Communications Blockset

For Use with Simulink®

Modeling

Simulation

Implementation

Reference

Version 2

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	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. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

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

Communications Blockset Reference

Printing History: May 2001 Online only New for Version 2.0.1 (Release 12.1)
July 2002 Online only Revised for Version 2.5 (Release 13)

Contents

1 Function Reference

2 Block Reference

1

Function Reference

Functions — Alphabetical List

Purpose

Display library link information for Communications Blockset blocks

Syntax

```
comm_links
comm_links(sys)
comm_links(sys,mode)
ret_blks = comm_links(...)
```

Description

`comm_links` displays library link information for blocks in the current model that are linked to the Communications Blockset. For each block in the current model, `comm_links` replaces the block name with the full path to the block's library link in the Communications Blockset. Blocks linked to the current Communications Blockset libraries are highlighted in blue. Blocks linked to older versions of the Simulink portion of the Communications Toolbox are highlighted in red. Blocks at all levels of the model are analyzed.

A summary report indicating the number of blocks linked to each blockset version is also displayed in the MATLAB command window. The highlighting and link display are disabled when the model is executed or saved, or when `comm_links` is executed a second time from the MATLAB command line.

`comm_links(sys)` toggles the display of block links in system `sys`. If `sys` is the current model (`gcs`), this is the same as the earlier `comm_links` syntax.

`comm_links(sys,mode)` directly sets the link display state, where *`mode`* can be '**`on`**', '**`off`**', or '**`toggle`**'. The default is '**`toggle`**'.

`ret_blks = comm_links(...)` returns a structure, each field of which is a cell array that lists the full paths to blocks' library links in the Communications Blockset. The different fields refer to different versions of the libraries.

See Also

`liblinks` (DSP Blockset)

commlib

Purpose

Open the main Communications Blockset library

Syntax

```
commlib
commlib(n)
commlib n
```

Description

`commlib` opens the current version of the main Communications Blockset library.

`commlib(n)` opens version number `n` of the main Communications Blockset library, where `n` can be either `1.3`, `1.5`, or `2.5`. Version 2.5 refers to the Release 13 Communications Blockset. Version 1.5 refers to the Simulink portion of the Communications Toolbox 1.5 (Release 11.1). Version 1.3 refers to the Simulink portion of the Communications Toolbox 1.3 (Release 10).

`commlib n` is the same as `commlib(n)`.

See Also

`simulink` (Simulink), `dsplib` (DSP Blockset)

Purpose

Default Simulink model settings for Communications Blockset

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 DSP Blockset 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 **ode45 (Dormand-Prince)** solver, which is a variable-step solver
- Default to a fixed step size and **SingleTasking** mode, in case you change the solver manually to a fixed-step solver
- Use starting and ending times of `0` and `Inf`, respectively
- Avoid saving time or output information to the workspace
- Produce an error upon detecting an algebraic loop
- Inline parameters and use a loop rolling threshold of `5`, in case you use the Real-Time Workshop® to generate code from the model

See Also

`startup`

randseed

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 is designed for producing random prime numbers that work well as seeds for random source blocks or noisy channel blocks in the Communications Blockset.

`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.

`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`

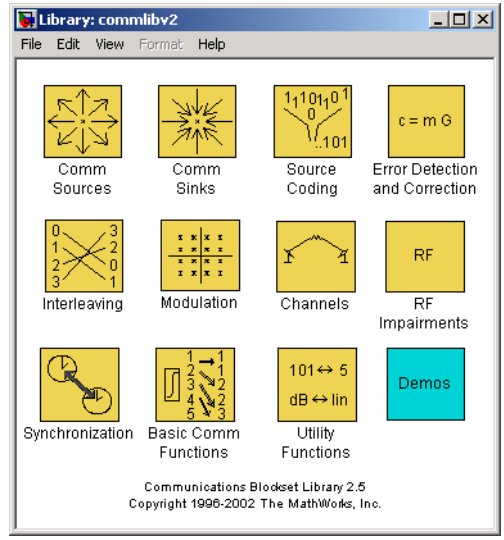
2

Block Reference

Blocks — By Category

This chapter contains detailed descriptions of all Communications Blockset blocks. It first shows the libraries and lists their contents, and then presents the block reference entries in alphabetical order. More detailed discussions of the core libraries' capabilities are in the "Using the Libraries" section.

Below is the main library of the Communications Blockset. You can open it by typing `commlib` at the MATLAB prompt. Each yellow icon in this window represents a library. In Simulink, double-clicking on a library icon opens the library. In this document, clicking on one of the yellow icons below jumps to an overview of that library.



To access an older version of the library (for example, if you are modifying one of your legacy models), then you should use one of these alternative syntaxes of the `commlib` command.

```
commlib 1.3 % To open version 1.3
commlib 1.5 % To open version 1.5
```

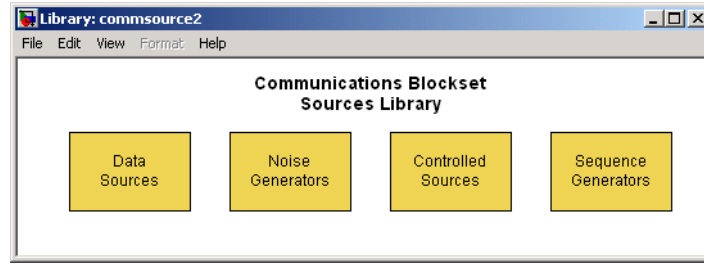
The main library is divided into eleven sublibraries:

- Communications Sources
- Communications Sinks
- Source Coding
- Error Detection and Correction
- Interleaving
- Modulation
- Channels
- RF Impairments
- Synchronization
- Basic Communications Functions
- Utility Functions

Communications Sources

Every communication system contains one or more sources. You can open the Comm Sources library by double-clicking its icon in the main Communications Blockset library (`commlib`), or by typing `commsource2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.

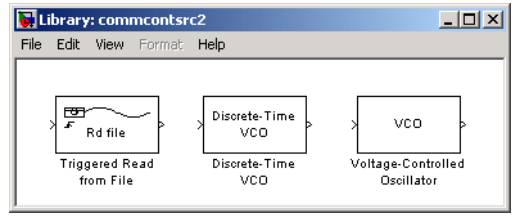


The Comms Sources library contains four sublibraries:

- Controlled Sources, which contains blocks that simulate nonrandom signals by reading from a file or by simulating a voltage-controlled oscillator (VCO).
- Data Sources, which contains blocks that generate random data to simulate signal sources.
- Noise Generators, which contains blocks that generate random data to simulate channel noise.
- Sequence Generators, which contains blocks that generate sequences for spreading or synchronization in a communication system.

Controlled Sources

You can open the Controlled Sources sublibrary by double-clicking on its icon in the Comm Sources library (`commsource2`), or by typing `commcontsrc2` at the MATLAB prompt.

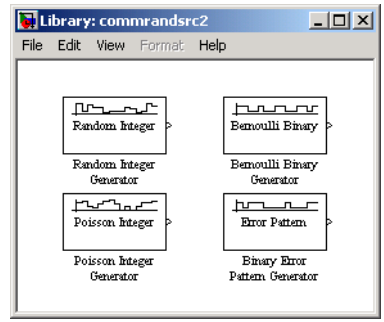


The table below lists and describes the blocks in the Controlled Sources library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Discrete-Time VCO	Implement a voltage-controlled oscillator in discrete time
Triggered Read From File	Read from a file, refreshing the output at rising edges of an input signal
Voltage-Controlled Oscillator	Implement a voltage-controlled oscillator

Data Sources

You can open the Data Sources sublibrary by double-clicking on its icon in the Comm Sources library (`commsource2`), or by typing `commrandsrc2` at the MATLAB prompt.

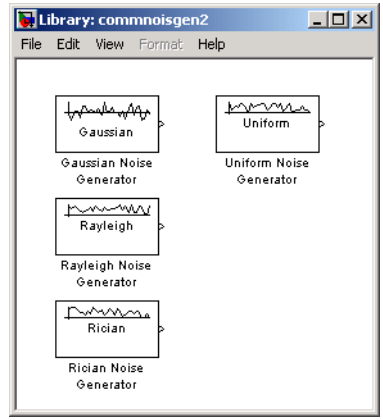


The table below lists and describes the blocks in the Data Sources sublibrary. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Bernoulli Binary Generator	Generate Bernoulli-distributed random binary numbers
Binary Error Pattern Generator	Generate a binary vector while controlling the number of 1s
Poisson Integer Generator	Generate Poisson-distributed random integers
Random Integer Generator	Generate integers randomly distributed in the range [0, M-1]

Noise Generators

You can open the Noise Generators sublibrary by double-clicking on its icon in the Comm Sources library (`commsource2`), or by typing `commnoisgen2` at the MATLAB prompt.

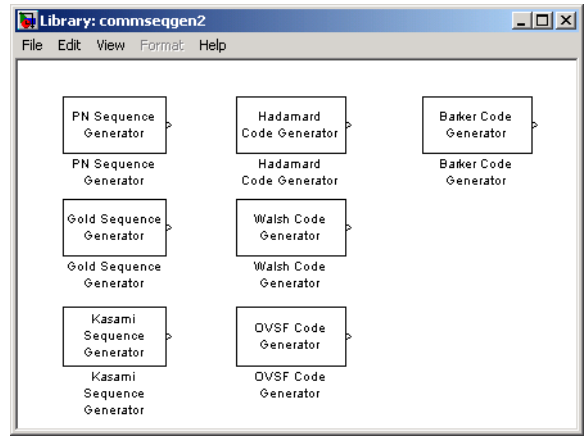


The table below lists and describes the blocks in the Noise Generators sublibrary. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
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 the upper and lower bounds

Sequence Generators

You can open the Sequence Generators sublibrary by double-clicking on its icon in Comm Sources library (`commsource2`), or by typing `commseqgen2` at the MATLAB prompt.



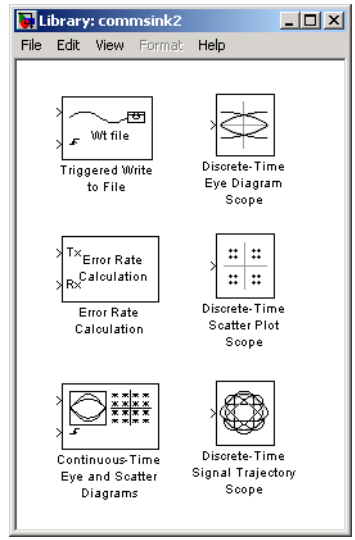
The table below lists and describes the blocks in the Sequence Generators sublibrary. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Barker Code Generator	Generate a Barker Code
Gold Sequence Generator	Generate a Gold sequence from a set of sequences
Kasami Sequence Generator	Generate a Kasami sequence from the set of Kasami sequences
Hadamard Code Generator	Generate a Hadamard code from an orthogonal set of codes
OVSF Code Generator	Generate an orthogonal variable spreading factor (OVSF) code from a set of orthogonal codes
PN Sequence Generator	Generate a pseudonoise sequence
Walsh Code Generator	Generate a Walsh code from an orthogonal set of codes

Communications Sinks

The Comm Sinks library provides sinks and display devices that facilitate analysis of communication system performance. You can open the Comm Sinks library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commsink2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Comm Sinks library. For information about a specific block, see the reference pages that follow.

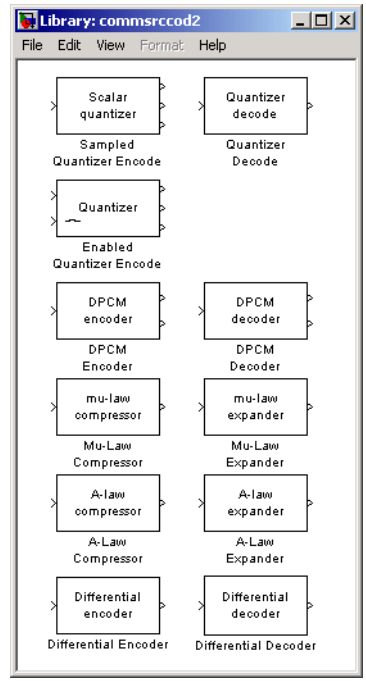
Block Name	Purpose
Error Rate Calculation	Compute the bit error rate or symbol error rate of input data
Continuous-Time Eye and Scatter Diagrams	Produce eye diagram, scatter, or x-y plots, using trigger to set decision timing

Block Name (Continued)	Purpose (Continued)
Discrete-Time Eye Diagram Scope	Display multiple traces of a modulated signal
Discrete-Time Scatter Plot Scope	Display a modulated signal in its signal space by plotting its in-phase component versus its quadrature component
Discrete-Time Signal Trajectory Scope	Display a modulated signal in its signal space by plotting its in-phase component versus its quadrature component
Triggered Write to File	Write to a file at each rising edge of an input signal

Source Coding

This blockset supports companders, scalar quantization and predictive quantization. You can open the Source Coding library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commsrccod2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Source Coding library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
A-Law Compressor	Implement A-law compressor for source coding
A-Law Expander	Implement A-law expander for source coding
Differential Decoder	Decode a binary signal using differential coding technique.
Differential Encoder	Encode a binary signal using differential coding technique.
DPCM Decoder	Decode differential pulse code modulation
DPCM Encoder	Encode using differential pulse code modulation

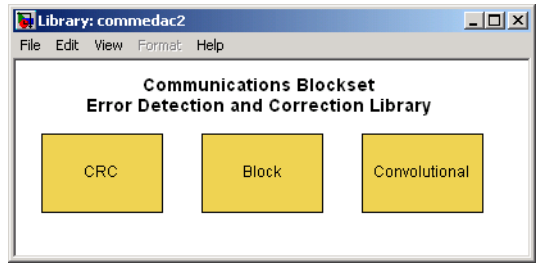
Block Name (Continued)	Purpose (Continued)
Mu-Law Compressor	Implement m-law compressor for source coding
Mu-Law Expander	Implement m-law expander for source coding
Quantizer Decode	Decode quantization index according to codebook
Sampled Quantizer Encode	Quantize a signal, indicating quantization index, coded signal, and distortion
Enabled Quantizer Encode	Quantize a signal, using trigger to control processing

Error Detection and Correction

The Error Detection and Correction library contains three sublibraries:

- Block, which contains blocks that implement the encoding and decoding of linear, cyclic, BCH, Hamming, and Reed-Solomon codes
- Convolutional, which contains blocks that implement convolutional encoding and decoding
- CRC, which contains blocks that append cyclic redundancy check (CRC) bits to data, and detect errors

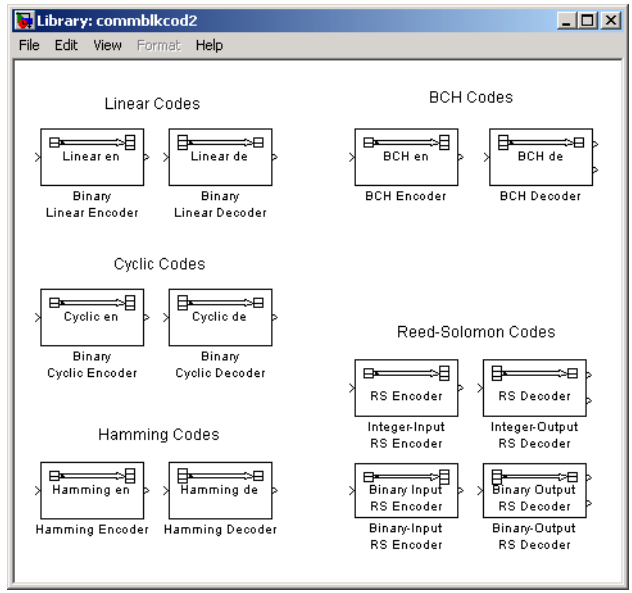
The main Error Detection and Correction library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commedac2` at the MATLAB prompt. Each icon in the Error Detection and Correction window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Block Coding

You can open the Block sublibrary by double-clicking on the Block icon in the main Error Detection and Correction library, or by typing `commblkcod2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



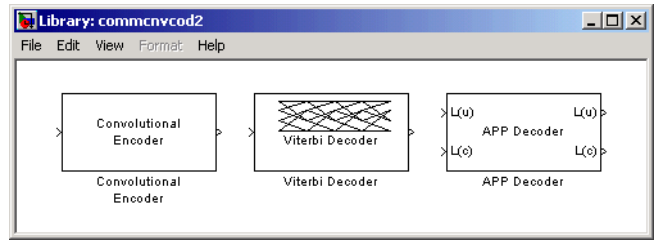
The table below lists and describes the blocks in the Block sublibrary of the Error Detection and Correction library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
BCH Decoder	Decode a BCH code to recover binary vector data
BCH Encoder	Create a BCH code from binary vector data
Binary Cyclic Decoder	Decode a systematic cyclic code to recover binary vector data
Binary Cyclic Encoder	Create a systematic cyclic code from binary vector data
Binary-Output RS Decoder	Decode a Reed-Solomon code to recover binary vector data
Binary-Input RS Encoder	Create a Reed-Solomon code from binary vector data
Binary Linear Decoder	Decode a linear block code to recover binary vector data
Binary Linear Encoder	Create a linear block code from binary vector data
Hamming Decoder	Decode a Hamming code to recover binary vector data
Hamming Encoder	Create a Hamming code from binary vector data
Integer-Output RS Decoder	Decode a Reed-Solomon code to recover integer vector data
Integer-Input RS Encoder	Create a Reed-Solomon code from integer vector data

Convolutional Coding

You can open the Convolutional sublibrary by double-clicking on the Convolutional icon in the main Error Detection and Correction library, or by typing `commcnvcod2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



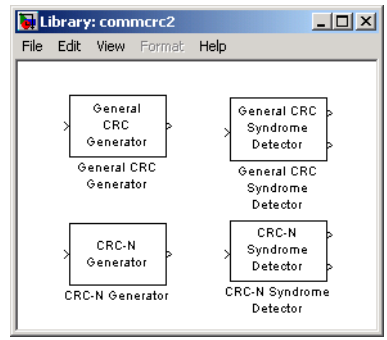
The table below lists and describes the blocks in the Convolutional sublibrary of the Error Detection and Correction library. For information about a specific block, see the reference pages that follow.

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

Cyclic Redundancy Check Coding

You can open the CRC sublibrary by double-clicking on the CRC icon in the main Error Detection and Correction library, or by typing `commcrc2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the CRC sublibrary of the Error Detection and Correction library. For information about a specific block, see the reference pages that follow.

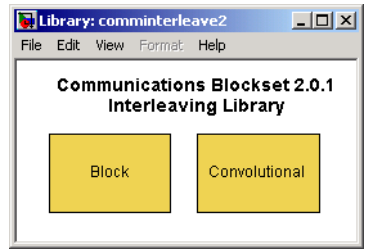
Block Name	Purpose
CRC-N Generator	Generate CRC bits according to the selected CRC method and append them to input data
CRC-N Syndrome Detector	Detect errors in the input data according to the specified CRC method
General CRC Generator	Generate CRC bits according to the generator polynomial and append them to input data
General CRC Syndrome Detector	Detect errors in the input data according to the generator polynomial

Interleaving

The Interleaving library contains two sublibraries:

- Block
- Convolutional

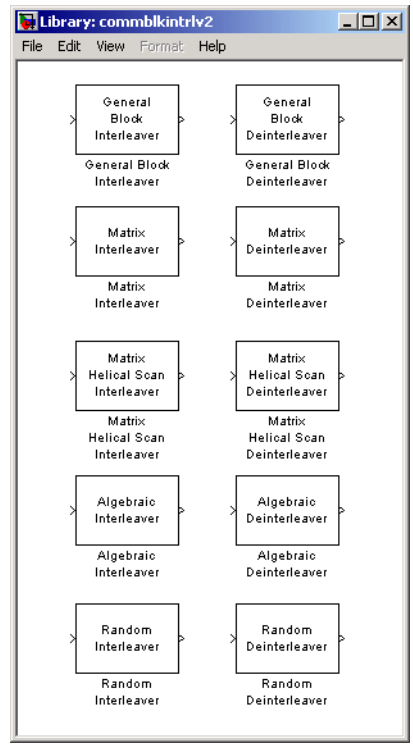
The main Interleaving library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `comminterleave2` at the MATLAB prompt. Each icon in the Interleaving window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Block Interleaving

You can open the Block sublibrary by double-clicking on the Block icon in the main Interleaving library, or by typing `commblkintrlv2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Block sublibrary of the Interleaving library. For information about a specific block, see the reference pages that follow.

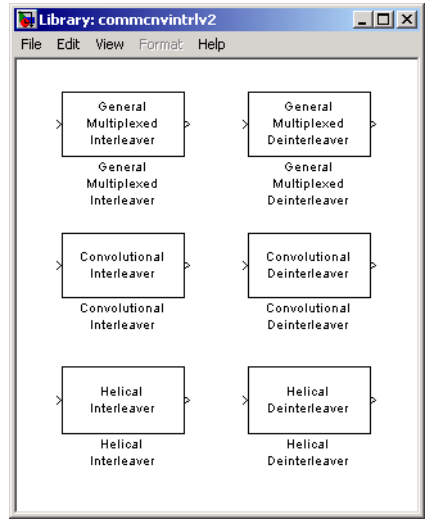
Block Name	Purpose
Algebraic Deinterleaver	Restore ordering of the input symbols using algebraically derived permutation
Algebraic Interleaver	Reorder the input symbols using algebraically derived permutation table
General Block Deinterleaver	Restore ordering of the symbols in the input vector
General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

Block Name (Continued)	Purpose (Continued)
Matrix Deinterleaver	Permute input symbols by filling a matrix by columns and emptying it by rows
Matrix Helical Scan Deinterleaver	Restore ordering of input symbols by filling a matrix along diagonals
Matrix Helical Scan Interleaver	Permute input symbols by selecting matrix elements along diagonals
Matrix Interleaver	Permute input symbols by filling a matrix by rows and emptying it by columns
Random Deinterleaver	Restore ordering of the input symbols using a random permutation
Random Interleaver	Reorder the input symbols using a random permutation

Convolutional Interleaving

You can open the Convolutional sublibrary by double-clicking on the Convolutional icon in the main Interleaving library, or by typing `commcnvintrlv2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Convolutional sublibrary of the Interleaving library. For information about a specific block, see the reference pages that follow.

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

Modulation

The Modulation library contains four sublibraries, each of which addresses a category of modulation:

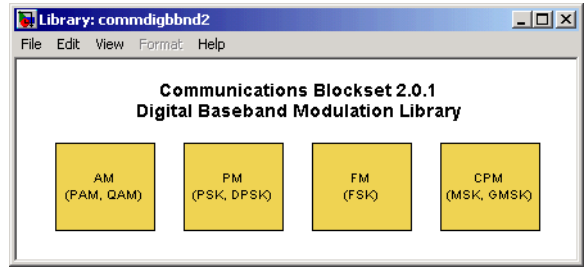
- Digital Baseband Modulation
- Analog Baseband Modulation
- Digital Passband Modulation
- Analog Passband Modulation

The main Modulation library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commmod2` at the MATLAB prompt. Each icon in the Modulation window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.

The first column shows the sublibraries for baseband simulation; the second column shows the sublibraries for passband simulation. The first row shows the sublibraries for digital modulation and demodulation. The second row shows the sublibraries for analog modulation and demodulation.

Digital Baseband Modulation

You can open the Digital Baseband sublibrary of Modulation by double-clicking on the Digital Baseband icon in the main Modulation library, or by typing `commdigbbnd2` at the MATLAB prompt. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.

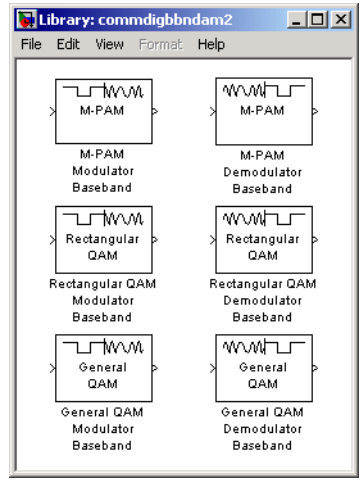


Digital Baseband is further divided into sublibraries according to specific modulation techniques:

- Amplitude modulation (PAM, QAM)
- Phase modulation (PSK, DPSK)
- Frequency modulation (FSK)
- Continuous phase modulation (MSK, GMSK)

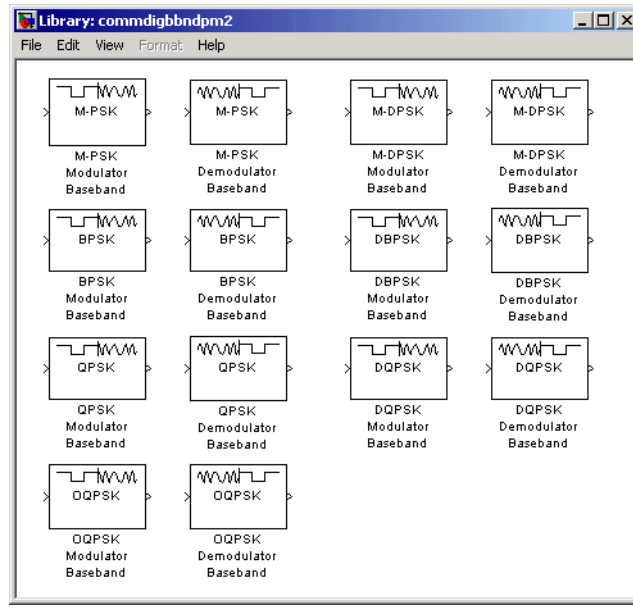
The figures and tables below show and list the blocks in the method-specific sublibraries. For information about a specific block, see the reference pages that follow.

AM Sublibrary



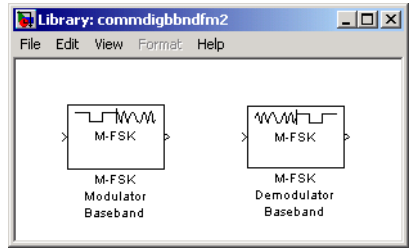
Block Name	Purpose
General QAM Demodulator Baseband	Demodulate QAM-modulated data
General QAM Modulator Baseband	Modulate using quadrature amplitude modulation
M-PAM Demodulator Baseband	Demodulate PAM-modulated data
M-PAM Modulator Baseband	Modulate using M-ary pulse amplitude modulation
Rectangular QAM Demodulator Baseband	Demodulate QAM-modulated data
Rectangular QAM Modulator Baseband	Modulate using M-ary quadrature amplitude modulation

PM Sublibrary



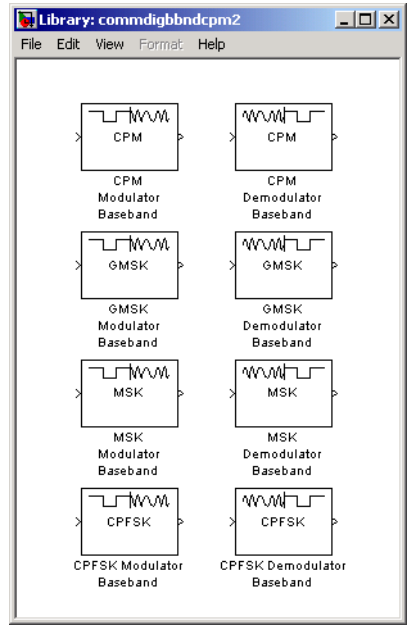
Block Name	Purpose
BPSK Demodulator Baseband	Demodulate BPSK-modulated data
BPSK Modulator Baseband	Modulate using the binary phase shift keying method
DBPSK Demodulator Baseband	Demodulate DBPSK-modulated data
DBPSK Modulator Baseband	Modulate using the differential binary phase shift keying method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

FM Sublibrary



Block Name	Purpose
M-FSK Demodulator Baseband	Demodulate FSK-modulated data
M-FSK Modulator Baseband	Modulate using the M-ary frequency shift keying method

CPM Sublibrary



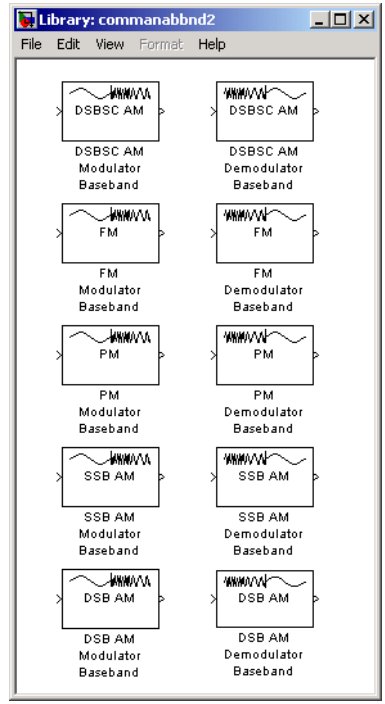
Block Name	Purpose
CPFSK Demodulator Baseband	Demodulate CPFSK-modulated data
CPFSK Modulator Baseband	Modulate using the continuous phase frequency shift keying method
CPM Demodulator Baseband	Demodulate CPM-modulated data
CPM Modulator Baseband	Modulate using continuous phase modulation
GMSK Demodulator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

Block Name (Continued)	Purpose (Continued)
MSK Demodulator Baseband	Demodulate MSK-modulated data
MSK Modulator Baseband	Modulate using the minimum shift keying method

Analog Baseband Modulation

You can open the Analog Baseband sublibrary of Modulation by double-clicking on the Analog Baseband icon in the main Modulation library, or by typing `commanabbnd2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.

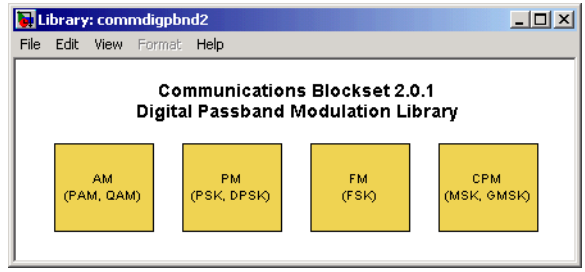


The table below lists and describes the blocks in the Analog Baseband sublibrary of the Modulation library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
DSB AM Demodulator Baseband	Demodulate DSB-AM-modulated data
DSB AM Modulator Baseband	Modulate using double-sideband amplitude modulation
DSBSC AM Demodulator Baseband	Demodulate DSBSC-AM-modulated data
DSBSC AM Modulator Baseband	Modulate using double-sideband suppressed-carrier amplitude modulation
FM Demodulator Baseband	Demodulate FM-modulated data
FM Modulator Baseband	Modulate using frequency modulation
PM Demodulator Baseband	Demodulate PM-modulated data
PM Modulator Baseband	Modulate using phase modulation
SSB AM Demodulator Baseband	Demodulate SSB-AM-modulated data
SSB AM Modulator Baseband	Modulate using single-sideband amplitude modulation

Digital Passband Modulation

You can open the Digital Passband sublibrary of Modulation by double-clicking on the Digital Passband icon in the main Modulation library, or by typing `commdigpbnd2` at the MATLAB prompt. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.

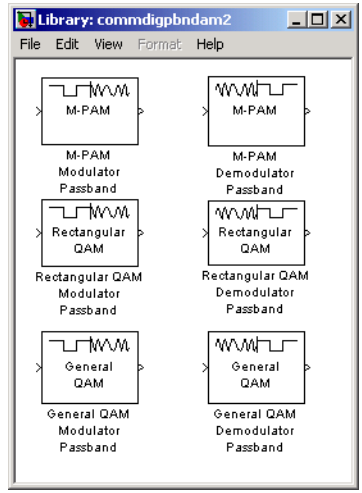


Digital Passband is further divided into sublibraries according to specific modulation techniques:

- Amplitude modulation (PAM, QAM)
- Phase modulation (PSK, DPSK)
- Frequency modulation (FSK)
- Continuous phase modulation (MSK, GMSK)

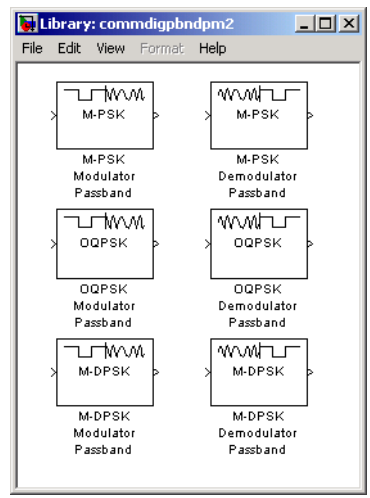
The figures and tables below show and list the blocks in the method-specific sublibraries. For information about a specific block, see the reference pages that follow.

AM Sublibrary



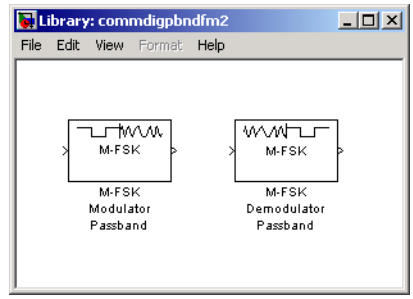
Block Name	Purpose
General QAM Demodulator Passband	Demodulate QAM-modulated data
General QAM Modulator Passband	Modulate using the pulse amplitude modulation phase shift keying method
M-PAM Demodulator Passband	Demodulate PAM-modulated data
M-PAM Modulator Passband	Modulate using M-ary pulse amplitude modulation
Rectangular QAM Demodulator Passband	Demodulate QAM-modulated data
Rectangular QAM Modulator Passband	Modulate using M-ary quadrature amplitude modulation

PM Sublibrary



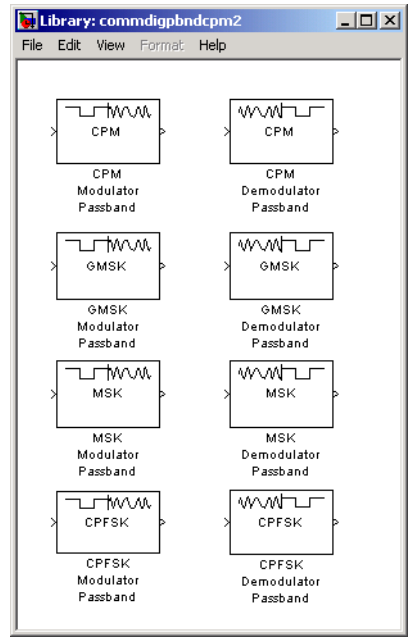
Block Name	Purpose
M-DPSK Demodulator Passband	Demodulate DPSK-modulated data
M-DPSK Modulator Passband	Modulate using the M-ary differential phase shift keying method
M-PSK Demodulator Passband	Demodulate PSK-modulated data
M-PSK Modulator Passband	Modulate using the M-ary phase shift keying method
OQPSK Demodulator Passband	Demodulate OQPSK-modulated data
OQPSK Modulator Passband	Modulate using the offset quadrature phase shift keying method

FM Sublibrary



Block Name	Purpose
M-FSK Demodulator Passband	Modulate using the M-ary frequency shift keying method
M-FSK Modulator Passband	Modulate using the M-ary frequency shift keying method

CPM Sublibrary



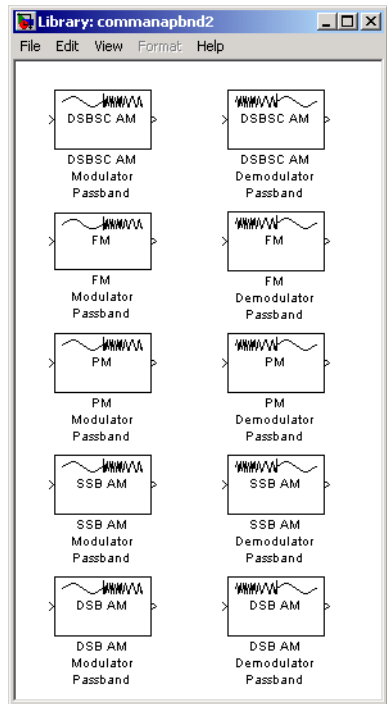
Block Name	Purpose
CPFSK Demodulator Passband	Demodulate CPFSK-modulated data
CPFSK Modulator Passband	Modulate using the continuous phase frequency shift keying method
CPM Demodulator Passband	Demodulate CPM-modulated data
CPM Modulator Passband	Modulate using continuous phase modulation
GMSK Demodulator Passband	Demodulate GMSK-modulated data
GMSK Modulator Passband	Modulate using the Gaussian minimum shift keying method

Block Name (Continued)	Purpose (Continued)
MSK Demodulator Passband	Demodulate MSK-modulated data
MSK Modulator Passband	Modulate using the minimum shift keying method

Analog Passband Modulation

You can open the Analog Passband sublibrary of Modulation by double-clicking on the Analog Passband icon in the main Modulation library, or by typing `commanapbnd2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



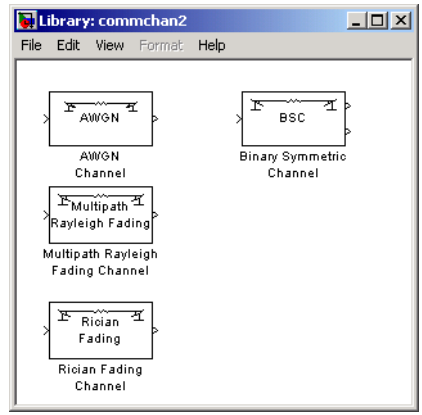
The table below lists and describes the blocks in the Analog Passband sublibrary of the Modulation library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
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

Channels

The Channels library provides passband and baseband channels. You can open the Channels library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commchan2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



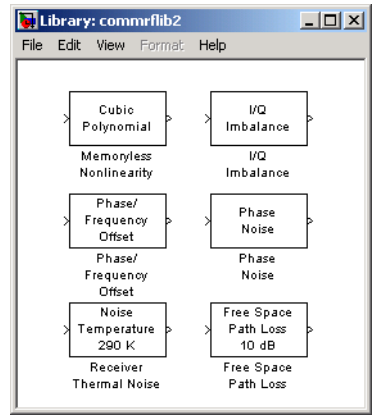
The table below lists and describes the blocks in the Channels library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
AWGN Channel	Add white Gaussian noise to the input signal
Binary Symmetric Channel	Introduce binary errors
Multipath Rayleigh Fading Channel	Simulate a multipath Rayleigh fading propagation channel
Rician Fading Channel	Simulate a Rician fading propagation channel

RF Impairments

The RF Impairments library provides blocks that simulate radio frequency (RF) impairments at the receiver. You can open the RF Impairments library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commrflib2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the RF Impairments library. For information about a specific block, see the reference pages that follow.

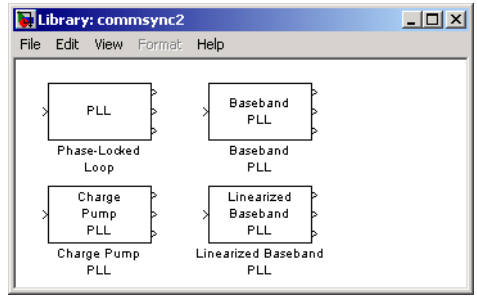
Block Name	Purpose
Free Space Path Loss	Reduce the amplitude of the input signal by the amount specified
I/Q Imbalance	Create a complex baseband model of the signal impairments caused by imbalances between in-phase and quadrature receiver components
Memoryless Nonlinearity	Apply a memoryless nonlinearity to a complex baseband signal.

Block Name (Continued)	Purpose (Continued)
Phase/Frequency Offset	Apply phase and frequency offsets to a complex baseband signal.
Phase Noise	Apply receiver phase noise to a complex baseband signal
Receiver Thermal Noise	Apply receiver thermal noise to a complex baseband signal

Synchronization

The Synchronization library provides four phase-locked loop models. You can open the Synchronization library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commsync2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Synchronization library. For information about a specific block, see the reference pages that follow.

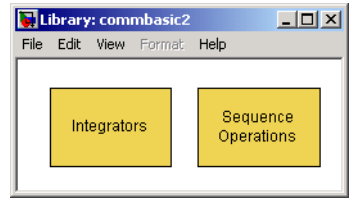
Block Name	Purpose
Baseband PLL	Implement a baseband phase-locked loop
Charge Pump PLL	Implement a charge pump phase-locked loop using a digital phase detector
Linearized Baseband PLL	Implement a linearized version of a baseband phase-locked loop
Phase-Locked Loop	Implement a phase-locked loop to recover the phase of the input signal

Basic Communications Functions

The Basic Comm Functions library contains these sublibraries:

- Integrators
- Sequence Operations

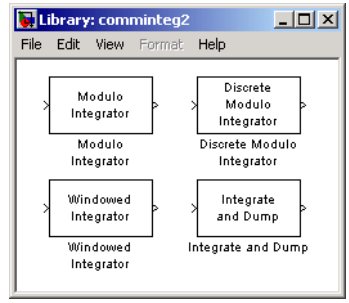
The main Basic Comm Functions library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commbasic2` at the MATLAB prompt. Each icon in the Basic Comm Functions window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Integrators

You can open the Integrators sublibrary by double-clicking on the Integrators icon in the main Basic Comm Functions library, or by typing `comminteg2` at the MATLAB prompt.

Note In this document, you can jump to a block's reference page by clicking on its icon below.



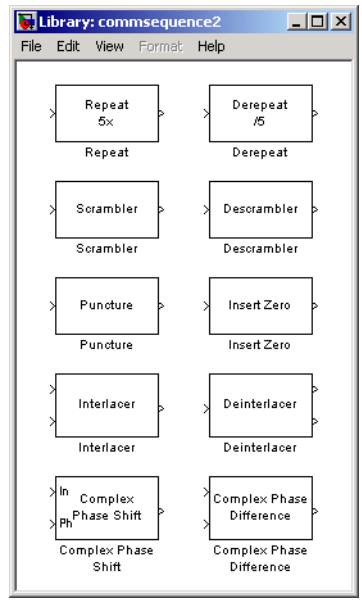
The table below lists and describes the blocks in the Integrators library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Discrete Modulo Integrator	Integrate in discrete time and reduce by a modulus
Integrate and Dump	Integrate, resetting to zero periodically and reducing by a modulus
Modulo Integrator	Integrate in continuous time and reduce by a modulus
Windowed Integrator	Integrate over a time window of fixed length

Sequence Operations

You can open the Sequence Operations sublibrary by double-clicking on the Sequence Operations icon in the main Basic Comm Functions library, or by typing `commsequence2` at the MATLAB prompt.

Note In this document, you can jump to a Communications Blockset block's reference page by clicking on its icon below. (The Repeat block is in the DSP Blockset, not the Communications Blockset; the icon in the Sequence Operations sublibrary is merely a link to the DSP Blockset.)



The table below lists and describes the blocks in the Sequence Operations library. For information about a specific block, see the reference pages that follow.

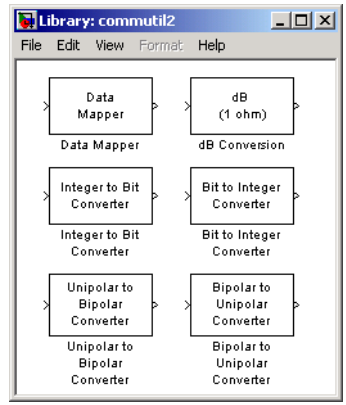
Block Name	Purpose
Complex Phase Difference	Output the phase difference between the two complex input signals
Complex Phase Shift	Shift the phase of the complex input signal by the second input value
Deinterlacer	Distribute elements of input vector alternately between two output vectors

Block Name (Continued)	Purpose (Continued)
Derepeat	Reduce sampling rate by averaging consecutive samples
Descrambler	Descramble the input signal
Insert Zero	Distribute input elements in output vector
Interlacer	Alternately select elements from two input vectors to generate output vector
Puncture	Output the elements which correspond to 1s in the binary Puncture vector
Repeat	Resample an input at a higher rate by repeating values
Scrambler	Scramble the input signal

Utility Functions

You can open the Utility Functions library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commutil2` at the MATLAB prompt.

Note In this document, you can jump to a Communications Blockset block's reference page by clicking on its icon below. (The dB Conversion block is in the DSP Blockset, not the Communications Blockset; the icon in the Utility Functions library is merely a link to the DSP Blockset.)



The table below lists and describes the blocks in the Utility Functions library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Bipolar to Unipolar Converter	Map a vector of bits to a corresponding vector of integers
Bit to Integer Converter	Map a vector of bits to a corresponding vector of integers
dB Conversion	Convert magnitude data to decibels (dB or dBm)
Data Mapper	Map integer symbols from one coding scheme to another

Block Name (Continued)	Purpose (Continued)
Integer to Bit Converter	Map a vector of integers to a vector of bits
Unipolar to Bipolar Converter	Map a unipolar signal in the range [0, M-1] into a 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

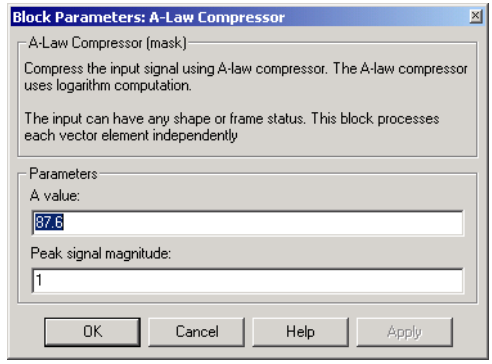
$$y = \begin{cases} \dfrac{A|x|}{1+\log A}\operatorname{sgn}(x) & \text{for } 0 \le |x| \le \dfrac{V}{A} \\ \dfrac{V(1+\log(A|x|/V))}{1+\log A}\operatorname{sgn}(x) & \text{for } \dfrac{V}{A} < |x| \le V \end{cases}$$

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).

