms to equalize a linearly modulated baseband signal through
               a dispersive channel. The supported algorithms, corresponding to the
               Update algorithm parameter, are

- Sign LMS

- Sign Regressor LMS

- Sign Sign LMS

During the simulation, the block uses the particular signed LMS
algorithm to update the weights, once per symbol. If the **Number of
samples per symbol** parameter is 1, then the block implements a
symbol-spaced equalizer; otherwise, the block implements a fractionally
spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as
a scalar or a frame-based column vector. The port labeled Desired
receives a training sequence whose length is less than or equal to the
number of symbols in the Input signal. Valid training symbols are
those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference
tap** is equal to or greater than the frame size, the block will not work
properly.

The port labeled Equalized outputs the result of the equalization
process.

You can configure the block to have one or more of these extra ports:

- `Mode` input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- `Err` output for the error signal, which is the difference between the `Equalized` output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- `Weights` output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications BlocksetUser's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap of the forward filter.

Sign LMS Decision Feedback Equalizer

Function Block Parameters: Sign LMS Decision Feedback Equalizer

Sign LMS Decision Feedback Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using a decision feedback equalizer and the signed LMS algorithm.

The block computes filter weights with the signed LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. For every frame in which the mode input is 1 or not present, the equalizer trains at the beginning of the frame for the length of the desired signal.

Parameters

Update algorithm | Sign LMS

Number of forward taps:
4

Number of feedback taps:
4

Number of samples per symbol:
1

Signal constellation:
[-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j]

Reference tap:
2

Step size:
0.01

Leakage factor:
1

Initial weights:
0

☑ Mode input port

☑ Output error

☑ Output weights

OK   Cancel   Help   Apply

Update algorithm

The specific type of signed LMS algorithm that the block uses to update the equalizer weights.

Number of forward taps

The number of taps in the forward filter of the decision feedback equalizer.

Number of feedback taps

The number of taps in the feedback filter of the decision feedback equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of forward taps in the equalizer.

Step size

The step size of the signed LMS algorithm.

Leakage factor

The leakage factor of the signed LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

A vector that concatenates the initial weights for the forward and feedback taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Sign LMS Decision Feedback Equalizer

Output error
> If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights
> If you check this box, the block outputs the current forward and feedback weights, concatenated into one vector.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

[2] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

See Also   Sign LMS Linear Equalizer, LMS Decision Feedback Equalizer

Purpose   Equalize using linear equalizer that updates weights with signed LMS algorithm

Library   Equalizers
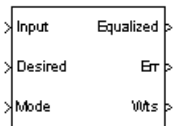
Description   The Sign LMS Linear Equalizer block uses a linear equalizer and an algorithm from the family of signed LMS algorithms to equalize a linearly modulated baseband signal through a dispersive channel. The supported algorithms, corresponding to the **Update algorithm** parameter, are

- `Sign LMS`

- `Sign Regressor LMS`

- `Sign Sign LMS`

During the simulation, the block uses the particular signed LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

Sign LMS Linear Equalizer

- `Mode` input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications BlocksetUser's Guide*.

- `Err` output for the error signal, which is the difference between the `Equalized` output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- `Weights` output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap.

Dialog Box

Function Block Parameters: Sign LMS Linear Equalizer

—Sign LMS Linear Equalizer (mask) (link)——————————————

Equalize a linearly modulated signal through a dispersive channel using the signed LMS algorithm.

The block computes filter weights with the signed LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. For every frame in which the mode input is 1 or not present, the equalizer trains at the beginning of the frame for the length of the desired signal.

—Parameters——————————————————————————————

Update algorithm Sign LMS

Number of taps:

4

Number of samples per symbol:

1

Signal constellation:

[-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j]

Reference tap:

2

Step size:

0.01

Leakage factor:

1

Initial weights:

0

☑ Mode input port

☑ Output error

☑ Output weights

OK   Cancel   Help   Apply

Sign LMS Linear Equalizer

Update algorithm

The specific type of signed LMS algorithm that the block uses to update the equalizer weights.

Number of taps

The number of taps in the filter of the linear equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of taps in the equalizer.

Step size

The step size of the signed LMS algorithm.

Leakage factor

The leakage factor of the signed LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

A vector that lists the initial weights for the taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current weights.

Examples   See the Adaptive Equalization demo.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

[2] Kurzweil, Jack, *An Introduction to Digital Communications*, New York, Wiley, 2000.

See Also   Sign LMS Decision Feedback Equalizer, LMS Linear Equalizer

Squaring Timing Recovery

Purpose   Recover symbol timing phase using squaring method

Library   Timing Phase Recovery sublibrary of Synchronization

Description   The Squaring Timing Recovery block recovers the symbol timing phase of the input signal using a squaring method. This frame-based, feedforward, non-data-aided method is similar to the conventional squaring loop. This block is suitable for systems that use linear baseband modulation types such as pulse amplitude modulation (PAM), phase shift keying (PSK) modulation, and quadrature amplitude modulation (QAM).

Typically, the input to this block is the output of a receive filter that is matched to the transmitting pulse shape. The input to this block must be a frame-based column vector of type `double` or `single`. The input represents **Symbols per frame** symbols using **Samples per symbol** samples for each symbol. Typically, **Symbols per frame** is approximately 100, **Samples per symbol** is at least 4, and the input signal is shaped using a raised cosine filter.

Note The block assumes that the phase offset is constant for all symbols in the entire input frame. If necessary, use the Buffer block to reorganize your data into frames over which the phase offset can be assumed constant. If the assumption of constant phase offset is valid, then a larger frame length yields a more accurate phase offset estimate.

The block estimates the phase offset for the symbols in each input frame and applies the estimate uniformly over the input frame. The block outputs frame-based signals, each containing one sample per symbol. The frame size of each output therefore equals the **Symbols per frame** parameter value. The outputs are as follows:

- The output port labeled Sym gives the result of applying the phase estimate uniformly over the input frame. This output is the signal value for each symbol, which can be used for decision purposes.

• The output port labeled Ph gives the phase estimate for each symbol in the input frame. All elements in this output frame are the *same* nonnegative real number less than the **Samples per symbol** parameter value. Noninteger values for the phase estimate correspond to interpolated values that lie between two values of the input signal.

Dialog Box
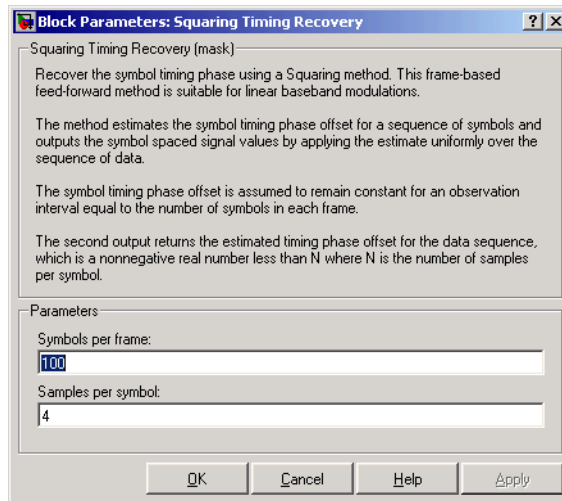
Symbols per frame
> The number of symbols in each frame of the input signal.

Samples per symbol
> The number of input samples that represent each symbol. This must be greater than 1.

Algorithm   This block uses a timing estimator that returns

$$-\frac{1}{2\pi}\arg\left(\sum_{m=0}^{LN-1}|x_{m+1}|^2\exp(\text{-j}2\pi m/N)\right)$$

as the normalized phase between -1/2 and 1/2, where *x* is the input vector, L is the **Symbols per frame** parameter and N is the **Samples per symbol** parameter.

For more information about the role that the timing estimator plays in this block's algorithm, see "Feedforward Method for Timing Phase Recovery" in *Communications Blockset User's Guide*.

Examples   See "Squaring Timing Phase Recovery Example" in *Communications Blockset User's Guide*.

References   [1] Oerder, M. and H. Myer, "Digital Filter and Square Timing Recovery," *IEEE Transactions on Communications*, Vol. COM-36, No. 5, May 1988, pp. 605-612.

[2] Mengali, Umberto and Aldo N. D'Andrea, *Synchronization Techniques for Digital Receivers*, New York, Plenum Press, 1997.