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

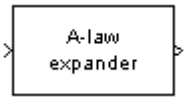
Purpose

Implement A-law expander for source coding

Library

Source Coding

Description

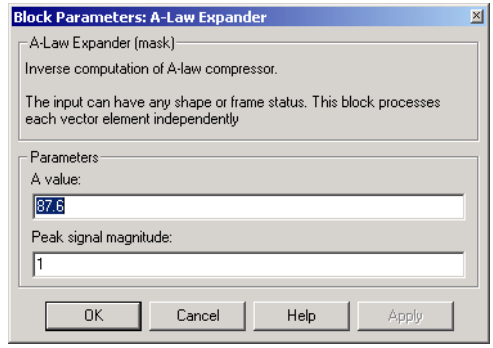


The A-Law Expander block recovers data that the A-Law Compressor block compressed. The formula for the A-law expander, shown below, is the inverse of the compressor function.

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

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 the input symbols using algebraically derived permutation

Library

Block sublibrary of Interleaving

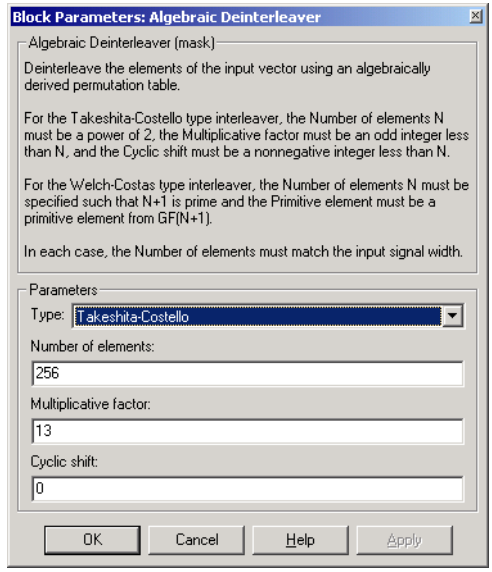
Description

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 **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**.

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.

Algebraic Interleaver

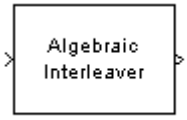
Purpose

Reorder the 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 **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 $\mathrm{mod}(A^k,N+1) - 1$.

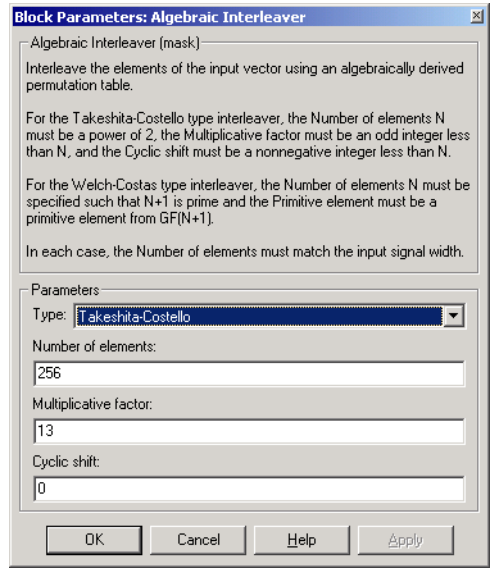
- If **Type** is set to **Takeshita-Costello**, then N must be 2^m for some integer m. The **Multiplicative factor** parameter, h, must be an odd integer less than N. The **Cyclic shift** parameter, k, must be a nonnegative integer less than N.

  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.

Purpose

Decode a convolutional code using the 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. You can use this block to build a turbo decoder.

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

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 the section, "Trellis Description of a Convolutional Encoder," in the *Communications Toolbox User's Guide*. You can use this parameter field in two ways:

- 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 preferable because it causes Simulink to spend less time updating the

diagram at the beginning of each simulation, compared to the usage in the next bulleted item.

- 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)
  ```

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, while 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 **Reset** parameter set to **On each frame**, then use the **Truncated** option in this block.

Specifying Details of the Algorithm

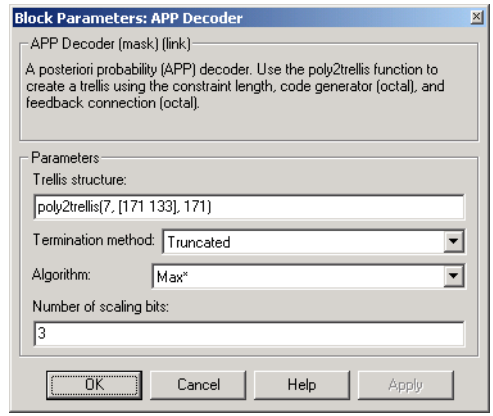
You can control part of the decoding algorithm using the **Algorithm** parameter. The **True APP** option implements a posteriori probability. To gain speed, both the **Max*** and **Max** options approximate expressions like

$$\log \sum_i \exp a_i$$

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.

The **Max*** option enables the **Scaling bits** parameter in the mask. This parameter is the number of bits by which the block scales the data it processes internally. You can use this parameter to avoid losing precision during the computations. It is especially appropriate if your implementation uses fixed-point components. For more information about the **Max*** option, see the article by Viterbi in the “References” section below.

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` (Communications Toolbox)

References

[1] Benedetto, Sergio and Guido Montorsi. “Performance of Continuous and Blockwise Decoded Turbo Codes.” *IEEE Communications Letters*, vol. 1, May 1997. 77-79.

[2] 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 TMO Progress Report*, vol. 42-127, November 1996. [This electronic journal is available at `http://tmo.jpl.nasa.gov/tmo/progress_report/index.html`.]

[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.

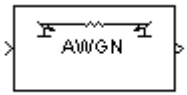
Purpose

Add white Gaussian noise to the 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 DSP Blockset's Random Source block to generate the noise. 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 *DSP Blockset User's Guide*.

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 four modes:

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

  - **Es/No**, the ratio of signal energy to noise power spectral density
  - **Input signal power**, the power of the input symbols
  - **Symbol period**
- **Signal to noise ratio (SNR)**, where the block calculates the variance from these quantities that you specify in the block mask:
  - **SNR**, the ratio of signal power to noise power
  - **Input signal power**, the power of the input samples
- **Variance from mask**, where you specify the variance in the block mask. 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.

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.

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 Between Es/No and SNR Modes

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

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

where

- E_s = Signal energy (Joules)
- N_0 = Noise power spectral density (Watts/Hz)
- T_{sym} is the **Symbol period** of the block in **Es/No** mode (s)
- T_{samp} is the inherited **Sample time** of the block (s)

You can derive this relationship as follows:

$$\begin{aligned} E_s/N_0 &= (S \cdot T_{sym})/(N/B_n) \\ &= (S/N) \cdot (T_{sym} \cdot F_s) \\ &= SNR \cdot (T_{sym}/T_{samp}) \end{aligned}$$

where

- S = **Input signal power (watts)**
- N = Noise power (Watts)
- B_n = Noise bandwidth (Hz)
- F_s = Sampling frequency (Hz)

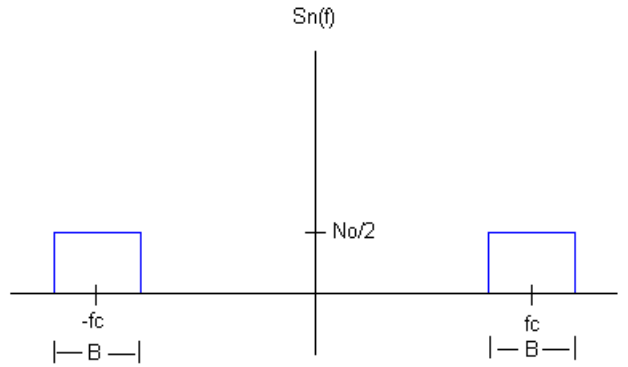
Note that $B_n = F_s = 1/T_{samp}$. The quantity E_s/N_0 is the signal-to-noise ratio with the noise measured in a symbol rate bandwidth. The quantity S/N is measured in a sample rate bandwidth.

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

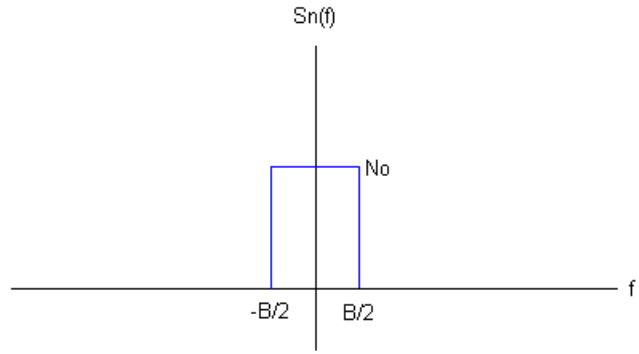
$$E_s/N_0 = 2 \cdot SNR \cdot (T_{sym}/T_{samp})$$

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.

The following figures illustrate the difference between the real and complex cases by showing the noise power spectral densities Sn(f) of a real bandpass white noise process and its complex lowpass equivalent.

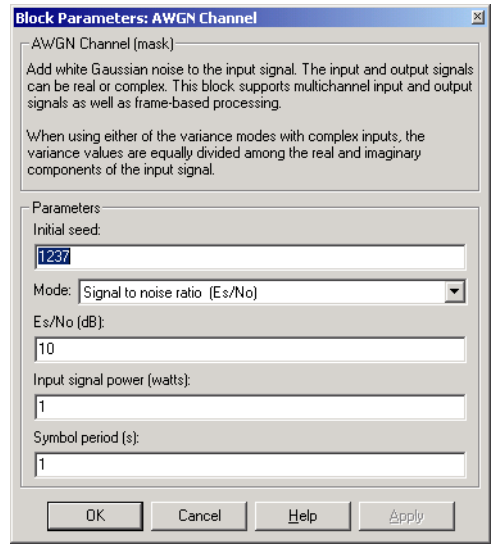


Real Bandpass Noise Power Spectral Density



Complex Lowpass Noise Power Spectral Density

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 (Es/No)**, **Signal to noise ratio (SNR)**, **Variance from mask**, or **Variance from port**.

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**.

Input signal power (watts)

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

Symbol period (s)

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

Variance

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

See Also

Random Source (DSP Blockset)

Reference

[1] Proakis, John G., *Digital Communications*, 4th Ed., McGraw-Hill, 2001.

Purpose

Generate a 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 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

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

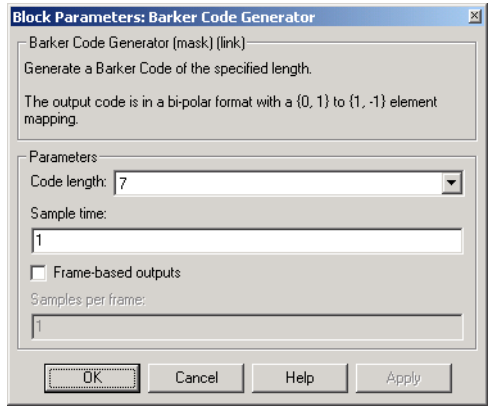
where X_j is an individual code symbol taking values +1 or -1, for $1 \le i \le N$, and the adjacent symbols are assumed to be zero.

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]`

Barker Code Generator

Dialog Box



Code length

The length of the Barker code.

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.

See Also

PN Sequence Generator

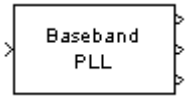
Purpose

Implement a baseband phase-locked loop

Library

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 the Phase-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** mask 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 the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

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

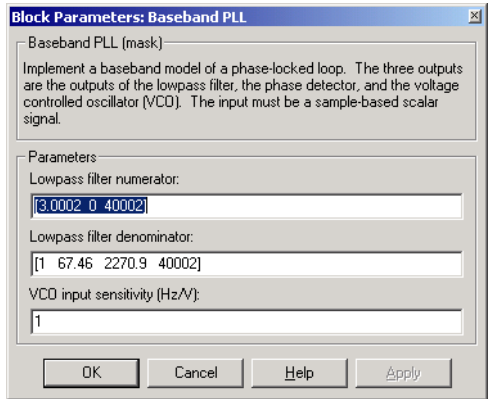
- A voltage-controlled oscillator (VCO). You specify the sensitivity of the VCO signal to its input using the **VCO input sensitivity** parameter. This parameter, measured in Hertz per volt, is a scale factor that determines how much the VCO shifts from its quiescent frequency.

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 the Linearized 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 Using the Communications Blockset.

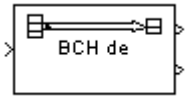
Purpose

Decode a 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.

The input is the binary codeword vector and the first output is the corresponding binary message vector. If the BCH code has message length K and codeword length N, then the input has length N and the first output has length K. If the input is frame-based, then it must be a column vector.

The number N must have the form 2^M-1, where M is an integer greater than or equal to 3. For a given codeword length N, only specific message lengths K are valid for a BCH code. To see which values of K are valid, use the `bchpoly` function in the Communications Toolbox. No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability.

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.

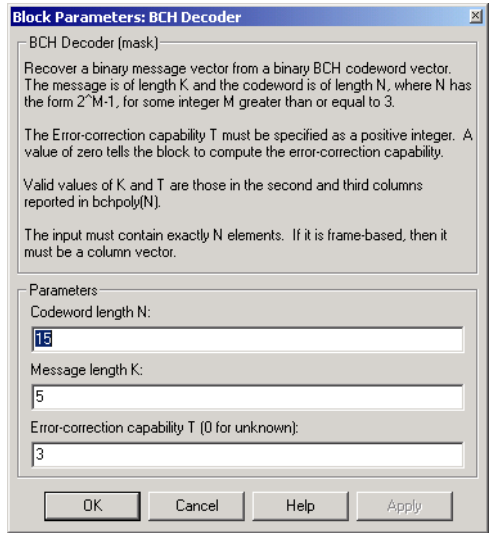
The sample times of all input and output signals are equal.

The **Error-correction capability T** parameter either:

- Indicates the error-correction capability of the code as a positive integer, or
- Tells the block to compute the error-correction capability, if you enter zero

The block runs faster in the first case above. You can use the `bchpoly` function in the Communications Toolbox to calculate the error-correction capability.

Dialog Box



Codeword length N

The codeword length, which is also the vector length of the first input.

Message length K

The message length, which is also the vector length of the first output.

Error-correction capability T

Either the error-correction capability of the code, or zero. A zero forces the block to calculate the error-control capability when initializing.

Pair Block

BCH Encoder

See Also

`bchpoly` (Communications Toolbox)

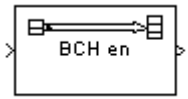
Purpose

Create a BCH code from binary vector data

Library

Block sublibrary of Channel Coding

Description



The BCH Encoder block creates a BCH code with message length K and codeword length N. You specify both N and K directly in the block mask.

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.

N must have the form 2^M-1, where M is an integer greater than or equal to 3. For a given codeword length N, only specific message lengths K are valid for a BCH code. To see which values of K are valid, use the `bchpoly` function in the Communications Toolbox. For example, in the output below, the second column lists all possible message lengths that correspond to a codeword length of 15. The third column lists the corresponding error-correction capabilities.

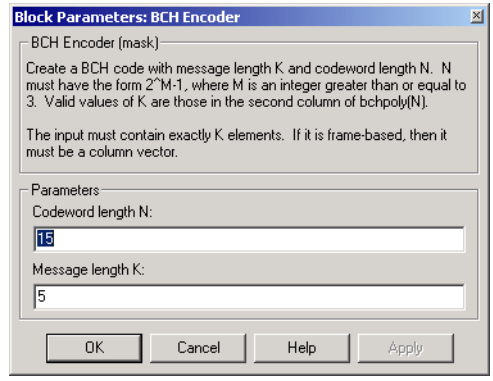
```
params = bchpoly(15)

params =

    15    11     1
    15     7     2
    15     5     3
```

No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability.

Dialog Box



Codeword length N

The codeword length, which is also the output vector length.

Message length K

The message length, which is also the input vector length.

Pair Block

BCH Decoder

See Also

`bchpoly` (Communications Toolbox)

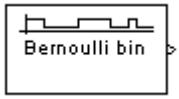
Purpose

Generate Bernoulli-distributed random binary numbers

Library

Data Sources sublibrary of Comm Sources

Description



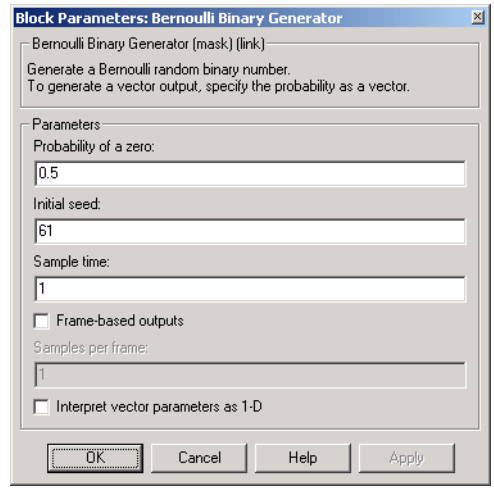
The Bernoulli Binary Generator block generates random binary numbers using a Bernoulli distribution. The Bernoulli distribution with parameter p produces zero with probability p and one with probability 1-p. The Bernoulli distribution has mean value 1-p and variance p(1-p). The **Probability of a zero** 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 Using the Communications Blockset 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



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.

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.

See Also

Binary Error Pattern Generator, Random Integer Generator, Binary Symmetric Channel; `randint` (Communications Toolbox), `rand` (built-in MATLAB function)

Purpose

Decode a 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 corresponding Binary Cyclic Encoder block.

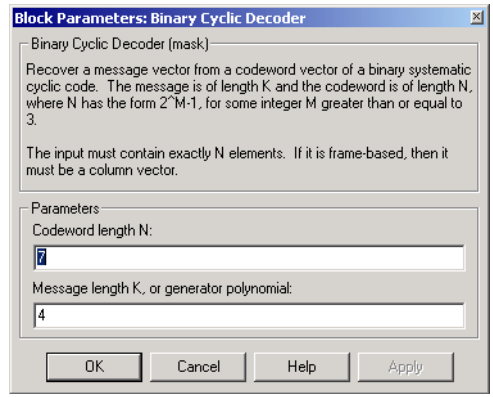
If the cyclic code has message length K and codeword length N, then N must have the form 2^M-1 for some integer M greater than or equal to 3.

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 mask 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 `cyclpoly` function in the Communications Toolbox.

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 a systematic cyclic code from binary vector data

Library

Block sublibrary of Channel Coding

Description

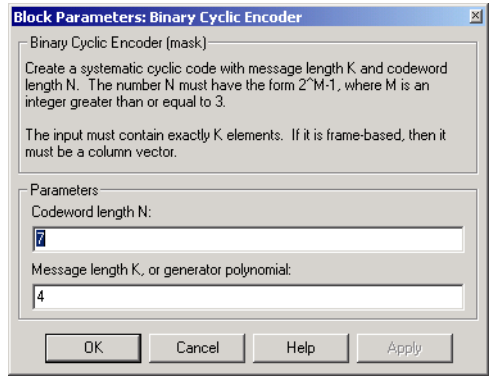
The Binary Cyclic Encoder block creates a systematic cyclic code with message length K and codeword length N. The number N must have the form 2^M-1, where M is an integer greater than or equal to 3.

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 mask 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 `cyclpoly` function in the Communications Toolbox.

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` (Communications Toolbox)

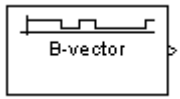
Purpose

Generate a binary vector while controlling the number of 1s

Library

Data Sources sublibrary Comm Sources

Description



The Binary Error Pattern Generator block outputs a random binary vector whose length is the **Binary vector length** parameter. The **Probabilities** parameter helps determine how many 1s appear in each output vector. Once the number of 1s is determined, their placement is determined according to a uniform distribution.

If $p_1, p_2,...p_m$ are the entries in the **Probabilities** parameter, then p_1 is the probability that the output vector will have a single 1, p_2 is the probability that the output vector will have exactly two 1s, and so on. Note that **Probabilities** must have sum less than or equal to one, and length less than or equal to the **Binary vector length**. Also, the probability of a zero vector is one minus the sum of **Probabilities**.

This block is useful in testing error-control coding algorithms.

Initial Seed

The scalar **Initial seed** parameter initializes the random number generator that the block uses to generate randiom errors. 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 block using the Communications Blockset's `randseed` function. At the MATLAB prompt, type the command