[3] Meyr, Heinrich, Marc Moeneclaey, and Stefan A. Fechtel, *Digital Communication Receivers*, Vol 2, New York, Wiley, 1998.

See Also   Gardner Timing Recovery, Early-Late Gate Timing Recovery

Purpose   Demodulate SSB-AM-modulated data

Library   Analog Passband Modulation, in Modulation

Description   The SSB AM Demodulator Passband block demodulates a signal that
was modulated using single-sideband amplitude modulation. The input
is a passband representation of the modulated signal. Both the input
and output signals are real sample-based scalar signals.

This block works only with real inputs of type double. This block is not
suited to be placed inside a triggered subsystem.

**Dialog
Box**
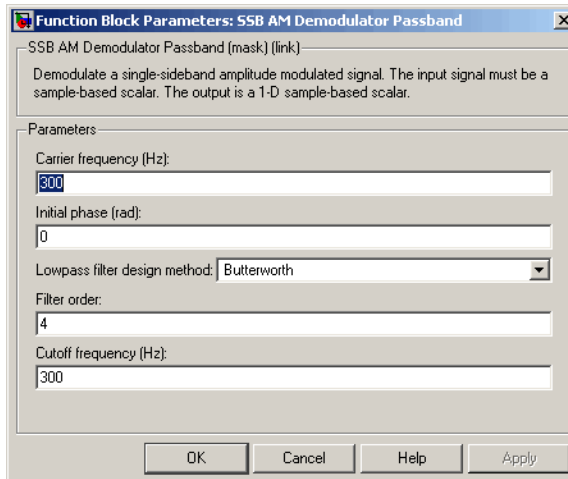
Carrier frequency (Hz)
> The carrier frequency in the corresponding SSB AM Modulator
> Passband block.

Initial phase (rad)
> The phase offset, θ, of the modulated signal.

SSB AM Demodulator Passband

Lowpass filter design method

The method used to generate the filter. Available methods are Butterworth, Chebyshev type I, Chebyshev type II, and Elliptic.

Filter order

The order of the lowpass digital filter specified in the **Lowpass filter design method** field .

Cutoff frequency

The cutoff frequency of the lowpass digital filter specified in the **Lowpass filter design method** field in Hertz.

Passband ripple

Applies to Chebyshev type I and Elliptic filters only. This is peak-to-peak ripple in the passband in dB.

Stopband ripple

Applies to Chebyshev type II and Elliptic filters only. This is the peak-to-peak ripple in the stopband in dB.

Pair Block   SSB AM Modulator Passband

See Also   DSB AM Demodulator Passband, DSBSC AM Demodulator Passband

Purpose   Modulate using single-sideband amplitude modulation

Library   Analog Passband Modulation, in Modulation

Description   The SSB AM Modulator Passband block modulates using
single-sideband amplitude modulation with a Hilbert transform filter.
The output is a passband representation of the modulated signal. Both
the input and output signals are real sample-based scalar signals.

SSB AM Modulator Passband transmits either the lower or upper
sideband signal, but not both. To control which sideband it transmits,
use the **Sideband to modulate** parameter.

If the input is $u(t)$ as a function of time t, then the output is

$$u(t)\cos(f_c t + \theta) \mp \hat{u}(t)\sin(f_c t + \theta)$$

where:

- f_c is the **Carrier frequency** parameter.

- θ is the **Initial phase** parameter.

- $\hat{u}(t)$ is the Hilbert transform of the input $u(t)$.

- The minus sign indicates the upper sideband and the plus sign
  indicates the lower sideband.

Hilbert Tranform Filter

This block uses the Analytic Signal block from the Signal Processing
Blockset Transforms library.

The Analytic Signal block computes the complex analytic signal
corresponding to each channel of the real M-by-N input, u

$$y = u + j\mathrm{H}\{u\}$$

where $j = \sqrt{-1}$ and H{} denotes the Hilbert transform. The real part
of the output in each channel is a replica of the real input in that

channel; the imaginary part is the Hilbert transform of the input. In the frequency domain, the analytic signal retains the positive frequency content of the original signal while zeroing-out negative frequencies and doubling the DC component.

The block computes the Hilbert transform using an equiripple FIR filter with the order specified by the Filter order parameter, n. The linear phase filter is designed using the Remez exchange algorithm, and imposes a delay of n/2 on the input samples.

For best results, use a carrier frequency which is estimated to be larger than 10% of your input signal's sample time. This is due to the implementation of the Hilbert transform by means of a filter.

In the following example, we sample a 10Hz input signal at 8000 samples per second. We then designate a Hilbert Transform filter of order 100. Below is the response of the Hilbert Transform filter as returned by `fvtool`.

Note the bandwidth of the filter's magnitude response. By choosing a carrier frequency larger than 10% (but less than 90%) of the input signal's sample time (8000 samples per second, in this example) or equivalently, a carrier frequency larger than 400Hz, we ensure that the Hilbert Transform Filter will be operating in the flat section of the filter's magnitude response (shown in blue), and that our modulated signal will have the desired magnitude and form.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. By the Nyquist sampling theorem, the reciprocal of the model's sample time (defined by the model's signal source) must exceed twice the **Carrier frequency** parameter.

SSB AM Modulator Passband

This block works only with real inputs of type `double`. This block is not suited to be placed inside a triggered subsystem.

Dialog Box
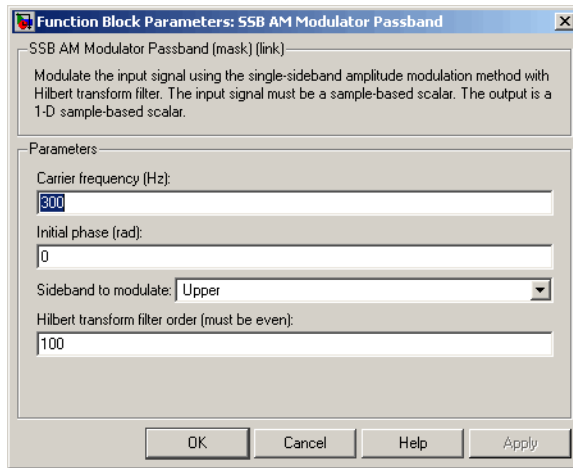
Carrier frequency (Hz)
> The frequency of the carrier.

Initial phase (rad)
> The phase offset, θ, of the modulated signal.

Sideband to modulate
> This parameter specifies whether to transmit the upper or lower sideband.

Hilbert Transform filter order
> The length of the FIR filter used to compute the Hilbert transform.

Pair Block   SSB AM Demodulator Passband

See Also   DSB AM Modulator Passband, DSBSC AM Modulator Passband; `hilbiir` (Communications Toolbox)

References   [1] Peebles, Peyton Z, Jr. *Communication System Principles*. Reading,
Mass.: Addison-Wesley, 1976.

Uniform Noise Generator

Purpose   Generate uniformly distributed noise between upper and lower bounds

Library   Noise Generators sublibrary of Comm Sources
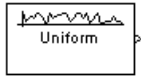
Description   The Uniform Noise Generator block generates uniformly distributed noise. The output data of this block is uniformly distributed between the specified lower and upper bounds. The upper bound must be greater than or equal to the lower bound.

You must specify the **Initial seed** in the simulation. When it is a constant, the resulting noise is repeatable.

If all the elements of the output vector are to be independent and identically distributed (i.i.d.), then you can use a scalar for the **Noise lower bound** and **Noise upper bound** parameters. Alternatively, you can specify the range for each element of the output vector individually, by using vectors for the **Noise lower bound** and **Noise upper bound** parameters. If the bounds are vectors, then their length must equal the length of the **Initial seed** parameter.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See "Signal Attribute Parameters for Random Sources" in *Communications Blockset User's Guide* for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Noise lower bound, Noise upper bound
　　The lower and upper bounds of the interval over which noise is uniformly distributed.

Initial seed
　　The initial seed value for the random number generator.

Sample time
　　The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs
　　Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame
> The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D
> If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

Output data type
> The output can be set to `double` or `single` data types.

See Also Random Source (Signal Processing Blockset documentation); `rand` (built-in MATLAB function)

Purpose   Map unipolar signal in range [0, M-1] into bipolar signal

Library   Utility Blocks
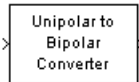
Description   The Unipolar to Bipolar Converter block maps the unipolar input signal to a bipolar output signal. If the input consists of integers between 0 and M-1, where M is the **M-ary number** parameter, then the output consists of integers between -(M-1) and M-1. If M is even, then the output is odd. If M is odd, then the output is even. This block is only designed to work when the input value is within the set {0,1,2...(M-1)}, where M is the **M-ary number** parameter. If the input value is outside of this set of integers the output may not be valid.

The table below shows how the block's mapping depends on the **Polarity** parameter.

Polarity Parameter Value	Output Corresponding to Input Value of k
Positive	2k-(M-1)
Negative	-2k+(M-1)

Dialog Box
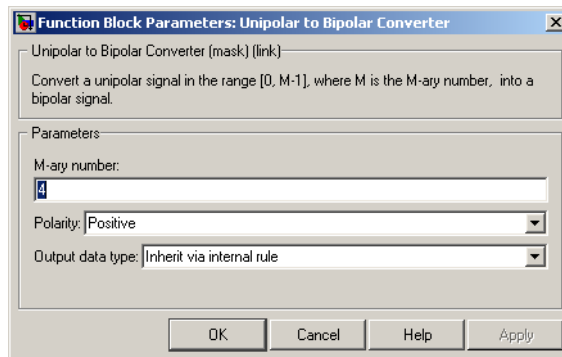
M-ary number
    The number of symbols in the bipolar or unipolar alphabet.

Unipolar to Bipolar Converter

Polarity

A value of `Positive` causes the block to maintain the relative ordering of symbols in the alphabets. A value of `Negative` causes the block to reverse the relative ordering of symbols in the alphabets.

Output Data Type

The type of bipolar signal produced at the block's output.

The block supports the following output data types:

- `Inherit via internal rule`

- `Same as input`

- `double`

- `int8`

- `int16`

- `int32`

When the parameter is set to its default setting, `Inherit via internal rule`, the block determines the output data type based on the input data type.

- If the input signal is floating-point (either `single` or `double`), the output data type is the same as the input data type.

- If the input data type is not floating-point:

  - Based on the **M-ary** number parameter, an ideal signed integer output word length required to contain the range [-(M-1)M-1] is computed as follows:

    ideal word length = ceil(log2(M))+1

    Note The +1 is associated with the need for the sign bit.

— The block sets the output data type to be a signed integer, based on the smallest word length (in bits) that can fit best the computed ideal word length.

> **Note** The selections in the **Hardware Implementation** pane pertaining to word length constraints do not affect how this block determines output data types.

Examples   If the input is [0; 1; 2; 3], the **M-ary number** parameter is 4, and the **Polarity** parameter is Positive, then the output is [-3; -1; 1; 3]. Changing the **Polarity** parameter to Negative changes the output to [3; 1; -1; -3].

If the value for the **M-ary number** is 2^7 the block gives an output of int8.

If the value for the **M-ary number** is 2^7+1 the block gives an output of int16.

Pair Block   Bipolar to Unipolar Converter

Variable Step LMS Decision Feedback Equalizer

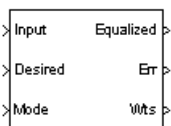**Purpose**   Equalize using decision feedback equalizer that updates weights with
                   variable-step-size LMS algorithm

Library   Equalizers

Description   The Variable Step LMS Decision Feedback Equalizer block uses a
                   decision feedback equalizer and the variable-step-size LMS algorithm
                   to equalize a linearly modulated baseband signal through a dispersive
                   channel. During the simulation, the block uses the variable-step-size
                   LMS algorithm to update the weights, once per symbol. If the **Number
                   of samples per symbol** parameter is 1, then the block implements a
                   symbol-spaced equalizer; otherwise, the block implements a fractionally
                   spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as
a scalar or a frame-based column vector. The port labeled Desired
receives a training sequence whose length is less than or equal to the
number of symbols in the Input signal. Valid training symbols are
those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference
tap** is equal to or greater than the frame size, the block will not work
properly.

The port labeled Equalized outputs the result of the equalization
process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or
  Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the
  Equalized output and the reference signal. The reference signal
  consists of training symbols in training mode, and detected symbols
  otherwise.

- `Weights` output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide*.

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide*.

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

1+(**Reference tap**-1)/(**Number of samples per symbol**)

Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap of the forward filter.

Variable Step LMS Decision Feedback Equalizer

Dialog Box

Function Block Parameters: Variable Step LMS Decision Feedback Equalizer

Variable Step LMS Decision Feedback Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using a decision feedback equalizer and the variable step LMS algorithm.

The block computes filter weights with the variable step LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. The equalizer will train for the length of the Desired signal. If the mode input is not present, the equalizer will train at the beginning of every frame for the length of the Desired signal.

Parameters

Number of forward taps:

```
5
```

Number of feedback taps:

```
6
```

Number of samples per symbol:

```
1
```

Signal constellation:

```
[-1+j -1-j 1+j 1-j]
```

Reference tap:

```
3
```

Initial step size:

```
.01
```

Increment step size:

```
.001
```

Minimum step size:

```
.001
```

Maximum step size:

```
.1
```

Leakage factor:

```
1
```

Initial weights:

```
0
```

☑ Mode input port

☑ Output error

☑ Output weights

OK   Cancel   Help   Apply

Number of forward taps

> The number of taps in the forward filter of the decision feedback equalizer.

Number of feedback taps

> The number of taps in the feedback filter of the decision feedback equalizer.

Number of samples per symbol

> The number of input samples for each symbol.

Signal constellation

> A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

> A positive integer less than or equal to the number of forward taps in the equalizer.

Initial step size

> The step size that the variable-step-size LMS algorithm uses at the beginning of the simulation.

Increment step size

> The increment by which the step size changes from iteration to iteration

Minimum step size

> The smallest value that the step size can assume.

Maximum step size

> The largest value that the step size can assume.

Leakage factor

> The leakage factor of the variable-step-size LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

> A vector that concatenates the initial weights for the forward and feedback taps.

Variable Step LMS Decision Feedback Equalizer

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current forward and feedback weights, concatenated into one vector.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

See Also   Variable Step LMS Linear Equalizer, LMS Decision Feedback Equalizer

Purpose   Equalize using linear equalizer that updates weights with variable-step-size LMS algorithm

Library   Equalizers

Description   The Variable Step LMS Linear Equalizer block uses a linear equalizer and the variable-step-size LMS algorithm to equalize a linearly modulated baseband signal through a dispersive channel. During the simulation, the block uses the variable-step-size LMS algorithm to update the weights, once per symbol. If the **Number of samples per symbol** parameter is 1, then the block implements a symbol-spaced equalizer; otherwise, the block implements a fractionally spaced equalizer.

Input and Output Signals

The port labeled Input receives the signal you want to equalize, as a scalar or a frame-based column vector. The port labeled Desired receives a training sequence whose length is less than or equal to the number of symbols in the Input signal. Valid training symbols are those listed in the **Signal constellation** vector.

This block accepts only frame-based signals. If the value of **Reference tap** is equal to or greater than the frame size, the block will not work properly.

The port labeled Equalized outputs the result of the equalization process.

You can configure the block to have one or more of these extra ports:

- Mode input, as described in "Controlling the Use of Training or Decision-Directed Mode" in *Communications Blockset User's Guide*.

- Err output for the error signal, which is the difference between the Equalized output and the reference signal. The reference signal consists of training symbols in training mode, and detected symbols otherwise.

- Weights output, as described in "Retrieving the Weights and Error Signal" in *Communications Blockset User's Guide.*

Decision-Directed Mode and Training Mode

To learn the conditions under which the equalizer operates in training or decision-directed mode, see "Using Adaptive Equalizers" in *Communications Blockset User's Guide.*

Equalizer Delay

For proper equalization, you should set the **Reference tap** parameter so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay, in symbols, between the modulator output and the equalizer *output* is equal to

$$1+(\textbf{Reference tap}\text{-}1)/(\textbf{Number of samples per symbol})$$

Since the channel delay is typically unknown, a common practice is to set the reference tap to the center tap.

Dialog Box

Function Block Parameters: Variable Step LMS Linear Equalizer

Variable Step LMS Linear Equalizer (mask) (link)

Equalize a linearly modulated signal through a dispersive channel using the variable step LMS algorithm.