```
randseed
```

This returns a random prime number greater than 30. Typing `randseed` again produces a different prime number. If you add an integer argument, `randseed` always returns the same prime for that integer. For example, `randseed(5)` always returns the same answer.

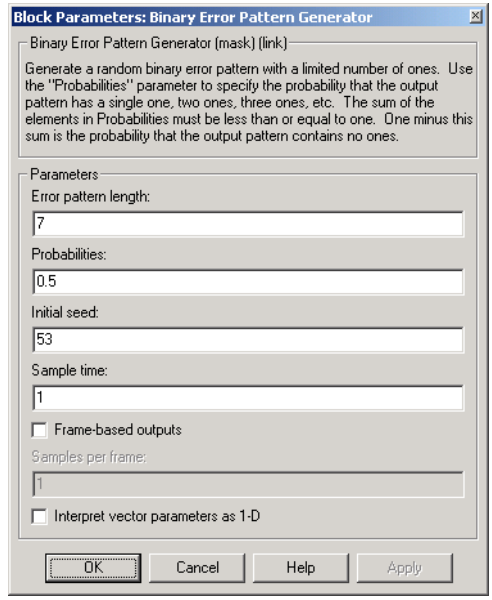
Attributes of Output Signal

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 Using the Communications Blockset for more details.

The **Binary vector length** 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 **Probabilities** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box



Error pattern length

The output vector length.

Probabilities

A vector whose *k*th entry indicates the probability that the output vector has exactly *k* 1s.

Initial seed

The initial seed value for the random number generator. This must be a

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.

See Also

Bernoulli Binary Generator; `randerr` (Communications Toolbox)

Binary-Input RS Encoder

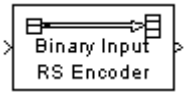
Purpose

Create a Reed-Solomon code from binary vector data

Library

Block sublibrary of Channel Coding

Description



The Binary-Input RS Encoder block creates a Reed-Solomon code with message length K and codeword length N. You specify both N and K directly in the block mask. 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 the section "Restrictions on the M and the Codeword Length N" below. The difference N-K must be an even integer.

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 output is a frame-based column vector whose length is the same integer multiple of M*N. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon only)."

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 the section "Specifying the Primitive Polynomial" following. If N is less than 2^M-1, the block uses a shortened Reed-Solomon code.

Each M*K input bits represent K integers between 0 and 2^M-1. Similarly, each M*N 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 the box next to **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 the box next to **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 the box next to **Specify primitive polynomial**, N must lie in the range $3 < N < 2^{16} - 1$.
- If you do select the box next to **Specify primitive polynomial**, N must lie in the range $3 \le N < 2^{M} - 1$ and M must lie in the range $3 \le M \le 16$.

Specifying the Generator Polynomial

You can specify the generator polynomial for the Reed-Solomon code. To do so, first select the box next to **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

$$g(x) = (x + \alpha^{b})(x + \alpha^{b+1})(x + \alpha^{b+2})\ldots(x + \alpha^{b+N-K-1})$$

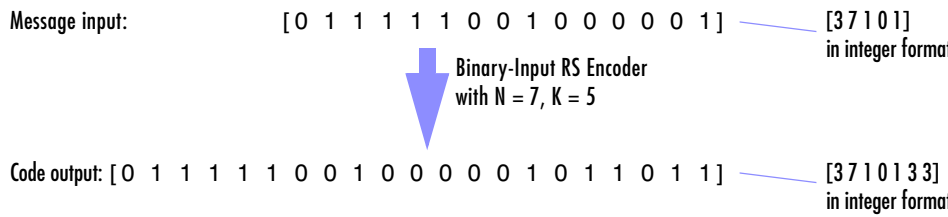
where α is the primitive element of the Galois field over which the input message is defined, and b is a non-negative integer.

If you do not select the box next to **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 typing `rsgenpoly(N1,K1)`, where `N1=2M-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)`.

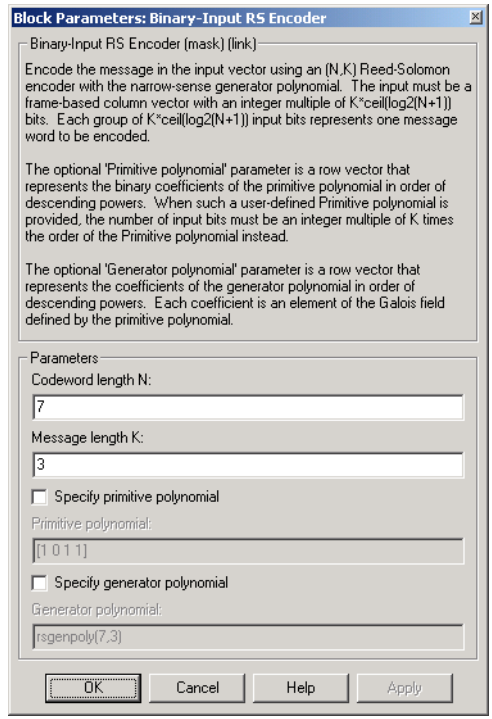
Examples

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.



Dialog Box



Codeword length N

The codeword length. The output has vector length M*N.

Message length K

The message length. The input has vector length M*K.

Specify primitive polynomial

When you select this box, you can specify the primitive polynomial as a binary row vector.

Primitive polynomial

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

Specify generator polynomial

When you select this box, you can specify the generator polynomial as an integer row vector.

Generator polynomial

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

Pair Block

Binary-Output RS Decoder

See Also

Integer-Input RS Encoder

Binary Linear Decoder

Purpose

Decode a linear block code to recover binary vector data

Library

Block sublibrary of Channel Coding

Description

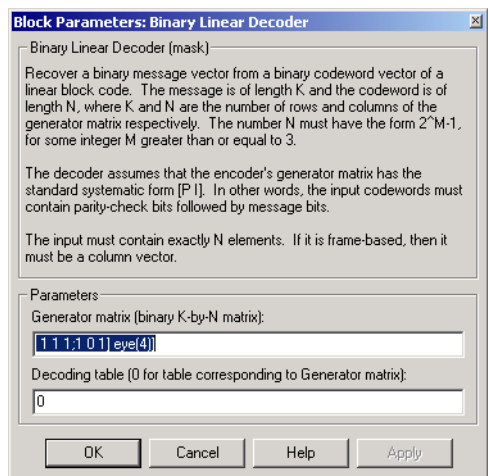
The Binary Linear Decoder block recovers a binary message vector from a binary codeword vector of a linear block code.

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.

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 a linear block code from binary vector data

Library

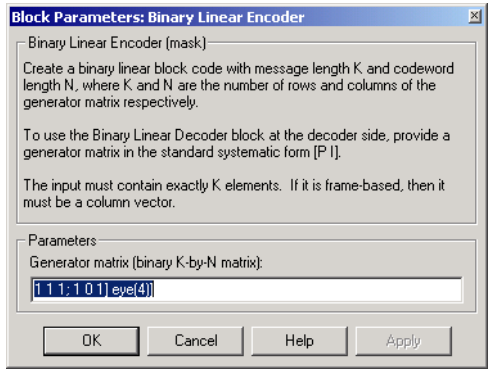
Block sublibrary of Channel Coding

Description

The Binary Linear Encoder block creates a binary linear block code using a generator matrix that you provide in the parameter mask. 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.

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

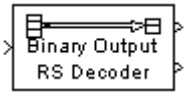
Purpose

Decode a 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.

The Reed-Solomon code has message length K and codeword length N. You specify both N and K directly in the block mask. 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 the section "Restrictions on the M and the Codeword Length N" on page 2-85. The difference N-K must be an even integer.

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 output is a frame-based column vector whose length is the same integer multiple of M*N. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon only)."

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 the section "Specifying the Primitive Polynomial" below. If N is less than 2^M-1, the block uses a shortened Reed-Solomon code.

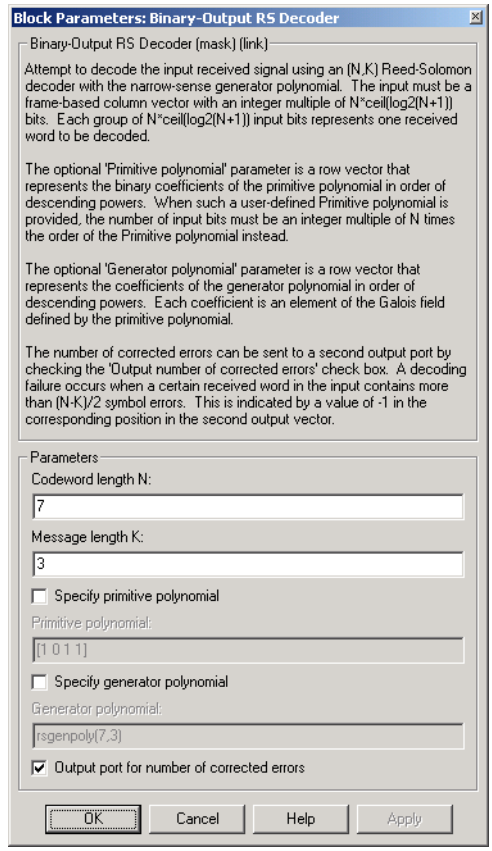
You can also specify the generator polynomial for the Reed-Solomon code, as described in the section "Specifying the Generator Polynomial" on page 2-85.

Each M*K input bits represent K integers between 0 and 2^M-1. Similarly, each M*N 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.

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

Dialog Box



Codeword length N

The codeword length. The input has vector length M*N.

Message length K

The message length. The first output has vector length M*K.

Specify primitive polynomial

When you select this box, you can specify the primitive polynomial as a binary row vector.

Primitive polynomial

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

Specify generator polynomial

When you select this box, you can specify the generator polynomial as an integer row vector.

Generator polynomial

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

Output port for 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.

Pair Block

Binary-Input RS Encoder

See Also

Integer-Output RS Decoder

Binary Symmetric Channel

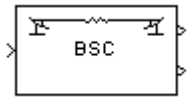
Purpose

Introduce binary errors

Library

Channels

Description

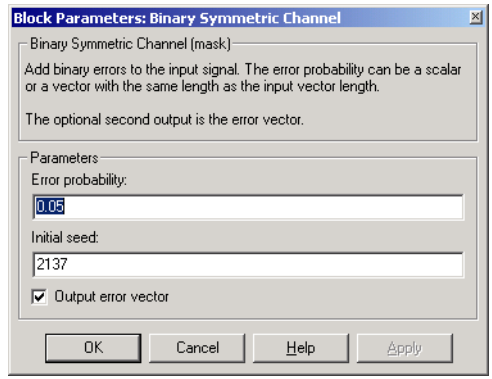


The Binary Symmetric Channel block introduces binary errors to the signal transmitted through this channel.

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**.

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, uncheck the **Output error vector** check box.

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.

See Also

Bernoulli Binary Generator

Purpose

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

Library

Utility Functions

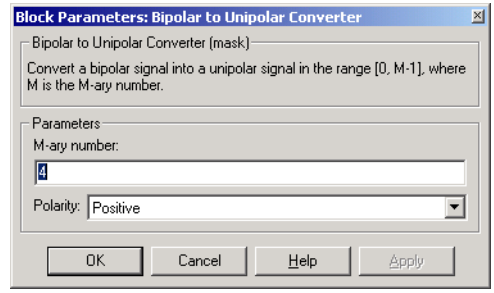
Description

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.

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.

Polarity

A value of **Positive** (respectively, **Negative**) causes the block to maintain (respectively, reverse) the relative ordering of symbols in the alphabets.

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].

Bipolar to Unipolar Converter

Pair Block Unipolar to Bipolar Converter

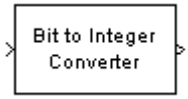
Purpose

Map a vector of bits to a corresponding vector of integers

Library

Utility Functions

Description

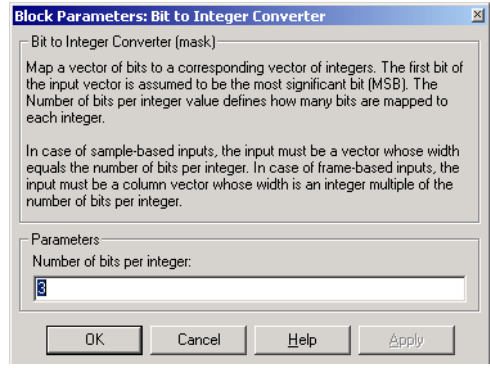


The Bit to Integer Converter block maps groups of bits in the input vector to integers in the output vector. If M is the **Number of bits per integer** parameter, then the block maps each group of M bits to an integer between 0 and 2^M-1. As a result, the output vector length is 1/M times the input vector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

If the input is sample-based input, then it must be a vector whose length equals 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 interprets the first bit in each group as the most significant bit.

Dialog Box



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 31.

Examples

If the input is [0; 1; 1; 1; 1; 1; 0; 1] and the **Number of bits per integer** parameter is 4, then the output is [7; 13]. The block maps the first group of four bits (0, 1, 1, 1) to 7 and the second group of four bits (1, 1, 0, 1) to 13. Notice that the output length is one-fourth of the output length.

Pair Block

Integer to Bit Converter

BPSK Demodulator Baseband

Purpose

Demodulate BPSK-modulated data

Library

PM, in Digital Baseband sublibrary of Modulation

Description

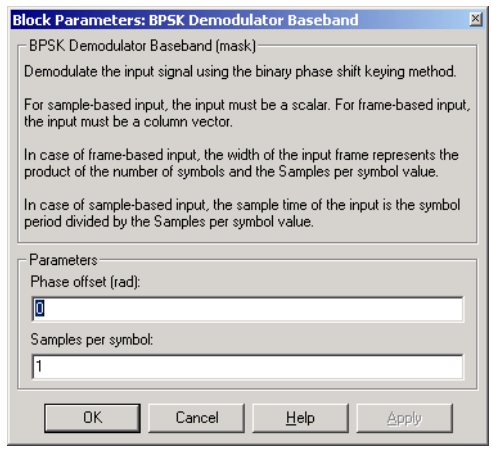
The BPSK Demodulator Baseband block demodulates a signal that was modulated using the binary phase shift keying method. The input is a baseband representation of the modulated signal. The input can be either a scalar or a frame-based column vector.

The 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.

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 Using the Communications Blockset.

Dialog Box



Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block BPSK Modulator Baseband

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

BPSK Modulator Baseband

Purpose

Modulate using the binary phase shift keying method

Library

PM, in Digital Baseband sublibrary of Modulation

Description

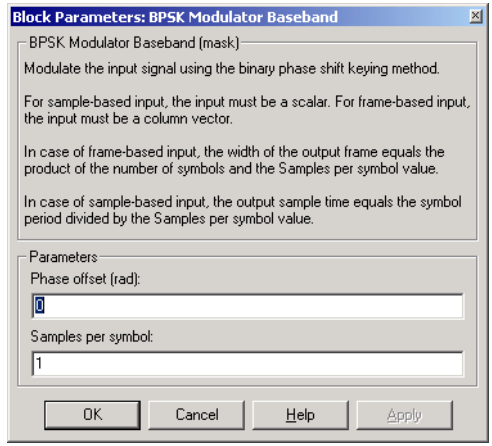
The BPSK Modulator Baseband block modulates using the binary phase shift keying method. The output is a baseband representation of the modulated signal.

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

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 Using the Communications Blockset.

Dialog Box



Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of output samples that the block produces for each input bit.

Pair Block BPSK Demodulator Baseband

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

Charge Pump PLL

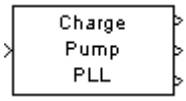
Purpose

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

Library

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** mask 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 the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

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

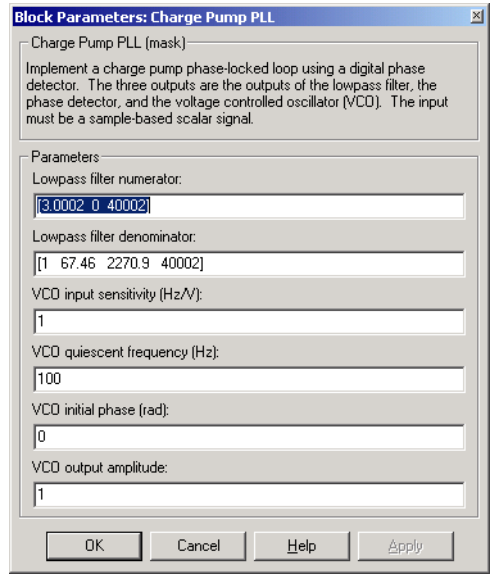
- A voltage-controlled oscillator (VCO). You specify characteristics of the VCO using the **VCO input sensitivity**, **VCO quiescent frequency**, **VCO initial phase**, and **VCO output amplitude** parameters.

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 Using the Communications Blockset.

Purpose

Output the phase difference between the two complex input signals

Library

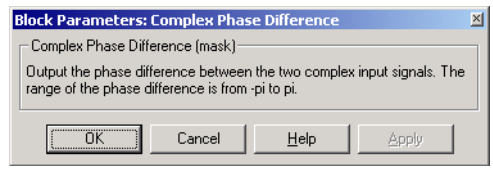
Sequence Operations, in Basic Comm Functions

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

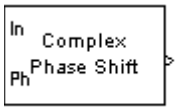
Purpose

Shift the phase of the complex input signal by the second input value

Library

Sequence Operations, in Basic Comm Functions

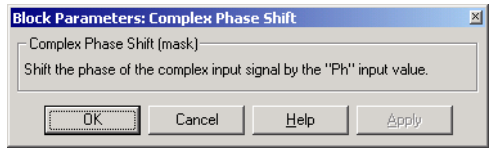
Description



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

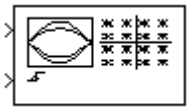
Purpose

Produce eye diagram, scatter, or x-y plots, using trigger to set decision timing

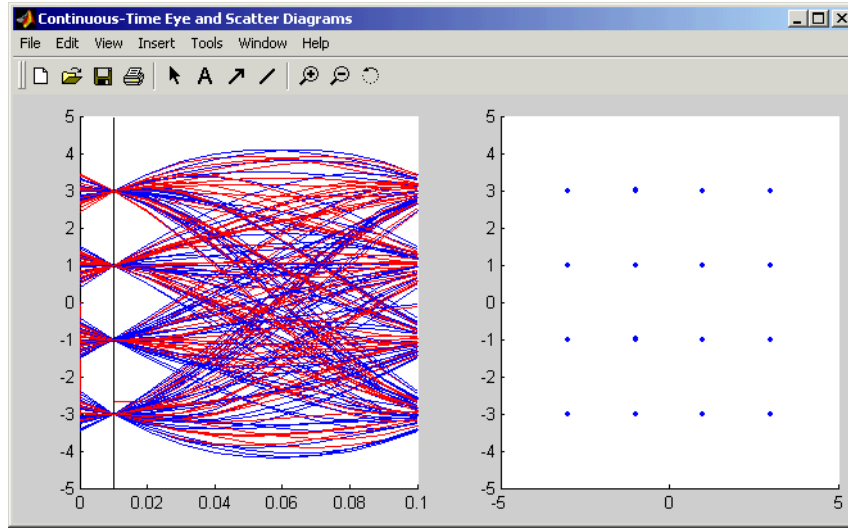
Library

Comm Sinks

Description



The Continuous-Time Eye and Scatter Diagrams block plots eye diagrams, scatter diagrams, and X-Y diagrams from a continuous-time input signal. The **Diagram type** parameter determines which plots the block produces. The block draws the diagrams in a single window, as in the case of the eye diagram and scatter diagram below.



The first input is a complex message signal. It must be a sample-based scalar signal. The eye diagram and the X-Y diagram both record the trajectories of the message signal in continuous time.

The second input is a scalar trigger signal that determines the decision timing for the scatter diagram. At each rising edge of the trigger signal, the block plots a vertical line in the eye diagram and adds a new point to the scatter plot.

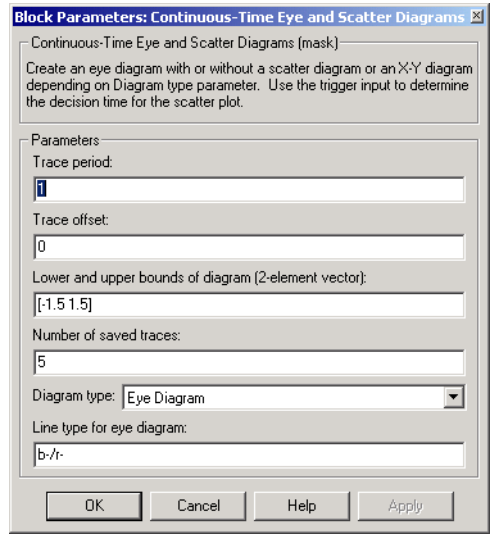
The **Trace period** parameter is the number of seconds represented by the horizontal axis in the eye diagram. The **Trace offset** parameter is the time value at the left edge of the horizontal axis of the eye diagram.

Tip This block works better if the model's simulation step size is suitable for your data. From the model window's **Simulation** menu, choose **Simulation parameters** and then choose a value for the **Max step size** parameter. Try multiplying this block's **Trace period** parameter by 1/8 or 1/16.

To specify the plotting color as well as the line type and/or marker type, use the **Line type** parameters that appear after you select a value for the **Diagram type** parameter. In the **Line type for eye diagram** parameter, use a slash (/) to separate the specifications for the real and imaginary components of the input signal. Choices for the color, marker, and line types are in the table below.

Color Characters		Marker-Type Characters		Line-Type Characters	
`y`	Yellow	`.`	Point	`-`	Solid
`m`	Magenta	`o`	Circle	`:`	Dotted
`c`	Cyan	`x`	Cross	`-.`	Dash-dot
`r`	Red	`+`	Plus sign	`--`	Dashed
`g`	Green	`*`	Asterisk		
`b`	Blue	`s`	Square		
`w`	White	`d`	Diamond		
`k`	Black	`v`	Triangle (down)		
		`^`	Triangle (up)		
		`<`	Triangle (left)		
		`>`	Triangle (right)		
		`p`	Five-pointed star		
		`h`	Six-pointed star		

Dialog Box



Trace period

The duration of the horizontal axis of the eye diagram, in seconds.

Trace offset

The time at the leftmost edge of the horizontal axis of the eye diagram.

Lower and upper bounds of diagram

A two-element vector containing the minimum and maximum signal values in the diagrams.

Number of saved traces

The number of curves in the eye diagram, or points in the scatter plot, that are visible after you resize or restore the figure window.

Diagram type

The diagram(s) that the block produces.

Line type for eye diagram

A string that specifies the color and the line type for the eye diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an eye diagram.

Line type for scatter diagram

A string that specifies the color and the marker type for the scatter diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an scatter diagram.

Line type for X-Y diagram

A string that specifies the color and the line type for the X-Y diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an X-Y diagram.

See Also

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

Purpose

Restore ordering of symbols that were permuted using shift registers

Library

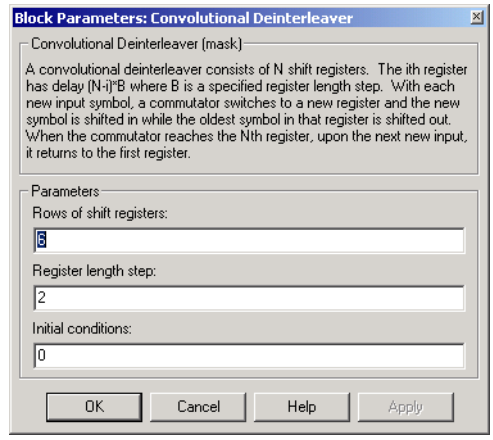
Convolutional sublibrary of Interleaving

Description

The Convolutional Deinterleaver block recovers a signal that was interleaved using the Convolutional Interleaver block. The parameters in the two blocks should have the same values.

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.

Dialog Box



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.

Initial conditions

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

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.

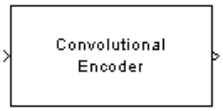
Purpose

Create a 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), then 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), then 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.

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 User's Guide*. You can use this parameter field in two ways:

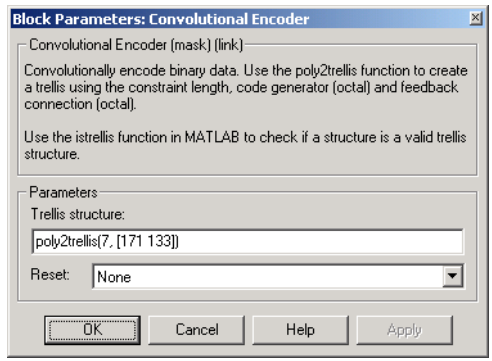
- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the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

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, use one of these values of the **Reset** parameter:

- The value **None** indicates that the encoder never resets.
- The value **On each frame** indicates that the encoder resets at the beginning of each frame, before processing the next frame of input data
- The value **On nonzero Rst input** causes the block to have 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.

Reset

Determines whether and under what circumstances the encoder resets to the all-zeros state before processing the input data. Choices are **None**, **On each frame**, and **On nonzero Rst input**. The last option causes the block to have a second input port, labeled `Rst`.

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.

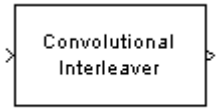
Purpose

Permute input symbols using a set of shift registers

Library

Convolutional sublibrary of Interleaving

Description

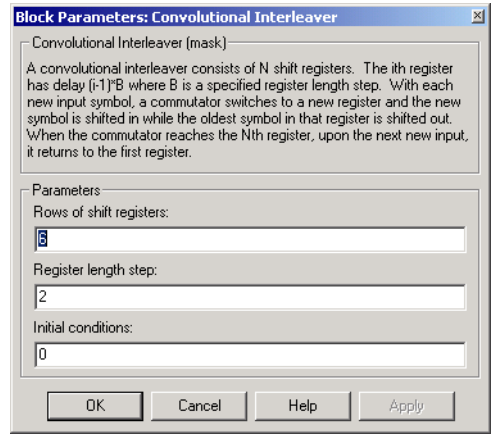


The Convolutional Interleaver block permutes the symbols in the input signal. Internally, it uses a set of shift registers. The delay value of the kth shift register is (k-1) times the **Register length step** parameter. The number of shift registers is the value of the **Rows of shift registers** parameter.

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.

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.

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.

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*.

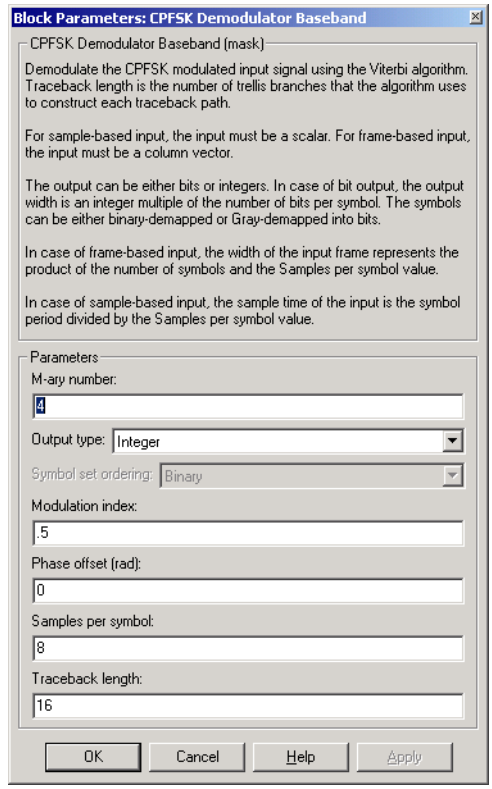
In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer (k+M-1)/2. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether 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 Using the Communications Blockset.

The input can be either a scalar or a frame-based column vector.

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 Using the Communications Blockset.

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.

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 Viterbi Decoder block uses to construct each traceback path.

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.

CPFSK Demodulator Passband

Purpose

Demodulate CPFSK-modulated data

Library

CPM, in Digital Passband sublibrary of Modulation

Description

The CPFSK Demodulator Passband block demodulates a signal that was modulated using the continuous phase frequency shift keying method. The input is a passband 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.

This block converts the input to an equivalent baseband representation, using downconversion and then FIR decimation. The block then uses the baseband equivalent block, CPFSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Signal set ordering**
- **Modulation index**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$

- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < $[2*\textbf{Carrier frequency} + 2*F_{max})]^{-1}$

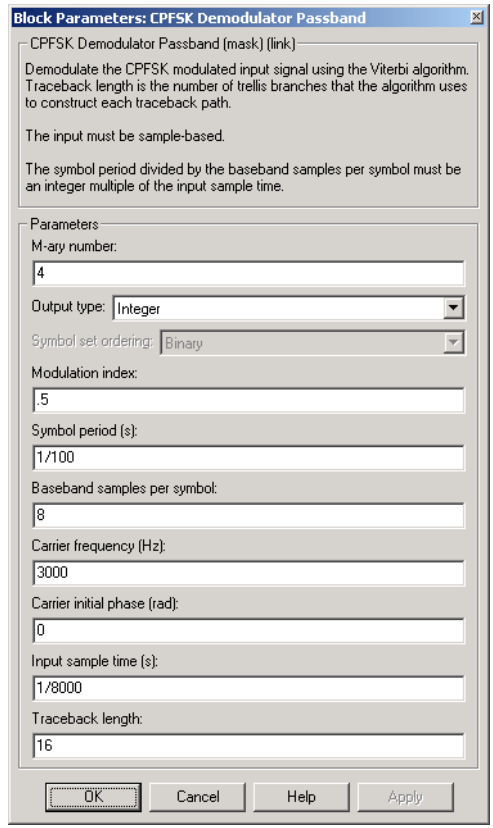
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The CPSK Demodulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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.

Symbol period(s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time (s)

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

CPFSK Modulator Passband

See Also

CPFSK Demodulator Baseband, Viterbi Decoder, M-FSK Demodulator Passband

References

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

CPFSK Modulator Baseband

Purpose

Modulate using the 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 listed in "References" on page 2-126.

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.

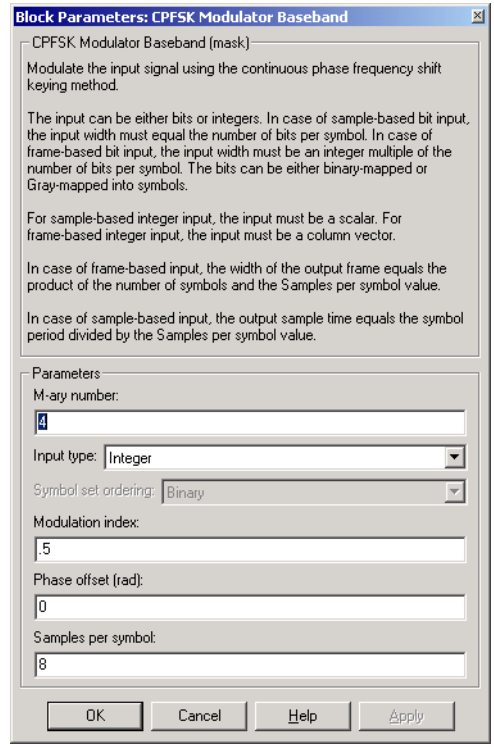
In binary input mode, the block maps each binary word to an integer between 0 and M-1, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. The block then maps the integer k to the intermediate value 2k-(M-1) and proceeds as in the integer input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

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 Using the Communications Blockset.

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.

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.

Purpose

Modulate using the continuous phase frequency shift keying method

Library

CPM, in Digital Passband sublibrary of Modulation

Description

The CPFSK Modulator Passand block modulates using the continuous phase frequency shift keying method. The output is a passband 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.

This block uses the baseband equivalent block, CPFSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation, using FIR interpolation and then upconversion. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Modulation index**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$

- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < $[2*\textbf{Carrier frequency} + 2*F_{max})]^{-1}$

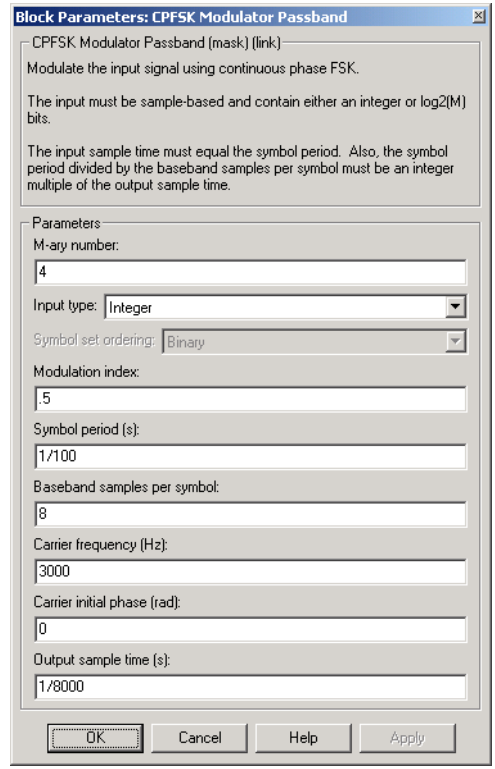
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The CPFSK Modulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time(s)

The sample time of the output signal.

Pair Block

CPFSK Demodulator Passband

See Also

CPFSK Modulator Baseband, M-FSK Modulator Passband

References

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

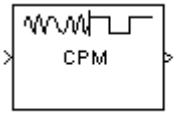
Purpose

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.

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 the CPM 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.

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*.

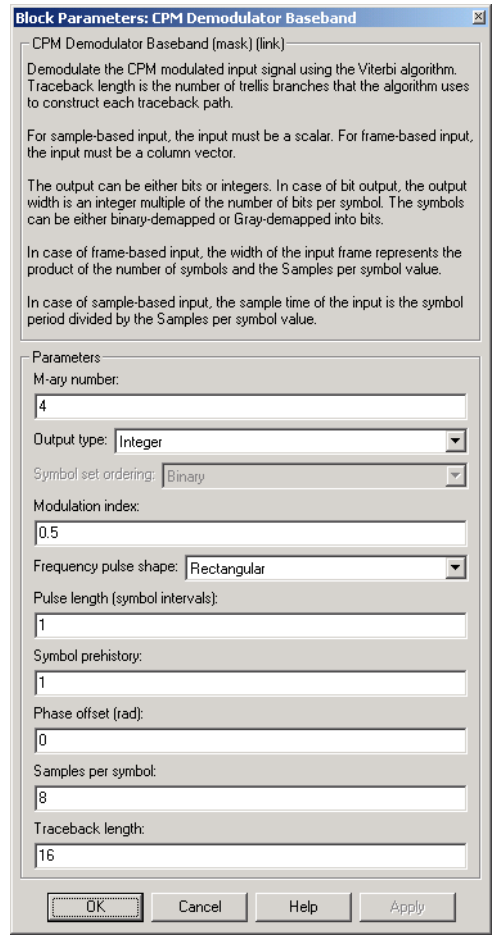
In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer (k+M-1)/2. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether 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 Using the Communications Blockset.

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 Using the Communications Blockset.

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.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

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.

CPM Demodulator Passband

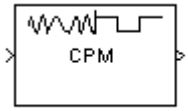
Purpose

Demodulate CPM-modulated data

Library

CPM, in Digital Passband sublibrary of Modulation

Description



The CPM Demodulator Passband block demodulates a signal that was modulated using continuous phase modulation. The input is a passband 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.

This block converts the input to an equivalent baseband representation using downconversion and then FIR decimation. The block then uses the baseband equivalent block, CPM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Symbol set ordering**
- **Modulation index**
- **Frequency pulse shape**
- **Rolloff**
- **BT product**
- **Pulse length**
- **Symbol prehistory**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per**

symbol parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < [2*__Carrier frequency__ + 2*F_{max})]$^{-1}$

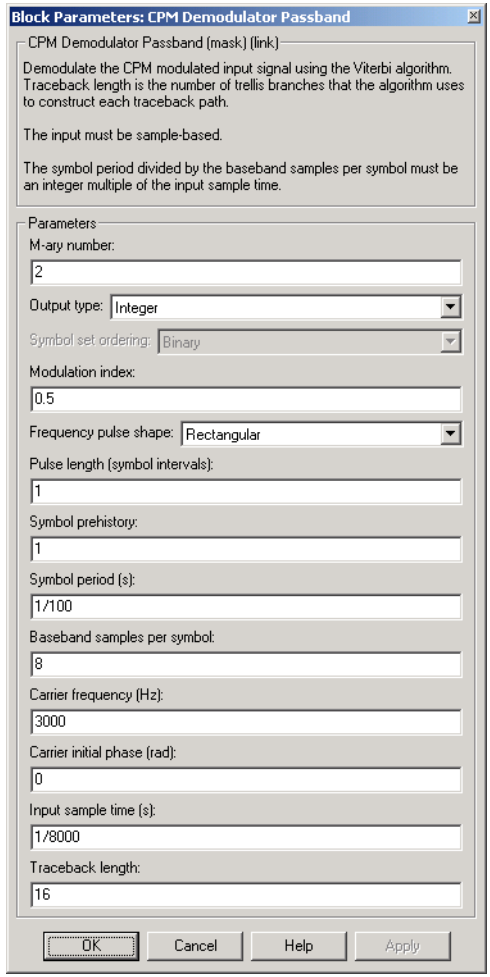
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The CPM Demodulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time(s)

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

CPM Modulator Passband

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.

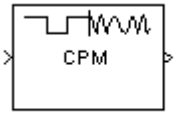
Purpose

Modulate using 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 listed in "References" on page 2-145. 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.

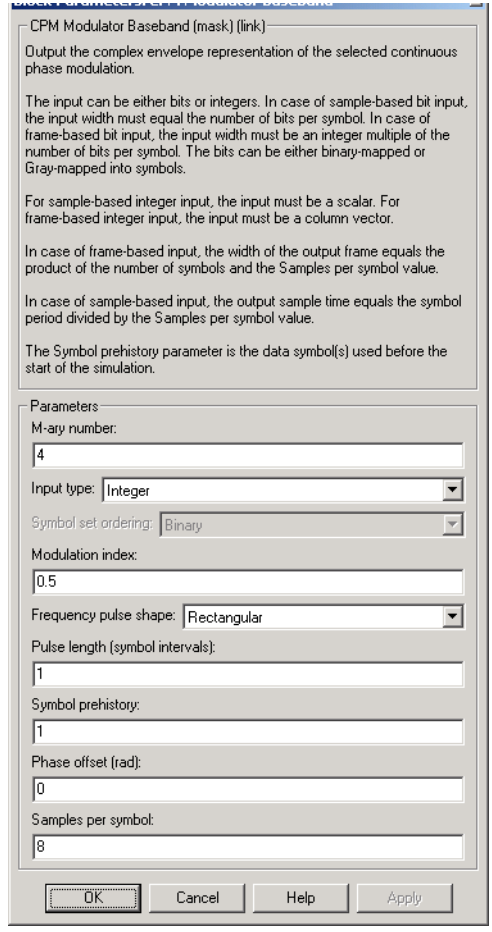
In binary input mode, the block maps each binary word to an integer between 0 and M-1, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. The block then maps the integer k to the intermediate value 2k-(M-1) and proceeds as in the integer input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

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 Using the Communications Blockset.

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.

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 Modulator Passband

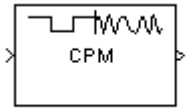
Purpose

Modulate using continuous phase modulation

Library

CPM, in Digital Passband sublibrary of Modulation

Description



The CPM Modulator Passband block modulates using continuous phase modulation. The output is a passband 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.

This block uses the baseband equivalent block, CPM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation, using FIR interpolation and then upconversion. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Modulation index**
- **Frequency pulse shape**
- **Rolloff**
- **BT product**
- **Pulse length**
- **Symbol prehistory**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each

integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < [2*__Carrier frequency__ + 2*F_{max})]$^{-1}$

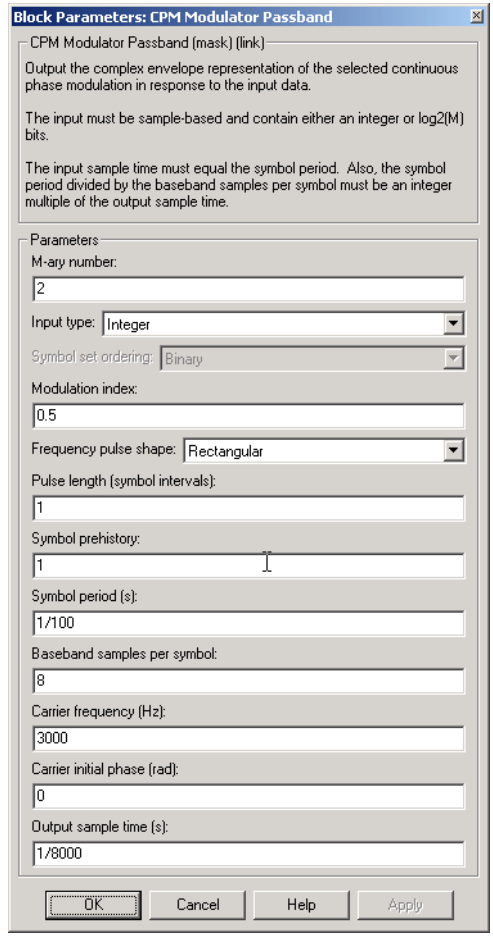
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The CPM Modulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time(s)

The sample time of the output signal.

Pair Block

CPM Demodulator Passband

See Also

CPM Modulator Baseband

References

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

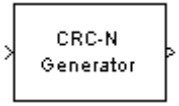
Purpose

Generate cyclic redundancy check (CRC) bits, according to the selected CRC method, and append them 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 end of 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 mask. The value of 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

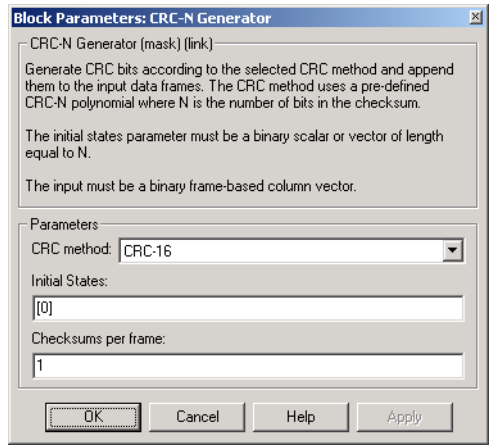
For a more detailed description of the CRC algorithm, see the section "Cyclic Redundancy Check Coding."

You specify the initial state of the internal shift register by the **Initial states** parameter in block's mask. 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 Generator block.

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.

Pair Block

CRC-N Syndrome Detector

See Also

General CRC Generator, General CRC Syndrome Detector

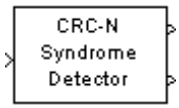
Purpose

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

Library

CRC Sublibrary of Error Detection and Correction

Description

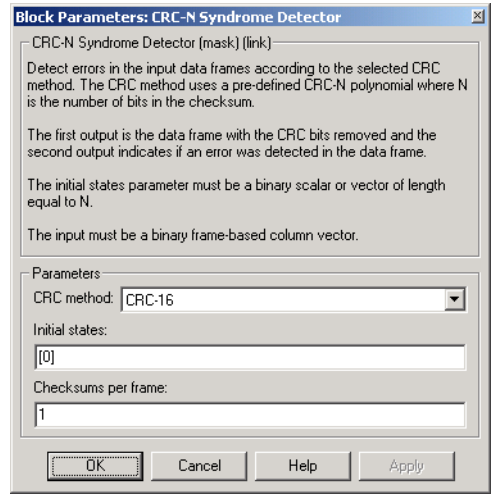


The CRC-N Syndrome Detector block receives a message word and removes the checksum. The block then calculates a new checksum, and compares the received checksum with the new checksum. The CRC-N Syndrome Detector block is a simplified version of the General CRC Syndrome Detector block. With the CRC-N 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 mask. The value of N is 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 in block's mask. 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.

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.

Pair Block

CRC-N Generator

See Also

General CRC Generator, General CRC Syndrome Detector

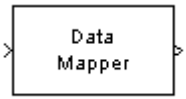
Purpose

Map integer symbols from one coding scheme to another

Library

Utility Functions

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 either a scalar, a sample-based vector, or a frame-based column vector.

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.

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

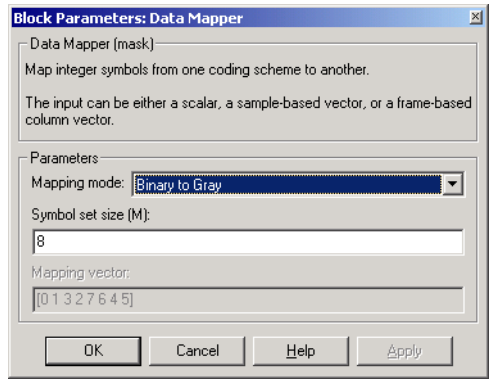
Binary to Gray Mode		Gray to Binary Mode	
Input	**Output**	**Input**	**Output**
5	7 *(111)*	5 *(101)*	6
6	5 *(101)*	6 *(110)*	4
7	4 *(100)*	7 *(111)*	5

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 is active only when **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.

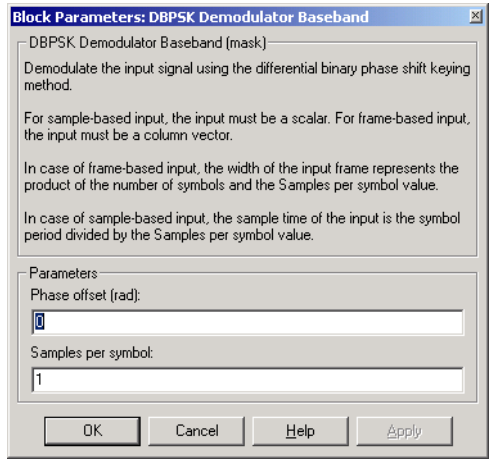
The input must be a discrete-time complex signal. The block compares the current symbol to the previous symbol. It maps phase differences of θ and $\pi+\theta$, respectively, to outputs of 0 and 1, respectively, where θ is the **Phase offset** parameter. The first element of the block's output is the 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.

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 Using the Communications Blockset.

Dialog Box



Phase offset (rad)

This phase difference between the current and previous modulated symbols results in an output of zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

DBPSK Modulator Baseband

See Also

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

DBPSK Modulator Baseband

Purpose

Modulate using the differential binary phase shift keying method

Library

PM, in Digital Baseband sublibrary of Modulation

Description

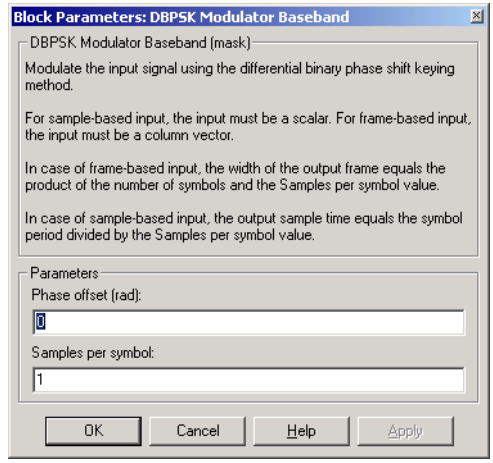
The DBPSK Modulator Baseband block modulates using the differential binary phase shift keying method. The output is a baseband representation of the modulated signal.

The input must be a discrete-time binary-valued signal. The input can be either a scalar or a frame-based column vector. These rules govern this modulation method when the **Phase offset** 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.

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 Using the Communications Blockset.

Dialog Box



Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of output samples that the block produces for each input bit.

Pair Block

DBPSK Demodulator Baseband

See Also

M-DPSK Modulator Passband, DQPSK Modulator Baseband, BPSK Modulator Baseband

Deinterlacer

Purpose

Distribute elements of input vector alternately between two output vectors

Library

Sequence Operations, in Basic Comm Functions

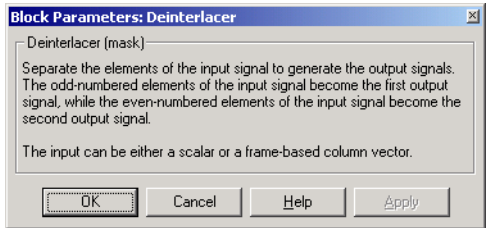
Description

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.

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)

Purpose

Reduce sampling rate by averaging consecutive samples

Library

Sequence Operations, in Basic Comm Functions

Description

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 Repeat block (DSP Blockset). The positive integer N is the **Derepeat factor** parameter in the Derepeat mask.

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 input can have any shape or frame status.

Sample-Based Operation

If the input is sample-based, 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.

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.

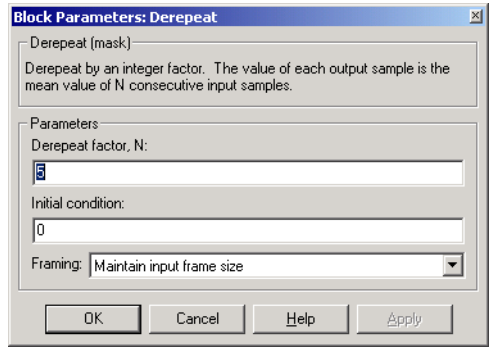
For example, if a single-channel input with a frame period of 1 second is derepeated by a factor of 4, then the output has a frame period of 4 seconds. The input and output frame sizes are equal.

- **Maintain input frame rate**

  The block reduces the sampling rate by using a proportionally smaller frame *size* than the input. 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.

Dialog Box



Derepeat factor, N

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

Initial condition

The value with which to initialize the block.

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.

See Also

Repeat (DSP Blockset), Downsample (DSP Blockset)

Descrambler

Purpose

Descramble the input signal

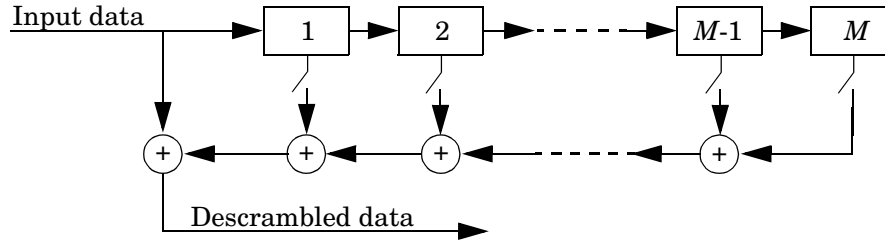
Library

Sequence Operations, in Basic Comm Functions

Description

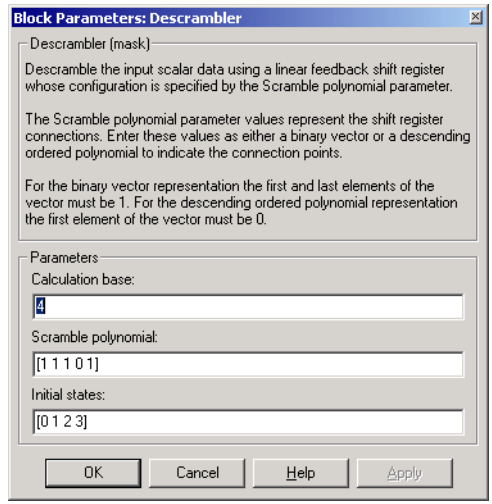
The Descrambler block descrambles the scalar input signal. The Descrambler block is the inverse of the Scrambler block. If you use the Scrambler block in the transmitter, then you should use the Descrambler block in the receiver.

Below is a schematic of the descrambler. All adders perform addition 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. The **Initial states** can be different in the two blocks, considering the transmitting and receiving filter delay. See the reference page for the Scrambler block for more information about these parameters.

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

Differential Decoder

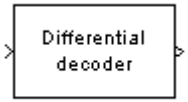
Purpose

Decode a binary signal using differential coding technique.

Library

Source Coding

Description



The Differential Decoder block decodes the binary input signal. The output of the Differential Decoder block is the decoded binary signal.

The block's input m and output d are related by

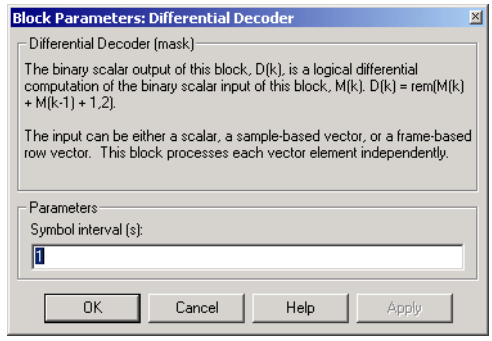
$$d(t_0) = m(t_0) + 1 \mod 2$$

$$d(t_k) = m(t_{k-1}) + m(t_k) + 1 \mod 2$$

where t_k is the kth time step.

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

Dialog Box



Symbol interval (s)

The sample time of the input symbol.

Pair Block

Differential Encoder

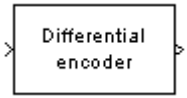
Purpose

Encode a binary signal using differential coding technique.

Library

Source Coding

Description



The Differential Encoder block encodes and outputs the binary input signal.

The input m and output d are related by

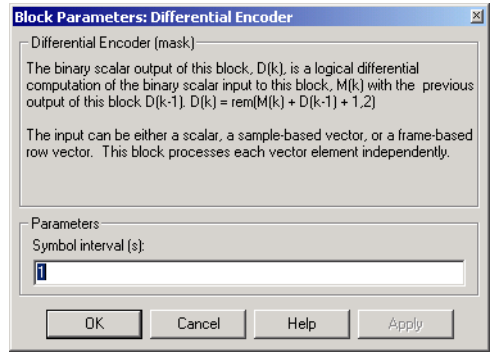
$$d(t_0) = m(t_0) + 1 \mod 2$$

$$d(t_k) = d(t_{k-1}) + m(t_k) + 1 \mod 2$$

where t_k is the kth time step.

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

Dialog Box



Symbol interval (s)

The sample time of the input symbol.

Pair Block

Differential Decoder

Discrete Modulo Integrator

Purpose

Integrate in discrete time and reduce by a modulus

Library

Integrators, in Basic Comm Functions

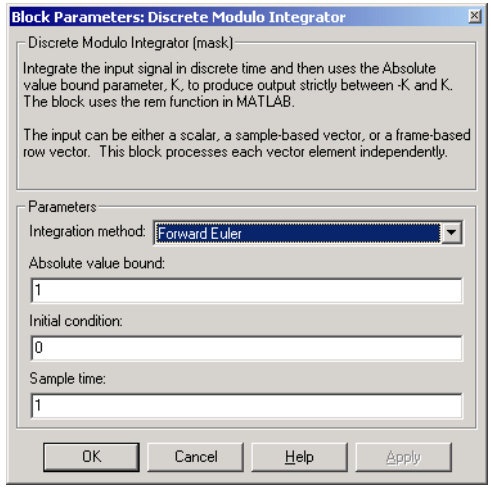
Description

The Discrete Modulo Integrator block integrates its input signal in discrete time and then uses the **Absolute value bound** parameter, K, to produce output strictly between -K and K. The block uses the `rem` function in MATLAB.

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

You can choose one of three integration methods: **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

Dialog Box



Integration method

The integration method. Choices are **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

Absolute value bound

The modulus by which the integration result is reduced. This parameter must be nonzero.

Initial condition

The initial condition for integration.

Sample time

The integration sample time.

See Also

Modulo Integrator, Windowed Integrator, Integrate and Dump, Discrete-Time Integrator (Simulink); `rem` (MATLAB)

Discrete-Time Eye Diagram Scope

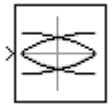
Purpose

Display multiple traces of a modulated signal

Library

Comm Sinks

Description

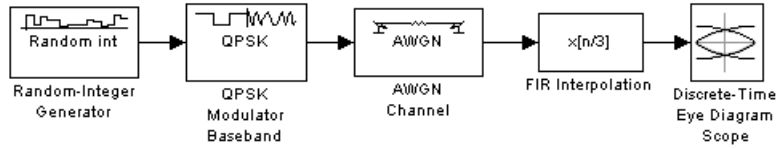


The Discrete Eye Diagram Scope block displays multiple traces of a modulated signal to produce an eye diagram. You can use the block to reveal the modulation characteristics of the signal, such as pulse shaping or channel distortions.

The Discrete-Time Eye Diagram Scope block has one input port. The input signal can be either real or complex. The input signal must be a sample-based scalar in sample-based mode. The input must be a frame-based column vector or a scalar in frame-based mode.

Example: Viewing the Eye Diagram of a Modulated Signal

The following model creates an eye diagram for a modulated signal. The model modulates a random signal using the QPSK Modulator block and then filters the signal with a raised cosine interpolation filter.



To build the model, gather and configure the following blocks:

- Random Integer Generator, in the Data Sources sublibrary of the Comm Sources library, with default parameters
- QPSK Modulator Baseband, in PM in the Digital Baseband sublibrary of the Modulation library of the Communications Blockset, with default parameters
- AWGN Channel, in the Channels library of the Communications Blockset, with the following changes to the default parameter settings:
  - Set **Mode** to **Signal-to-noise ratio (SNR)**.
  - Set **SNR (dB)** to 15.

- FIR Interpolation, in the Multirate Filters sublibrary of the Filtering library of the DSP Blockset, with the following changes to the default parameter settings:
  - Set **FIR Filter Coefficients** to `rcosine(1, 8,[], 0.5,3)`.
  - Set **Interpolation factor** to `8`.
- Discrete-Time Eye Diagram Scope, in the Comms Sinks library, with the following changes to the default parameter settings:
  - Set **Samples per symbol** to `8`.

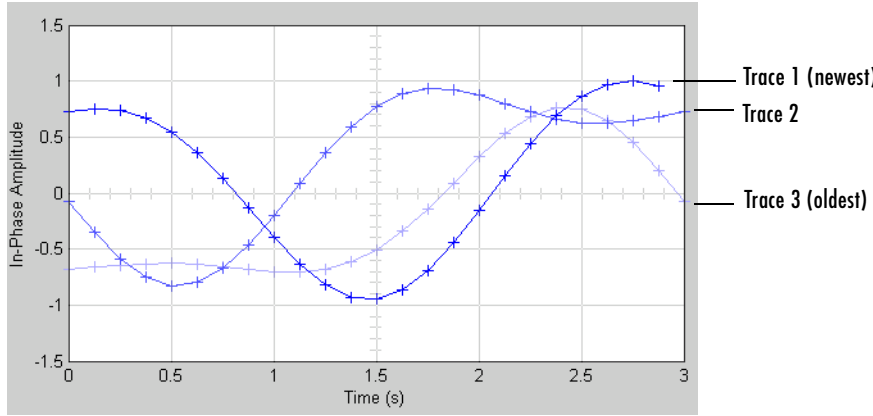
    Samples per symbol specifies the number of samples in each channel symbol. Each sample corresponds to a plotted point in the eye diagram.
  - Set **Symbols per trace** to `3`.

    Symbols per trace specifies the number of symbols that are displayed in each trace of the eye diagram. A trace is any one of the individual lines in the eye diagram.
  - Set **Traces displayed** to `3`.

    Traces displayed specifies the number of traces that are displayed at any instant.
  - Set **New traces per display** to `1`.

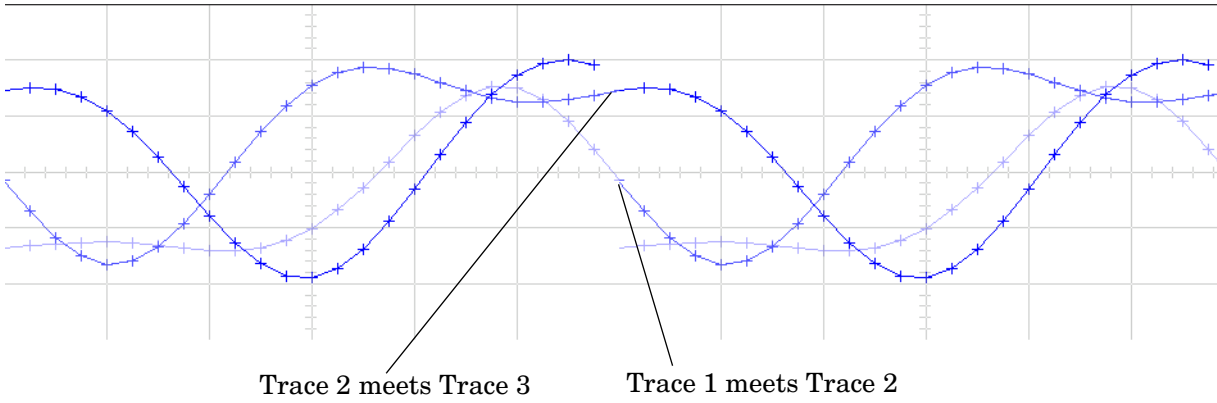
    New traces per display specifies the number of new traces that appear each time the diagram is refreshed. The number of traces that remain in the diagram from one refresh to the next is **Traces displayed** minus **New traces per display**.
  - Check **Show Rendering Properties** and set **Markers** to `+` to indicate the points plotted at each sample. The default value of **Markers** is empty, which indicates no marker.
  - Check **Show Figure Properties** and set **Eye diagram to display** to **In-phase only**.

When you run the model for 10 seconds, the Discrete-Time Eye Diagram Scope displays the following diagram.

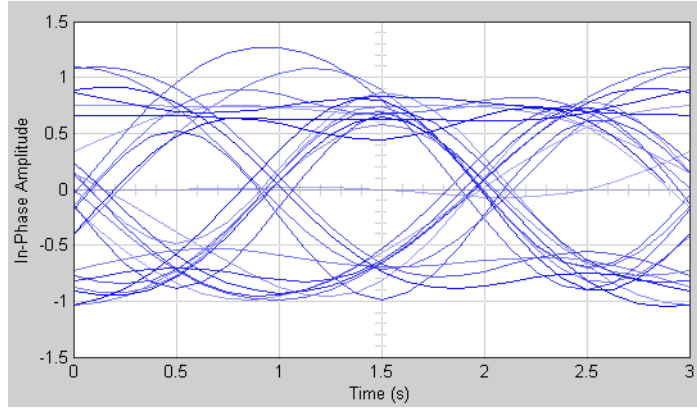


Note that three traces are displayed. Traces 2 and 3 are faded because the **Show Rendering Properties** and **Color fading** check boxes are selected. This causes traces to be displayed less brightly the older they are. In this picture, Trace 1 is the most recent and Trace 3 is the oldest. Because **New traces per display** is set to 1, only Trace 1 is appearing for the first time. Traces 2 and 3 also appear in the previous display.

Because **Symbols per trace** is set to 3, each trace contains three symbols, and because **Samples per trace** is set to 8, each symbol contains eight samples. Note that trace 1 contains 24 points, which is the product of **Symbols per trace** and **Samples per symbol**. However, traces 2 and 3 contain 25 points each. The last point in trace 2, at the right border of the scope, represents the same sample as the first point in trace 1, at the left border of the scope. Similarly, the last point in trace 3 represents the same sample as the first point in trace 2. These duplicate points indicate where the traces would meet if they were displayed side by side, as illustrated in the following picture.



You can view a more realistic eye diagram by changing the value of **Number of simultaneous traces** to `40` and clearing the **Line markers** field.



Note that when the **Offset (samples)** parameter is set to `0`, the plotting starts at the center of the first symbol, so that the open part of the eye diagram is in the middle of the plot for most points.

For another example, see Example: Viewing a Sinusoid.

Marker and Line Styles

The **Marker**, **Line style**, and **Line color** parameters, which are visible when you check **Show Rendering Properties**, 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	*	
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)	

Color	RGB Equivalent	Appearance
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`)

Parameter	Recommended Setting
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.
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.
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

Scope Options

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

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

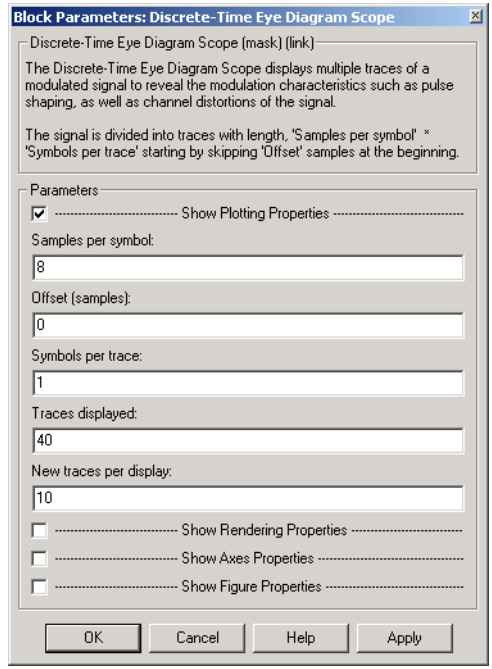
The properties listed in the **Axes** menu apply to all channels. Many of the parameters in this menu are also accessible through the block parameter dialog box. These are **Autoscale**, **Show grid**, **Frame #**, and **Save Position**. Below are descriptions of the other parameters listed in the **Axes** 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/Axes properties** field 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



Show Plotting Properties

Select to display plotting properties.

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 value of **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

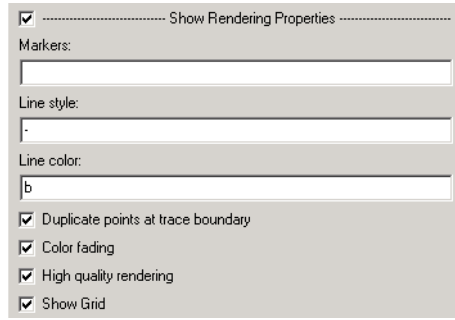
Number of traces plotted.

New traces per display

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

Show Rendering Properties

Select to display rendering properties, as shown in the following figure.



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

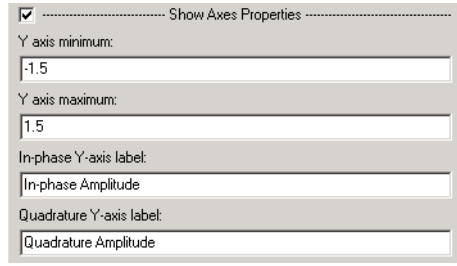
When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid

Toggles the scope grid on and off. Tunable.

Show Axes Properties

Select to display axes properties, as shown in the following figure.



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

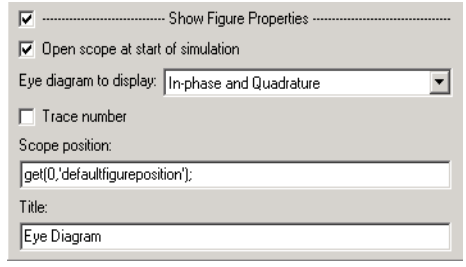
Label for *y*-axis of the in-phase diagram. Tunable.

Quadrature Y-axis label

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

Show Figure Properties

Select to display figure properties, as shown in the following figure.



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.

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.

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

- CPM Phase Tree Example – `cpmphasetree.mdl`
- Filtered Offset QPSK vs. Filtered QPSK – `foqpskvsfqpsk.mdl`
- Rayleigh Fading Channel – `rayleighfading.mdl`
- QPSK vs. MSK – `qpskvsmsk.mdl`

See Also

Continuous-Time Eye and Scatter Diagrams, Discrete-Time Scatter Plot Scope, Discrete-Time Signal Trajectory Scope

Discrete-Time Scatter Plot Scope

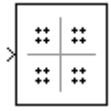
Purpose

Display the in-phase and quadrature components of the constellation of a modulated signal

Library

Comm Sinks

Description

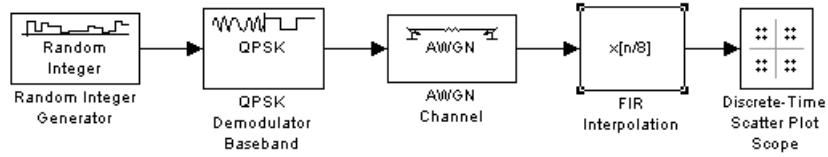


The Discrete-Time Scatter Plot Scope block displays scatter plots of a modulated signal, to reveal the modulation characteristics, such as pulse shaping or channel distortions of the signal.

The Discrete-Time Scatter Plot Scope block has one input port. The input signal must be complex. The input signal must be a sample-based scalar in sample-based mode. The input must be a frame-based column vector or a scalar in frame-based mode.

Example: Viewing the Scatter Plot of a Modulated Signal

The following model creates a scatter plot for a modulated signal. The model modulates a random signal using the QPSK Modulator block and then filters the signal with a raised cosine interpolation filter.



To build the model, gather and configure the following blocks:

- Random Integer Generator, in the Data Sources sublibrary of the Comm Sources library, with default parameters
- QPSK Modulator Baseband, in PM in the Digital Baseband sublibrary of the Modulation library of the Communications Blockset, with default parameters
- AWGN Channel, in the Channels library of the Communications Blockset, with the following changes to the default parameter settings:
  - Set **Mode** to **Signal-to-noise ratio (SNR)**.
  - Set **SNR (dB)** to 15.

- FIR Interpolation, in the Multirate Filters sublibrary of the Filtering library of the DSP Blockset, with the following changes to the default parameter settings:
  - Set **FIR Filter Coefficients** to `rcosine(1, 8,[], 0.5,3)`.
  - Set **Interpolation factor** to `8`.
- Discrete-Time Scatter Plot Scope, in the Comms Sinks library, with the following changes to the default parameter settings:
  - Set **Samples per symbol** to `2`.

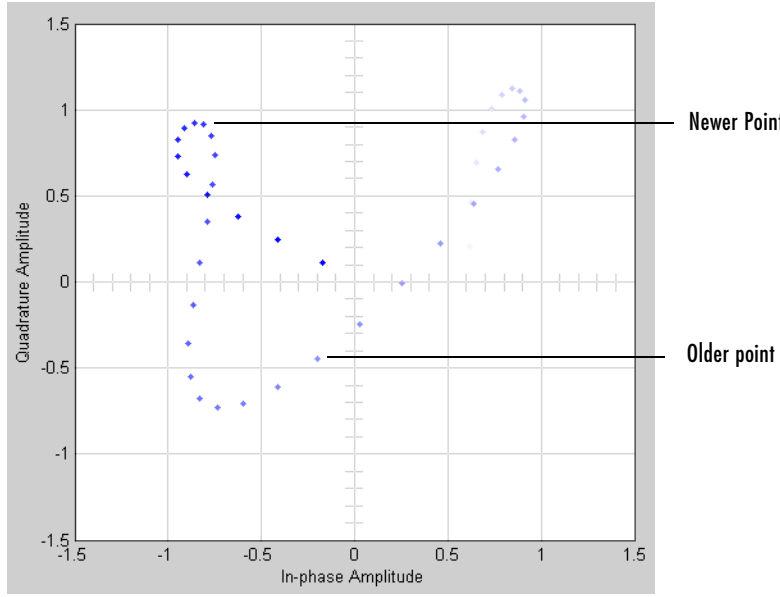
    Samples per symbol specifies the number of samples in each channel symbol. Each sample corresponds to a plotted point in the scatter plot.
  - Set **Offset (samples)** to `0`.

    Offset (samples) specifies the number of samples to skip before plotting the first point.
  - Set **Points displayed** to `40`.

    Points displayed specifies the number of points that are displayed at any instant.
  - Set **New points per display** to `10`.

    New points per display specifies the number of new points that appear each time the diagram is refreshed. The number of points that remain in the diagram from one refresh to the next is **Points displayed** minus **New points per display**.

When you run the model for 50 seconds, the Discrete-Time Scatter Plot Scope block displays the following plot.

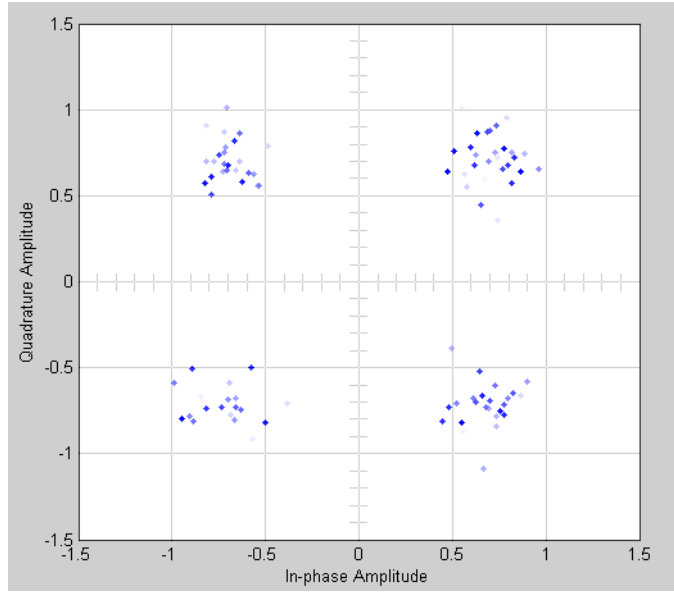


The plot displays 30 points. Because the **Color fading** check box in the **Rendering Properties** group is selected, points are displayed less brightly the older they are.

For another example, see Example: Viewing a Sinusoid.

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



Markers and Color

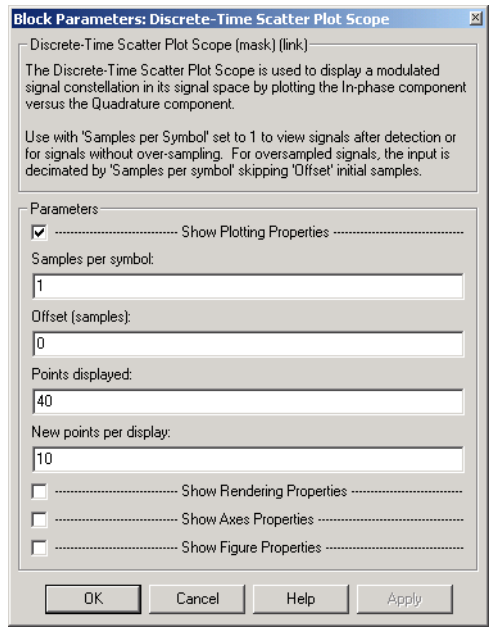
The **Markers** and **Color** parameters, which are visible when **Rendering Properties** is checked, specify the style and color of markers in the scatter plot. For details on the options for these parameters, see the reference page for the Discrete-Time Eye Diagram Scope block.

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.
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

Dialog Box



Show Plotting Properties

Select to display plotting properties.

Samples per symbol

Number of samples per symbol.

Offset (samples)

Samples to skip before plotting points.

Points displayed

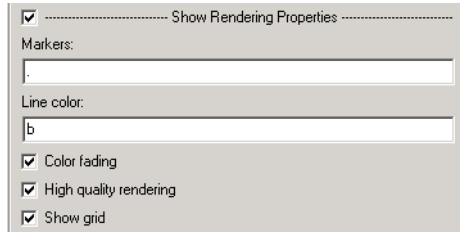
Total number of points plotted.

New points per display

Number of new points that appear in each display.

Show Rendering Properties

Select to display rendering properties, as shown in the following figure:



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

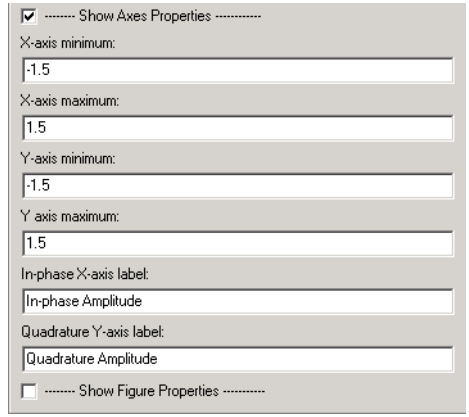
When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid

Toggles the scope grid on and off. Tunable

Show Axes Properties

Select to display axes properties, as shown in the following figure.



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

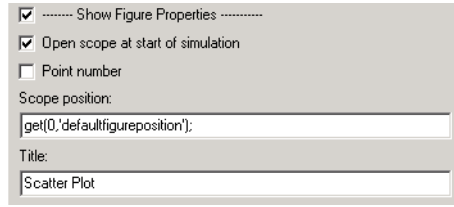
Label for *x*-axis. Tunable.

Quadrature Y-axis label

Label for *y*-axis. Tunable.

Show Figure Properties

Select to display figure properties, as shown in the following figure.



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.

The following demos in the Communications Blockset illustrate how to use the Discrete-Time Scatter Plot Scope block:

- Digital Video Broadcasting Model – `dvbt_sim.mdl`
- DS Spread Spectrum Example – `spreadspectrum.mdl`
- HiperLAN/2 – `hiperlan2.mdl`
- Phase Noise Effects in 256 QAM - `phasenoise_sim.mdl`
- Rayleigh Fading Channel – `rayleighfading.mdl`

See Also

Continuous-Time Eye and Scatter Diagrams, Discrete-Time Eye Diagram Scope, Discrete-Time Signal Trajectory Scope, Real-Imag to Complex

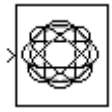
Purpose

Display a modulated signal in its signal space by plotting its in-phase component versus its quadrature component

Library

Comm Sinks

Description

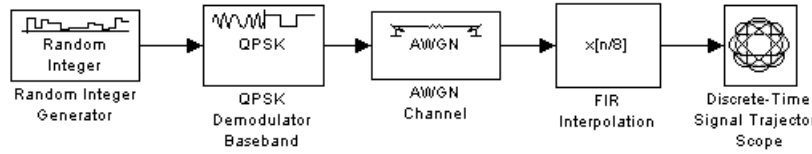


The Discrete-Time Signal Trajectory Scope displays the trajectory of a modulated signal in its signal space by plotting its in-phase component versus its quadrature component.

The Discrete-Time Signal Trajectory Scope block has one input port. The input signal must be complex. The input signal must be a sample-based scalar in sample-based mode. The input must be a frame-based column vector or a scalar in frame-based mode.

Example: Viewing the Signal Trajectory of a Modulated Signal

The following model shows the signal trajectory of a modulated signal. The model modulates a random signal using the QPSK Modulator block and then filters the signal with a raised cosine interpolation filter.



To build the model, gather and configure the following blocks:

- Random Integer Generator, in the Data Sources sublibrary of the Comm Sources library, with default parameters
- QPSK Modulator Baseband, in PM in the Digital Baseband sublibrary of the Modulation library of the Communications Blockset, with default parameters
- AWGN Channel, in the Channels library of the Communications Blockset, with the following changes to the default parameter settings:
  - Set **Mode** to **Signal-to-noise ratio (SNR)**.
  - Set **SNR (dB)** to 15.

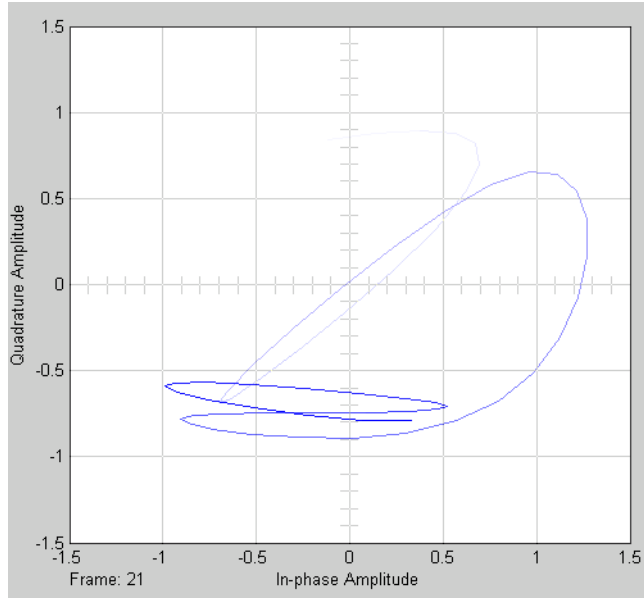
- FIR Interpolation, in the Multirate Filters sublibrary of the Filtering library of the DSP Blockset, with the following changes to the default parameter settings:
  - Set **FIR Filter Coefficients** to `rcosine(1, 8,[], 0.5,3)`.
  - Set **Interpolation factor** to `8`.
- Discrete-Time Signal Trajectory Scope, in the Comms Sinks library, with the following changes to the default parameter settings:
  - Set **Samples per symbol** to `8`.

    Samples per symbol specifies the number of samples in each channel symbol. Each sample corresponds to a plotted point in the signal trajectory.
  - Set **Symbols displayed** to `7`.

    Symbols displayed specifies the number of symbols displayed in the signal trajectory. The total number of points displayed is the product of **Samples per symbol** and **Symbols displayed**.
  - Set **New symbols per display** to `10`.

    New symbols per display specifies the number of new symbols that appear each time the diagram is refreshed. The number of symbols that remain in the diagram from one refresh to the next is **Symbols displayed** minus **New symbols per display**.

When you run the model for 50 seconds, the Discrete-Time Signal Trajectory Scope displays the following trajectory.

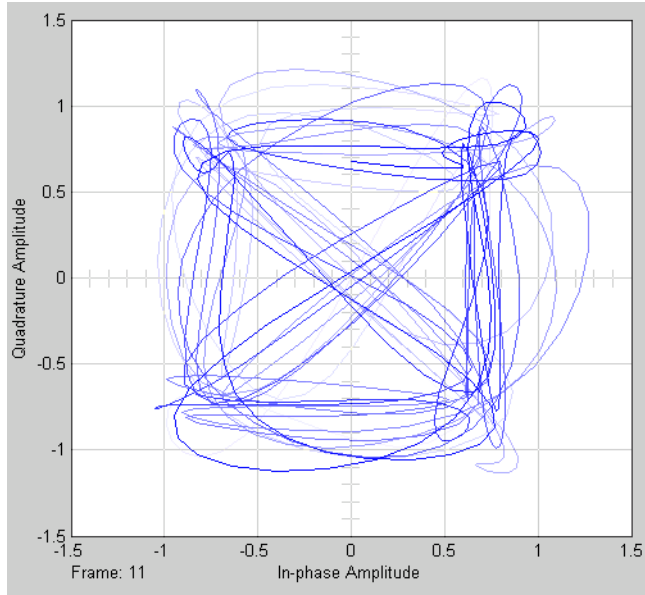


The plot displays 40 symbols. Because the **Color fading** check box in the **Rendering Properties** group is selected, symbols are displayed less brightly the older they are.

For another example, see Example: Viewing a Sinusoid.

See the reference page for the Discrete-Time Scatter Plot Scope block to compare the preceding signal trajectory with the scatter plot 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.

If you increase **Symbols displayed** to 100, the model produces the following signal trajectory.



The total number of points displayed at any instant is 800, which is the product of the parameters **Samples per symbol** and **Symbols displayed**.

For another example, see Example: Viewing a Sinusoid.

Line Style and Color

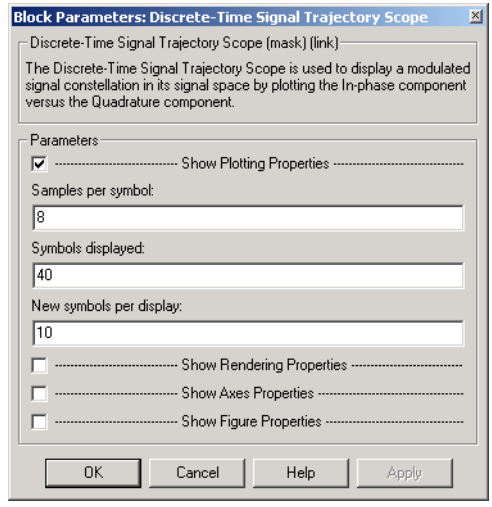
The **Line style** and **Line color** parameters in the **Rendering Properties** group control the appearance of the signal trajectory. The **Line style** parameter specifies the style for lines in the signal trajectory. For details on the options for these parameters, see the reference page for the 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.
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

Dialog Box



Show Plotting Properties

Select to display plotting properties.

Samples per symbol

Number of samples per symbol.

Symbols displayed

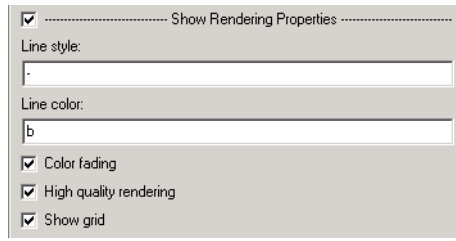
Total number of symbols plotted.

New symbols per display

Number of new symbols that appear in each display.

Show Rendering Properties

Select to display rendering properties, as shown in the following figure.



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

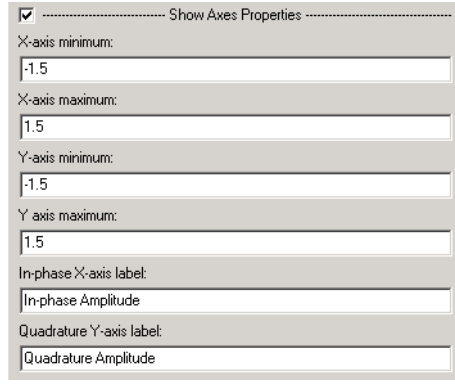
When selected, the block renders a slow, higher-quality picture with overwrite raster operations. When cleared, the block renders a fast, lower-quality picture with XOR raster operations. Tunable.

Show grid

Toggles the scope grid on and off. Tunable.

Show Axes Properties

Select to display axes properties, as shown in the following figure.



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

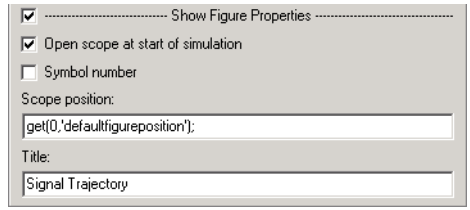
Label for *x*-axis. Tunable.

Quadrature Y-axis label

Label for *y*-axis. Tunable.

Show Figure Properties

Select to display figure properties, as shown in the following figure.



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.

The following demos in the Communcations Blockset illustrate how to use the Discrete-Time Signal Trajectory Scope:

- Filtered Offset QPSK vs. Filtered QPSK – `foqpskvsfqpsk.mdl`
- GMSK vs. MSK – `gmskvsmsk.mdl`

See Also

Continuous-Time Eye and Scatter Diagrams, Discrete-Time Eye Diagram Scope, Discrete-Time Scatter Plot Scope

Discrete-Time VCO

Purpose

Implement a voltage-controlled oscillator in discrete time

Library

Controlled Sources sublibrary of Comm Sources

Description

The Discrete-Time VCO (voltage-controlled oscillator) block generates a signal whose frequency shift from the **Oscillation 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

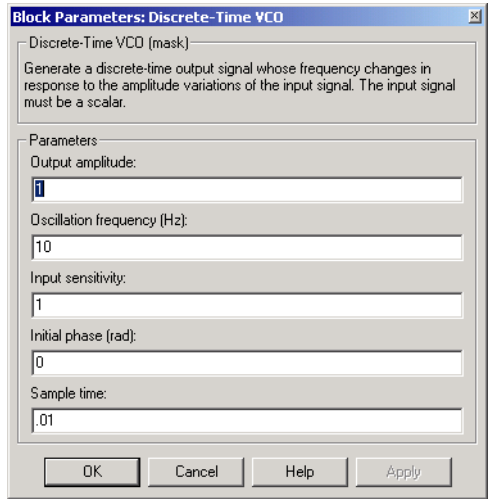
$$y(t) = A_c \cos(2\pi f_c t + 2\pi k_c \int_0^t u(\tau) d\tau + \varphi)$$

where A_c is the **Output amplitude**, f_c is the **Oscillation 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 are both scalars.

Dialog Box



Output amplitude

The amplitude of the output.

Oscillation 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 **Oscillation 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

Voltage-Controlled Oscillator

DPCM Decoder

Purpose

Decode differential pulse code modulation

Library

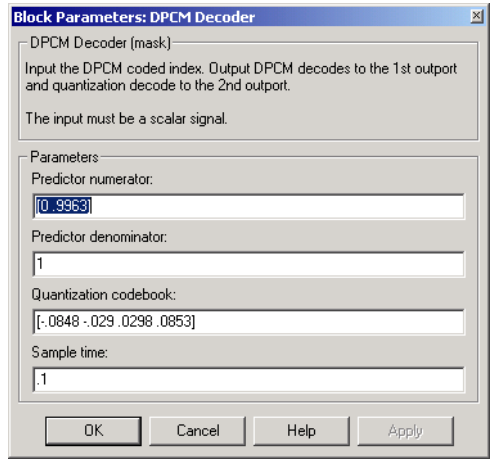
Source Coding

Description

The DPCM Decoder block recovers a message from a quantized signal using differential pulse code demodulation. The input represents a DPCM-encoded quantization index. The input must be a scalar signal. Its two outputs are the recovered signal and the quantized predictive error.

The description of the Sampled Quantizer Encode block gives more detailed information about quantization indices and quantization-encoded signals. The description of the DPCM Encoder block gives more information about implementing DPCM.

Dialog Box



Predictor numerator

The vector of coefficients of the numerator of the predictor transfer function, in order of ascending powers of z^{-1}. The first entry must be zero.

Predictor denominator

The vector of coefficients of the denominator of the predictor transfer function, in order of ascending powers of z^{-1}. Usually this parameter is 1.

Quantization codebook

The vector of output values that the quantizer assigns to each partition.

Sample time

The block's sample time.

Match these parameters to the ones in the corresponding DPCM Encoder block.

Pair Block

DPCM Encoder

References

[1] Kondoz, A. M. *Digital Speech*. Chichester, England: John Wiley & Sons, 1994.

DPCM Encoder

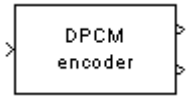
Purpose

Encode using differential pulse code modulation

Library

Source Coding

Description



The DPCM Encoder block quantizes the input signal using differential pulse code modulation. The input must be a scalar signal. Its two outputs are the quantization index and the quantization-encoded signal.

This block uses the Sampled Quantizer Encode block. The description of that block gives more detailed information about quantization indices and quantization-encoded signals.

Quantization partition is a vector whose entries give the endpoints of the partition intervals. **Quantization codebook**, a vector whose length exceeds the length of **Quantization partition** by one, 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 **Quantization partition**.

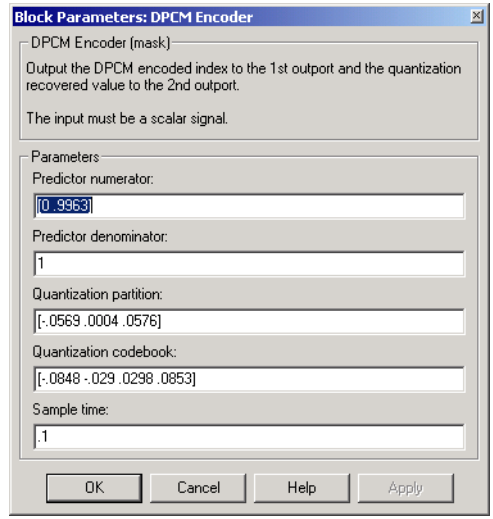
You can think of the predictor as a transfer function for an IIR filter, hence a rational function of z^{-1}. Specify the predictor's numerator and denominator by listing their coefficients in the vectors **Predictor numerator** and **Predictor denominator**, respectively. List the coefficients in order of increasing powers of z^{-1}.

Note The first entry of **Predictor numerator** must be zero. A nonzero entry there would fail to make sense conceptually, and would create an algebraic loop in the implementation.

You can use the function `dpcmopt` in the Communications Toolbox to train the **Predictor numerator**, **Predictor denominator**, **Quantization partition**, and **Quantization codebook** parameters. The output of `dpcmopt` omits the denominator of the predictor, assuming that it will be 1. In most DPCM applications, the denominator of the predictor transfer function is 1.

If **Predictor numerator** has the form [0, x] and **Predictor denominator** is 1, then the modulation is called *delta modulation*.

Dialog Box



Predictor numerator

The vector of coefficients of the numerator of the predictor transfer function, in order of ascending powers of z^{-1}. The first entry must be zero.

Predictor denominator

The vector of coefficients of the denominator of the predictor transfer function, in order of ascending powers of z^{-1}. Usually this parameter is 1.

Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values that the quantizer assigns to each partition.

Sample time

The block's sample time.

Pair Block

DPCM Decoder

References

[1] Kondoz, A. M. *Digital Speech*. Chichester, England: John Wiley & Sons, 1994.

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.

Outputs and Constellation Types

If the **Output type** parameter is set to **Integer**, then the block maps a phase difference of

$$\theta + \pi m/2$$

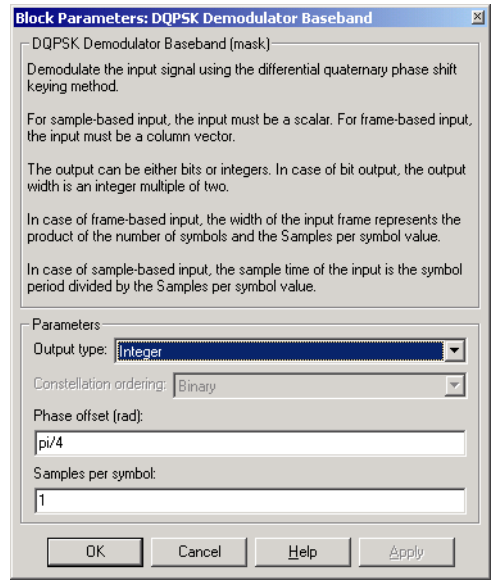
to m, where θ is the **Phase offset** parameter and m is 0, 1, 2, or 3.

If the **Output type** parameter is set to **Bit**, then the output contains pairs of binary values. The reference page for the DQPSK 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**.

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 Using the Communications Blockset.

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 offset (rad)

This phase difference between the current and previous modulated symbols results in an output of zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

DQPSK Modulator Baseband

See Also

M-DPSK Demodulator Baseband, DBPSK Demodulator Baseband, QPSK Demodulator Baseband

DQPSK Modulator Baseband

Purpose

Modulate using the 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.

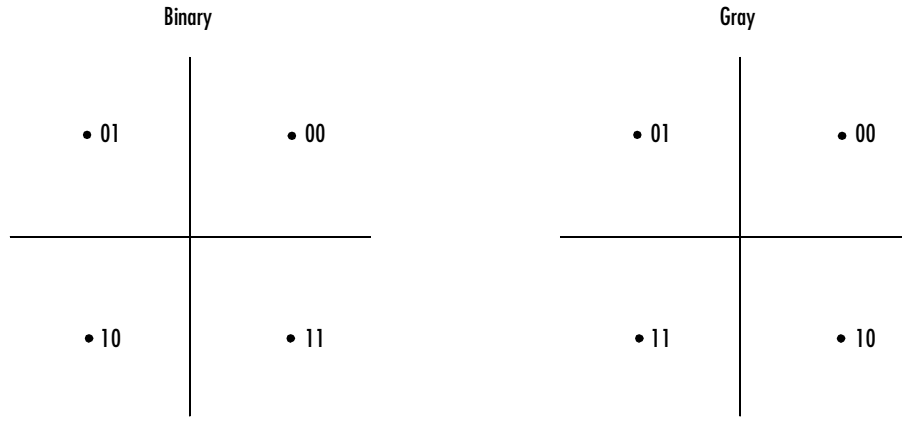
Inputs and Constellation Types

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

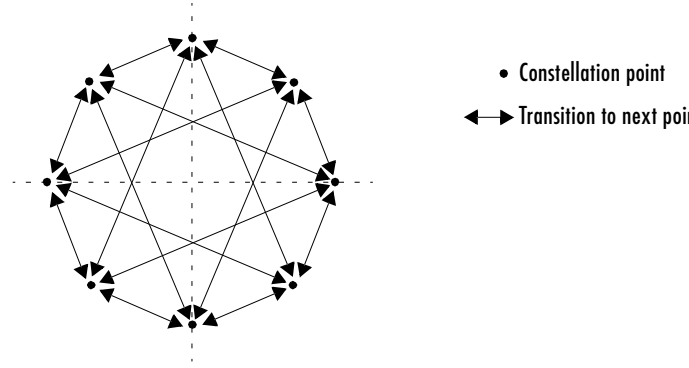
$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** 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 offset** parameter is set to `pi/4`; in other cases, the two schematics would be rotated accordingly.



The figure below shows the signal constellation for the DQPSK modulation method when the **Phase offset** 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.

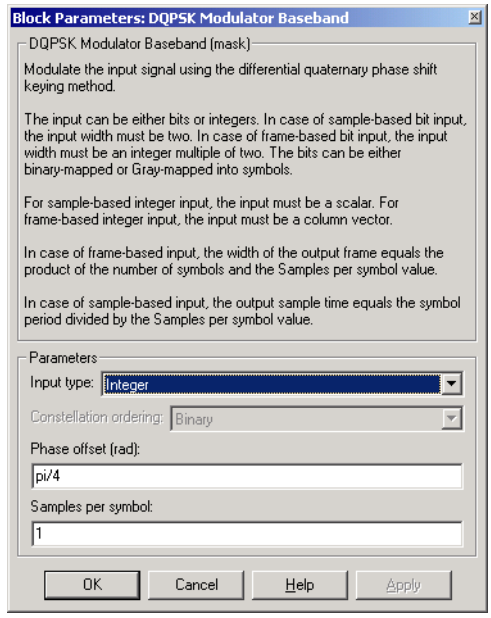


More generally, if the **Phase offset** parameter has the form π/k for some integer k, then the signal constellation has 2k points.

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 Using the Communications Blockset.

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 offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

Pair Block

DQPSK Demodulator Baseband

See Also

M-DPSK Modulator Baseband, DBPSK Modulator Baseband, QPSK Modulator Baseband

DSB AM Demodulator Baseband

Purpose

Demodulate DSB-AM-modulated data

Library

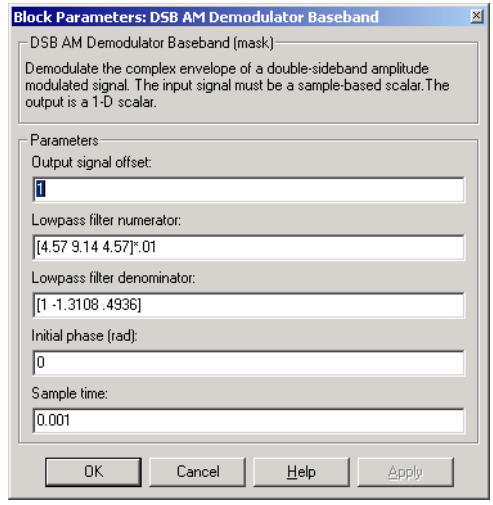
Analog Baseband Modulation, in Modulation

Description

The DSB AM Demodulator Baseband block demodulates a signal that was modulated using double-sideband amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Output signal offset

The same as the **Input signal offset** parameter in the corresponding DSB AM Modulator Baseband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Initial phase (rad)

The initial phase in the corresponding DSB AM Modulator Baseband block.

Sample time

The sample time of the output signal.

Pair Block

DSB AM Modulator Baseband

DSB AM Demodulator Passband

Purpose

Demodulate DSB-AM-modulated data

Library

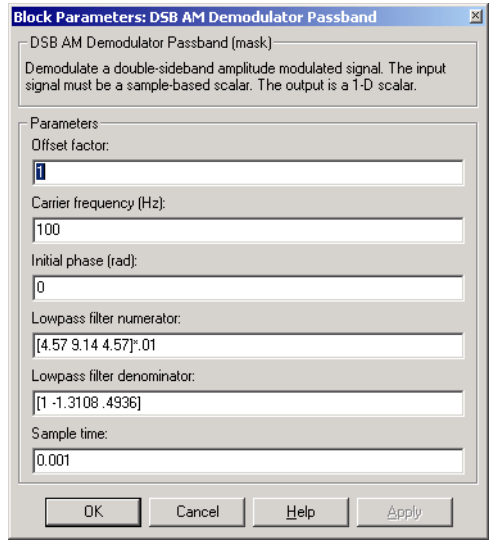
Analog Passband Modulation, in Modulation

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 transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Block



Offset factor

The same as the **Input signal offset** parameter in the corresponding AM with Carrier block.

Carrier frequency (Hz)

The frequency of the carrier in the corresponding AM with Carrier block.

Initial phase (rad)

The initial phase of the carrier in radians.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Sample time

The sample time of the output signal.

Pair Block

DSB AM Modulator Passband

See Also

DSB AM Demodulator Baseband

DSB AM Modulator Baseband

Purpose

Modulate using double-sideband amplitude modulation

Library

Analog Baseband Modulation, in Modulation

Description

The DSB AM Modulator Baseband block modulates using double-sideband amplitude modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

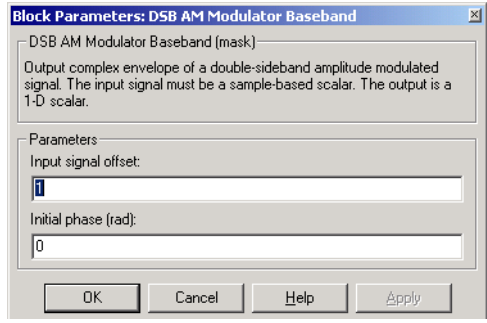
If the input is $u(t)$ as a function of time t, then the output is

$$(u(t) + k)e^{j\theta}$$

where:

- θ is the **Initial phase** parameter.
- k is the **Input signal offset** parameter.

Dialog Box



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.

Initial phase (rad)

The phase of the modulated signal.

Pair Block

DSB AM Demodulator Baseband

See Also

DSBSC AM Modulator Baseband, SSB AM Modulator Baseband

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

$$(u(t)+k)\cos(2\pi f_c t+\theta)$$

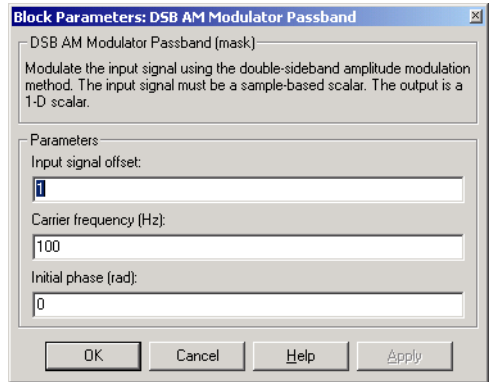
where:

- 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. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation instead of passband simulation.

Dialog Box



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

DSB AM Modulator Baseband, DSBSC AM Modulator Passband, SSB AM Modulator Passband

Purpose

Demodulate DSBSC-AM-modulated data

Library

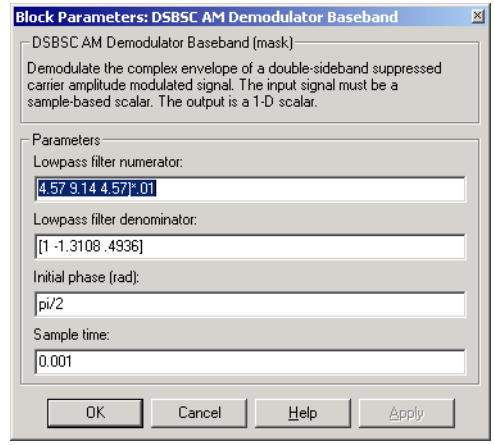
Analog Baseband Modulation, in Modulation

Description

The DSBSC AM Demodulator Baseband block demodulates a signal that was modulated using double-sideband suppressed-carrier amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Initial phase (rad)

The initial phase in the corresponding DSBSC AM Modulator Baseband block.

Sample time

The sample time of the output signal.

Pair Block

DSBSC AM Modulator Baseband

See Also

DSB AM Demodulator Baseband, SSB AM Demodulator Baseband

Purpose

Demodulate DSBSC-AM-modulated data

Library

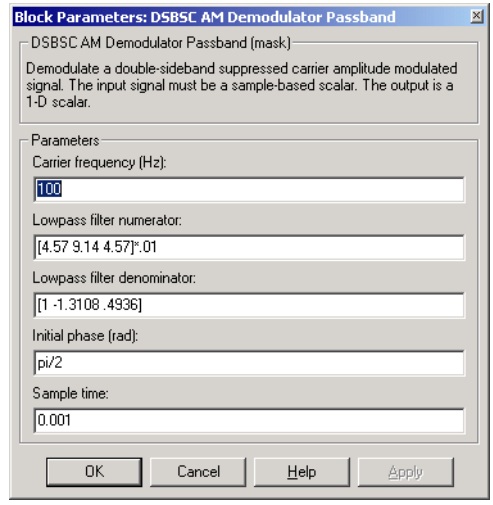
Analog Passband Modulation, in Modulation

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 transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Carrier frequency (Hz)

The carrier frequency in the corresponding DSBSC AM Modulator Passband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Initial phase (rad)

The initial phase of the carrier in radians.

Sample time

The sample time of the output signal.

Pair Block

DSBSC AM Modulator Passband

See Also

DSBSC AM Demodulator Baseband, DSB AM Demodulator Passband, SSB AM Demodulator Passband

Purpose

Modulate using double-sideband suppressed-carrier amplitude modulation

Library

Analog Baseband Modulation, in Modulation

Description

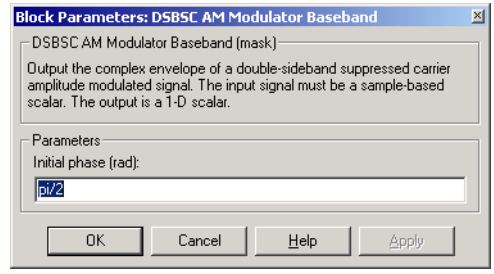
The DSBSC AM Modulator Baseband block modulates using double-sideband suppressed-carrier amplitude modulation. The output is a baseband representation of the modulated signal. The block accepts a real input signal and produces a complex output signal. The input may be continuous-time or discrete-time; the output sample time matches the input sample time. The input must be a sample-based scalar signal.

If the input is $u(t)$ as a function of time t, then the output is

$$u(t)e^{j\theta}$$

where θ is the **Initial phase** parameter.

Dialog Box



Initial phase (rad)

The phase of the modulated signal in radians.

Pair Block

DSBSC AM Demodulator Baseband

See Also

DSB AM Modulator Baseband, SSB AM Modulator Baseband

DSBSC AM Modulator 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.

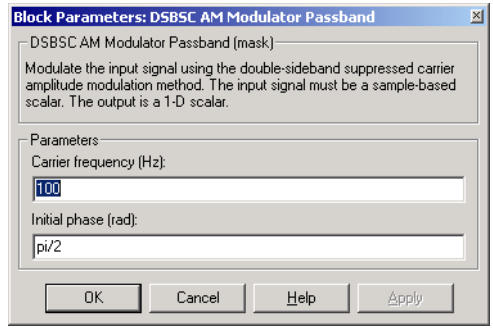
If the input is $u(t)$ as a function of time t, then the output is

$$u(t)\cos(2\pi f_c t + \theta)$$

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. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (DSBSC AM Modulator Baseband block) instead of passband simulation.

Dialog Box



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 DSBSC AM Modulator Baseband, DSB AM Modulator Passband, SSB AM Modulator Passband

Enabled Quantizer Encode

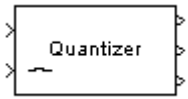
Purpose

Quantize a signal, using trigger to control processing

Library

Source Coding

Description

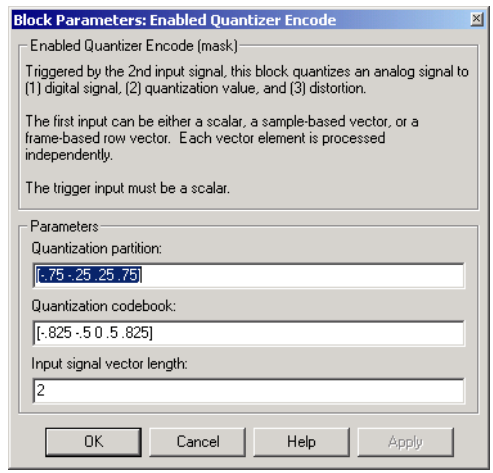


The Enabled Quantizer Encode block performs quantization when a trigger signal occurs. This block is similar to the Sampled Quantizer Encode block, except that a trigger signal at the second input port controls the quantization processing. This block renews its output when the scalar trigger signal is nonzero. For more about quantization, see the reference page for the Sampled Quantizer Encode block.

This block has two input ports and three output ports. The first input signal is the data to be quantized, while the second is the trigger signal that controls the timing of quantization. The three output signals represent the quantization index, quantization value, and mean square distortion, respectively.

The first input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the first input signal. The trigger input must be a scalar.

Dialog Box



Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values assigned to each partition.

Input signal vector length

The length of the input signal.

Pair Block

Quantizer Decode

See Also

Sampled Quantizer Encode

Error Rate Calculation

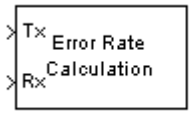
Purpose

Compute the 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.

Input Data

This block has between two and four input ports, depending on how you set the mask 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. 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 DSP Blockset's Repeat block using the **Maintain input frame rate** option.)

If you check the **Reset port** box in the mask, then an additional inport appears, labeled `Rst`. The `Rst` input must be a sample-based scalar signal 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** mask 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.

The guidelines below indicate how you should configure the inputs and the mask 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 mask. 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** mask 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. (See the Zero Pad block in the DSP Blockset.) 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 workspace or to an output port, depending on how you set the **Output data** parameter in the mask:

- If you set the **Output data** parameter to **Workspace** and fill in the **Variable name** parameter, then that variable 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, 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.

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 **Simulation 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

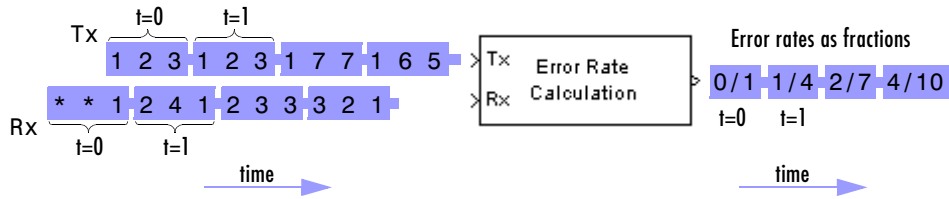
The figure below shows how the block compares pairs of elements and counts the number of error events. This example assumes that the sample time of each input signal is 1 second and that the block's parameters are as follows:

- **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.

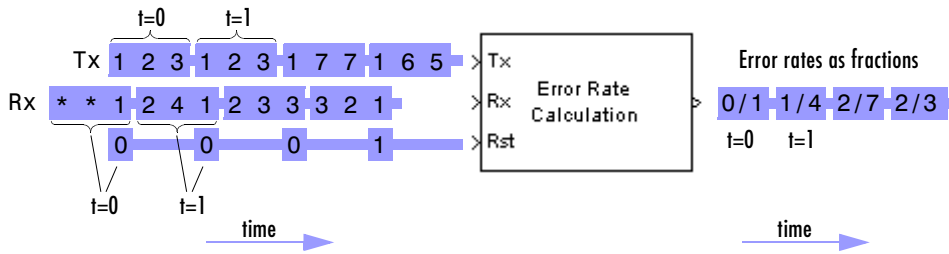
The values of the first two elements of Rx appear as asterisks because they do not influence the output. Similarly, the 6 and 5 in the Tx signal do not influence the output up to time 3, though they *would* influence the output at time 4.

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*.



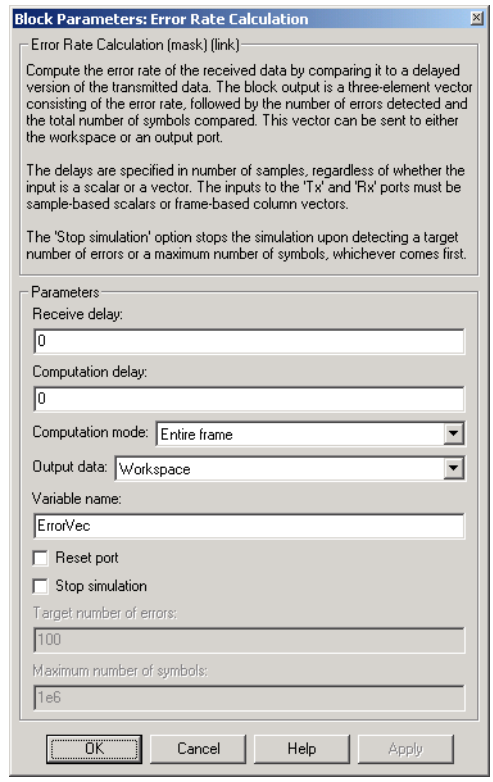
Note: Tx and Rx inputs are frame-based column vectors.

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.

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 workspace variable for the output data vector. 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.

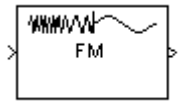
Purpose

Demodulate FM-modulated data

Library

Analog Baseband Modulation, in Modulation

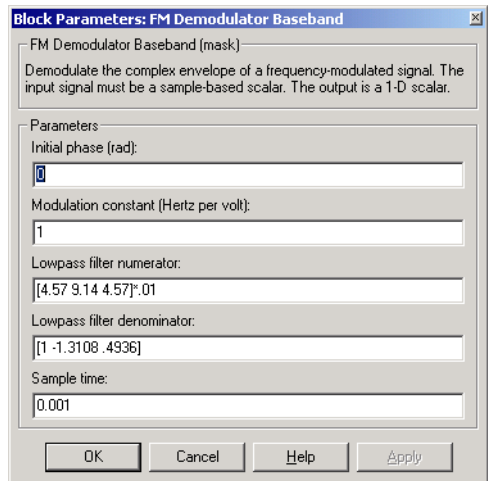
Description



The FM Demodulator Baseband block demodulates a signal that was modulated using frequency modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Initial phase (rad)

The initial phase in the corresponding FM Modulator Baseband block.

Modulation constant (Hertz per volt)

The modulation constant in the corresponding FM Modulator Baseband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Sample time

The sample time of the output signal.

Pair Block

FM Modulator Baseband

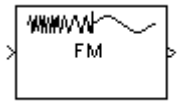
Purpose

Demodulate FM-modulated data

Library

Analog Passband Modulation, in Modulation

Description

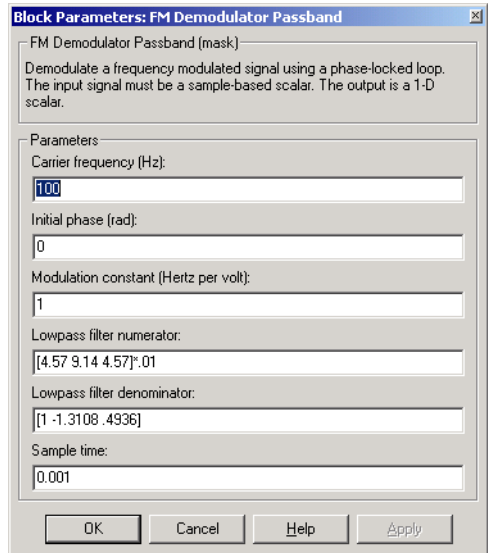


The FM Demodulator Passband block demodulates a signal that was modulated using frequency modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

The block uses a voltage-controlled oscillator (VCO) in the demodulation. The **Initial phase** parameter gives the initial phase of the VCO.

Dialog Box



Carrier frequency (Hz)

The carrier frequency in the corresponding FM Modulator Passband block.

Initial phase (rad)

The initial phase of the VCO in radians.

Modulation constant (Hertz per volt)

The modulation constant in the corresponding FM Modulator Passband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Sample time

The sample time in the corresponding FM Modulator Passband block.

Pair Block

FM Modulator Passband

See Also

FM Demodulator Baseband

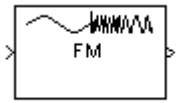
Purpose

Modulate using frequency modulation

Library

Analog Baseband Modulation, in Modulation

Description



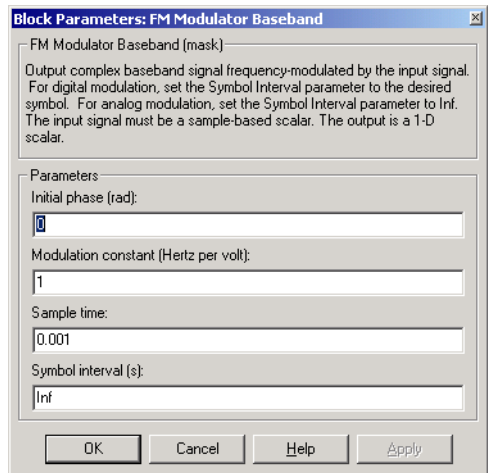
The FM Modulator Baseband block modulates using frequency modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal. In the frequency modulation technique, the frequency of the modulated signal varies according to the amplitude of the input signal.

If the input is $u(t)$ as a function of time t, then the output is

$$\exp(j\theta + 2\pi j K_c \int_t u(\tau) d\tau)$$

where θ is the **Initial phase** parameter and K_c is the **Modulation constant** parameter.

Dialog Box



Initial phase (rad)

The initial phase of the modulated signal in radians.

Modulation constant (Hertz per volt)

The modulation constant K_c.

Sample time

The sample time of the output signal. It must be a positive number.

Symbol interval (s)

`Inf` by default. To use this block to model FSK, set this parameter to the length of time required to transmit a single information bit.

Pair Block

FM Demodulator Baseband

Purpose

Reduce the amplitude of the input signal by the amount specified

Library

RF Impairments

Description

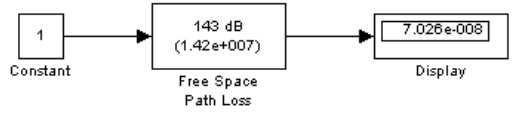
The Free Space Path Loss block simulates the loss of signal power due to the distance between transmitter and receiver. The block reduces the amplitude of the input signal by an amount that is determined in either of two ways:

- By the **Distance (km)** and **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

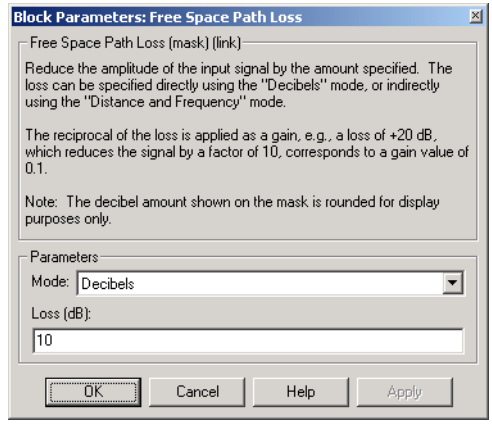
The model shown in the following figure illustrates the effect of the Free Space Path Loss Block with the following parameter settings:

Mode is set to **Distance and Frequency**.

- **Distance (km)** is set to `0.5`
- **Frequency (MHz)** is set to `180`



Dialog Box

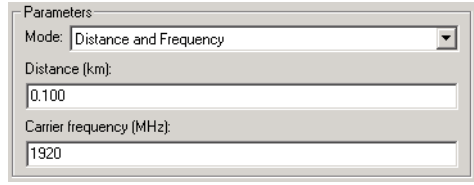


Mode

Method of specifying the amount by which the signal power is reduced. The choices are **Decibels** and **Distance and Frequency**.

Loss

The signal loss in decibels. This parameter is visible when you select **Decibels** in the **Mode** field.



Distance

Distance between transmitter and receiver in kilometers. This parameter is visible when you select **Distance and Frequency** in the **Mode** field.

Carrier frequency (MHz)

The carrier frequency in megahertz. This parameter is visible when you select **Distance and Frequency** in the **Mode** field.

See Also

Memoryless Nonlinearity

FM Modulator Passband

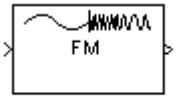
Purpose

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

$$\cos(2\pi f_c t + 2\pi K_c \int_t u(\tau) d\tau + \theta)$$

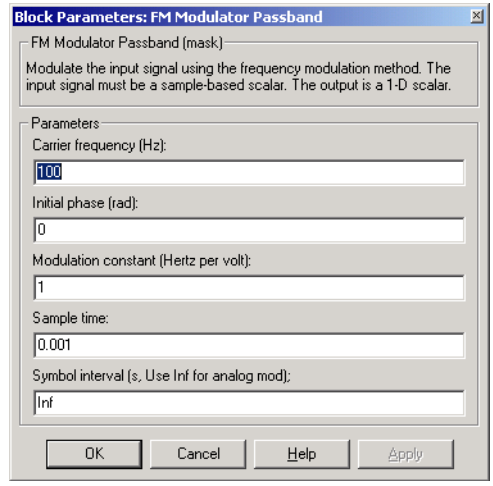
where:

- 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. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (FM Modulator Baseband block) instead of passband simulation.

By the Nyquist sampling theorem, the reciprocal of the **Sample time** parameter must exceed twice the **Carrier frequency** parameter.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The initial phase of the carrier in radians.

Modulation constant (Hertz per volt)

The modulation constant K_c.

Sample time

The sample time of the output signal. It must be a positive number.

Symbol interval

`Inf` by default. To use this block to model FSK, set this parameter to the length of time required to transmit a single information bit.

Pair Block

FM Demodulator Passband

See Also

FM Modulator Baseband

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

$$f(x) = ((2\pi)^n \det K)^{-\frac{1}{2}} \exp(-(x-\mu)^T K^{-1}(x-\mu)/2)$$

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's `randseed` function. At the MATLAB prompt, type the command