The block computes filter weights with the variable step LMS algorithm and filters the input signal. When the number of samples per symbol is 1, the filter weights are updated once for each symbol, for a symbol spaced (i.e. T-spaced) equalizer. When the number of samples per symbol is greater than one, the weights are updated once every Nth sample, for a T/N-spaced equalizer.

The Desired input is used for training the equalizer. It expects complex constellation points.

The Leakage factor must be in the range 0 to 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

If the Mode Input box is checked, the mode input toggles between training and decision directed mode. For training, the mode input must be 1, for decision directed, the mode should be 0. For every frame in which the mode input is 1 or not present, the equalizer trains at the beginning of the frame for the length of the desired signal.

Parameters

Number of taps:

    4

Number of samples per symbol:

    1

Signal constellation:

    [-3+3j -3+j -3-j -3-3j -1+3j -1+j -1-j -1-3j 1+3j 1+j 1-j 1-3j 3+3j 3+j 3-j 3-3j]

Reference tap:

    2

Initial step size:

    .01

Increment step size:

    .01

Minimum step size:

    .001

Maximum step size:

    .1

Leakage factor:

    1

Initial weights:

    0

☑ Mode input port

☑ Output error

☑ Output weights

    OK   Cancel   Help   Apply

Variable Step LMS Linear Equalizer

Number of taps

The number of taps in the filter of the linear equalizer.

Number of samples per symbol

The number of input samples for each symbol.

Signal constellation

A vector of complex numbers that specifies the constellation for the modulation.

Reference tap

A positive integer less than or equal to the number of taps in the equalizer.

Initial step size

The step size that the variable-step-size LMS algorithm uses at the beginning of the simulation.

Increment step size

The increment by which the step size changes from iteration to iteration

Minimum step size

The smallest value that the step size can assume.

Maximum step size

The largest value that the step size can assume.

Leakage factor

The leakage factor of the LMS algorithm, a number between 0 and 1. A value of 1 corresponds to a conventional weight update algorithm, and a value of 0 corresponds to a memoryless update algorithm.

Initial weights

A vector that lists the initial weights for the taps.

Mode input port

If you check this box, the block has an input port that enables you to toggle between training and decision-directed mode.

Output error

If you check this box, the block outputs the error signal, which is the difference between the equalized signal and the reference signal.

Output weights

If you check this box, the block outputs the current weights.

Examples   See the Adaptive Equalization demo.

References   [1] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Chichester, England, Wiley, 1998.

See Also   Variable Step LMS Decision Feedback Equalizer, LMS Linear Equalizer

Viterbi Decoder

Purpose   Decode convolutionally encoded data using Viterbi algorithm

Library   Convolutional sublibrary of Channel Coding

Description

The Viterbi Decoder block decodes input symbols to produce binary output symbols. This block can process several symbols at a time for faster performance.
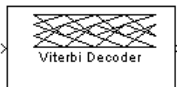
Input and Output Sizes

If the convolutional code uses an alphabet of 2^n possible symbols, this block's input vector length is L*n for some positive integer L. Similarly, if the decoded data uses an alphabet of 2^k possible output symbols, this block's output vector length is L*k. The integer L is the number of frames that the block processes in each step.

The input can be either a sample-based vector with L = 1, or a frame-based column vector with any positive integer for L.

The block supports nondouble data typed input and output signals based on the **Decision type** selected from the mask. For unquantized decisions, the block accepts double or single typed inputs.

For hard decisions, the block accepts the input data types `ufix(1)`, `double`, `single`, `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

For soft decisions, the block accepts fixed-point signals (e.g., `ufix(3)`), `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`. The fixed-point input signal must have a fraction length of 0 and be unsigned.

Input Values and Decision Types

The entries of the input vector are either bipolar, binary, or integer data, depending on the **Decision type** parameter.

Decision type Parameter	Possible Entries in Decoder Input	Interpretation of Values
Unquantized	Real numbers	Positive real: logical zero Negative real: logical one
Hard Decision	0, 1	0: logical zero 1: logical one
Soft Decision	Integers between 0 and 2^b-1, where b is the **Number of soft decision bits** parameter.	0: most confident decision for logical zero 2^b-1: most confident decision for logical one Other values represent less confident decisions.

To illustrate the soft decision situation more explicitly, the table below lists interpretations of values for 3-bit soft decisions.

Input Value	Interpretation
0	Most confident zero
1	Second most confident zero
2	Third most confident zero
3	Least confident zero
4	Least confident one
5	Third most confident one
6	Second most confident one
7	Most confident one

Operation Modes for Frame-Based Inputs

If the input signal is frame-based, the block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses:

- In Continuous mode, the block saves its internal state metric at the end of each frame, for use with the next frame. Each traceback path is treated independently.

- In Truncated mode, the block treats each frame independently. The traceback path starts at the state with the best metric and always ends in the all-zeros state. This mode is appropriate when the corresponding Convolutional Encoder block has its **Operation mode** set to Truncated (reset every frame).

- In Terminated mode, the block treats each frame independently, and the traceback path always starts and ends in the all-zeros state. This mode is appropriate when the uncoded message signal (that is, the input to the corresponding Convolutional Encoder block) has enough zeros at the end of each frame to fill all memory registers of the feed-forward encoder. If the encoder has k input streams and constraint length vector constr (using the polynomial description), "enough" means k*max(constr-1). For feedback encoders, this mode is appropriate if the corresponding Convolutional Encoder block has **Operation mode** set to Terminate trellis by appending bits.

In the special case when the frame-based input signal contains only one symbol, the Continuous mode is most appropriate.

Traceback Depth and Decoding Delay

The **Traceback depth** parameter, D, influences the decoding delay. The decoding delay is the number of zero symbols that precede the first decoded symbol in the output.

- If the input signal is sample-based, the decoding delay consists of D zero symbols

- If the input signal is frame-based and the **Operation mode** parameter is set to `Continuous`, the decoding delay consists of D zero symbols

- If the **Operation mode** parameter is set to `Truncated` or `Terminated`, there is no output delay and the **Traceback depth** parameter must be less than or equal to the number of symbols in each frame.

If the code rate is 1/2, a typical **Traceback depth** value is about five times the constraint length of the code.

Reset Port

The reset port is usable only when the **Operation mode** parameter is set to `Continuous`. Selecting the **Enable reset input port** check box gives the block an additional input port, labeled `Rst`. When the `Rst` input is nonzero, the decoder returns to its initial state by configuring its internal memory as follows:

- Sets the all-zeros state metric to zero.

- Sets all other state metrics to the maximum value.

- Sets the traceback memory to zero.

Using a reset port on this block is analogous to setting **Operation mode** in the Convolutional Encoder block to `Reset on nonzero input via port`.

The reset port supports `double` or `boolean` typed signals.

Fixed-Point Signal Flow Diagram

There are three main components to the Viterbi decoding algorithm. They are branch metric computation (BMC), add-compare and select (ACS), and traceback decoding (TBD). The following diagram illustrates the signal flow for a k/n rate code.

Viterbi Decoder
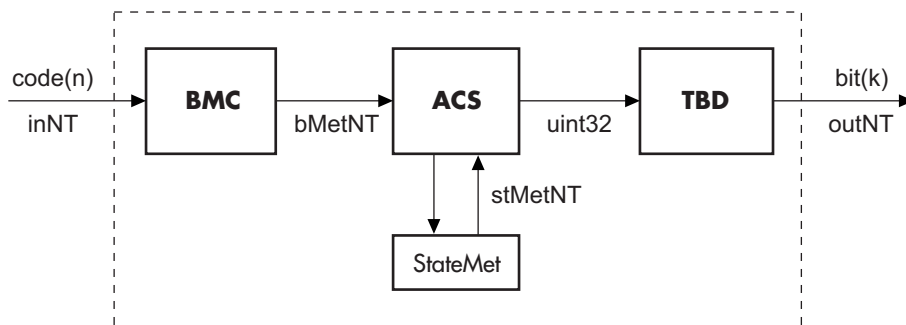
As an example of a BMC diagram, a 1/2 rate, *nsdec* = 3 signal flow would be as follows.
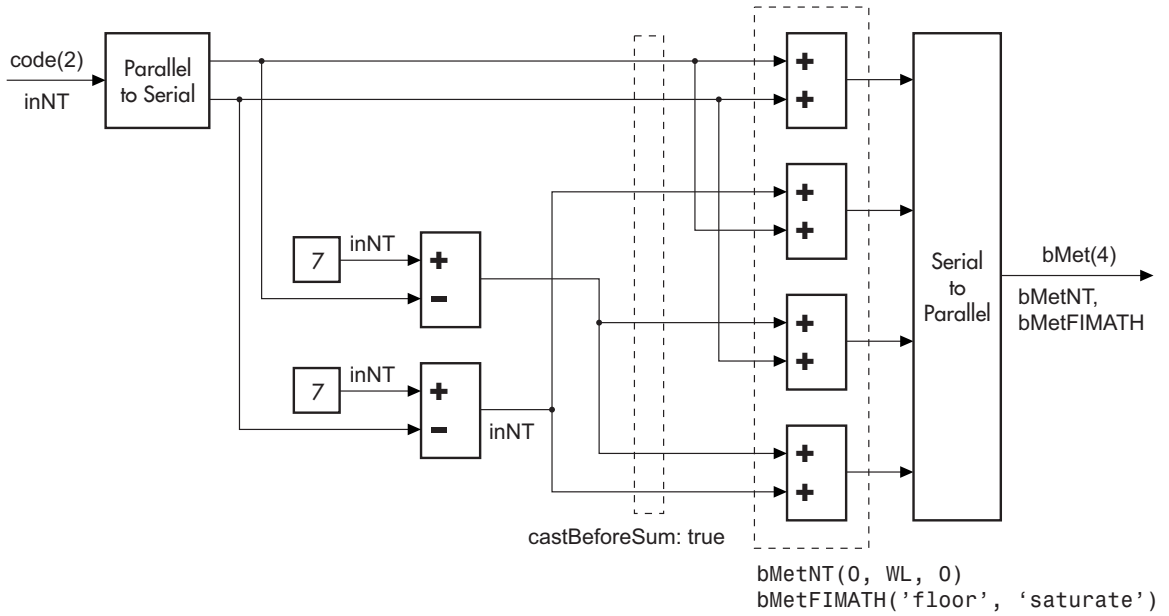
$$WL = nsdec + n - 1$$
$$n = 2 \Rightarrow WL = 4$$

The ACS component is generally illustrated as shown in the following diagram.
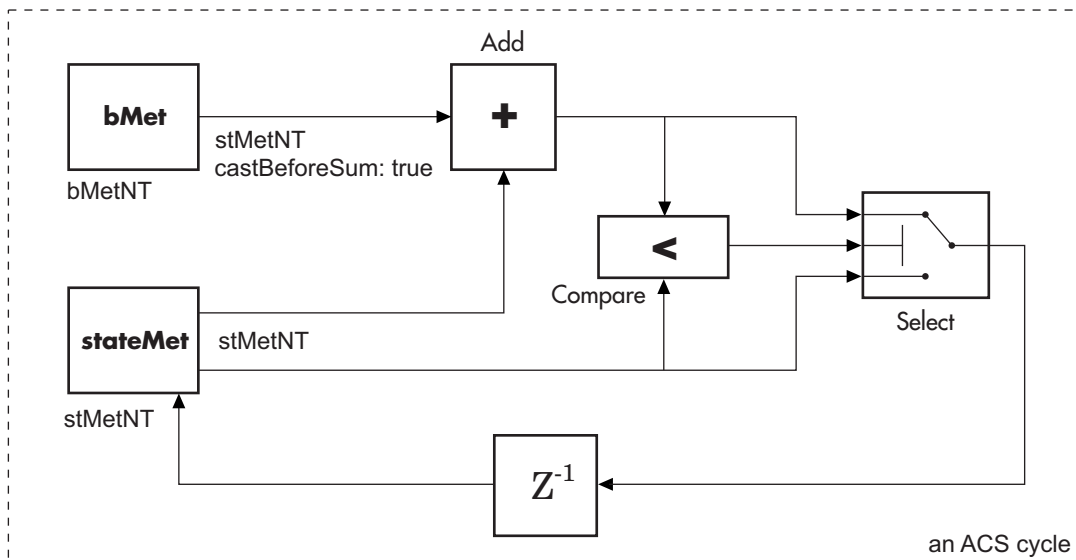

```
stMetNT(0, WL2, 0)
stMetFIMATH('floor', 'saturate')
```

Where `WL2` is specified on the mask by the user.

In the flow diagrams above, inNT, bMetNT , stMetNT, and outNT are `numerictype` objects, and bMetFIMATH and stMetFIMATH, are `fimath` objects.

Puncture Pattern Examples

For some commonly used puncture patterns for specific rates and polynomials, see the last three references.

Fixed-Point Viterbi Decoding Examples

The following two example models showcase the fixed-point Viterbi decoder block used for both hard- and soft-decision convolutional decoding.

If you are reading this reference page in the MATLAB Help Browser, click here (hard-decision) and here (soft-decision) to open the models. These can also be found as doc_fixpt_vitharddec.mdl and doc_fixpt_vitsoftdec.mdl under help\toolbox\commblks\examples.

Fixed-point Hard-Decision Viterbi Decoding
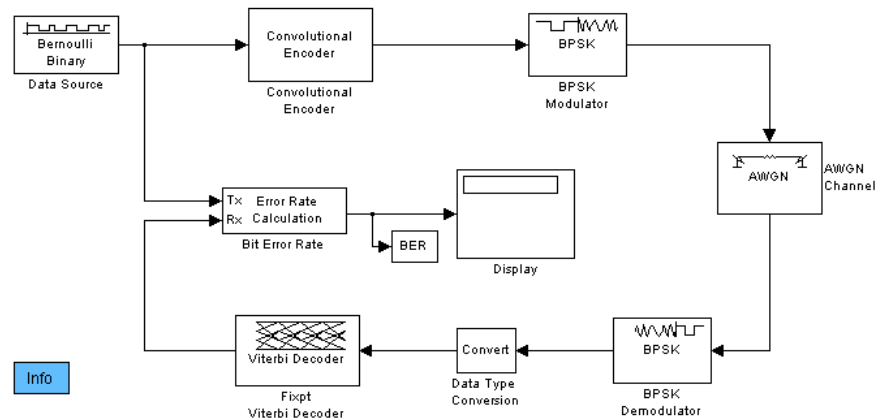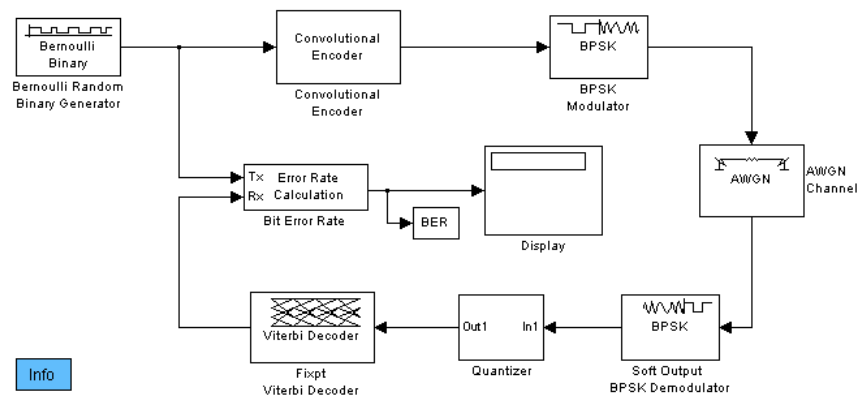
The layout of the soft decision model example is also similar to the existing doc example on Soft-Decision Decoding (click here to open the model, which can be found at `help\toolbox\commblks\examples\doc_softdecision.mdl`).

The purpose of this model is to highlight the fixed-point modeling attributes of the Viterbi decoder, using a familiar layout.

Overview of the Simulations

The two simulations have a similar structure and have most parameters in common. A data source produces a random binary sequence that is convolutionally encoded, BPSK modulated, and passed through an AWGN channel.

For the hard-decision case, the BPSK demodulator produces hard decisions, at the receiver, which are passed onto the decoder.