```
randseed
```

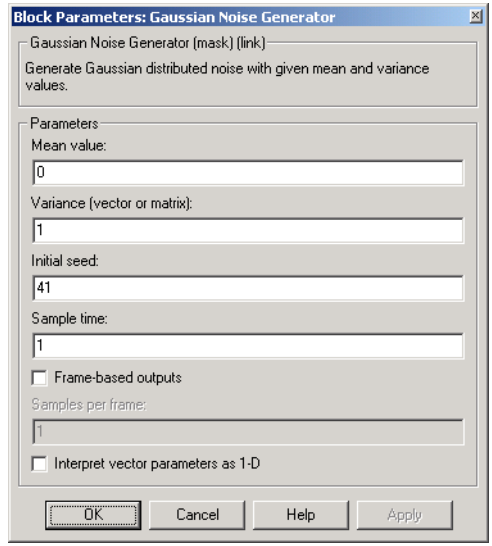
This returns a random prime number greater than 30. Typing `randseed` again produces a different prime number. If you add an integer argument, `randseed` always returns the same prime for that integer. For example, `randseed(5)` always returns the same answer.

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 Using the Communications Blockset 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.

Dialog Box



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.

See Also

Random Source (DSP Blockset), AWGN Channel, `rand` (built-in MATLAB function), `randseed`

General Block Deinterleaver

Purpose

Restore ordering of the symbols in the 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,

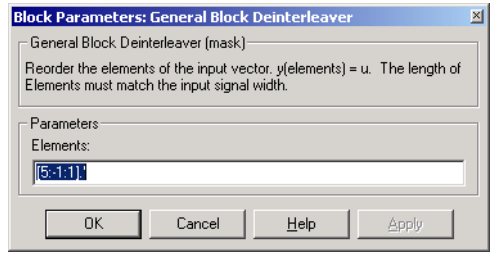
Output(**Elements**(k)) = Input(k)

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.

To use this block as an inverse of the General 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 the symbols in the input vector

Library

Block sublibrary of Interleaving

Description

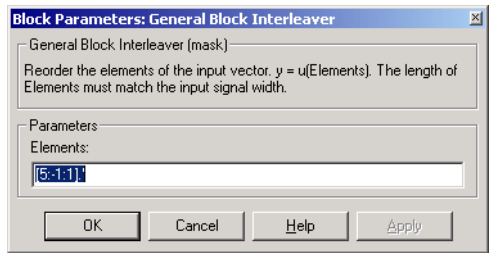
The General Block Interleaver block rearranges the elements of its input vector without repeating or omitting any elements. The input can be real or complex. If the input contains N elements, then the **Elements** parameter is a vector of length N that indicates the indices, in order, of the input elements 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.

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)

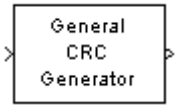
Purpose

Generate cyclic redundancy code (CRC) bits according to the generator polynomial and append them to the input data frames.

Library

CRC sublibrary of Error Correction and Detection

Description



The General CRC Generator block generates cyclic redundancy code (CRC) bits for each input data frame and appends them to the end of the frame. You specify the generator polynomial for the CRC algorithm by the **Generator polynomial** parameter in the block's mask. You represent the polynomial in either of two ways:

- As a binary row vector containing the coefficients in descending order of powers. For example, the vector `[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, the vector `[3 2 0]` represents the polynomial $x^3 + x^2 + 1$.

For a more detailed description of the CRC algorithm, see the section "Cyclic Redundancy Check Coding."

You specify the initial state of the internal shift register by the **Initial states** parameter in block's mask. 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 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

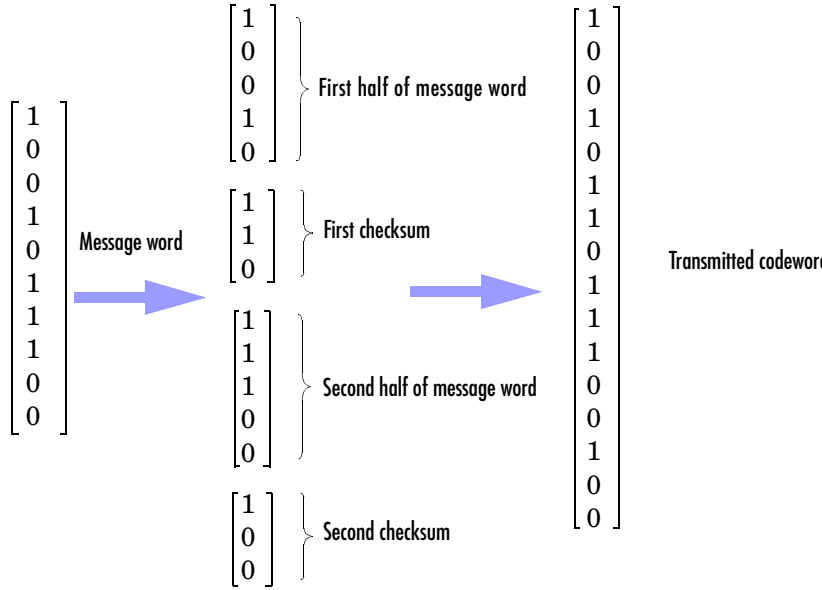
3 Applies the CRC algorithm to each augmented subframe

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.

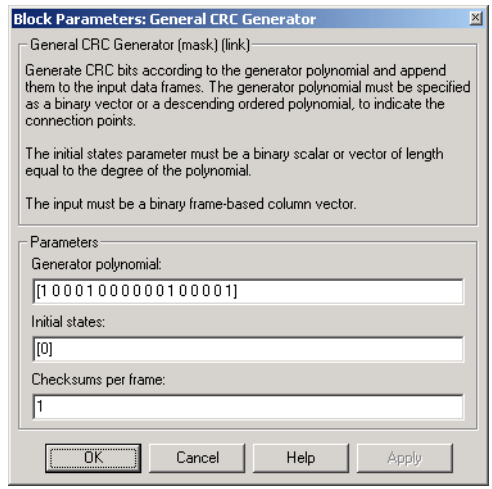
For example, suppose the size of the input frame is 10, the degree of the generator polynomial is 3, **Initial states** is set to `[0]`, and **Checksums per frame** is set to 2. The block divides each input frame into two subframes of size 5 and appends a checksum of size 3 to each subframe, as shown 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 output frame then has size 5 + 3 + 5 + 3 = 16.



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.

Pair Block

General CRC Syndrome Detector

See Also

CRC-N Generator, CRC-N Syndrome Detector

General CRC Syndrome Detector

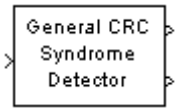
Purpose

Detect errors in the input data frames according to the generator polynomial

Library

CRC sublibrary of Error Correction and Detection

Description



The General CRC Syndrome Detector block receives a message word and removes the checksum. The block then calculates a new checksum and compares the received checksum with the new checksum. The block has two outputs. The first output is the message word with the checksum removed. The second output is a Boolean error flag, which is 0 if the received checksum agrees with the new checksum, and 1 otherwise. For a more detailed description of the CRC algorithm, see the section "Cyclic Redundancy Check Coding."

The block's parameter settings should agree with those in the General CRC Generator block.

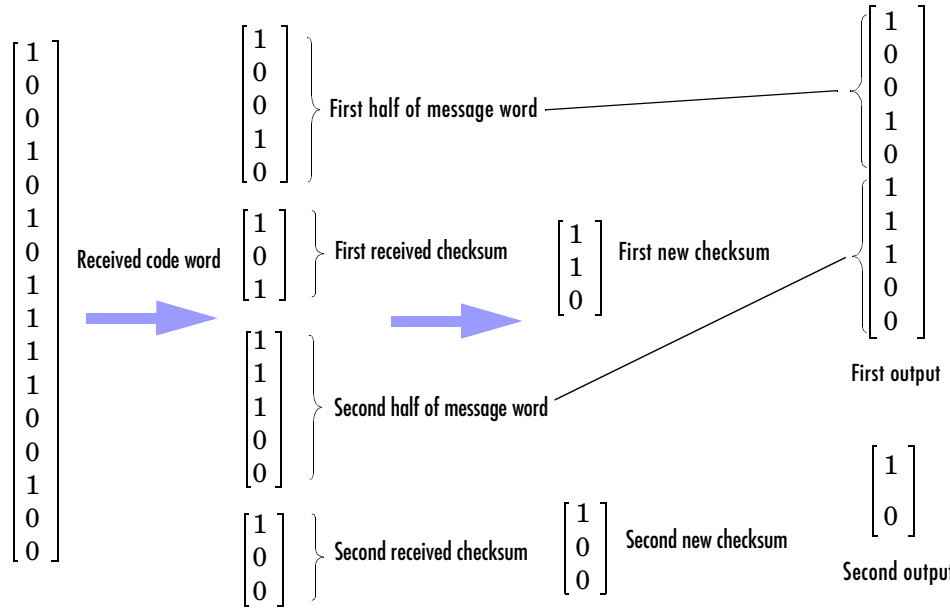
You specify the generator polynomial for the CRC algorithm by the **Generator polynomial** parameter in the block's mask. You represent the polynomial in either of two ways:

- As a binary row vector containing the coefficients in descending order of powers. For example, the vector `[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, the vector `[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 in the block's mask. 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 the block calculates for each frame by the **Checksums per frame** parameter. The parameter equals the size of the second output. 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.

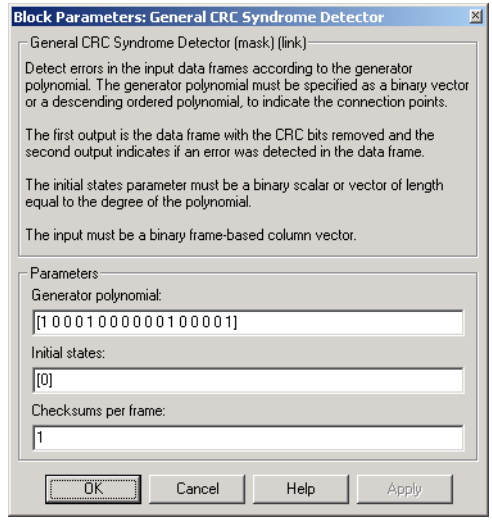
As an example, suppose the received codeword has size 16, the generator polynomial has degree 3, **Initial states** is set to `[0]`, and **Checksums per frame** is set to `2`. The block removes the two checksums of size 3, one from the end of the first half of the received codeword, and the other from the end of 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 then concatenates the two truncated halves as a single vector of size 10 and outputs this vector through the first output port. The block outputs a 2x1 Boolean frame vector, whose entries are 0 or 1, depending on whether the corresponding received and new checksums agree. The following figure shows an example in which the first checksums disagree and the second checksums agree. This indicates that an error occurred in transmitting the first half of the codeword.



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.

Pair Block

General CRC Generator

See Also

CRC-N Generator, CRC-N Syndrome Detector

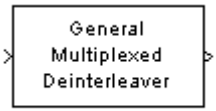
Purpose

Restore ordering of symbols using specified-delay shift registers

Library

Convolutional sublibrary of Interleaving

Description

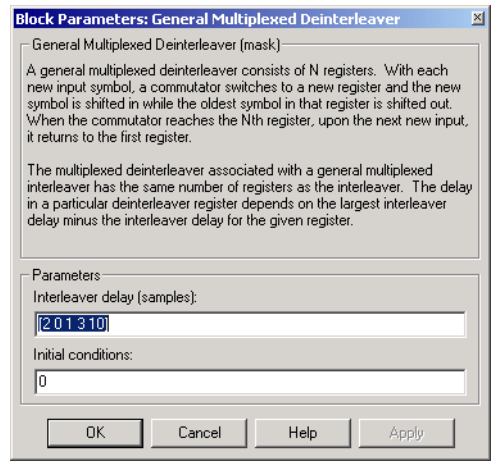


The General Multiplexed Deinterleaver block restores the original ordering of a sequence that was interleaved using the General Multiplexed Interleaver block.

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.

Dialog Box



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.

General Multiplexed Deinterleaver

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 a set of shift registers with specified delays

Library

Convolutional sublibrary of Interleaving

Description

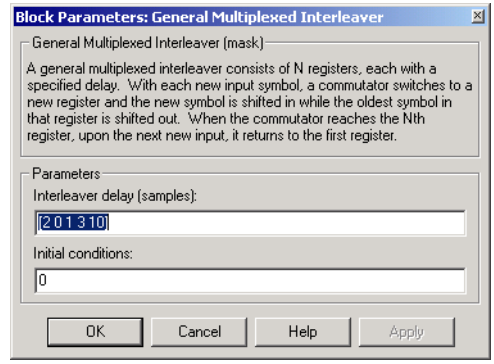
The General Multiplexed Interleaver block permutes the symbols in the input signal. Internally, it uses a set of shift registers, each with its own delay value.

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 **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.

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.

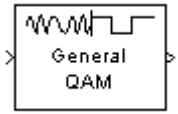
Purpose

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.

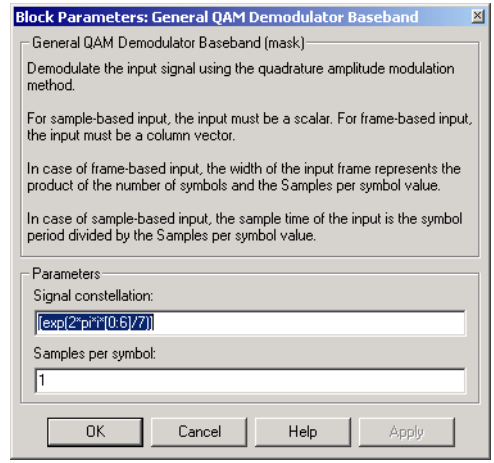
The input must be a discrete-time complex signal. The **Signal constellation** parameter defines the constellation by listing its points in a vector of complex numbers. The block maps the mth point in the **Signal constellation** vector to the integer m-1.

The input can be either a scalar or a frame-based column vector.

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 Using the Communications Blockset.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

General QAM Modulator Baseband

See Also

Rectangular QAM Demodulator Baseband

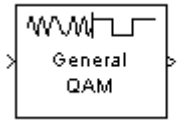
Purpose

Demodulate QAM-modulated data

Library

AM, in Digital Passband sublibrary of Modulation

Description



The General QAM Demodulator Passband block demodulates a signal that was modulated using the pulse amplitude phase shift keying method. The input is a passband representation of the modulated signal.

The input must be a sample-based scalar. Furthermore, it must be a discrete-time complex signal.

The **Signal constellation** parameter defines the constellation by listing its points in a vector of complex numbers.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

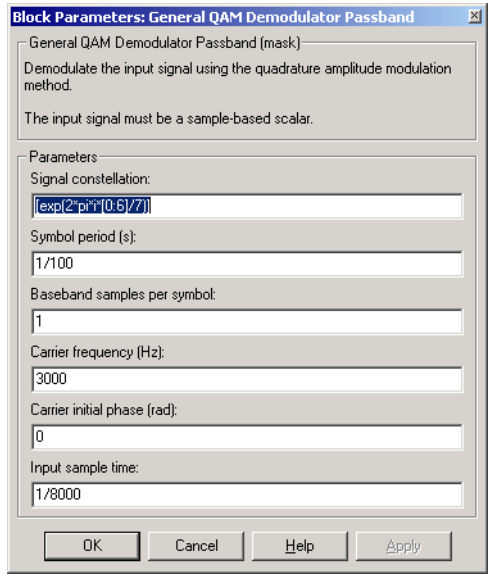
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Input sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block General QAM Modulator Passband

See Also General QAM Demodulator Baseband

General QAM Modulator Baseband

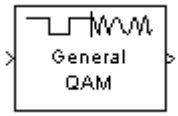
Purpose

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.

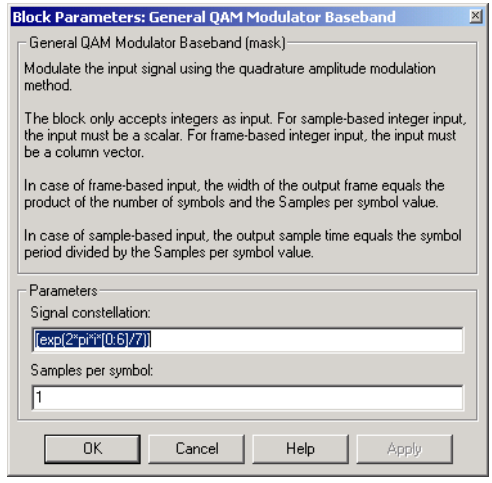
The **Signal constellation** parameter defines the constellation by listing its points in a length-M vector of complex numbers. The input signal values must be integers between 0 and M-1. The block maps an input integer m to the (m+1)st value in the **Signal constellation** vector.

The input can be either a scalar or a frame-based column vector.

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 Using the Communications Blockset.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Samples per symbol

The number of output samples that the block produces for each input integer.

Pair Block

General QAM Demodulator Baseband

See Also

Rectangular QAM Modulator Baseband

General QAM Modulator Passband

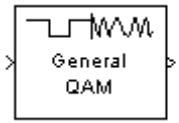
Purpose

Modulate using the pulse amplitude modulation phase shift keying method

Library

AM, in Digital Passband sublibrary of Modulation

Description



The General QAM Modulator Passband block modulates using the pulse amplitude modulation phase shift keying method. The output is a passband 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, and then converts these mapped values to a passband output signal.

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

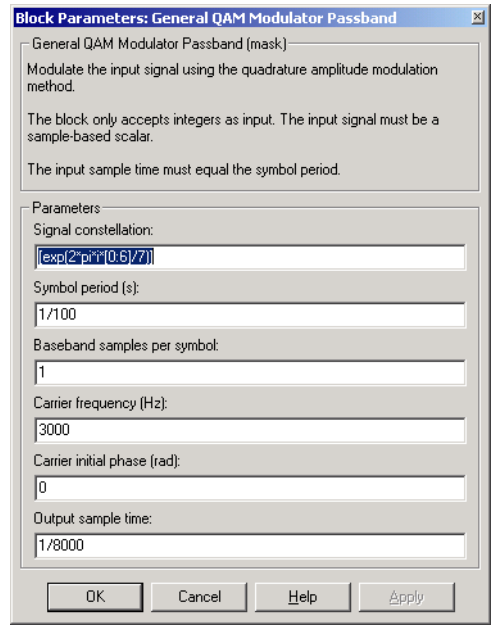
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2*__Carrier frequency__ + 2/(__Symbol period__)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)
The frequency of the carrier.

Carrier initial phase (rad)
The initial phase of the carrier in radians.

Output sample time
The sample time of the output signal.

Pair Block

General QAM Demodulator Passband

See Also

General QAM Modulator Baseband

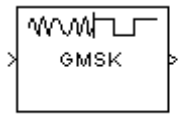
Purpose

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 the GMSK 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.

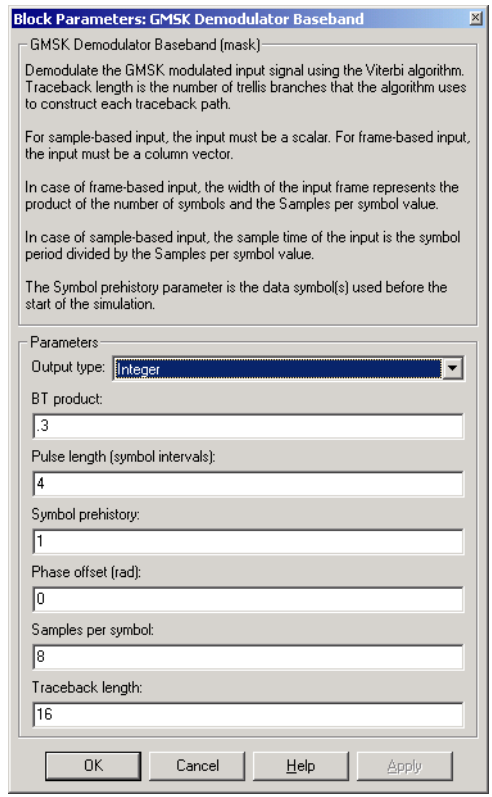
Inputs and Outputs

The input can be either a scalar or a frame-based column vector. 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 Using the Communications Blockset.

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 Viterbi Decoder block uses to construct each traceback path.

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 Demodulator Passband

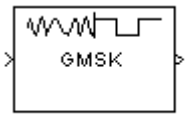
Purpose

Demodulate GMSK-modulated data

Library

CPM, in Digital Passband sublibrary of Modulation

Description



The GMSK Demodulator Passband block demodulates a signal that was modulated using the Gaussian minimum shift keying method. The input is a passband representation of the modulated signal.

This block converts the input to an equivalent baseband representation using downconversion and then FIR decimation. The block uses the baseband equivalent block, GMSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **Output type**
- **BT product**
- **Pulse length**
- **Symbol prehistory**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4

- **Output sample time** < [2*Carrier frequency + 2*F_{max})]$^{-1}$

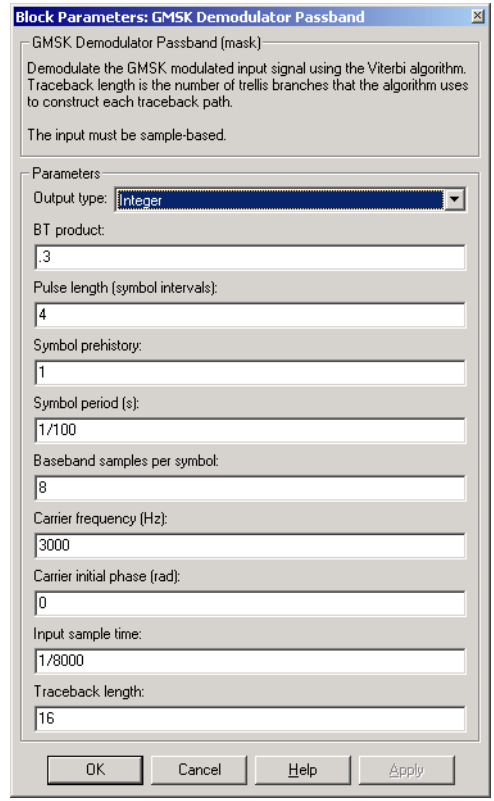
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The GMSK Demodulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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.

Symbol period (s)
The symbol period, which equals the sample time of the output.

Baseband samples per symbol
The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)
The frequency of the carrier.

Carrier initial phase (rad)
The initial phase of the carrier in radians.

Input sample time (s)
The sample time of the input signal.

Traceback length
The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

GMSK Modulator Passband

See Also

GMSK Demodulator Baseband, Viterbi Decoder

References

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

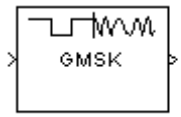
Purpose

Modulate using the 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 the exact definitions of the pulse shape, see the work by Anderson, Aulin, and Sundberg listed in "References" on page 2-283.

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 $\pi/2$ radians. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

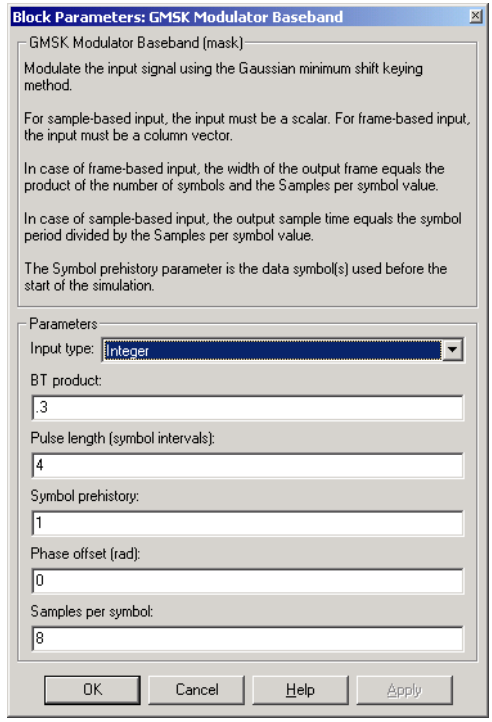
Input Attributes

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 Using the Communications Blockset.

Dialog Box



Input type

Indicates whether the input 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 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.

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.

GMSK Modulator Passband

Purpose

Modulate using the Gaussian minimum shift keying method

Library

CPM, in Digital Passband sublibrary of Modulation

Description

The GMSK Modulator Passband block modulates using the Gaussian minimum shift keying method. The output is a passband representation of the modulated signal.

This block uses the baseband equivalent block, GMSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation, using FIR interpolation and then upconversion. The following parameters in this block are the same as those of the baseband equivalent block:

- **Input type**
- **BT product**
- **Pulse length**
- **Symbol prehistory**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$

- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < [2*Carrier frequency + $2*F_{max}$)]$^{-1}$

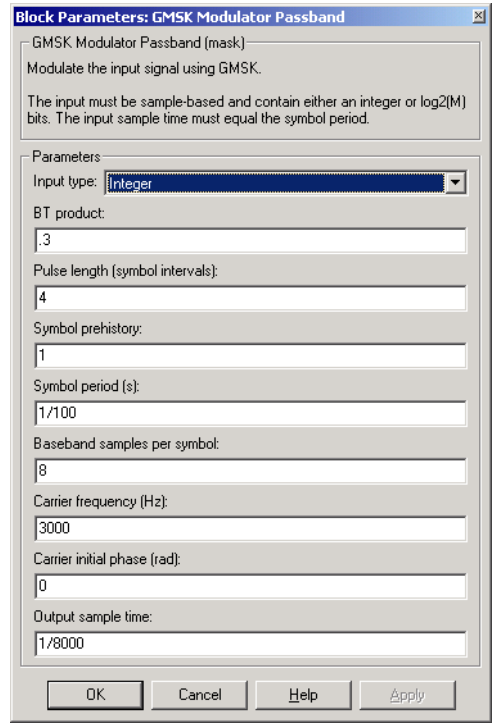
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The GMSK Modulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

Dialog Box



Input type

Indicates whether the input 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 before the start of the simulation, in reverse chronological order.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time (s)

The sample time of the output signal.

Pair Block

GMSK Demodulator Passband

See Also

GMSK Modulator Baseband

References

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

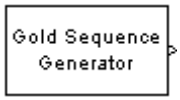
Purpose

Generate a Gold sequence from a 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.

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 the following section, "Preferred Pairs of Sequences". The set $G(u, v)$ of Gold sequences is defined by

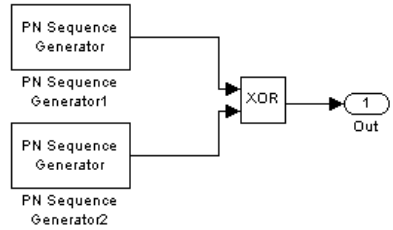
$$G(u, v) = \{u, v, u \oplus v, u \oplus Tv, u \oplus T^2 v, \ldots, u \oplus T^{N-1} v\}$$

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

$$t(n) = \begin{cases} 1 + 2^{\frac{n+1}{2}}, & \text{n odd} \\ 1 + 2^{\frac{n+2}{2}}, & \text{n even} \end{cases}$$

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.



You can specify the preferred pair by the **Preferred polynomial [1]** and **Preferred polynomial [2]** parameters in the mask 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]`
9	511	`[9 4 0]`	`[9 6 4 3 0]`

n	N	Preferred Polynomial[1]	Preferred Polynomial[2]
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 $[-2, -1, 0, 1, 2, \ldots, 2^n - 2]$. The correspondence between **Sequence index** and the output sequence is given in the following table.

Sequence Index	Output Sequence
-2	u
-1	v
0	$u \oplus v$
1	$u \oplus Tv$
2	$u \oplus T^2 v$

Sequence Index	Output Sequence
…	…
$2^n - 2$	$u \oplus T^{2^n-2}v$

You can shift the starting point of the Gold sequence with the **Shift** parameter, which is an integer representing the length of the shift.

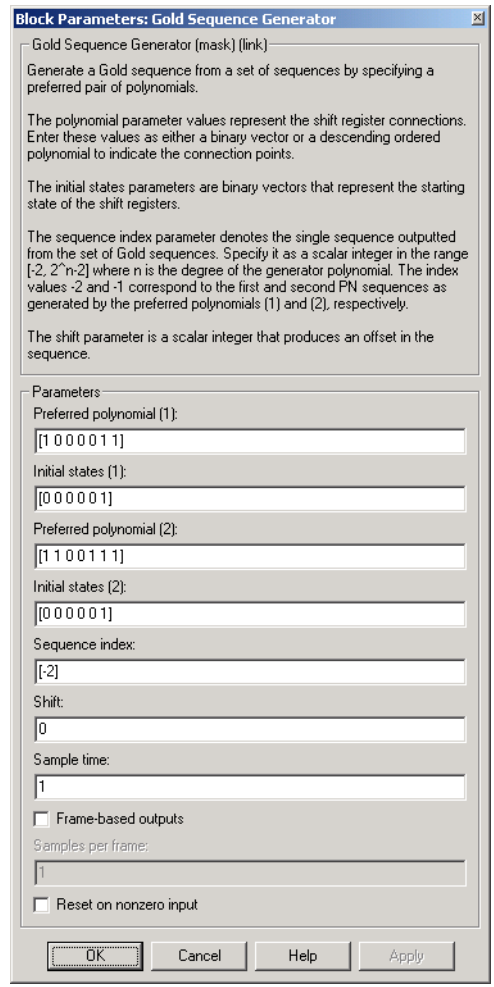
You can use an external signal to reset the values of the internal shift register to the initial state by selecting the **Reset on nonzero input** check box. This creates an input port for the external signal in the Gold Sequence Generator block. The way the block resets the internal shift register depends 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 2-461 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
- $gcd(n, k) = \begin{cases} 1, & n \equiv 1 \mod 2 \\ 2, & n \equiv 2 \mod 4 \end{cases}$

Dialog Box



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

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.

Hadamard Code Generator

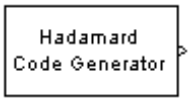
Purpose

Generate a Hadamard code from an 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.

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 $N \times N$ Hadamard matrix, denoted H_N, is defined recursively as follows.

$$H_1 = \begin{bmatrix} 1 \end{bmatrix}$$

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

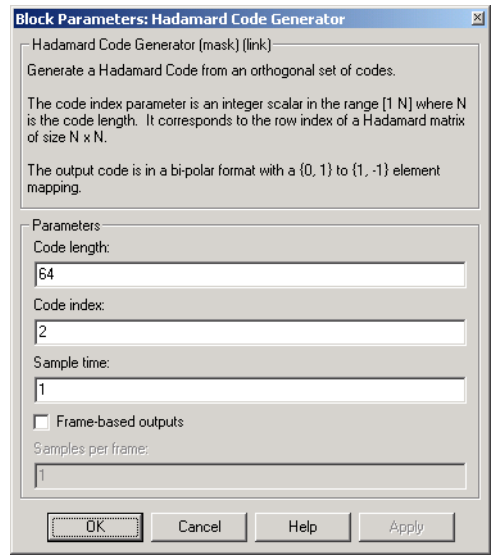
The $N \times N$ Hadamard matrix has the property that

$$H_N H_N^T = N I_N$$

where I_N is the $N \times N$ 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 in the block's mask. 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], where N is the **Code length**, by the **Code index** parameter.

Dialog Box



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.

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.

See Also

OVSF Code Generator, Walsh Code Generator

Hamming Decoder

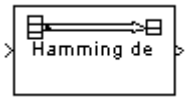
Purpose

Decode a 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.

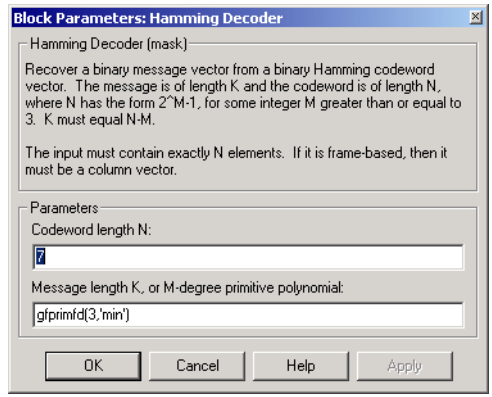
If the Hamming code has message length K and codeword length N, then N must have the form 2^M-1 for some integer M greater than or equal to 3. Also, K must equal N-M.

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 mask 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 `gfprimfd` function in the Communications Toolbox.

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)

Hamming Encoder

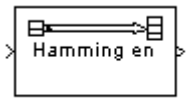
Purpose

Create a Hamming code from binary vector data

Library

Block sublibrary of Channel Coding

Description



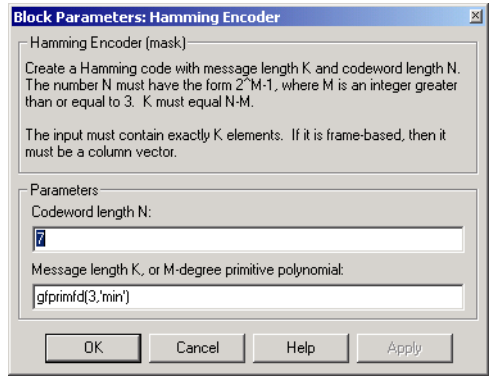
The Hamming Encoder block creates a Hamming code with message length K and codeword length N. The number N must have the form 2^M-1, where M is an integer greater than or equal to 3. Then K equals N-M.

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 mask 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 `gfprimfd` function in the Communications Toolbox.

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)

Helical Deinterleaver

Purpose

Restore ordering of symbols permuted by a 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.

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.

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

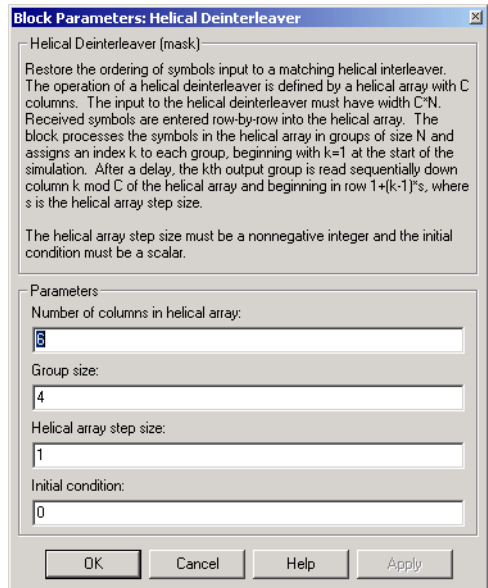
$$CN\left\lceil \frac{s(C-1)}{N} \right\rceil$$

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 Integer Delay block in the DSP Blockset 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.

Purpose

Permute input symbols using a helical array

Library

Convolutional sublibrary of Interleaving

Description

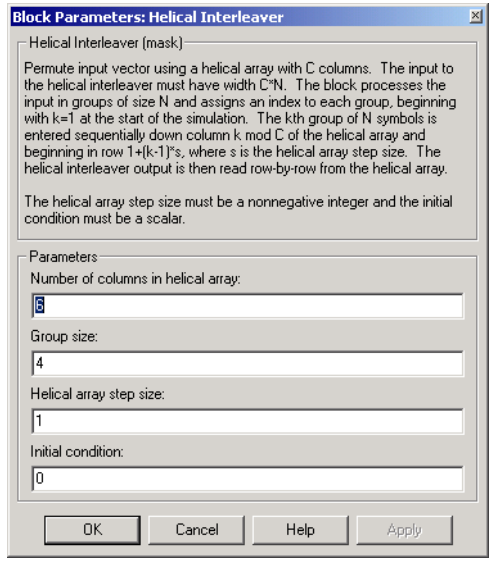
The Helical Interleaver block permutes the symbols in the input signal by placing them in an array in a helical fashion and then sending rows of the array to the output port.

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.

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

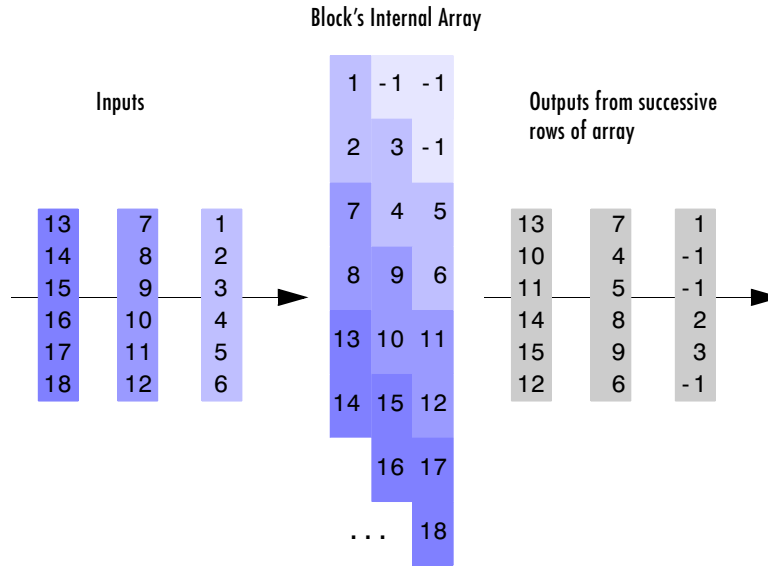
The number of rows of separation between consecutive input groups in their respective columns of the helical array.

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 `[13:19]'`, 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.)



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.

Insert Zero

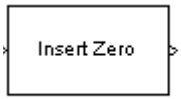
Purpose

Distribute input elements in output vector

Library

Sequence Operations, in Basic Comm Functions

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:

- 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

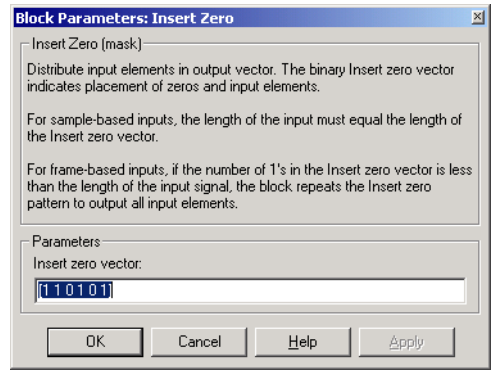
If the input signal is sample-based, then the input vector length must equal the number of 1s in the **Insert zero vector** parameter.

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.

Dialog Box



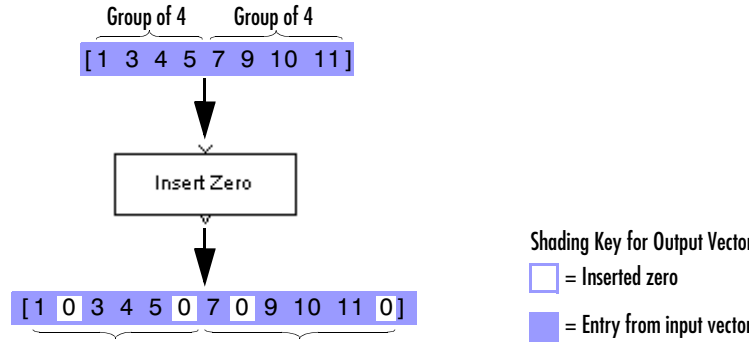
Insert zero vector

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.



Compare this example with that on the reference page for the Puncture block.

Insert Zero

See Also Puncture

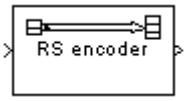
Purpose

Create a Reed-Solomon code from integer vector data

Library

Block sublibrary of Channel Coding

Description



The Integer-Input RS Encoder block creates a Reed-Solomon code with message length K and codeword length N. You specify both N and K directly in the block mask. 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 the section "Restrictions on M and the Codeword Length N" below. The difference N - K must be an even integer.

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 output is a frame-based column vector whose length is the same integer multiple of N. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon only)."

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 the section "Specifying the Primitive Polynomial" following. 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 check the box next to **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 the box next to **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 the box next to **Specify primitive polynomial**, N must lie in the range $3 < N < 2^{16} - 1$.
- If you do select the box next to **Specify primitive polynomial**, N must lie in the range $3 \le N < 2^{M} - 1$ and M must lie in the range $3 \le M \le 16$.

Specifying the Generator Polynomial

You can specify the generator polynomial for the Reed-Solomon code. To do so, first select the box next to **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

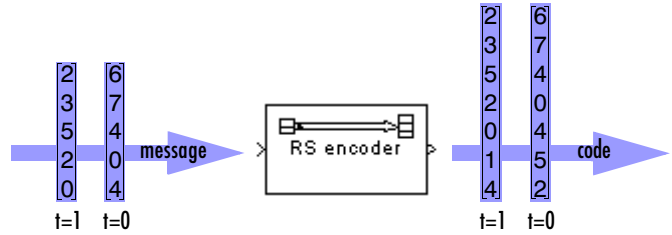
$$g(x) = (x + \alpha^{b})(x + \alpha^{b+1})(x + \alpha^{b+2})\ldots(x + \alpha^{b+N-K-1})$$

where α is the primitive element of the Galois field over which the input message is defined, and b is an integer.

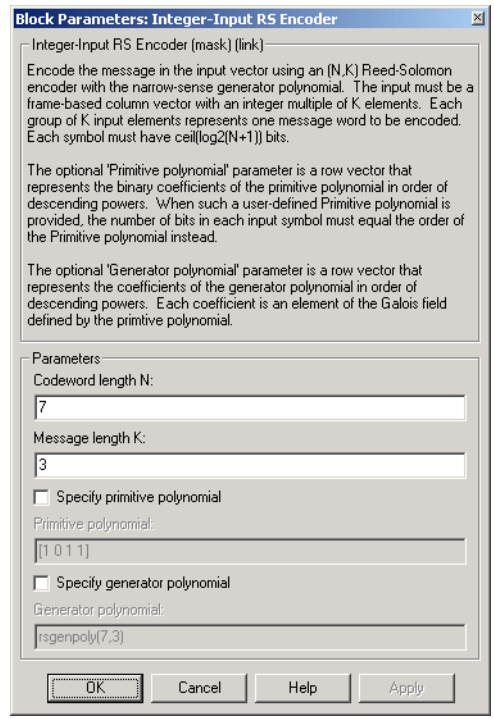
If you do not select the box next to **Specify generator polynomial**, the block uses the default generator polynomial, corresponding to b=1, for Reed-Solomon encoding. You can display the default generator polynomial by typing `rsgenpoly(N1,K1)`, where `N1 = 2M-1` and `K1 = K+(N1-N)`, at the MATLAB 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.



Dialog Box



Codeword length N

The codeword length.

Message length K

The message length.

Specify primitive polynomial

When you select this box, you can specify the primitive polynomial as a binary row vector.

Primitive polynomial

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

Specify generator polynomial

When you select this box, you can specify the generator polynomial as an integer row vector.

Generator polynomial

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

Pair Block

Integer-Output RS Decoder

See Also

Binary-Input RS Encoder

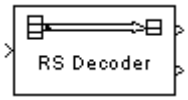
Purpose

Decode a Reed-Solomon code to recover integer vector data

Library

Block sublibrary of Channel Coding

Description



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. You specify both N and K directly in the block mask. 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 the section "Restrictions on M and the Codeword Length N" following. The difference N - K must be an even integer.

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 output is a frame-based column vector whose length is the same integer multiple of N. For more information on representing data for Reed-Solomon codes, see the section "Integer Format (Reed-Solomon only)."

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 the section "Specifying the Primitive Polynomial" 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 the section "Specifying the Generator Polynomial" on page 2-310.

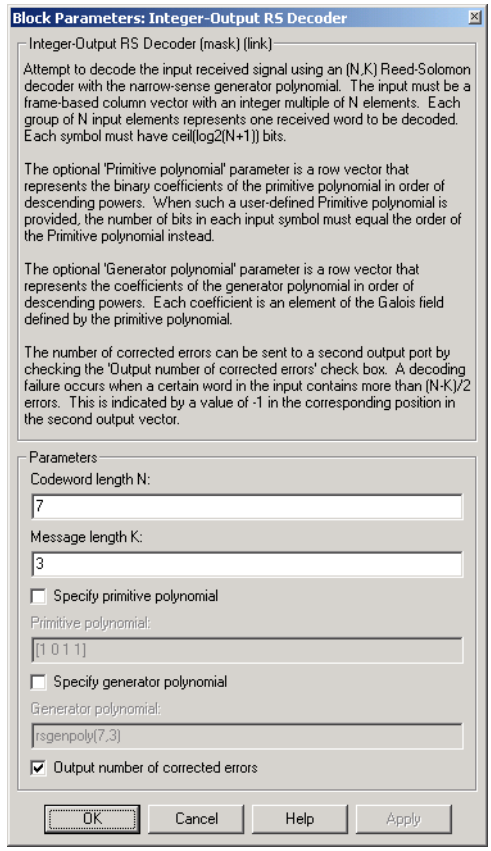
An (N, K) Reed-Solomon code can correct up to `floor((N-K)/2)` symbol errors (*not* bit errors) in each codeword.

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.

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

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

When you select this box, you can specify the primitive polynomial as a binary row vector.

Primitive polynomial

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

Specify generator polynomial

When you select this box, you can specify the generator polynomial as an integer row vector.

Generator polynomial

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

Output port for 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.

Pair Block

Integer-Input RS Encoder

See Also

Binary-Output RS Decoder

Integer to Bit Converter

Purpose

Map a vector of integers to a vector of bits

Library

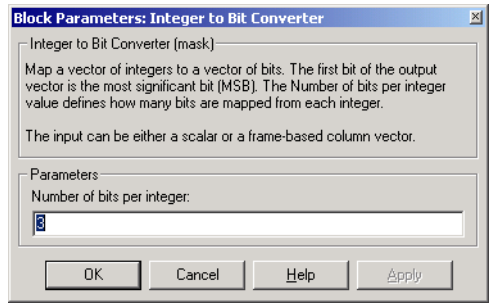
Utility Functions

Description

The Integer to Bit Converter block maps each integer in the input vector to a group of bits in the output vector. If M is the **Number of bits per integer** parameter, then the input integers must be between 0 and 2^M-1. The block maps each integer to a group of M bits, using the first bit as the most significant bit. As a result, the output vector length is M times the input vector length.

The input can be either a scalar or a frame-based column vector.

Dialog Box



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 31.

Examples

If the input is [7; 13] and the **Number of bits per integer** parameter is 4, then the output is [0; 1; 1; 1; 1; 1; 0; 1]. The first group of four bits (0, 1, 1, 1) represents 7 and the second group of four bits (1, 1, 0, 1) represents 13. Notice that the output length is four times the input length.

Pair Block

Bit to Integer Converter

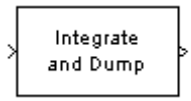
Purpose

Integrate, resetting to zero periodically and reducing by a modulus

Library

Integrators, in Basic Comm Functions

Description



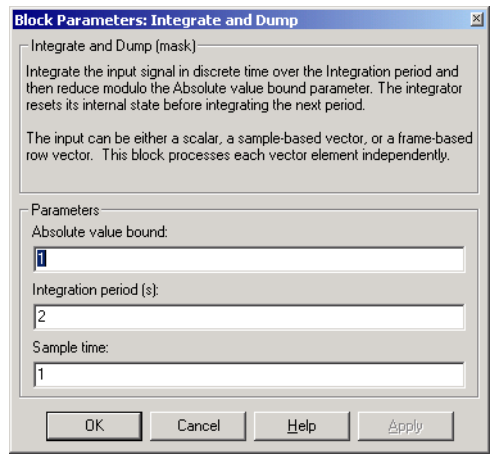
The Integrate and Dump block integrates the input signal in discrete time, resets to zero according to a fixed schedule, and reduces modulo the **Absolute value bound** parameter. If the **Absolute value bound** parameter is K, then the block output is strictly between -K and K.

The reset times are the positive integral multiples of the **Integration period** parameter. At each reset time, the block performs its final integration step, sends the result to the output port, and then clears its internal state for the next time step.

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

This block uses the Forward Euler integration method.

Dialog Box



Absolute value bound

The modulus by which the integration result is reduced. This parameter must be positive.

Integration period (s)

The first reset time. This is also the time interval between resets.

Sample time

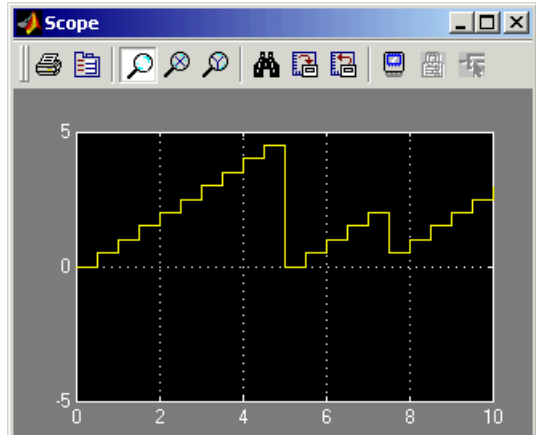
The integration sample time. This must not exceed the **Integration period**.

Examples

Integrate a constant signal whose value is 1 using these parameters:

- **Absolute value bound** = `5`
- **Integration period** = `7`
- **Sample time** = `.5`

You can use a Simulink Constant block for the input signal. The Simulink Scope block shows the output below.



Notice that the output is 0 at time 0 and that the output never exceeds 5. Also notice that the output at time 7.5 seconds (**Integration period** plus **Sample time**) is the result of resetting the integrator after the previous time step and then considering the input signal between times 7 and 7.5.

See Also

Discrete Modulo Integrator, Windowed Integrator, Discrete-Time Integrator (Simulink)

Purpose

Alternately select elements from two input vectors to generate output vector

Library

Sequence Operations, in Basic Comm Functions

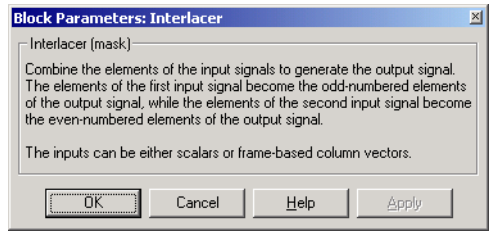
Description

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 and from the second input. 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.

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)

I/Q Imbalance

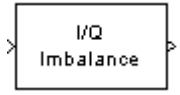
Purpose

Create a complex baseband model of the 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.

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

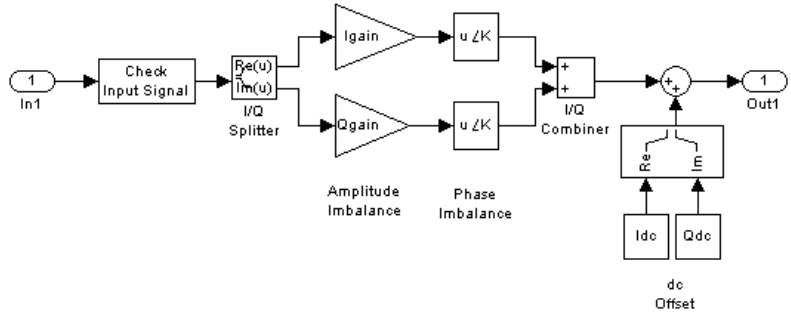
1 Separates the signal into its in-phase and quadrature components.

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 offsets, specified by the **I dc offset** parameter, and a quadrature offset, specified by the **Q dc offset** parameter, to the signal.

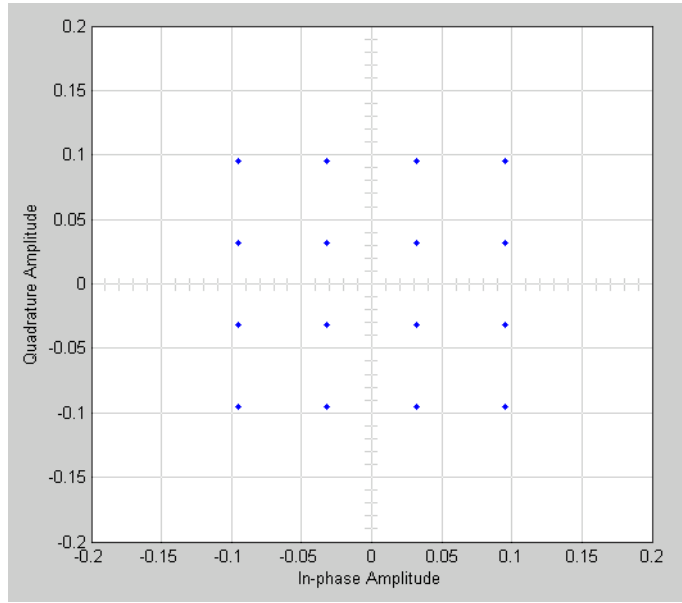
The block performs these operations in the subsystem shown in the following diagram, which you can view by right-clicking the block and selecting **Look under mask**:



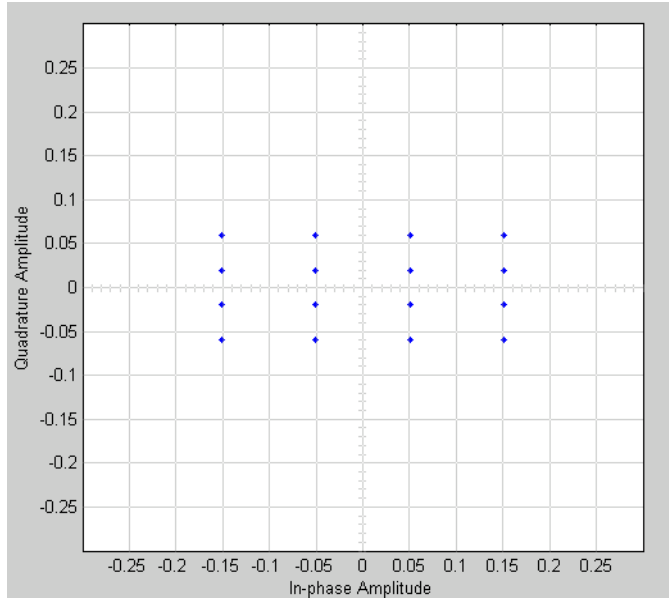
The value of the **I/Q amplitude imbalance (dB)** parameter is divided between the in-phase and quadrature components:

- 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:

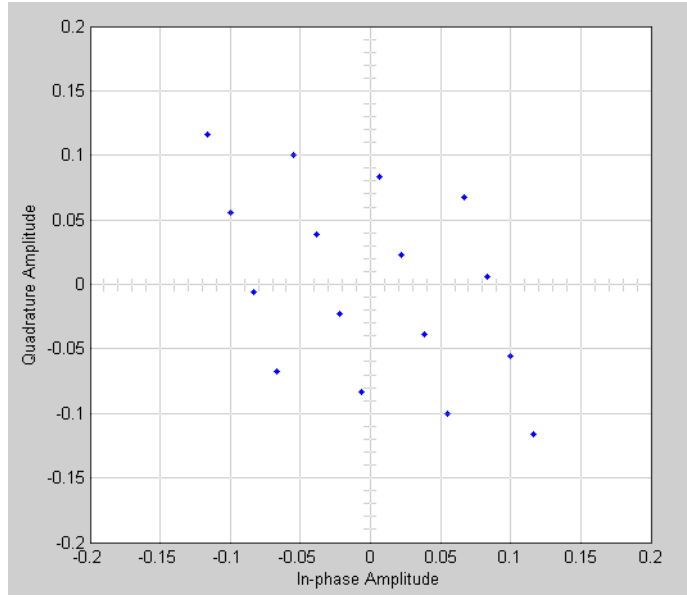


The following figure shows a scatter plot of an output signal, modulated by 16-ary QAM, from the I/Q block with **I/Q amplitude imbalance (dB)** set to `8` and all other parameters set to `0`:

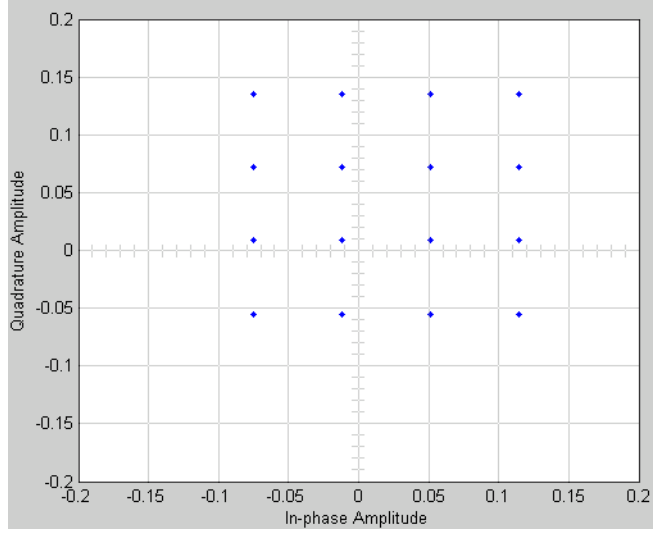


Observe that the scatter plot is stretched horizontally and compressed vertically compared to the undistorted constellation.

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 in the following figure:

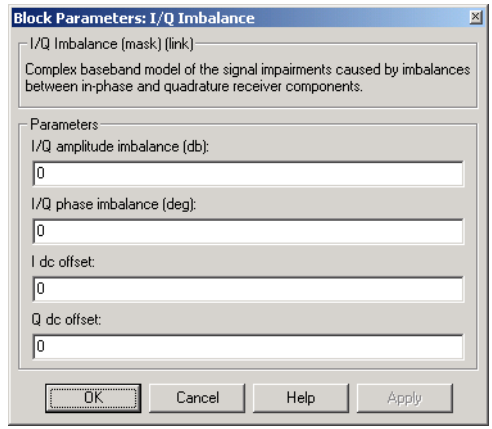


Setting the **I dc offset** to `0.02` and the **Q dc offset** to `0.04` shifts the constellation 0.02 to the right and 0.04 up, as shown in the following figure:



See “Scatter Plot Examples” for a description of the model that generates this plot.

Dialog Box



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

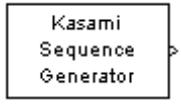
Purpose

Generate a Kasami sequence from the 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.

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.

$$K_S(u, n, m) = \begin{cases} u & m = -1 \\ u \oplus T^m w & m = 0, \ldots, 2^{n/2} - 2 \end{cases}$$

Figure 2-1: 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.

$$K_L(u,n,k,m) = \begin{cases} u & k=-2, m=-1 \\ v & k=-1, m=-1 \\ u \oplus T^k v & k=0,\ldots,2^n-2, m=-1 \\ u \oplus T^m w & k=-2, m=0,\ldots,2^{n/2}-2 \\ v \oplus T^m w & k=-1, m=0,\ldots,2^{n/2}-2 \\ u \oplus T^k v \oplus T^m w & k=0,\ldots,2^n-2, m=0,\ldots,2^{n/2}-2 \end{cases}$$

Figure 2-2: 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

$$t(n) = 1 + 2^{\frac{n+2}{2}}, \quad n \text{ even}$$

and

$$s(n) = \frac{1}{2}(t(n)+1)$$

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 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`	$m = 0, \ldots, 2^{n/2} - 2$	$u \oplus T^m w$

- 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]:

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.

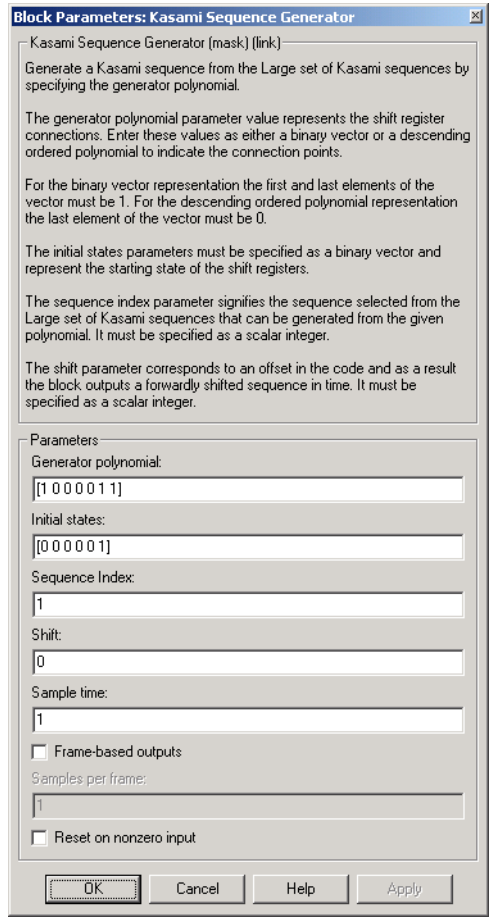
You can use an external signal to reset the values of the internal shift register to the initial state by selecting the **Reset on nonzero input** check box. This creates an input port for the external signal in the Kasami Sequence Generator block. The way the block resets the internal shift register depends on whether its output signal and the reset signal are sample-based or frame-based. See “Example: Resetting a Signal” on page 2-461 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
8	255	[8 4 3 2 0]	Small
10	1023	[10 3 0]	Large
12	4095	[12 6 4 1 0]	Small

Dialog Box



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.**

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.

Linearized Baseband PLL

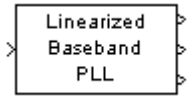
Purpose

Implement a linearized version of a baseband phase-locked loop

Library

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.

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** mask 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 the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

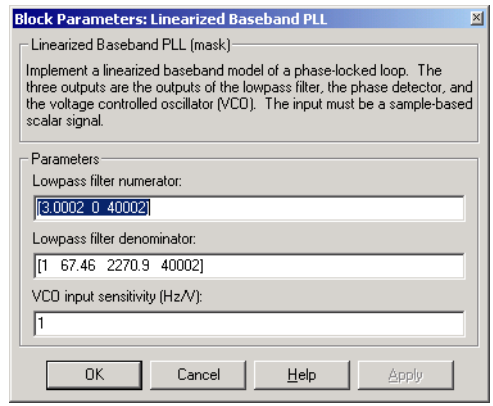
  ```
  [num, den] = cheby2(3,40,100,'s')
  ```

- 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

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

Baseband PLL, Phase-Locked Loop

References

For more information about phase-locked loops, see the works listed in "Selected Bibliography for Synchronization" in Using the Communications Blockset.

Matrix Deinterleaver

Purpose

Permute input symbols by filling a matrix by columns and emptying it by rows

Library

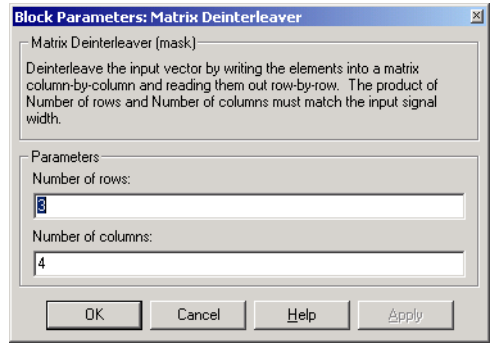
Block sublibrary of Interleaving

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.

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 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

Matrix Helical Scan Deinterleaver

Purpose

Restore ordering of input symbols by filling a matrix along diagonals

Library

Block sublibrary of Interleaving

Description

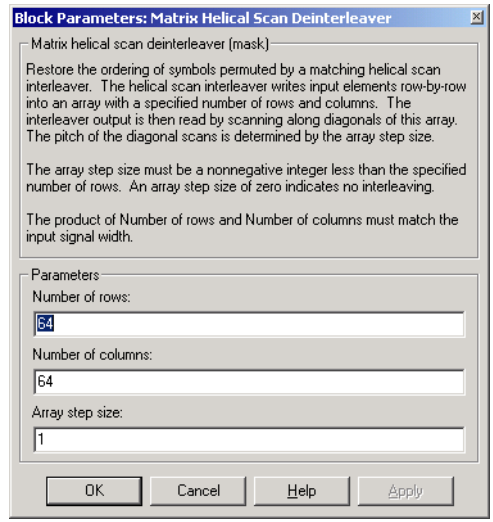
The Matrix Helical Scan Deinterleaver block performs block deinterleaving by filling a matrix with the input symbols in a helical fashion and then sending the matrix contents to the output port row by row. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that 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.

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

Matrix Helical Scan Interleaver

Purpose

Permute input symbols by selecting matrix elements along diagonals

Library

Block sublibrary of Interleaving

Description

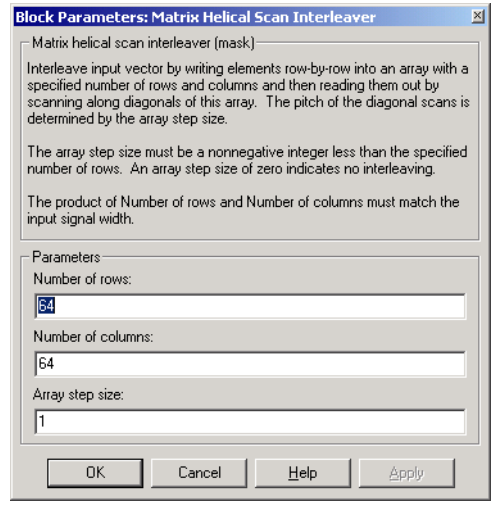
The Matrix Helical Scan Interleaver block performs block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port in a helical fashion. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the 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.

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

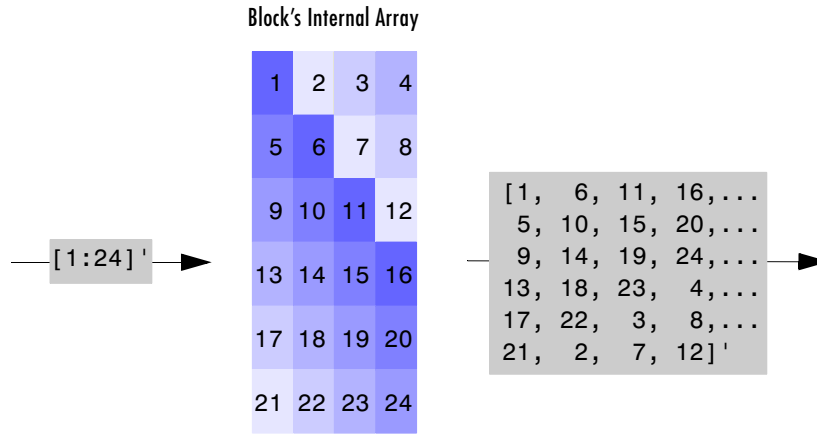
The slope of the diagonals that the block reads.

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

```
[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

Purpose

Permute input symbols by filling a matrix by rows and emptying it by columns

Library

Block sublibrary of Interleaving

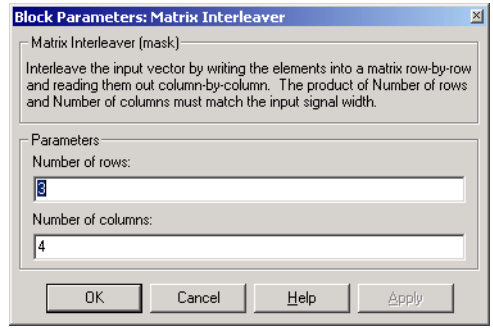
Description

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.

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]`.

Matrix Interleaver

Pair Block Matrix Deinterleaver

See Also General Block Interleaver

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 **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 maps a phase difference of

$$\theta + 2\pi m/M$$

to m, where θ is the **Phase offset** parameter and m 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.

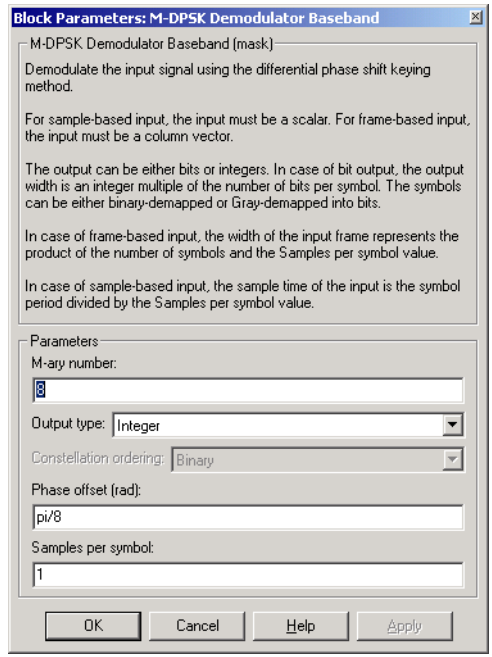
In binary output mode, the **Constellation ordering** parameter indicates how the block maps an integer to a corresponding group of K output bits. See the reference pages for the M-DPSK Modulator Baseband and M-PSK Modulator Baseband blocks for details.

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. If it is greater than 1, then the demodulated signal is delayed by one output sample. For more

information, see “Upsampled Signals and Rate Changes” in Using the Communications Blockset.

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. This field is active only when **Output type** is set to **Bit**.

Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

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.

M-DPSK Demodulator Passband

Purpose

Demodulate DPSK-modulated data

Library

PM, in Digital Passband sublibrary of Modulation

Description

The M-DPSK Demodulator Passband block demodulates a signal that was modulated using the M-ary differential phase shift keying method. The input is a passband representation of the modulated signal. The input and output for this block are discrete-time signals. The input must be a sample-based scalar signal.

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 because there is no previous symbol.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-DPSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

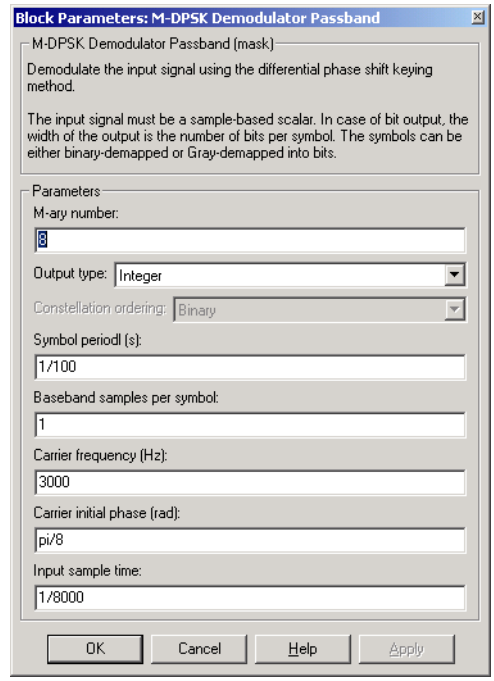
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Input sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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. This field is active only when **Output type** is set to **Bit**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block

M-DPSK Modulator Passband

See Also

M-DPSK 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 the 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.

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 m, then the modulated symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** 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** 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.

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 offset** parameter.

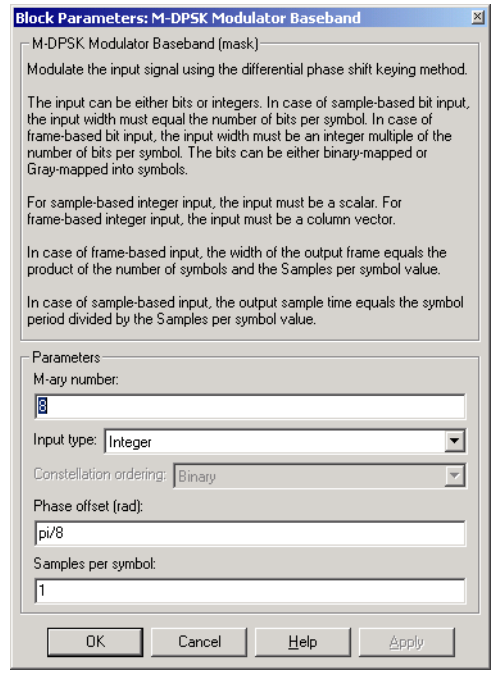
Input	Binary-Coded Phase Differences	Gray-Coded Phase Differences
[0 0 0]	$j\theta$	$j\theta$
[0 0 1]	$j\theta + j\pi/2$	$j\theta + j\pi/2$
[0 1 0]	$j\theta + j\pi 2/2$	$j\theta + j\pi 3/2$
[0 1 1]	$j\theta + j\pi 3/2$	$j\theta + j\pi 2/2$
[1 0 0]	$j\theta + j\pi 4/2$	$j\theta + j\pi 6/2$
[1 0 1]	$j\theta + j\pi 5/2$	$j\theta + j\pi 7/2$
[1 1 0]	$j\theta + j\pi 6/2$	$j\theta + j\pi 5/2$
[1 1 1]	$j\theta + j\pi 7/2$	$j\theta + j\pi 4/2$

For more details about the **Binary** and **Gray** options, see the reference page for the M-PSK Modulator Baseband block. The signal constellation for that block corresponds to the arrangement of phase differences for this block.

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 Using the Communications Blockset.

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. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

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

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.

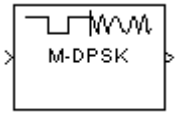
Purpose

Modulate using the M-ary differential phase shift keying method

Library

PM, in Digital Passband sublibrary of Modulation

Description



The M-DPSK Modulator Passband block modulates using the M-ary differential phase shift keying method. The output is a passband 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.

This block uses the baseband equivalent block, M-DPSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

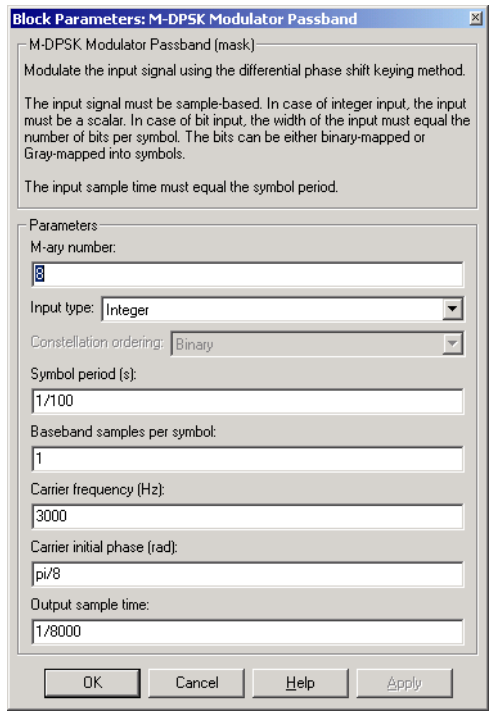
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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. This field is active only when **Input type** is set to **Bit**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

M-DPSK Demodulator Passband

See Also

M-DPSK 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.

Memoryless Nonlinearity

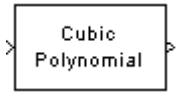
Purpose

Apply a memoryless nonlinearity to a 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.

The Memoryless Nonlinearity block provides five different methods for modeling the nonlinearity, which you specify by the **Method** parameter in the block's mask. 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 figure below.

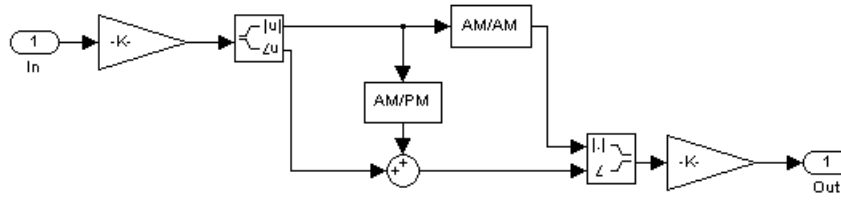


Figure 2-3: Nonlinearity Subsytem

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.

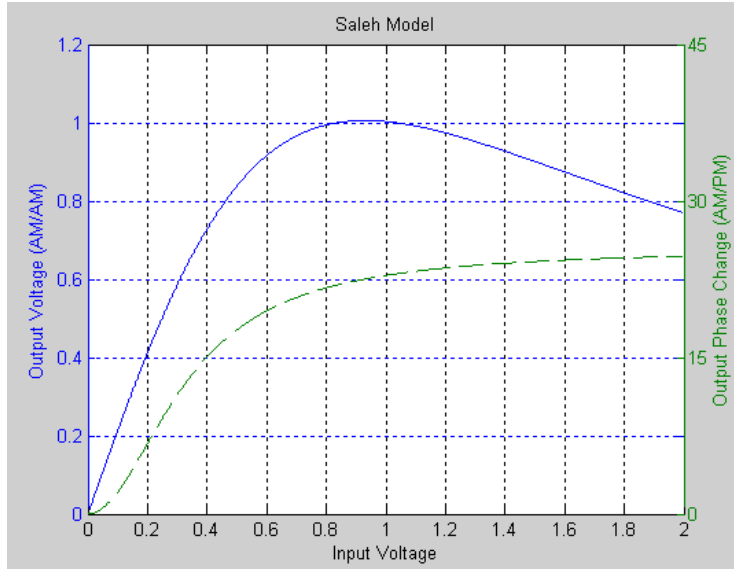
However, the subsystems implement the AM/AM and AM/PM conversions differently, according to the **Method** you specify.

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:

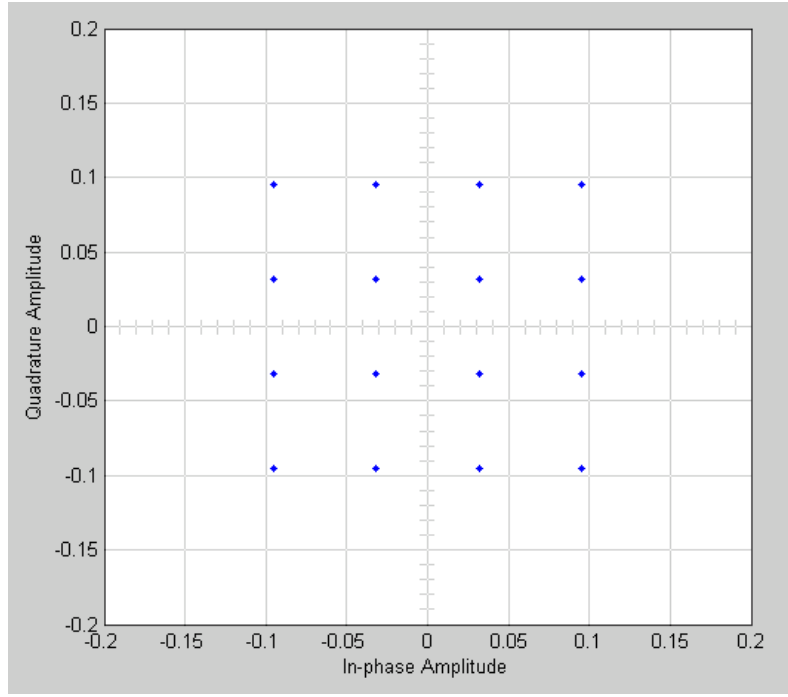
1. Right click on the Memoryless Nonlinearity block.
2. Select **Look under mask** in the pop-up menu. This displays the block's configuration underneath the mask. The block contains five subsystems corresponding to the five nonlinearity methods.
3. Double-click the subsystem for the method you are interested in. This displays the subsystem shown in the preceding figure, "Nonlinearity Subsytem".
4. Double-click on one of the subsystems labeled AM/AM or AM/PM to view how the block implements the conversions.

The following figure shows, for the Saleh method, plots of

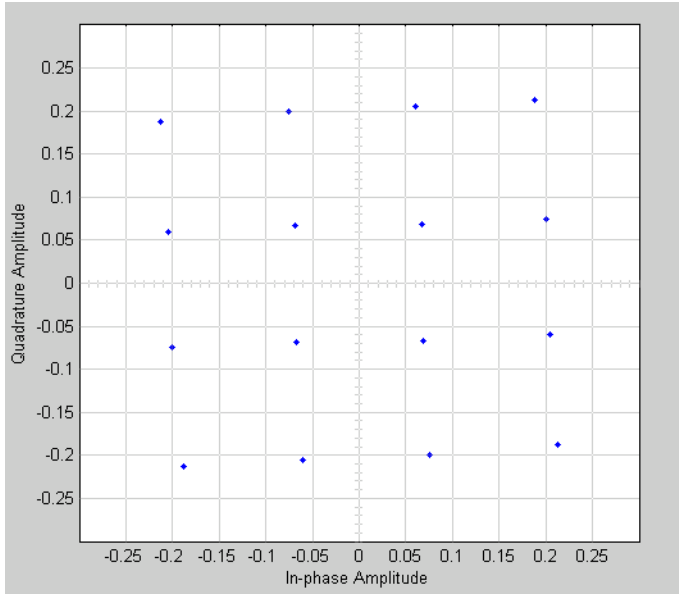
- Output voltage against input voltage for the AM/AM conversion
- Output phase against input voltage for the AM/PM conversion



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 Mode**l, as shown in the following figure.



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:

$$F_{AM/AM}(u) = \frac{\text{alpha}*u}{1 + beta*u^2}$$

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:

$$F_{AM/PM}(u) = \frac{\text{alpha}*u^2}{1 + beta*u^2}$$

where u is the magnitude of the input 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:

$$F_{AM/AM}(u) = \frac{x_1 u^{x_2}}{1 + x_3 u^{x_2}} + x_4 u$$

where u is the magnitude of the input signal.

The **Output scaling (dB)** parameter scales the output signal similarly.

Parameters for the Rapp Model

The **Smoothness factor** and **Output saturation level** parameters are used to compute the amplitude gain for an input signal by the following function:

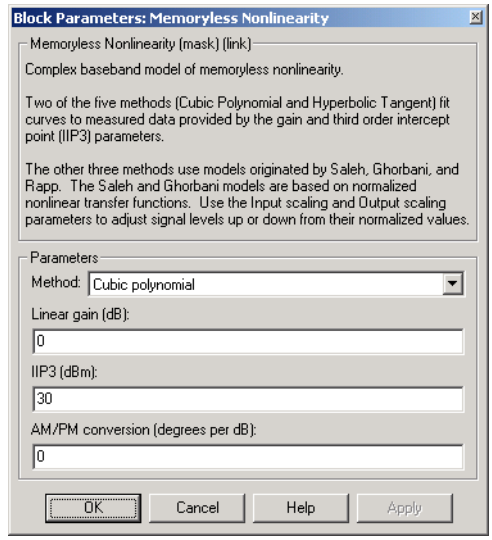
$$F_{AM/AM}(u) = \frac{u}{\left(1 + \left(\frac{u}{O_{sat}}\right)^{2S}\right)^{1/2S}}$$

where 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.

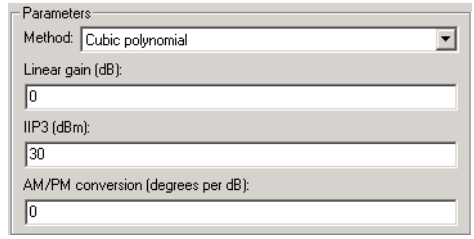
Dialog Box



Method

The nonlinearity method.

The following describes specific parameters for each method.



Linear gain (db)

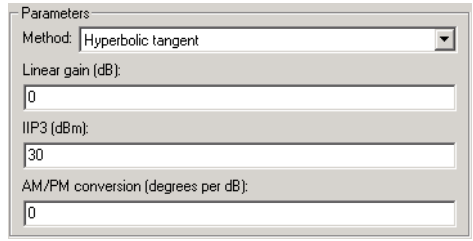
Scalar specifying the linear gain for the output function.

IIP3 (dBm)

Scalar specifying the third order intercept.

AM/PM conversion (degrees per dB)

Scaler specifying the AM/PM conversion in degrees per decibel.



Linear gain (db)

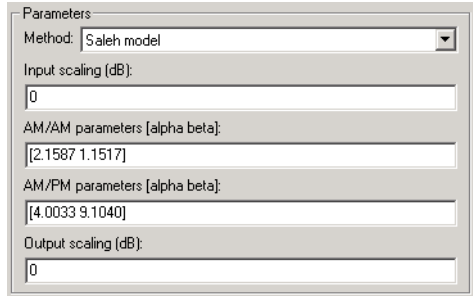
Scalar specifying the linear gain for the output function.

IIP3 (dBm)

Scalar specifying the third order intercept.

AM/PM conversion (degrees per dB)

Scaler specifying the AM/PM conversion in degrees per decibel.



Input scaling (dB)

Number that scales the input signal level.

AM/AM parameters [alpha beta]

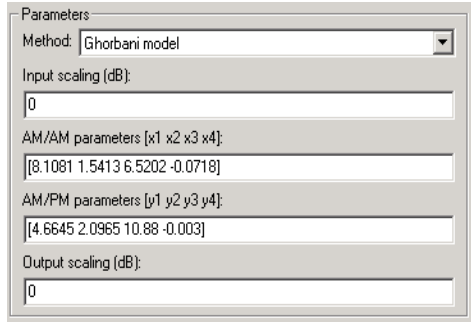
Vector specifying the AM/AM parameters.

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]

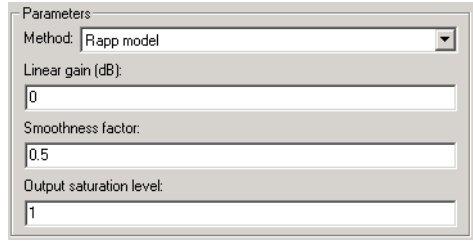
Vector specifying the AM/AM parameters.

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 European Conference on Satellite Communications, Liege, Belgium, Oct. 22-24, 1991, pp. 179-184.

M-FSK Demodulator Baseband

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.

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.

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.

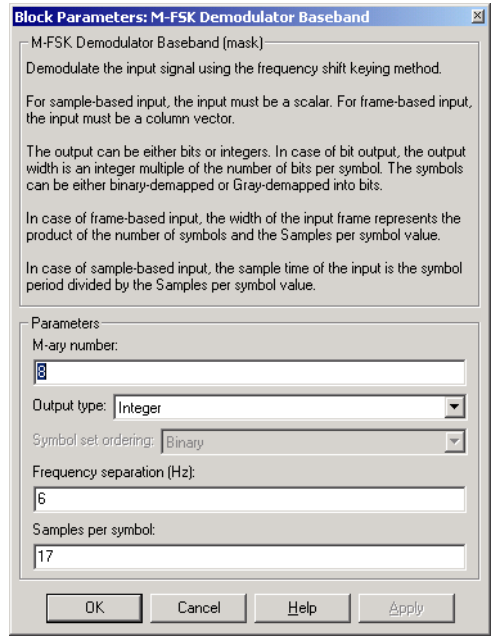
In binary output mode, the **Symbol set ordering** parameter indicates how the block maps an integer to a corresponding group of K output bits. See the reference pages for the M-FSK Modulator Baseband and M-PSK Modulator Baseband blocks for details.