For the soft-decision case, the BPSK demodulator produces soft decisions, at the receiver, using the log-likelihood ratio. These soft outputs are 3-bit quantized and passed onto the decoder.

After the decoding, the simulation compares the received decoded symbols with the original transmitted symbols in order to compute the

bit error rate. The simulation ends after processing 100 bit errors or 1e6 bits, whichever comes first.

Fixed-Point Modeling

Fixed-point modeling enables bit-true simulations which take into account hardware implementation considerations and the dynamic range of the data/parameters. For example, if the target hardware is a DSP microprocessor, some of the possible word lengths are 8, 16, or 32 bits, whereas if the target hardware is an ASIC or FPGA, there may be more flexibility in the word length selection.

To enable fixed-point Viterbi decoding, the block input must be of type ufix1 (unsigned integer of word length 1) for hard decisions. Based on this input (either a 0 or a 1), the internal branch metrics are calculated using an unsigned integer of word length = (number of output bits), as specified by the trellis structure (which equals 2 for the hard-decision example).

For soft decisions, the block input must be of type ufixN (unsigned integer of word length N), where N is the number of soft-decision bits, to enable fixed-point decoding. The block inputs must be integers in the range 0 to 2^{N-1}. The internal branch metrics are calculated using an unsigned integer of word length = (N + number of output bits - 1), as specified by the trellis structure (which equals 4 for the soft-decision example).

The **State metric word length** is specified by the user and usually must be greater than the branch metric word length already calculated. You can tune this to be the most suitable value (based on hardware and/or data considerations) by reviewing the logged data for the system.

Enable the logging by selecting **Tools > Fixed-Point Settings**. In the Fixed-Point Setting GUI, select the **Logging mode** to `Min, max and overflow`, and rerun the simulation. If you see overflows, it implies the data did not fit in the selected container. You could either increase the size of the word length (if your hardware allows it) or try scaling the data prior to processing it. Based on the minimum and maximum values of the data, you are also able to determine whether the selected container is of the appropriate size.

Try running simulations with different values of **State metric word length** to get an idea of its effect on the algorithm. You should be able to narrow down the parameter to a suitable value that has no adverse effect on the BER results.

Comparisons with Double-Precision Data

To run the same model with double precision data, Select **Tools > Fixed-Point Settings**. In the Fixed-point Settings GUI, select the **Data type override** to be `True doubles`. This selection overrides all data type settings in all the blocks to use double precision. For the Viterbi Decoder block, as **Output type** was set to `Boolean`, this parameter should also be set to double.

Upon simulating the model, note that the double-precision and fixed-point BER results are the same. They are the same because the fixed-point parameters for the model have been selected to avoid any loss of precision while still being most efficient.

Comparisons Between Hard and Soft-Decision Decoding

The two models are set up to run from within BERTool to generate a simulation curve that compares the BER performance for hard-decision versus soft-decision decoding.

To generate simulation results for `doc_fixpt_vitharddec.mdl`, do the following:

1 Type `bertool` at the MATLAB command prompt.

2 Go to the **Monte Carlo** pane.

3 Set the **Eb/No range** to `2:5`.

4 Set the **Simulation model** to `doc_fixpt_vitharddec.mdl`. Make sure that the model is on path.

5 Set the **BER variable name** to `BER`.

6 Set the **Number of errors** to `100`, and the **Number of bits** to `1e6`.

7 Press **Run** and a plot generates.
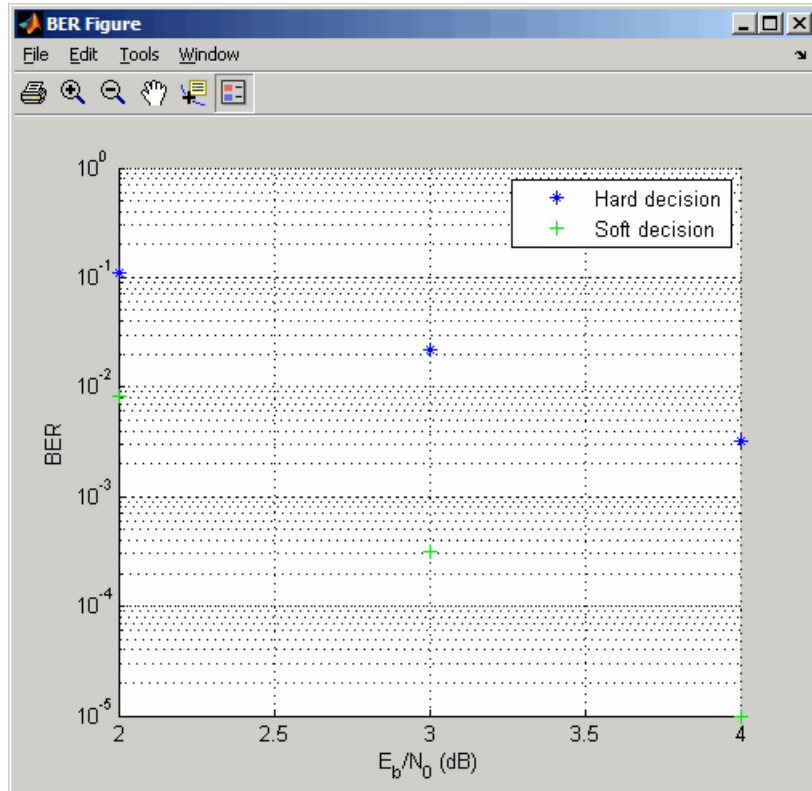
To generate simulation results for doc_fixpt_vitsoftdec.mdl, just change the **Simulation model** in step 4 and press **Run**.

Notice that, as expected, 3-bit soft-decision decoding is better than hard-decision decoding, roughly to the tune of 1.7 dB, and not 2 dB as commonly cited. The difference in the expected results could be attributed to the imperfect quantization of the soft outputs from the demodulator.

Viterbi Decoder
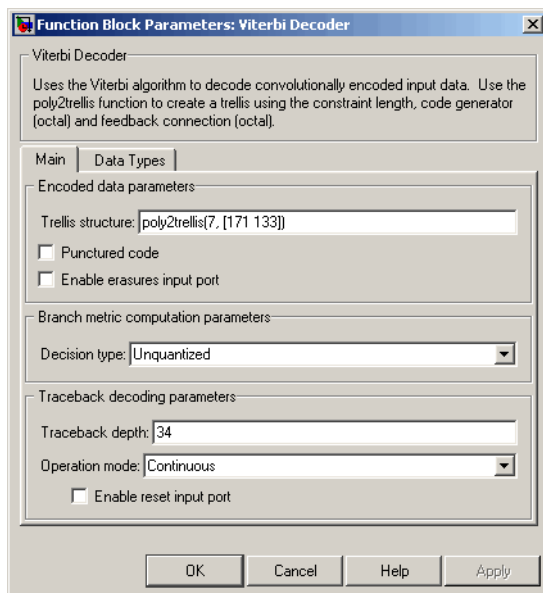
Trellis structure
> MATLAB structure that contains the trellis description of the
> convolutional encoder. Use the same value here and in the
> corresponding Convolutional Encoder block.

Punctured code
> Select this check box to specify a punctured input code. The field,
> **Punctured code**, appears.

Puncture vector
> Constant puncture pattern vector used at the transmitter
> (encoder). The puncture vector is a pattern of 1s and 0s, where
> the 0s indicate the punctured bits. This field appears when the
> check box **Punctured code** is selected.

Enable erasures input port
> When you check this box, the decoder opens an input port labeled
> Era. Through this port, you can specify an erasure vector pattern
> of 1s and 0s, where the 1s indicate the erased bits.

For these erasures in the incoming data stream, the decoder does not update the branch metric. The widths and the sample times of the erasure and the input data ports must be the same. The erasure input port can be of data type `double` or `Boolean`.

Decision type

Unquantized, `Hard Decision`, or `Soft Decision`.

Number of soft decision bits

The number of soft decision bits used to represent each input. This field is active only when **Decision type** is set to `Soft Decision`.

Error if quantized input values are out of range

Check this box to throw an error when quantized input values are out of range. This check box is active only when **Decision type** is set to `Soft Decision` or `Hard Decision`.

Traceback depth

The number of trellis branches used to construct each traceback path.

Operation mode

Method for transitioning between successive input frames. For frame-based input, the choices are `Continuous`, `Terminated`, and `Truncated`. Sample-based input must use the `Continuous` mode.