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.

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 Using the Communications Blockset.

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. This field is active only when **Output type** is set to **Bit**.

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.

M-FSK Demodulator Baseband

Pair Block M-FSK Modulator Baseband

See Also CPFSK Demodulator Baseband

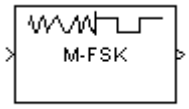
Purpose

Modulate using the M-ary frequency shift keying method

Library

FM, in Digital Passband sublibrary of Modulation

Description



The M-FSK Demodulator Passband block demodulates a signal that was modulated using the M-ary frequency shift keying method. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of frequencies in the modulated signal.

This block converts the input to an equivalent baseband representation using downconversion and then FIR decimation. The block then uses the baseband equivalent block, M-FSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Signal set ordering**
- **Frequency separation**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Output sample time** < [2***Carrier frequency** + 2*F_{max})]$^{-1}$

where F_{max} is defined as follows:

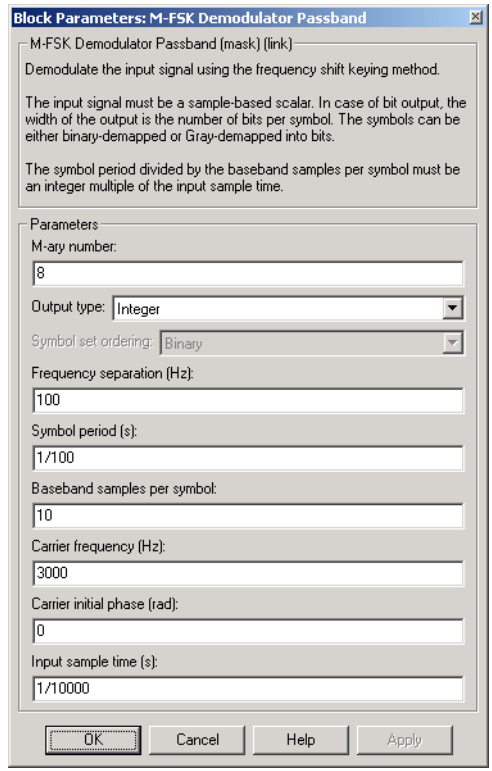
F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The M-FSK Demodulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

The **Symbol period** parameter must be an integer multiple of the product of **Output sample time** times **Baseband samples per symbol**.

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. This field is active only when **Output type** is set to **Bit**.

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time(s)

The sample time of the input signal.

Pair Block

M-FSK Modulator Passband

See Also

M-FSK Demodulator Baseband, CPFSK Demodulator Passband

M-FSK Modulator Baseband

Purpose

Modulate using the 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. 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.

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:

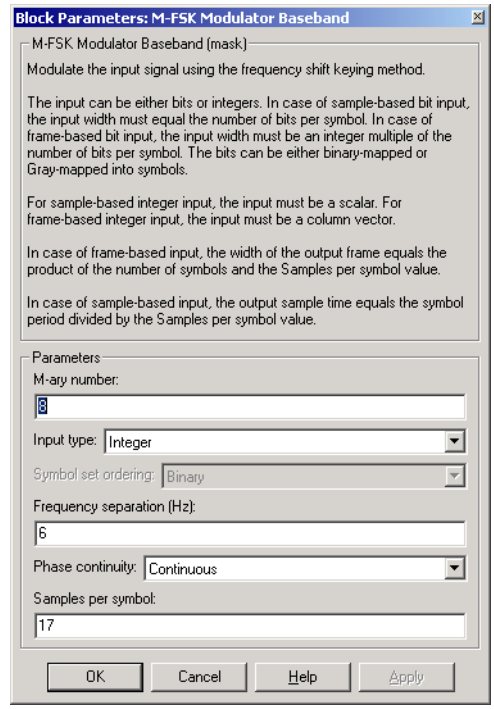
- If **Input type** is set to **Integer**, then the block accepts integers. The input can be either a scalar or a frame-based column vector.
- If **Input type** is set to **Bit**, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K. 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 the M-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.

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 Using the Communications Blockset.

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. This field is active only when **Input type** is set to **Bit**.

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.

Pair Block

M-FSK Demodulator Baseband

See Also

CPFSK Modulator Baseband

Purpose

Modulate using the M-ary frequency shift keying method

Library

FM, in Digital Passband sublibrary of Modulation

Description

The M-FSK Modulator Passband block modulates using the M-ary frequency shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of frequencies in the modulated signal.

This block uses the baseband equivalent block, M-FSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation, using FIR interpolation and then upconversion. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Frequency separation**
- **Phase continuity**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

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.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each

integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Output sample time** < [2*__Carrier frequency__ + 2*F_{max})]$^{-1}$

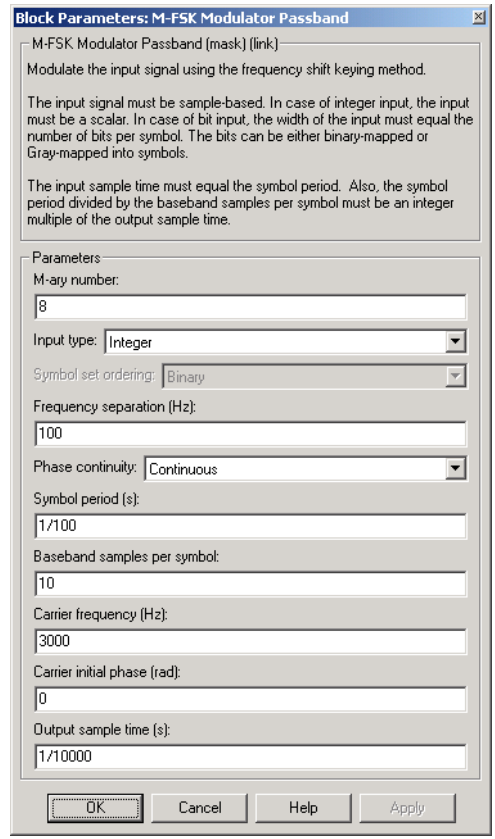
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The M-FSK Modulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

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. This field is active only when **Input type** is set to **Bit**.

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.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time(s)

The sample time of the output signal.

Pair Block

M-FSK Demodulator Passband

See Also

M-FSK Modulator Baseband, CPFSK Modulator Passband

Purpose

Integrate in continuous time and reduce by a modulus

Library

Integrators, in Basic Comm Functions

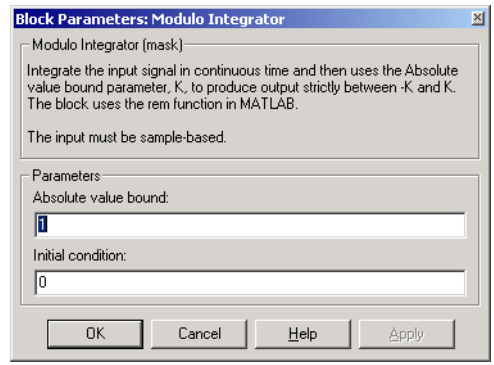
Description

The Modulo Integrator block integrates its input signal in continuous time and then reduces modulo the **Absolute value bound** parameter. If the **Absolute value bound** parameter is K, then the block output is strictly between -K and K.

The input must be sample-based. The block processes each vector element independently.

This block's functionality is useful for monotonically increasing or decreasing functions, but works with any integrable function. This block uses the Forward Euler integration method.

Dialog Box



Absolute value bound

The modulus by which the integration result is reduced. This parameter must be nonzero.

Initial condition

The initial condition for integration.

See Also

Discrete Modulo Integrator, Integrator (Simulink)

M-PAM Demodulator Baseband

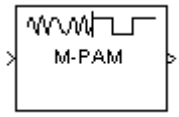
Purpose

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 the 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.

Output Signal Values

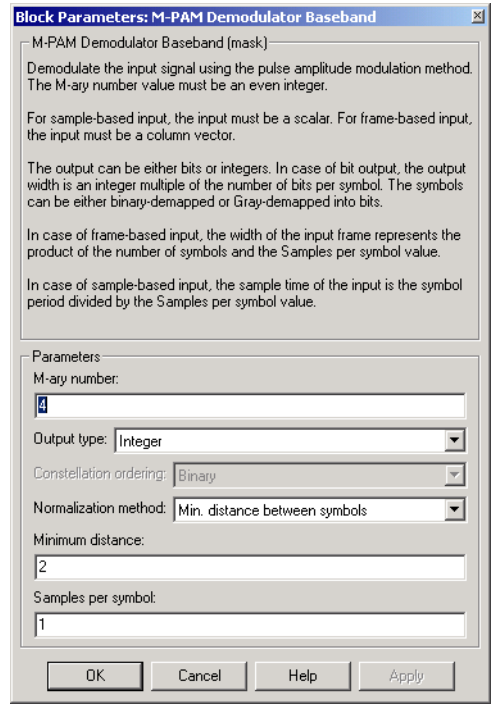
The **Output type** parameter determines whether the block produces integers or binary representations of integers. If **Output type** is set to **Integer**, then the block produces integers. If **Output type** is set to **Bit**, then the block produces a group of K bits, called a binary word, for each 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.

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 Using the Communications Blockset.

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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

M-PAM Modulator Baseband

See Also

General QAM Demodulator Baseband

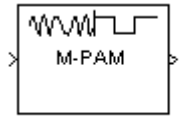
Purpose

Demodulate PAM-modulated data

Library

AM, in Digital Passband sublibrary of Modulation

Description



The M-PAM Demodulator Passband block demodulates a signal that was modulated using M-ary pulse amplitude modulation. The input is a passband representation of the modulated signal. The input must be a sample-based scalar signal.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PAM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

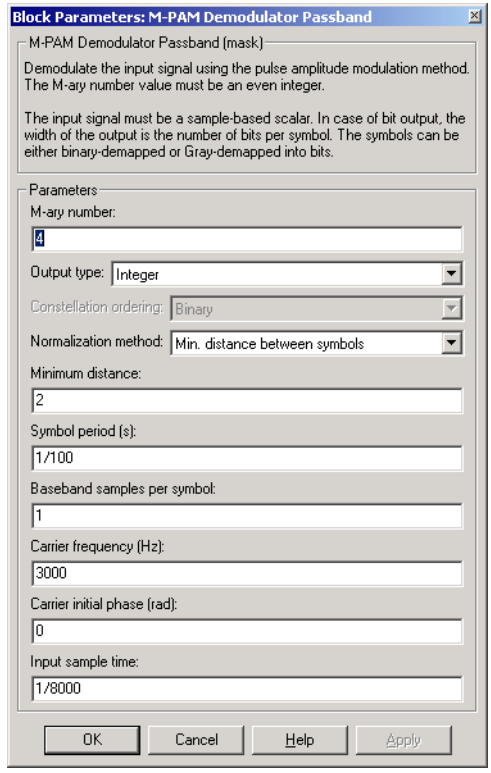
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Input sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time
The sample time of the input signal.

Pair Block M-PAM Modulator Passband

See Also M-PAM Demodulator 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.

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 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 table below 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

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.

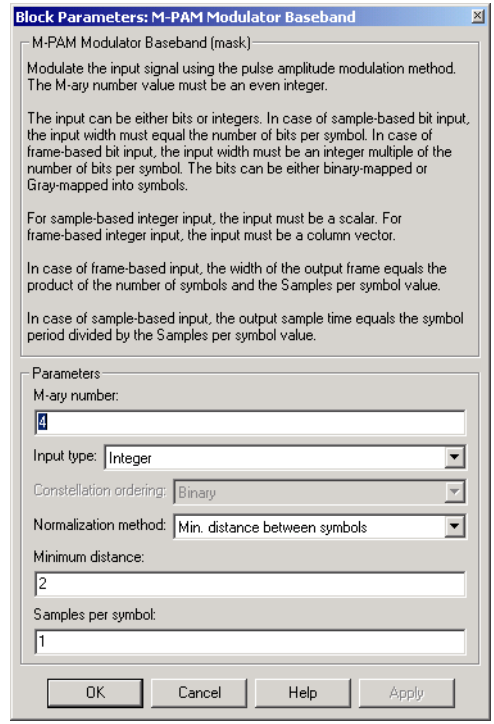
- If **Input type** is set to **Integer**, then the block accepts integers. The input can be either a scalar or a frame-based column vector.
- If **Input type** is set to **Bit**, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K. 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.

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 Using the Communications Blockset.

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. This field is active only when **Input 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Samples per symbol

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

Pair Block

M-PAM Demodulator Baseband

See Also

General QAM Modulator Baseband

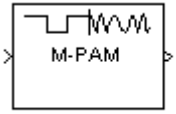
Purpose

Modulate using M-ary pulse amplitude modulation

Library

AM, in Digital Passband sublibrary of Modulation

Description



The M-PAM Modulator Passband block modulates using M-ary pulse amplitude modulation. The output is a passband 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.

This block uses the baseband equivalent block, M-PAM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

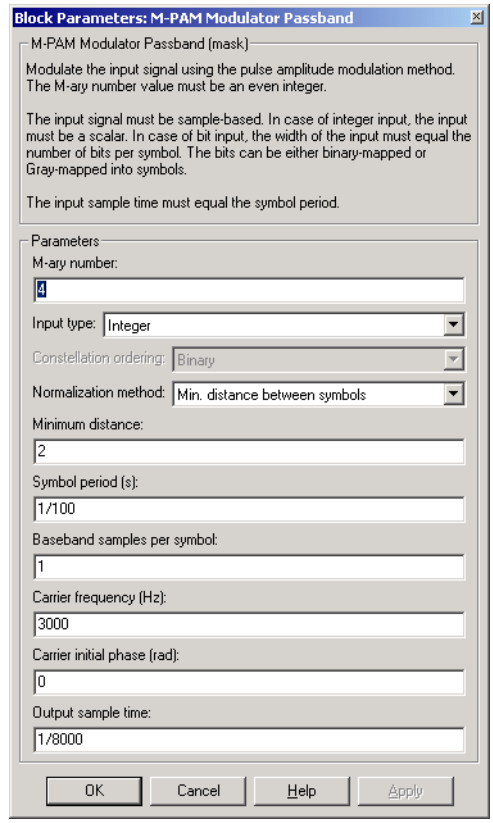
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2*__Carrier frequency__ + 2/(**Symbol period**)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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. This field is active only when **Input 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

M-PAM Demodulator Passband

See Also

M-PAM 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 input can be either a scalar or a frame-based column vector. The **M-ary number** parameter, M, is the number of points in the signal constellation.

Binary or Integer Outputs

If the **Output type** parameter is set to **Integer**, then the block maps the point

$$\exp(j\theta + j2\pi m/M)$$

to m, where θ is the **Phase offset** parameter and m is an integer between 0 and M-1.

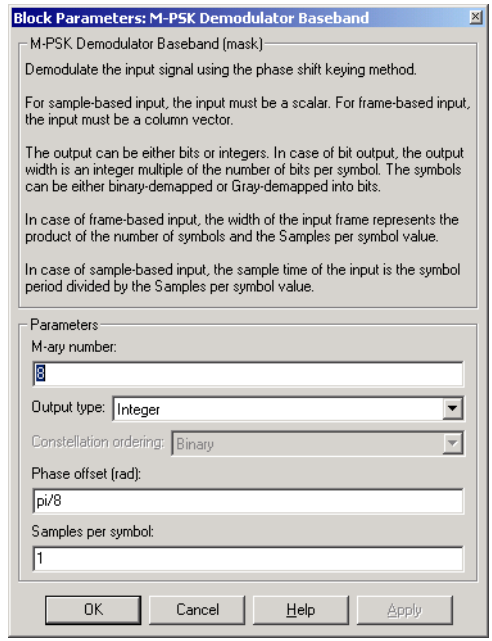
If the **Output type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K, then the block outputs binary representations of integers between 0 and M-1. It outputs a group of K bits, called a binary *word*, for each symbol.

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 page for the M-PSK Modulator Baseband block for details.

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 Using the Communications Blockset.

Dialog Box



M-ary number

The number of points in the signal constellation.

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**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block M-PSK Modulator Baseband

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

M-PSK Demodulator Passband

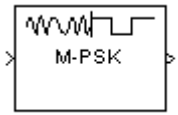
Purpose

Demodulate PSK-modulated data

Library

PM, in Digital Passband sublibrary of Modulation

Description



The M-PSK Demodulator Passband block demodulates a signal that was modulated using the M-ary phase shift keying method. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

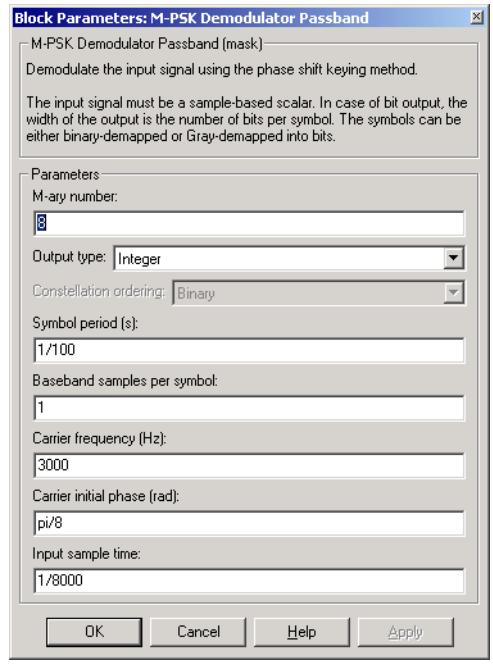
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Input sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



M-ary number

The number of points in the signal constellation.

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**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block

M-PSK Modulator Passband

See Also

M-PSK Demodulator Baseband

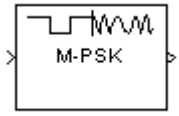
Purpose

Modulate using the 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.

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

$$\exp(j\theta + j2\pi m/M)$$

The input and output for this block are discrete-time signals. To use integers between 0 and M-1 as input values, set the **Input type** parameter to **Integer**. In this case, the input can be either a scalar or a frame-based column vector.

Alternative configurations of the block determine how the block interprets its input and arranges its output, as explained in the sections below.

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, 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.

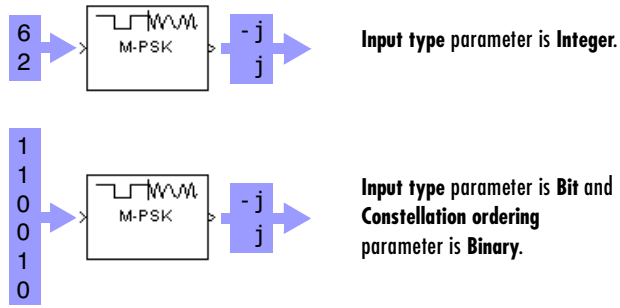
In binary input mode, the **Constellation ordering** parameter indicates how the block maps a group of K input bits to a corresponding integer. Choices are **Binary** and **Gray**. For more information, see "Binary-Valued and Integer-Valued Signals" in Using the Communications Blockset.

If **Constellation ordering** is set to **Gray**, then 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" below.

Frame-Based Inputs

If the input is a frame-based column vector, then the block processes several integers or several binary words, in each time step. (If the **Input type** parameter is set to **Bit**, then a binary word consists of $\log_2(M)$ bits.)

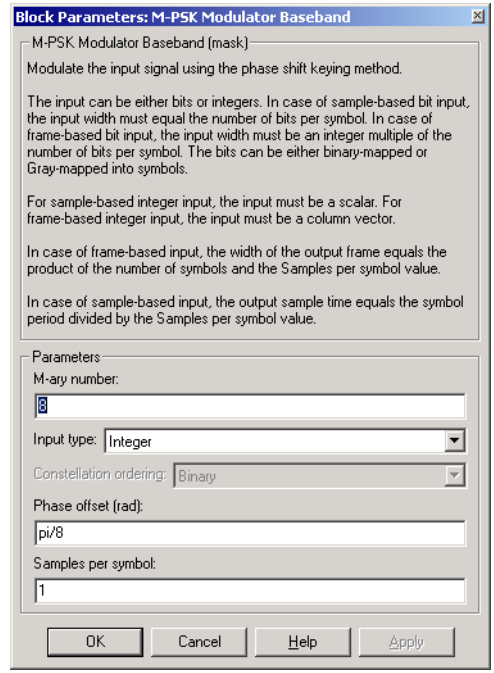
For example, the schematics below illustrate how the block processes two 8-ary integers or binary words in one time step. The signals involved are all frame-based column vectors. In both cases, the **Phase offset** parameter is 0.



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 Using the Communications Blockset.

Dialog Box



M-ary number

The number of points in the signal constellation.

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. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

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

Algorithm

If the **Constellation ordering** parameter is set to **Gray**, then 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 `m`th 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, then 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, then the binary representations that correspond to the zeroth through seventh phases are below.