Enable reset input port

When you check this box, the decoder opens an input port labeled `Rst`. Providing a nonzero input value to this port causes the block to set its internal memory to the initial state before processing the input data.

Viterbi Decoder
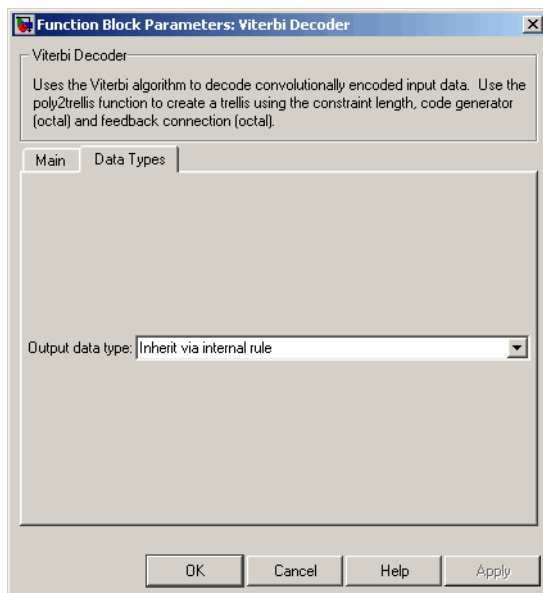
Output data type

> The output signal's data type can be `double`, `single`, `boolean`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or set to `'Inherit via internal rule'` or `'Smallest unsigned integer'`.
>
> When set to `'Smallest unsigned integer'`, the output data type is selected based on the settings used in the **Hardware Implementation** pane of the Configuration Parameters dialog box of the model. If `ASIC/FPGA` is selected in the **Hardware Implementation** pane, the output data type is `ufix(1)`. For all other selections, it is an unsigned integer with the smallest specified wordlength corresponding to the char value (e.g., `uint8`).
>
> When set to `'Inherit via internal rule'` (the default setting), the block selects double-typed outputs for double inputs, single-typed outputs for single inputs, and behaves similarly to

the `'Smallest unsigned integer'` option for all other typed inputs.

See Also   Convolutional Encoder, APP Decoder

References   [1] Clark, G. C. Jr. and J. Bibb Cain., *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

[2] Gitlin, R. D., J. F. Hayes, and S. B. Weinstein, *Data Communications Principles*, New York, Plenum, 1992.

[3] Heller, J. A. and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Transactions on Communication Technology*, Vol. COM-19, October 1971, pp 835–848.

[4] Yasuda, Y., et. al., "High-rate punctured convolutional codes for soft decision Viterbi decoding," *IEEE Transactions on Communications*, Vol. COM-32, No. 3, pp 315–319, March 1984.

[5] Haccoun, D., and Begin, G., "High-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Transactions on Communications*, Vol. 37, No. 11, pp 1113–1125, Nov. 1989.

[6] Begin, G., et.al., "Further results on high-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Transactions on Communications*, Vol. 38, No. 11, pp 1922–1928, Nov. 1990.

Walsh Code Generator

Purpose   Generate Walsh code from orthogonal set of codes

Library   Sequence Generators sublibrary of Comm Sources

Description   Walsh codes are defined as a set of *N* codes, denoted W_j, for $j = 0, 1, \ldots ,$ N - 1, which have the following properties:

Walsh Code
Generator
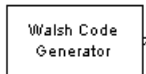
- W_j takes on the values +1 and -1.

- $W_j[0] = 1$ for all j.

- W_j has exactly j zero crossings, for $j = 0, 1, \ldots , N$ - 1.

- $W_j W_k^T = \begin{cases} 0 & j \neq k \\ N & j = k \end{cases}$
- Each code Wj is either even or odd with respect to its midpoint.

Walsh codes are defined using a Hadamard matrix of order *N*. The Walsh Code Generator block outputs a row of the Hadamard matrix specified by the **Walsh code index**, which must be an integer in the range [0, ..., *N* - 1]. If you set **Walsh code index** equal to an integer *j*, the output code has exactly *j* zero crossings, for $j = 0, 1, \ldots , N$ - 1.

Note, however, that the indexing in the Walsh Code Generator block is different than the indexing in the Hadamard Code Generator block. If you set the **Walsh code index** in the Walsh Code Generator block and the **Code index parameter** in the Hadamard Code Generator block, the two blocks output different codes.

Dialog Box
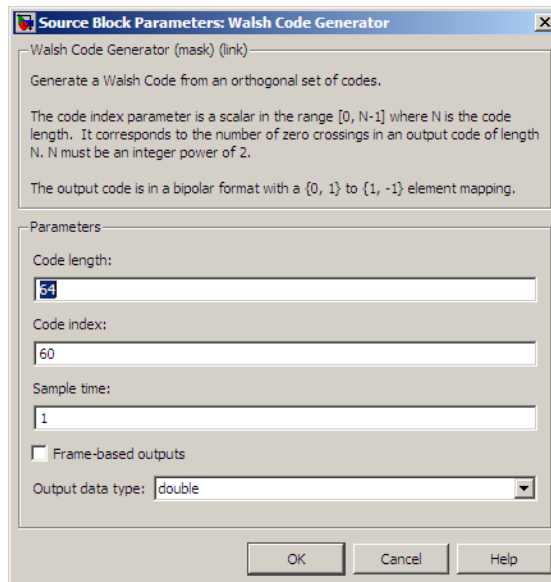
Source Block Parameters: Walsh Code Generator

Walsh Code Generator (mask) (link)

Generate a Walsh Code from an orthogonal set of codes.

The code index parameter is a scalar in the range [0, N-1] where N is the code length. It corresponds to the number of zero crossings in an output code of length N. N must be an integer power of 2.

The output code is in a bipolar format with a {0, 1} to {1, -1} element mapping.

Parameters

Code length:

64

Code index:

60

Sample time:

1

☐ Frame-based outputs

Output data type: double

OK   Cancel   Help

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Code length
Integer scalar that is a power of 2 specifying the length of the output code.

Code index
Integer scalar in the range [0, 1, ... , N - 1], where N is the **Code length**, specifying the number of zero crossings in the output code.

Sample time
A positive real scalar specifying the sample time of the output signal.

Frame-based outputs
When checked, the block outputs a frame-based signal. When cleared, the block outputs a [1] unoriented scalar.

Samples per frame

The number of samples in a frame-based output signal. This field is active only if you select the **Frame-based outputs** check box. If **Samples per frame** is greater than the **Code length**, the code is cyclically repeated.

Output data type

The output type of the block can be specified as an `int8` or `double`. By default, the block sets this to `double`.

See Also   Hadamard Code Generator, OVSF Code Generator

Purpose   Integrate over time window of fixed length

Library   Comm Filters
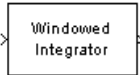
Description The Windowed Integrator block creates cumulative sums of the
input signal values over a sliding time window of fixed length. If the
Integration period parameter is N and the input samples are denoted
by x(1), x(2), x(3),..., then the nth output sample is the sum of the x(k)
values for k between n-N+1 and n. In cases where n-N+1 is less than 1,
the block uses an initial condition of 0 to represent those samples.

The input can be either a scalar or a frame-based matrix. If the input is
frame-based, then the block processes each column independently. The
output has the same sample time and matrix size as the input. double,
single, and fixed-point data types are supported.

Dialog Box
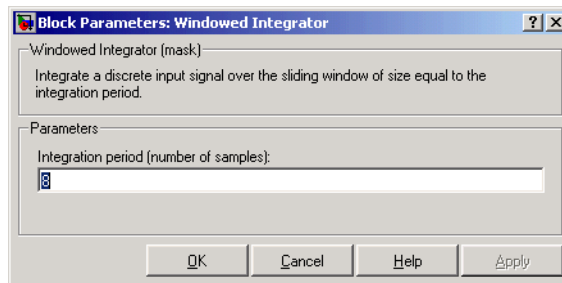
Integration period
The length of the interval of integration, measured in samples.

Rounding mode

Select the rounding mode for fixed-point operations. The block uses the **Rounding mode** when the result of a fixed-point calculation does not map exactly to a number representable by the data type and scaling storing the result. The filter coefficients do not obey this parameter; they always round to Nearest. For more information, see Rounding Modes in the Signal Processing Blockset User's Guide or "Rounding Mode: Simplest" in the Simulink Fixed Point documentation.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator). See "Filter Structure Diagrams" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you are able to enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the coefficients, in bits. If applicable, you are able to enter separate fraction lengths for the numerator and denominator coefficients.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the coefficients. If applicable, you are able to enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.

- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" in *Signal Processing Blockset Reference Guide* for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you are able to enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you are able to enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool.

Examples   If **Integration period** is 3 and the input signal is a ramp (1, 2, 3, 4,...), then some of the sums that form the output of this block are as follows:

- 0+0+1 = 1
- 0+1+2 = 3
- 1+2+3 = 6
- 2+3+4 = 9
- 3+4+5 = 12
- 4+5+6 = 15
- etc.

The zeros in the first few sums represent initial conditions. If the input signal is a sample-based scalar, then the values 1, 3, 6,... are successive values of the scalar output signal. If the input signal is a frame-based column vector, then the values 1, 3, 6,... are organized into output frames that have the same vector length as the input frames.

See Also   Integrate and Dump, Discrete-Time Integrator (Simulink documentation)

Windowed Integrator

Function Reference

comm_links

Purpose   Library link information for Communications Blockset blocks

Syntax
```
comm_links
comm_links(sys)
comm_links(sys,color)
```

Description   comm_links returns a structure with two elements. Each element contains a cell array of strings containing names of library blocks in the current system. The blocks are grouped into two categories: obsolete and current. Blocks at all levels of the model are analyzed.

comm_links(sys) works as above on the named system sys, instead of the current system.

comm_links(sys,*color*) additionally colors all obsolete blocks according to the specified *color*. *color* is one of the following strings: 'blue', 'green', 'red', 'cyan', 'magenta', 'yellow', or 'black'.

Obsolete blocks are blocks that are no longer supported. They might or might not work properly.

Current blocks are supported and represent the latest block functionality.

See Also   liblinks (Signal Processing Blockset documentation), commliblist

Purpose   Open main Communications Blockset library

Syntax   `commlib`

Description   `commlib` opens the latest version of the Communications Blockset library.

See Also   simulink (Simulink documentation), dsplib (Signal Processing Blockset documentation)

commstartup

Purpose   Default Simulink model settings for Communications Blockset software

Syntax   commstartup

Description commstartup changes the default Simulink model settings to values
more appropriate for the simulation of communication systems. The
changes apply to new models that you create later in the MATLAB
session, but not to previously created models.

Note The Signal Processing Blockset application includes a similar
dspstartup script, which assigns different model settings. For modeling
communication systems, you should use commstartup alone.

To install the communications-related model settings each time you
start MATLAB, invoke commstartup from your startup.m file.

To be more specific, the settings in commstartup cause models to:

- Use the variable-step discrete solver in single-tasking mode

- Use starting and ending times of 0 and Inf, respectively

- Avoid producing a warning or error message for inherited sample
  times in source blocks

- Set the Simulink Boolean logic signals parameter to Off

- Avoid saving output or time information to the workspace

- Produce an error upon detecting an algebraic loop

- Inline parameters if you use the Model Reference feature of Simulink

If your communications model does not work well with these default
settings, you can change each of the individual settings as the model
requires.

See Also   startup

Purpose   Generate prime numbers for use as random number seeds

Syntax   out = randseed
                   out = randseed(state)
                   out = randseed(state,m)
                   out = randseed(state,m,n)
                   out = randseed(state,m,n,rmin)
                   out = randseed(state,m,n,rmin,rmax)

Description   The randseed function produces random prime numbers that work
                   well as seeds for random source blocks or noisy channel blocks in
                   Communications Blockset software. It is recommended you use the
                   randseed function when specifying the initial seed parameters of the
                   following blocks: Gaussian, Rayleigh, and Rician Noise Generator.

Note The randseed function uses a local stream of numbers that
is independent from the global stream of numbers in the MATLAB
software. Use of this function does not affect the state of the global
random number stream.

---

out = randseed generates a random prime number between 31 and
2^{17}-1, using the MATLAB function rand.

out = randseed(state) generates a random prime number after
setting the state of rand to the positive integer state. This syntax
produces the same output for a particular value of state.

out = randseed(state,m) generates a column vector of m random
primes.

out = randseed(state,m,n) generates an m-by-n matrix of random
primes.

out = randseed(state,m,n,rmin) generates an m-by-n matrix of
random primes between rmin and 2^{17}-1.

randseed

`out = randseed(state,m,n,rmin,rmax)` generates an `m`-by-`n` matrix of random primes between `rmin` and `rmax`.

Examples

To generate a two-element sample-based row vector of random bits using the Bernoulli Random Binary Generator block, you can set **Probability of a zero** to `[0.1 0.5]` and set **Initial seed** to `randseed(391,1,2)`.

To generate three streams of random data from three different blocks in a single model, you can define `out = randseed(93,3)` in the MATLAB workspace and then set the three blocks' **Initial seed** parameters to `out(1)`, `out(2)`, and `out(3)`, respectively.

See Also   `rand`, `primes`

Purpose   Save Communications Blockset model using blocks of earlier version

Syntax   `flag = saveas_commblks(sys,new_sys,version)`

Description

Note This function is obsolete.

Simulink software provides a similar functionality through its **Save As...** option in the **File** menu of its models.

`saveas_commblks` is designed to replace Communications Blockset R14SP2 (version 3.1) blocks with R14 (version 3.0) or R14SP1 (version 3.0.1) blocks.

`flag = saveas_commblks(sys,new_sys,version)` saves the model `sys` to a previous version specified by `version`, with the name `new_sys`. `version` must be either R14 or R14SP1. `flag` is `true` if the operation is successful, else it is `false`. It displays the list of blocks that are updated in the command window.

`saveas_commblks` updates the following blocks in the model:

- AWGN Channel

- Binary Symmetric Channel

- Multipath Rician Fading Channel

- Voltage-Controlled Oscillator or Continuous-Time VCO

- All blocks from the AM sub-library of the Digital Baseband Modulation library (`commdigbbndam3`). These blocks are replaced by the corresponding blocks from the old AM library (`commdigbbndam2`).

- All blocks from the PM sub-library of the Digital Baseband Modulation library (`commdigbbndpm3`). These blocks are replaced by the corresponding blocks from the old PM library (`commdigbbndpm2`).

- Multipath Rayleigh Fading Channel block from the Channels library (`commchan3`). These blocks are replaced by the Multipath Rayleigh

saveas_commblks

Fading Channel block from the old Channels library (`commchan2`). Note that the default parameters of this block are used after the model is saved.

Example   `success =`
`saveas_commblks('myCommsSystem','myCommsSystemR14sp1','R14SP1')`
saves the system `myCommsSystem` to one using the blocks in the R14SP1 version of Communications Blockset, and returns `true` on successful operation.

Algorithm Used to Decode BCH and Reed-Solomon Codes

Errors-only Decoding

See Errors-only Decoding in the *Communications Toolbox User's Guide.*

Errors and Erasures Decoding

The process for decoding errors and erasures is very similar to errors-only decoding. Because the decoder knows the positions of the erasures, it only needs to determine the error magnitudes in the erased positions.

The errors-and-erasures decoder calculates an *erasure locator polynomial*, $\Gamma(z)$, according to

$$\Gamma(z) = \prod_{k=1}^{e} \left(1 - z\alpha^{j_k}\right)$$

where e is the number of erasures in a codeword, and j_k is the kth erasure in a codeword

The Berlekamp algorithm is then used to calculate a new *errors and erasures locator polynomial*, $\Psi(z)$, which is the convolution of $\Gamma(z)$ and $\Lambda(z)$. The error evaluator polynomial, $\Omega(z)$, is then calculated as the convolution of $\Psi(z)$ and $S(z)$. Finally, the error magnitudes are then calculated as

$$e_{i_m} = \frac{\Omega(\alpha^{-i_m})}{\Psi'(\alpha^{-i_m})}$$

References

[1] Clark, G. C., and Cain, J. B., *Error-Correction Coding for Digital Communications*, New York, Plenum Press, 1981.

[2] Wicker, S. B., *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, N.J., Prentice Hall, 1995.

Index