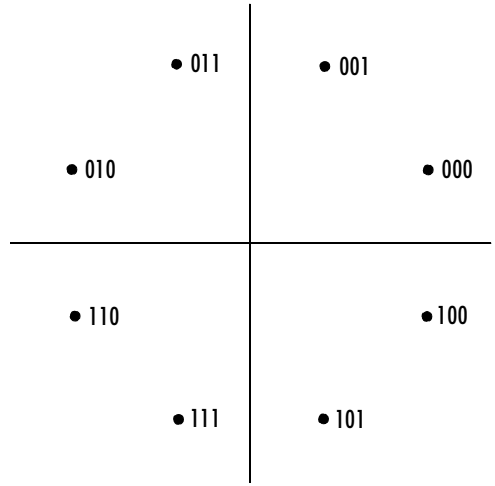
```
M = 8; m = [0:M-1]';
de2bi(bitxor(m,floor(m/2)), log2(M),'left-msb')

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

Below is 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 Modulator Passband

Purpose

Modulate using the M-ary phase shift keying method

Library

PM, in Digital Passband sublibrary of Modulation

Description

The M-PSK Modulator Passband block modulates using the M-ary phase shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation.

This block uses the baseband equivalent block, M-PSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

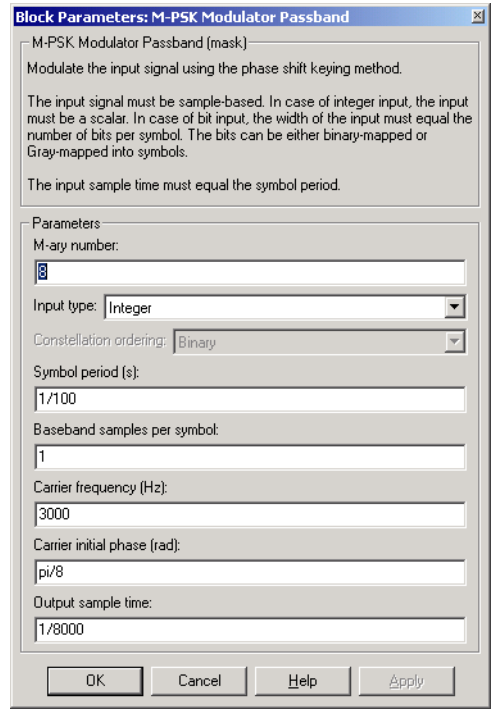
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2*__Carrier frequency__ + 2/(__Symbol period__)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



M-ary number

The number of points in the signal constellation.

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. This field is active only when **Input type** is set to **Bit**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

M-PSK Demodulator Passband

See Also

M-PSK Modulator 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.

Inputs and Outputs

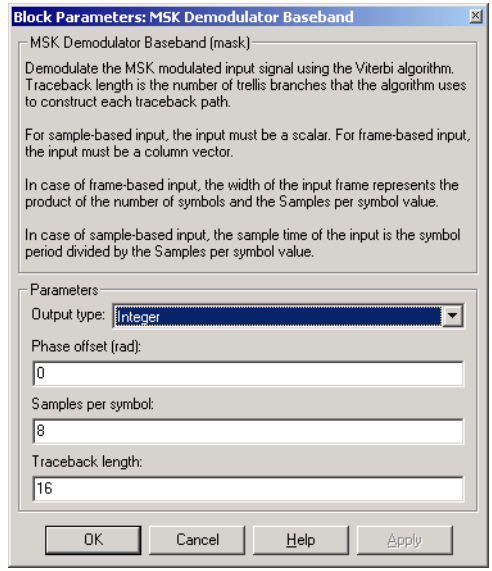
The input can be either a scalar or a frame-based column vector. If the **Output type** parameter is set to **Integer**, then the block produces values of 1 and -1. If the **Output type** parameter is set to **Bit**, then the block produces values of 0 and 1.

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 Using the Communications Blockset.

MSK Demodulator Baseband

Dialog Box



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 Viterbi Decoder block uses to construct each traceback path.

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.

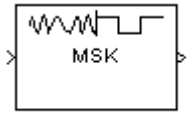
Purpose

Demodulate MSK-modulated data

Library

CPM, in Digital Passband sublibrary of Modulation

Description



The MSK Demodulator Passband block demodulates a signal that was modulated using the minimum shift keying method. The input is a passband representation of the modulated signal.

This block converts the input to an equivalent baseband representation using downconversion and then FIR decimation. The block then uses the baseband equivalent block, MSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **Output type**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < [2***Carrier frequency** + 2*F_{max})]$^{-1}$

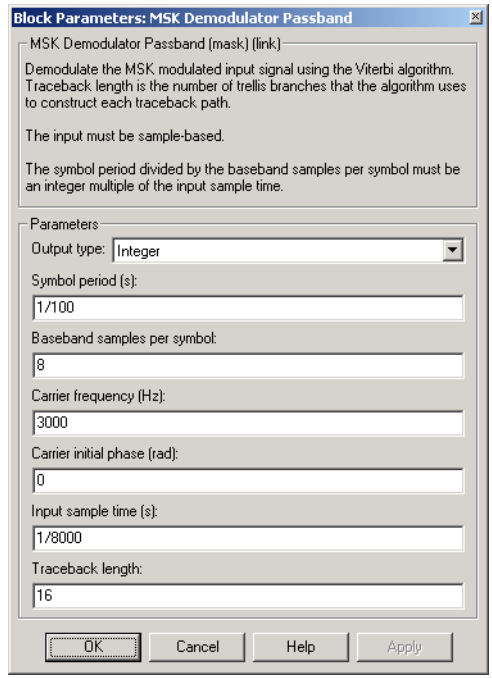
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The MSK Demodulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

Dialog Box



Output type

Determines whether the output consists of bipolar or binary values.

Symbol period(s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time (s)

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

MSK Modulator Passband

See Also

MSK Demodulator Baseband, Viterbi Decoder

References

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

MSK Modulator Baseband

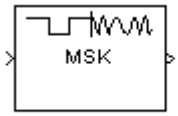
Purpose

Modulate using the 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 **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.

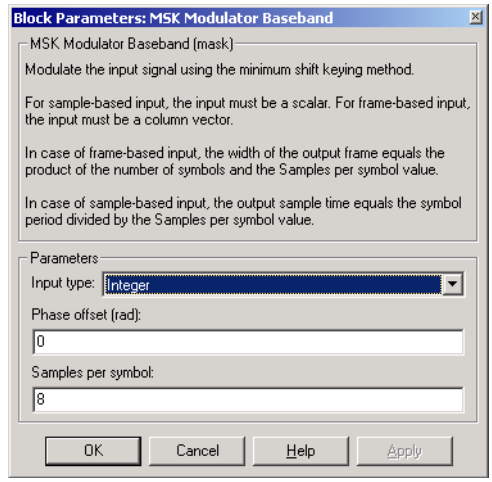
Input Attributes

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 Using the Communications Blockset.

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.

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.

MSK Modulator Passband

Purpose

Modulate using the minimum shift keying method

Library

CPM, in Digital Passband sublibrary of Modulation

Description

The MSK Modulator Passband block modulates using the minimum shift keying method. The output is a passband representation of the modulated signal.

This block uses the baseband equivalent block, MSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation using FIR interpolation and then upconversion. The **Input type** parameter of this block is the same as that of the baseband equivalent block.

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$, containing values of 0 and 1. If the **Input type** parameter is **Integer**, then the input must be a scalar containing values of 1 and -1.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Symbol period** must be an integer multiple of the product of **Output sample time** and **Baseband samples per symbol**.
- **Baseband samples per symbol** > 4
- **Output sample time** < [2***Carrier frequency** + 2*F_{max})]$^{-1}$

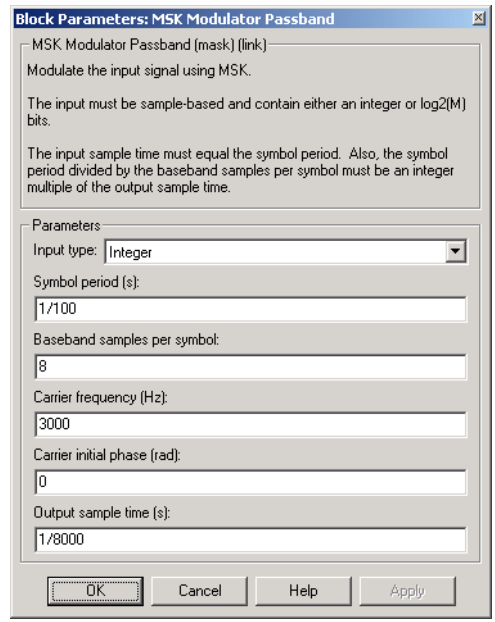
where F_{max} is defined as follows:

F_{max} = [**Frequency separation** * (**M-ary number** - 1) / 2] + 1 / **Symbol period**

The **Carrier frequency** parameter is typically much larger than the highest frequency of the baseband signal.

The MSK Modulator Passband block creates a delay in signals that it processes. This delay is caused by FIR filters in the block, whose tap length depends on signal and simulation parameters.

Dialog Box



Input type

Indicates whether the input consists of bipolar or binary values.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time (s)

The sample time of the output signal.

Pair Block

MSK Demodulator Passband

See Also

MSK Modulator Baseband

References

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

Purpose

Implement μ-law compressor for source coding

Library

Source Coding

Description

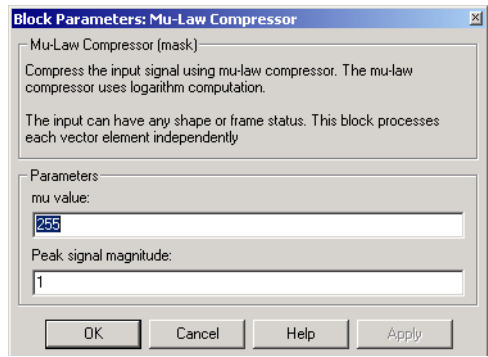
The Mu-Law Compressor block implements a μ-law compressor for the input signal. The formula for the μ-law compressor is

$$y = \frac{V\log(1+\mu|x|/V)}{\log(1+\mu)}\operatorname{sgn}(x)$$

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

See Also

A-Law Compressor

References

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

Mu-Law Expander

Purpose

Implement μ-law expander for source coding

Library

Source Coding

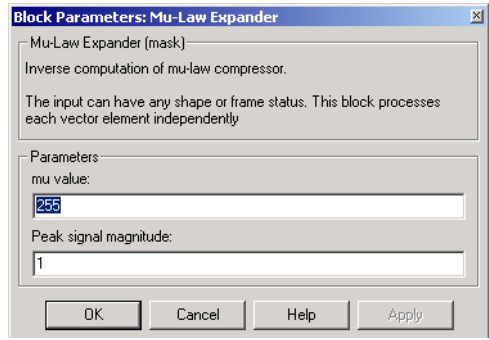
Description

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.

$$x = \frac{V}{\mu}(e^{|y|\log(1+\mu)/V} - 1)\,\mathrm{sgn}(y)$$

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

References

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

Purpose

Simulate a 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. This block is useful for modeling mobile wireless communication systems. For details about fading channels, see the works listed in "References" on page 2-423.

The input can be either a scalar or a frame-based column vector. The input is a complex signal.

Relative motion between the transmitter and receiver causes Doppler shifts in the signal frequency. The Jakes PSD (power spectral density) determines the spectrum of the Rayleigh process.

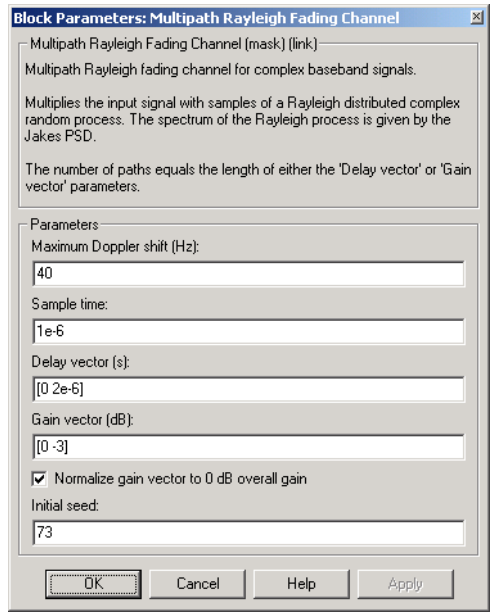
Since a multipath channel reflects signals at multiple places, a transmitted signal travels to the receiver along several paths that may have different lengths and hence different associated time delays. Fading occurs when signals traveling along different paths interfere with each other. In the block's parameter mask, the **Delay vector** specifies the time delay for each path. If the **Normalize gain vector to 0 dB overall gain** box is unchecked, then the **Gain vector** specifies the gain for each path. If the box is checked, then the block uses a multiple of **Gain vector** instead of the **Gain vector** itself, choosing the scaling factor so that the channel's effective gain considering all paths is 0 dB.

The number of paths is the length of **Delay vector** or **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 is a scalar, then the block expands it into a vector whose size matches that of the other **vector** parameter.

The **Sample time** parameter is the time between successive elements of the input signal. Note that if the input is a frame-based column vector of length n, then the frame period (as Simulink's Probe block reports, for example) is n*__Sample time__.

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.

Dialog Box



Maximum Doppler shift (Hz)

A positive scalar that indicates the maximum Doppler shift.

Sample time

The period of each element of the input signal.

Delay vector (s)

A vector that specifies the propagation delay for each 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 decibels.

Initial seed

The scalar seed for the Gaussian noise generator.

Algorithm

This implementation is based on the direct form simulator described in Reference [1] below.

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 is the maximum Doppler shift arising from motion of the mobile.

See Also

Rayleigh Noise Generator, Rician Fading Channel

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.

OQPSK Demodulator Baseband

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.

If the **Output type** parameter is set to **Integer**, then the block outputs integers between 0 and 3. If the **Output type** parameter is set to **Bit**, then the block outputs binary representations of such integers, in a binary-valued vector whose length is an even number.

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.

Frame-Based Inputs

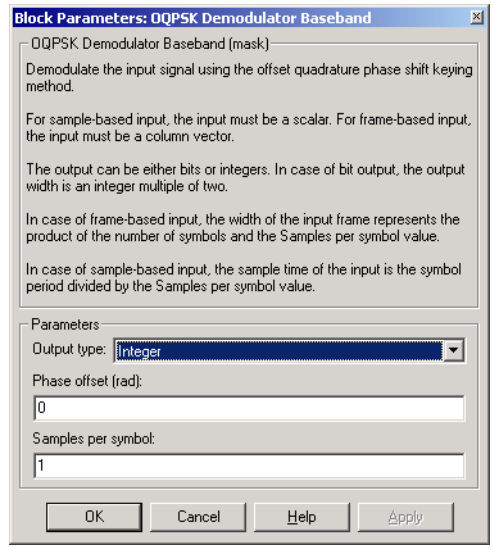
If the input is a frame-based column vector, then the block processes several integers or several pairs of bits, in each time step. In this case, the output sample time equals the input sample time, even though the symbol period is half the output period.

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.

- If the input is a frame-based column vector, then the output vector contains 1/(2S) integers or pairs of bits for each sample in the input vector, while the output sample time equals the input sample time.
- If the input is a sample-based scalar, then the output vector contains a single integer or pair of bits, while the output sample time is 2S times the input sample time.

Dialog Box



Output type

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

Phase offset (rad)

The amount by which the phase of the zeroth point of the signal constellation is shifted from $\pi/4$.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

OQPSK Modulator Baseband

See Also

QPSK Demodulator Baseband

OQPSK Demodulator Passband

Purpose

Demodulate OQPSK-modulated data

Library

PM, in Digital Passband sublibrary of Modulation

Description

The OQPSK Demodulator Passband block demodulates a signal that was modulated using the offset quadrature phase shift keying method. The input is a passband representation of the modulated signal.

If the **Output type** parameter is set to **Integer**, then the block outputs integers between 0 and 3. If the **Output type** parameter is set to **Bit**, then the block outputs binary representations of such integers, in binary-valued vectors of length two. The constellation used to map bit pairs to symbols is on the reference page for the OQPSK Modulator Passband block.

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

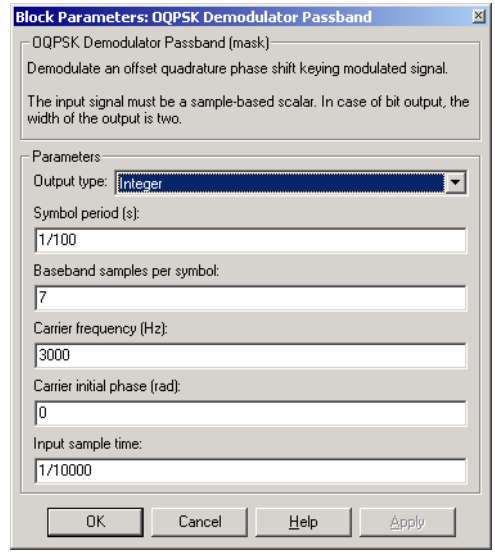
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Input sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



Output type

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

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)
The initial phase of the carrier in radians.

Input sample time
The sample time of the input signal.

Pair Block

OQPSK Modulator Passband

See Also

OQPSK Demodulator Baseband

Purpose

Modulate using the offset quadrature phase shift keying method

Library

PM, in Digital Baseband sublibrary of Modulation

Description

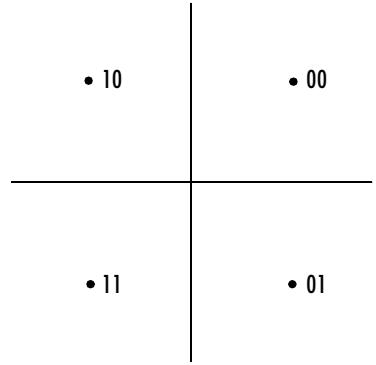
The OQPSK Modulator Baseband block modulates using the offset quadrature phase shift keying method. The output is a baseband representation of the modulated signal.

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.

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 figure below. If the block's **Phase offset** parameter is nonzero, then this constellation is rotated by that parameter value.



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

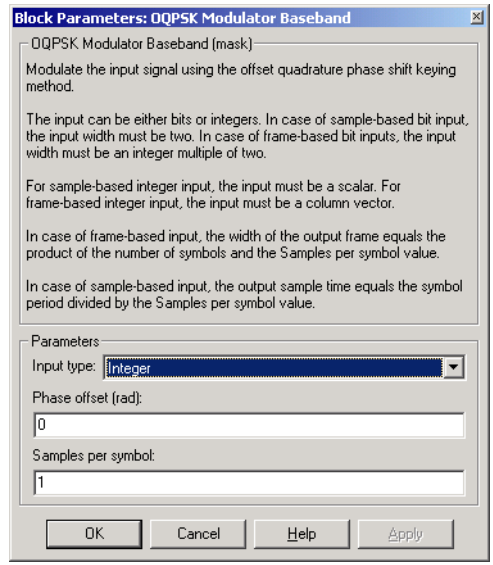
sample time equals the input sample time, even though the period of each output symbol is half the period of each integer or bit pair in the input.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter, S, is the upsampling factor. It must be a positive integer.

- If the input is a frame-based column vector, then the output vector length is 2S times the number of integers or pairs of bits in the input vector, while the output sample time equals the input sample time. The one-symbol initial condition inherent in this block corresponds to the first S elements of the first output vector.
- If the input is a sample-based scalar, then the output vector is a scalar, while the output sample time is 1/(2S) times the input sample time. The one-symbol initial condition inherent in this block corresponds to the first S samples.

Dialog Box



Input type

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

Phase offset (rad)

The amount by which the phase of the zeroth point of the signal constellation is shifted from π/4.

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

Pair Block

OQPSK Demodulator Baseband

See Also

QPSK Modulator Baseband

OQPSK Modulator Passband

Purpose

Modulate using the offset quadrature phase shift keying method

Library

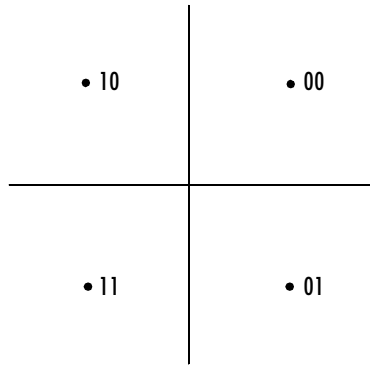
PM, in Digital Passband sublibrary of Modulation

Description

The OQPSK Modulator Passband block modulates using the offset quadrature phase shift keying method. The output is a passband representation of the modulated signal.

If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. In this case, the input must be a sample-based scalar. If the **Input type** parameter is set to **Bit**, then the input must be a binary-valued sample-based vector of length two.

The constellation used to map bit pairs to symbols is in the figure below.



Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

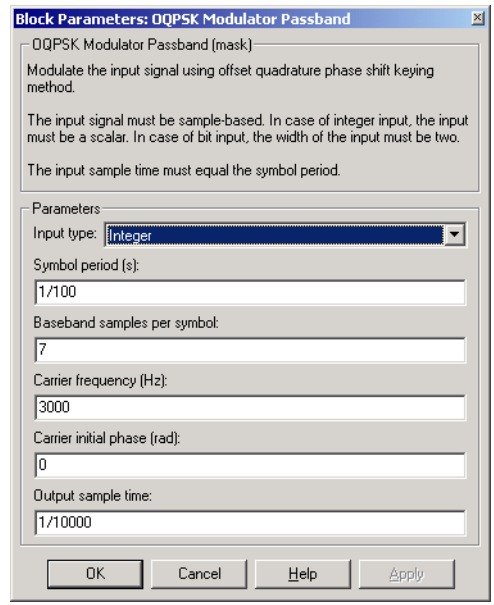
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2*__Carrier frequency__ + 2/(**Symbol period**)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

Dialog Box



Input type

Indicates whether the input consists of integers or pairs 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 period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or pair of bits in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

OQPSK Demodulator Passband

See Also

OQPSK Modulator Baseband

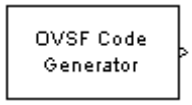
Purpose

Generate an orthogonal variable spreading factor (OVSF) code from a set of orthogonal codes

Library

Spreading Codes

Description



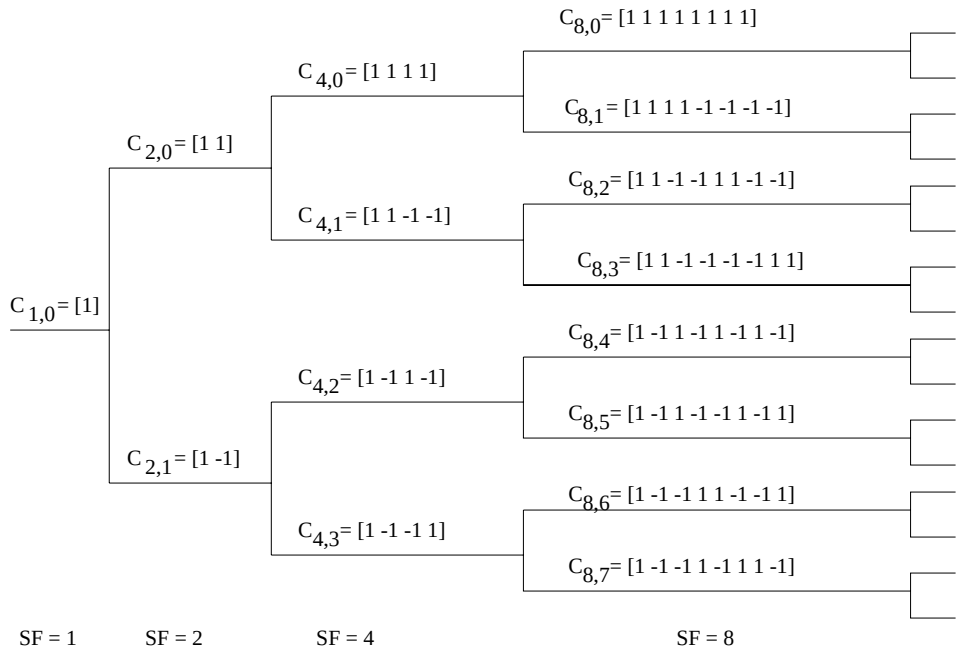
The OVSF Code Generator block generates an OVSF code from a set of orthogonal codes. OVSF codes were first introduced for 3G communication systems. OVSF codes are primarily used to preserve orthogonality between different channels in a communication system.

OVSF codes are defined as the rows of an $N \times N$ 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

$$C_{2N} = \begin{bmatrix} C_N(0) & C_N(0) \\ C_N(0) & -C_N(0) \\ C_N(1) & C_N(1) \\ C_N(1) & -C_N(1) \\ \cdots & \cdots \\ C_N(N-1) & C_N(N-1) \\ C_N(N-1) & -C_N(N-1) \end{bmatrix}$$

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.



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 OVSV Code Generator block outputs by two parameters in the block's mask: the **Spreading factor**, which is the length of the code, and the **Code index**, which must be an integer in the range [0, 1, ... , *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**.

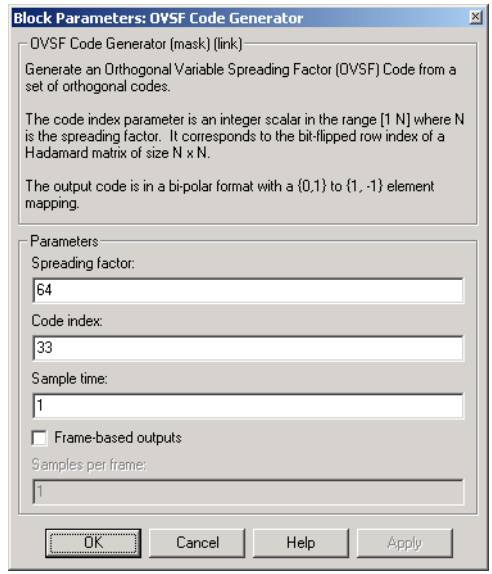
For example, to find the code with **Spreading factor** 16 and **Code index** 6, do the following:

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	$C_2 = C_1\ -C_1$ = [1 1] [-1 -1]
3	1	$C_3 = C_2\ -C_2$ = [1 1 -1 -1] [-1 -1 1 1]
4	0	$C_4 = C_3\ C_3$ = [1 1 -1 -1 -1 -1 1 1] [1 1 -1 -1 -1 -1 1 1]

The code C_4 has **Spreading factor** `16` and **Code index** `6`.

Dialog Box



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.

See Also

Hadamard Code Generator, Walsh Code Generator

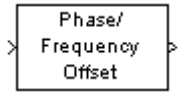
Purpose

Apply phase and frequency offsets to a complex baseband signal.

Library

RF Impairments

Description



The Phase/Frequency Offset block first applies a phase offset and then a frequency offset to a complex, baseband 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**:

You can view the implementation of the phase or frequency offsets by double-clicking the Phase Offset or Frequency Offset subsystems under the mask.

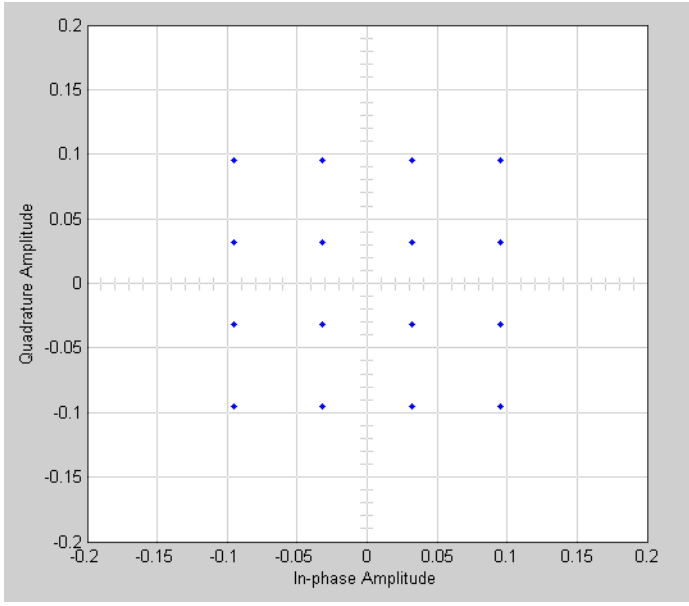
Phase Offset

The block applies a phase offset to the input signal, specified by the **Phase offset (deg)** parameter.

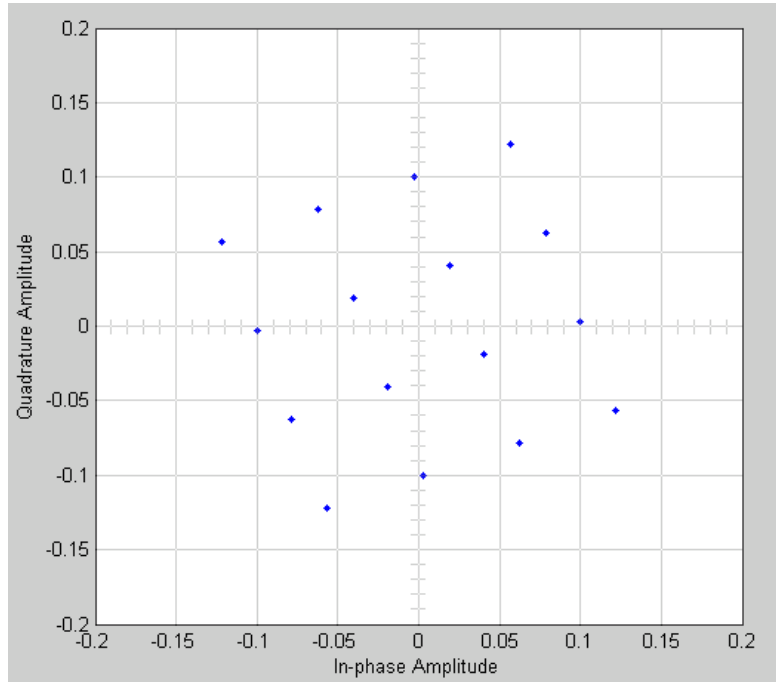
Frequency Offset

The block applies a frequency offset to the signal that is specified by the **Frequency offset (Hz)** parameter.

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:

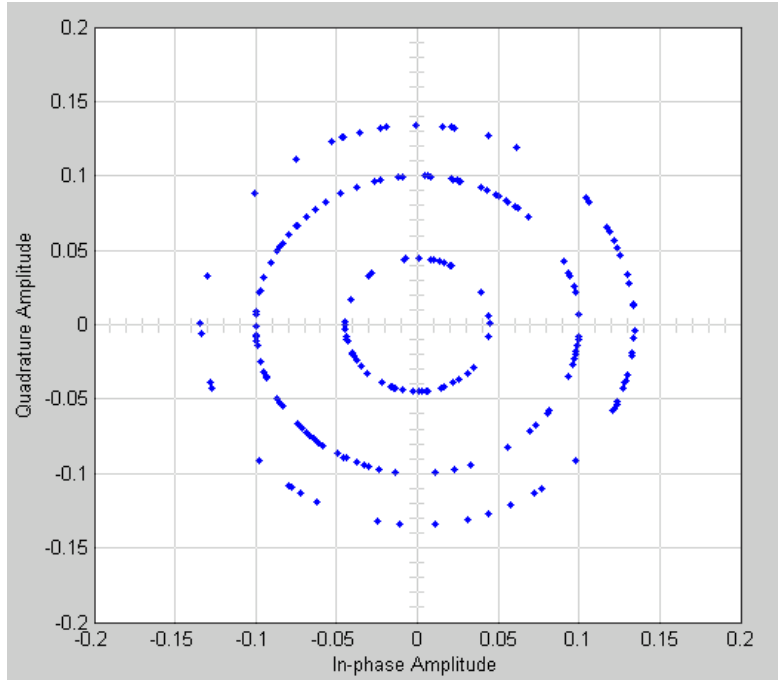

0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
Quadrature Amplitude
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
In-phase Amplitude

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 (deg)** set to 20 and **Frequency offset (Hz)** set to 0:



Observe that each point in the constellation is rotated by a 20 degree angle counterclockwise.

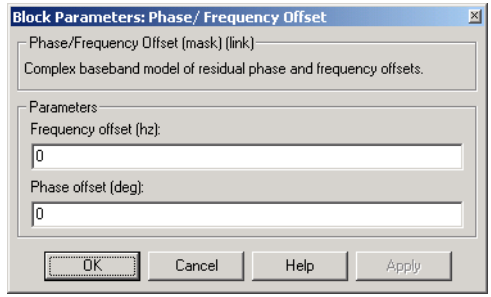
If you set **Frequency offset (Hz)** to 2 and **Phase offset (deg)** to 0, the angles of points in the constellation change linearly over time. This causes points in the scatter plot to shift radially, as shown in the following figure:



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.

Dialog Box



Frequency offset (hz)

Scalar specifying the frequency offset in Hertz.

Phase offset (deg)

Scalar specifying the phase offset in degrees.

See Also

Phase Noise

Phase-Locked Loop

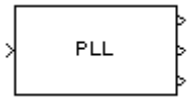
Purpose

Implement a phase-locked loop to recover the phase of the input signal

Library

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** mask 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 the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

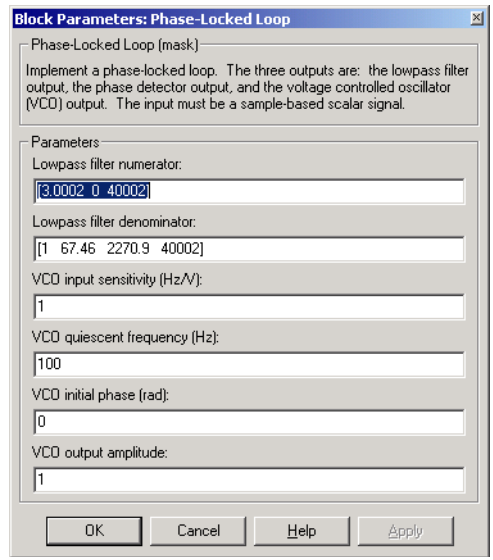
  ```
  [num, den] = cheby2(3,40,100,'s')
  ```

- 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

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 Using the Communications Blockset.

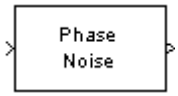
Purpose

Apply receiver phase noise to a complex baseband signal

Library

RF Impairments

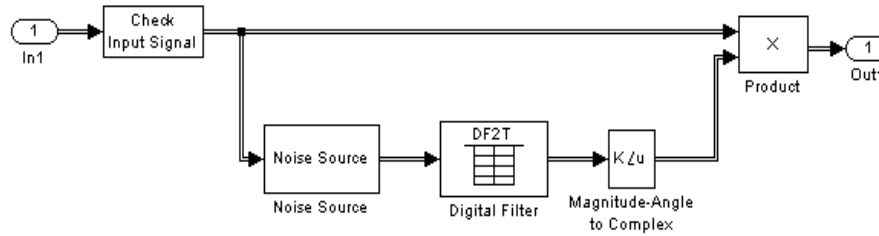
Description



The Phase Noise block appies phase noise to a complex, baseband signal. The block applies the phase noise as follows:

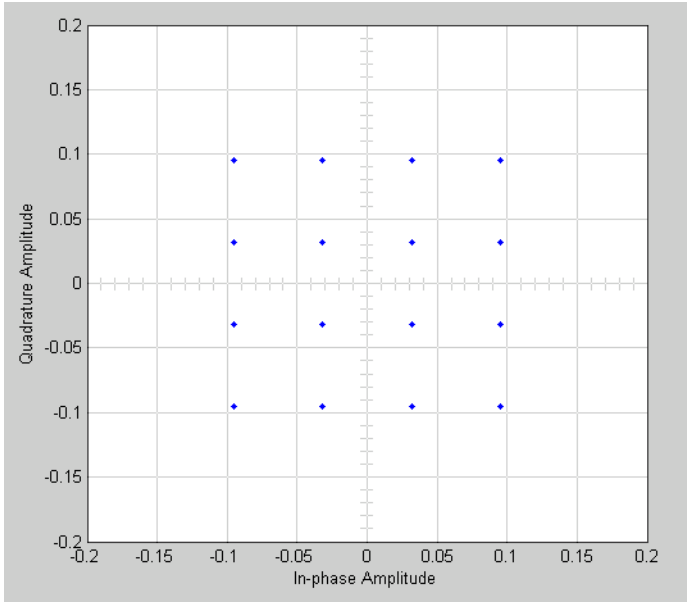
1. Generates additive white Gaussian noise (AWGN) and filters it with a digital filter.
2. Adds the resulting noise to the angle component of the input signal.

You can view the block's implementation of phase noise by right-clicking on the block and selecting **Look under mask** from the pop-up menu. This displays the following figure:

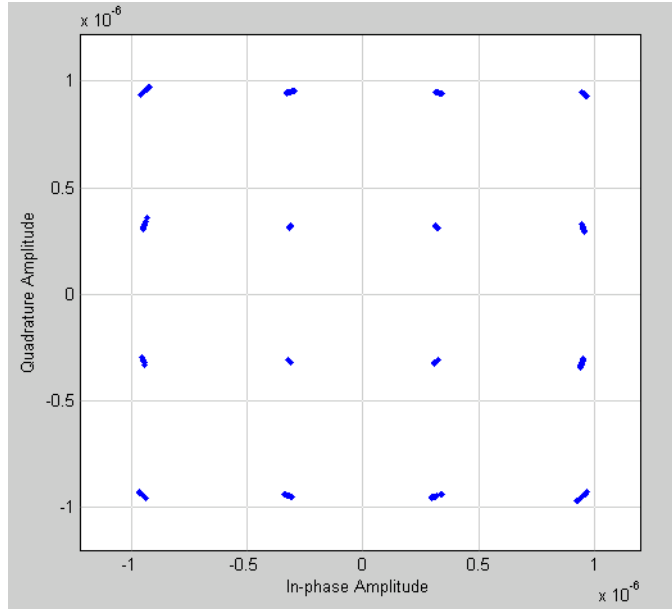


You can view the construction of the Noise Source subsystem by double-clicking it.

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`:

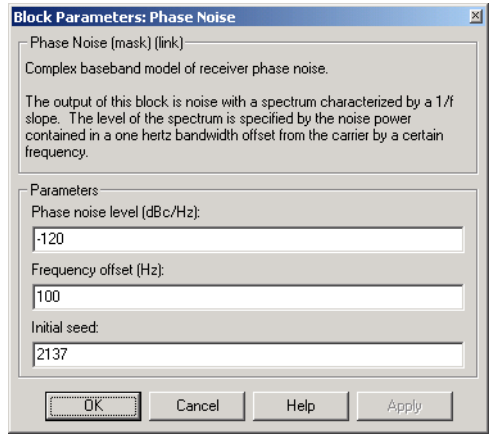


This plot is generated by the model described in “Scatter Plot Examples,” with the following parameter settings for the Rectangular QAM Modulator Baseband block:

- **Normalization method** set to **Average Power**
- **Average power (watts)** set to `1e-12`

Phase Noise

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

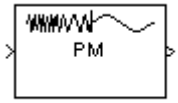
Purpose

Demodulate PM-modulated data

Library

Analog Baseband Modulation, in Modulation

Description

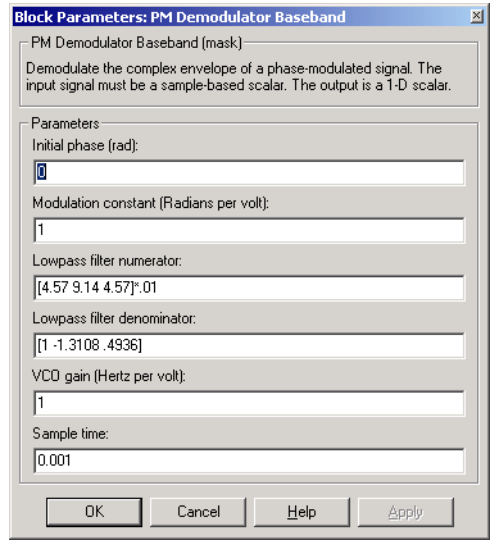


The PM Demodulator Baseband block demodulates a signal that was modulated using phase modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

This block uses a phase-locked loop containing a voltage-controlled oscillator (VCO). The **VCO Gain** parameter specifies the input sensitivity of the VCO.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Initial phase (rad)

The initial phase in the corresponding PM Modulator Baseband block.

Modulation constant (Radians per volt)

The modulation constant in the corresponding PM Modulator Baseband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

VCO gain (Hertz per volt)

The input sensitivity of the voltage-controlled oscillator.

Sample time

The sample time of the output signal.

Pair Block

PM Modulator Baseband

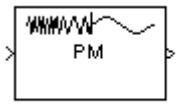
Purpose

Demodulate PM-modulated data

Library

Analog Passband Modulation, in Modulation

Description



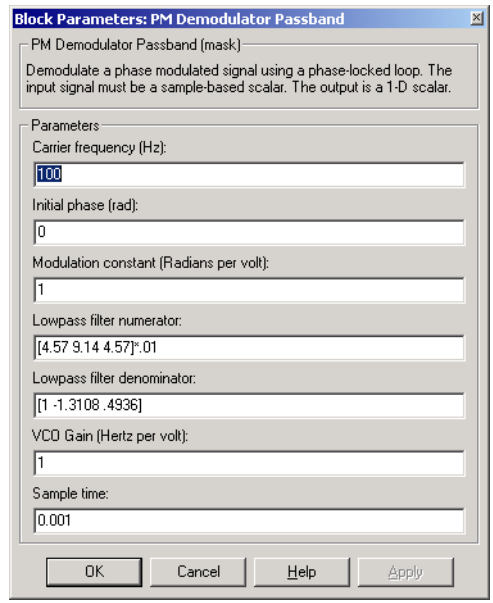
The PM Demodulator Passband block demodulates a signal that was modulated using phase modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

This block uses a phase-locked loop containing a voltage-controlled oscillator (VCO). The **VCO Gain** parameter specifies the input sensitivity of the VCO.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

By the Nyquist sampling theorem, the reciprocal of the **Sample time** parameter must exceed twice the **Carrier frequency** parameter.

Dialog Box



Carrier frequency (Hz)

The carrier frequency in the corresponding PM Modulator Passband block.

Initial phase (rad)

The carrier signal's initial phase in the corresponding PM Modulator Passband block.

Modulation constant (Radians per volt)

The modulation constant in the corresponding PM Modulator Passband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

VCO Gain (Hertz per volt)

The input sensitivity of the voltage-controlled oscillator.

Sample time

The sample time of the output signal.

Pair Block

PM Modulator Passband

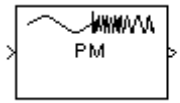
Purpose

Modulate using phase modulation

Library

Analog Baseband Modulation, in Modulation

Description



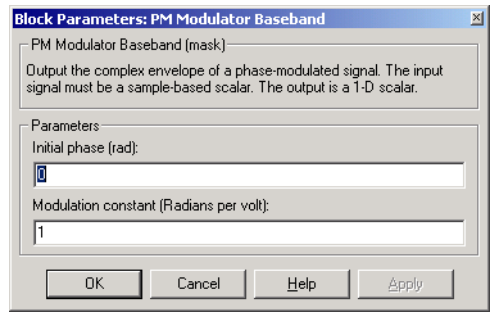
The PM Modulator Baseband block modulates using phase modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

If the input is $u(t)$ as a function of time t, then the output is

$$\exp(j\theta + jK_c u(t))$$

where θ is the **Initial phase** parameter and K_c is the **Modulation constant** parameter.

Dialog Box



Initial phase (rad)

The phase of the modulated signal when the input is zero.

Modulation constant (Radians per volt)

Modulation constant K_c.

Pair Block

PM Demodulator Baseband

PM Modulator Passband

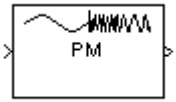
Purpose

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.

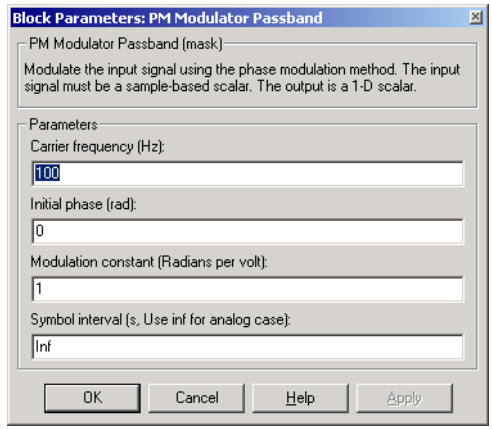
If the input is $u(t)$ as a function of time t, then the output is

$$\cos(2\pi f_c t + K_c u(t) + \theta)$$

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. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (PM Modulator Baseband block) instead of passband simulation.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The initial phase of the carrier in radians.

Modulation constant (Radians per volt)

The modulation constant K_c.

Symbol interval

`Inf` by default. To use this block to model PSK, set this parameter to the length of time required to transmit a single information bit.

Pair Block

PM Demodulator Passband

See Also

PM Modulator Baseband

PN Sequence Generator

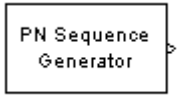
Purpose

Generate a pseudonoise sequence

Library

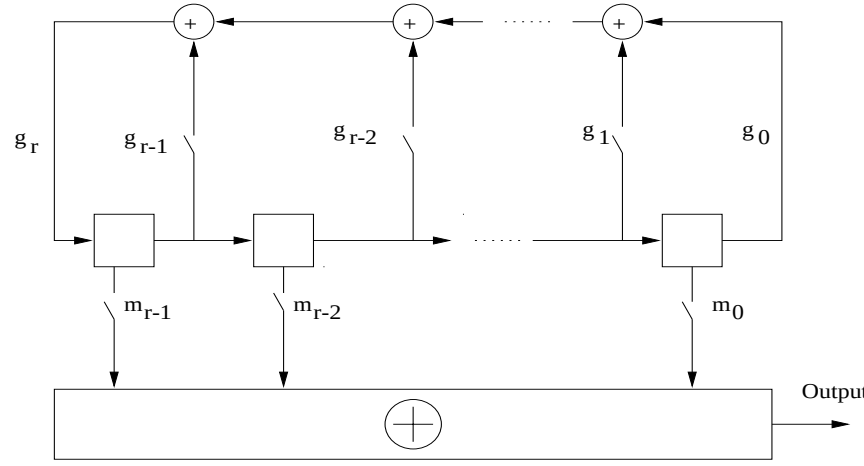
Sequence Generators sublibrary of Comm Sources

Description



The PN Sequence Generator block generates a sequence of pseudorandom binary numbers. A pseudonoise sequence can be used in a pseudorandom scrambler and descrambler. It can also be used in a direct-sequence spread-spectrum system.

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.

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 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]`

The **Shift** parameter 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 the **Shift** parameter. 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 **Shift (or mask)** to 0, versus a positive integer d, is shown in the following table.

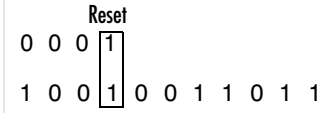
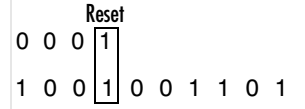
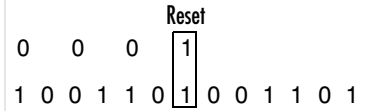
	T = 0	T = 1	T = 2	…	T = d	T = d+1
Shift = 0	x_0	x_1	x_2	…	x_d	x_{d+1}
Shift = d	x_d	x_{d+1}	x_{d+2}	…	x_{2d}	x_{2d+1}

Alternatively, you can set the **Shift** parameter to a binary vector, corresponding to a polynomial in z, $m_{r-1}z^{r-1} + m_{r-2}z^{r-2} + \ldots + m_1 z + m_0$, 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 the **Shift (or mask)** parameter is implemented when you specify it as a mask vector. The default setting for the **Shift (or mask)** parameter is `[0 0 0 1]`, which corresponds to d = 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	
Frame-based **Sample time** =1 **Samples per frame** = 2	Frame-based **Sample time** = 1 **Samples per frame** = 2	
Sample-based **Sample time** = 2 **Samples per frame** = 1	Frame-based **Sample time** = 1 **Samples per frame** = 2	

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.

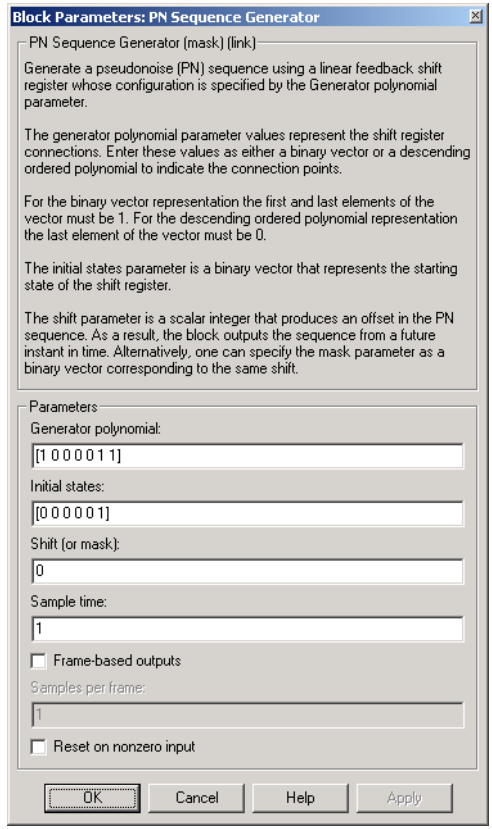
Sequences of Maximum Length

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.

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]

r	Generator Polynomial	r	Generator Polynomial
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]
45	[45 4 3 1 0]	52	[52 3 0]
46	[46 21 10 1 0]	53	[53 6 2 1 0]

Dialog Box



Generator polynomial

Polynomial that determines the shift register's feedback connections.

Initial states

Vector of initial states of the shift registers.

Shift (or mask)

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.

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.

Poisson Integer Generator

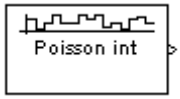
Purpose

Generate Poisson-distributed random integers

Library

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.

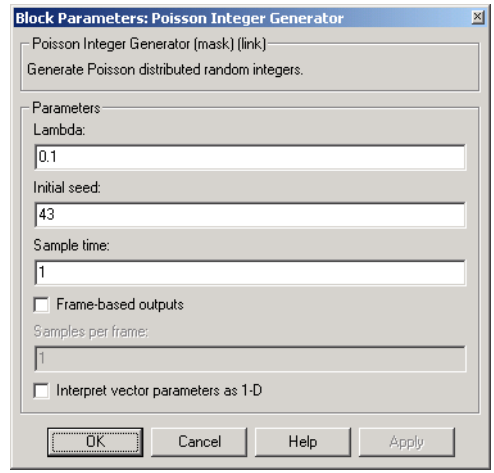
You can use the Poisson Integer Generator to generate noise in a binary transmission channel. In this case, the Poisson parameter **Lambda** should be less than 1, usually much less.

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 Using the Communications Blockset 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



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.

See Also Random Integer Generator; `poissrnd` (Statistics Toolbox)

Purpose

Output the elements which correspond to 1s in the binary Puncture vector

Library

Sequence Operations, in Basic Comm Functions

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.

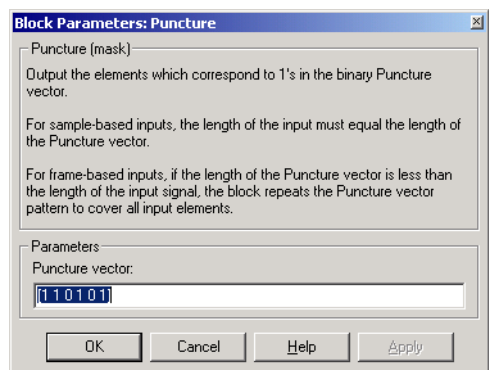
Frame-Based Processing

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

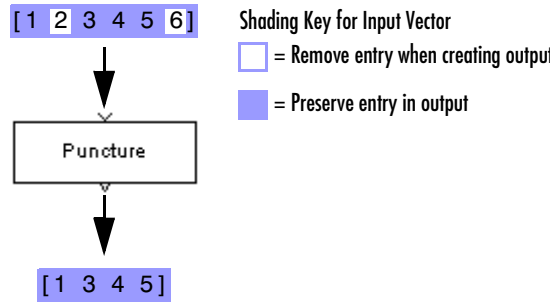
A binary vector whose pattern of 0s (1s) indicates which elements of the input the block should remove (preserve).

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.



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.

If the **Output type** parameter is set to **Integer**, 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.

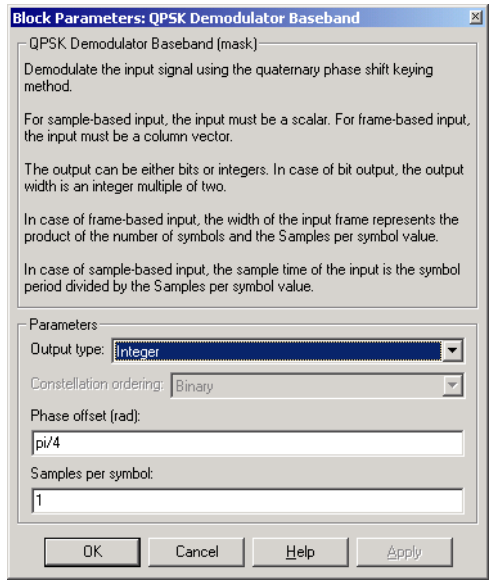
If the **Output type** parameter is set to **Bit**, then the output contains pairs of binary values. The reference page for the QPSK Modulator Baseband block shows the signal constellations for the cases when the **Constellation ordering** parameter is either **Binary** or **Gray**.

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 Using the Communications Blockset.

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 offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

QPSK Modulator Baseband

See Also

M-PSK Demodulator Baseband, BPSK Demodulator Baseband, DQPSK Demodulator Baseband

Purpose

Modulate using the 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.

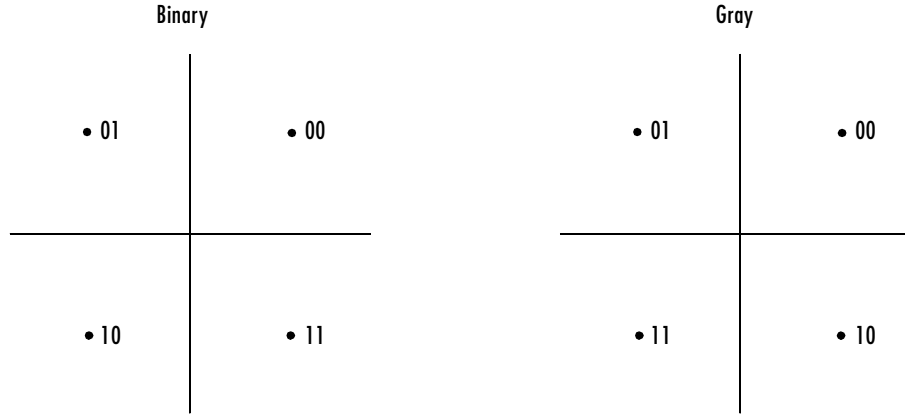
Inputs and Constellation Types

If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. If the input is m, then the output symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** parameter. In this case, the input can be either a scalar or a frame-based column vector.

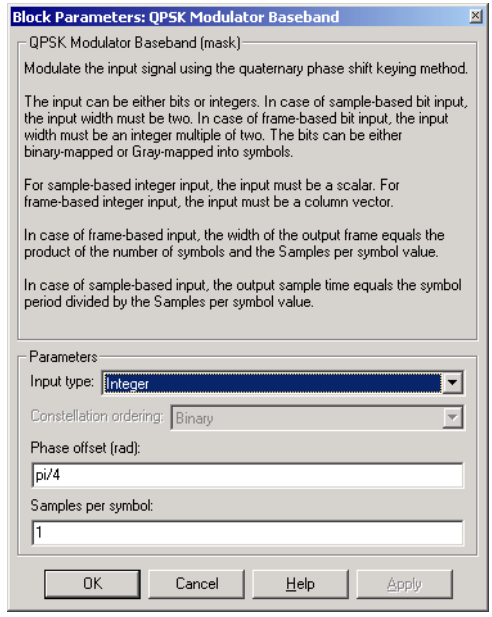
If the **Input type** parameter is set to **Bit**, then the input contains pairs of binary values. The input can be either a vector of length two or a frame-based column vector whose length is an even integer. If the **Phase offset** parameter is set to `pi/4`, then the block uses one of the signal constellations in the figure below, depending on whether the **Constellation ordering** parameter is set to **Binary** or **Gray**.



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 Using the Communications Blockset.

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 offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

Pair Block QPSK Demodulator Baseband

See Also M-PSK Modulator Baseband, BPSK Modulator Baseband, DQPSK Modulator Baseband

Quantizer Decode

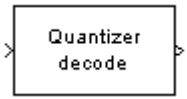
Purpose

Decode quantization index according to codebook

Library

Source Coding

Description

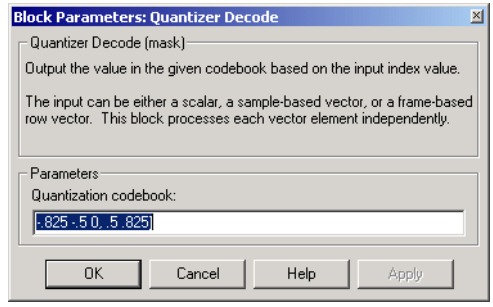


The Quantizer Decode block recovers a message from a quantized signal, converting the quantization index into the corresponding codebook value. The **Quantization codebook** parameter, a vector of length N, prescribes the possible output values. If the input is an integer k between 0 and N-1, then the output is the (k+1)st element of **Quantization codebook**.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the input signal.

Note The Sampled Quantizer Encode and Enabled Quantizer Encode blocks also use a **Quantization codebook** parameter. The first output of those blocks corresponds to the input of Quantizer Decode; the second output of those blocks corresponds to the output of Quantizer Decode.

Dialog Box



Quantization codebook

A real vector that prescribes the output value corresponding to each input integer.

Pair Block

Sampled Quantizer Encode or Enabled Quantizer Encode

Purpose

Restore ordering of the input symbols using a random permutation

Library

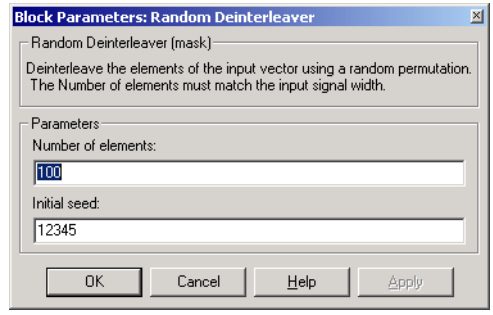
Block sublibrary of Interleaving

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 the Random 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.

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

See Also

General Block Deinterleaver

Random Integer Generator

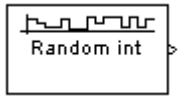
Purpose

Generate integers randomly distributed in the range [0, M-1]

Library

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.

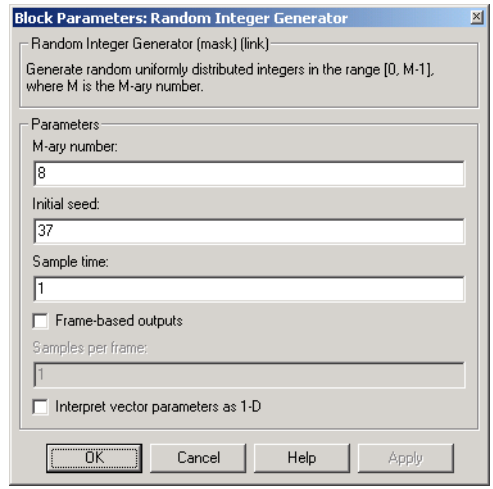
If the **Initial seed** parameter is a constant, then the resulting noise is repeatable.

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 Using the Communications Blockset 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



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.

See Also

`randint` (Communications Toolbox)

Purpose

Reorder the input symbols using a random permutation

Library

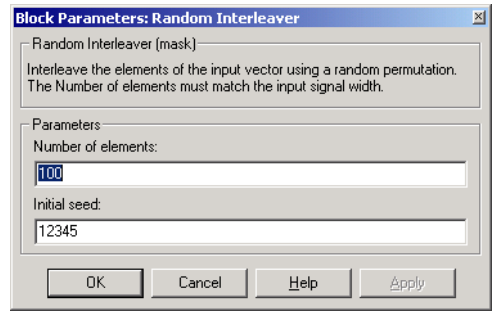
Block sublibrary of Interleaving

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.

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

$$f(x) = \begin{cases} \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

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's `randseed` function. At the MATLAB prompt, type the command

```
randseed
```

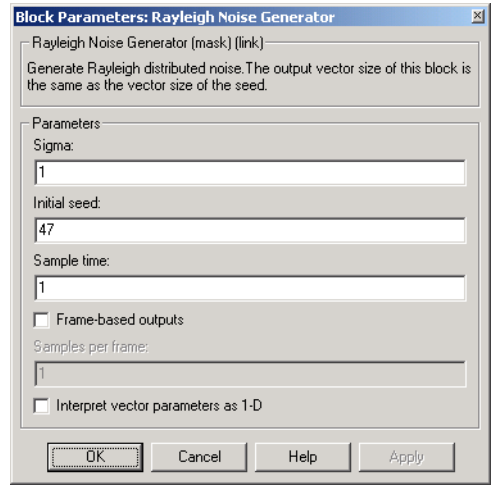
This returns a random prime number greater than 30. Typing `randseed` again produces a different prime number. If you add an integer argument, `randseed` always returns the same prime for that integer. For example, `randseed(5)` always returns the same answer.

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 Using the Communications Blockset 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



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.

See Also

Multipath Rayleigh Fading Channel; `raylrnd` (Statistics Toolbox)

References

[1] Proakis, John G. *Digital Communications*, Third edition. New York: McGraw Hill, 1995.

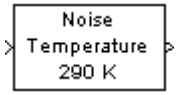
Purpose

Apply receiver thermal noise to a complex baseband signal

Library

RF Impairments

Description



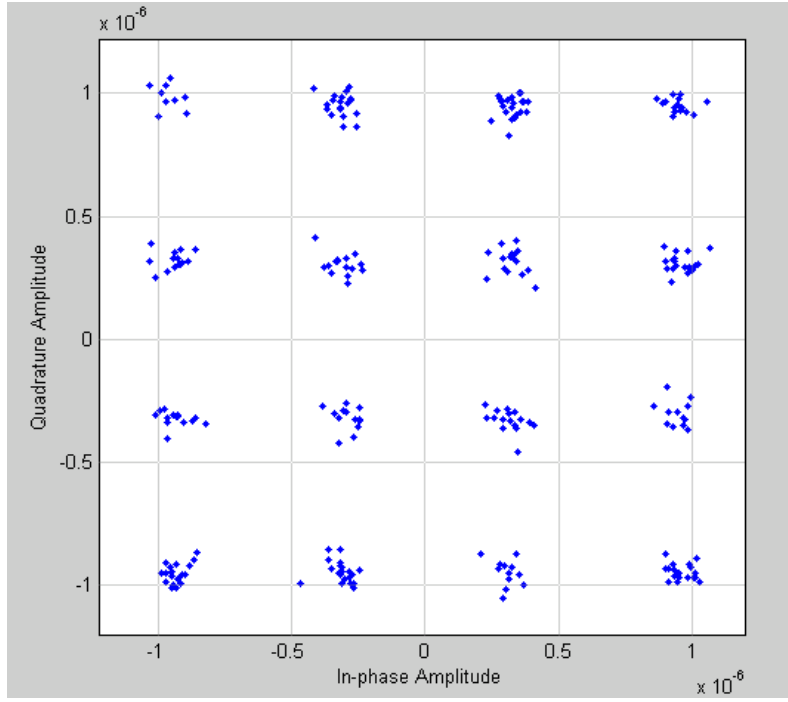
The Receiver Thermal Noise block simulates the effects of thermal noise on a complex, baseband signal. You can specify the amount of thermal noise in three ways, according to which **Specification method** you select:

- **Noise temperature** – specifies the noise in degrees Kelvin
- **Noise figure** – specifies the noise in decibels relative to a noise temperature of 290 degrees Kelvin
- **Noise factor** – specifies the noise in by the following equation6:

$$\text{Noise factor} = 1 + \frac{\text{Noise temperature}}{290}$$

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.

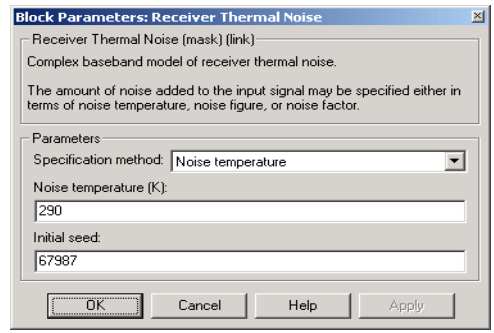
s



This plot is generated by the model described in "Scatter Plot Examples, with the following parameter settings:

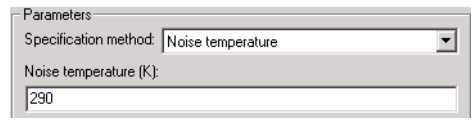
- Rectangular QAM Modulator Baseband
  - **Normalization method** set to **Average Power**
  - **Average power (watts)** set to `1e-12`
- Receiver Thermal Noise
  - **Specification method** set to **Noise figure**
  - **Noise figure (dB)** set to `3.01`

Dialog Box



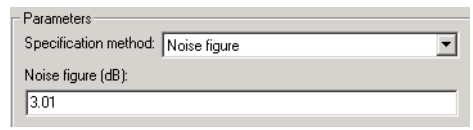
Specification method

The method by which you specify the amount of noise. The choices are **Noise temperature**, **Noise figure**, and **Noise factor**.



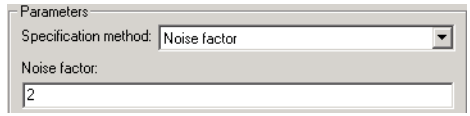
Noise temperature (K)

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

Purpose

Demodulate 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.

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.

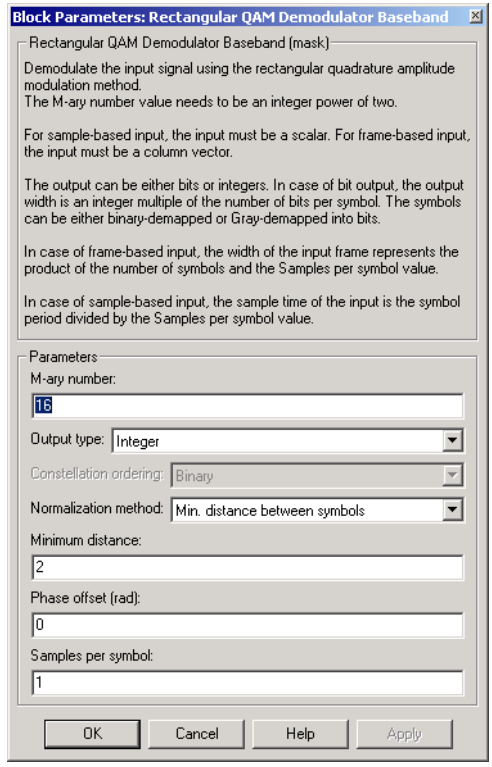
Output Signal Values

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 Rectangular QAM Modulator Baseband block.

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 Using the Communications Blockset.

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.

Output type

Indicates 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. 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power among the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Phase offset (rad)

The rotation of the signal constellation, in radians.

Samples per symbol

The number of input samples that represent each modulated symbol.

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.

Rectangular QAM Demodulator Passband

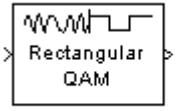
Purpose

Demodulate QAM-modulated data

Library

AM, in Digital Passband sublibrary of Modulation

Description



The Rectangular QAM Demodulator Passband block demodulates a signal that was modulated using quadrature amplitude modulation with a constellation on a rectangular lattice. 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.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PAM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

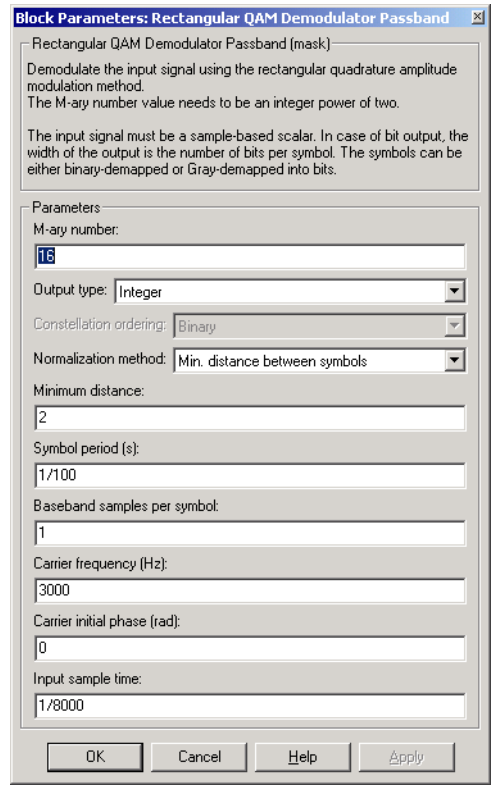
- **Symbol period** > (**Carrier frequency**)$^{-1}$

- **Input sample time** < [2*__Carrier frequency__ + 2/(__Symbol period__)]$^{-1}$

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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.

Output type

Indicates 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. 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power among the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block

Rectangular QAM Modulator Passband

See Also

General QAM Demodulator Passband, Rectangular QAM Demodulator Baseband

References

[1] Smith, Joel G, “Odd-Bit Quadrature Amplitude-Shift Keying,” *IEEE Transactions on Communications*, vol. COM-23, March 1975, pp. 385-389.

Rectangular QAM Modulator Baseband

Purpose

Modulate using M-ary 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.

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 table below 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.

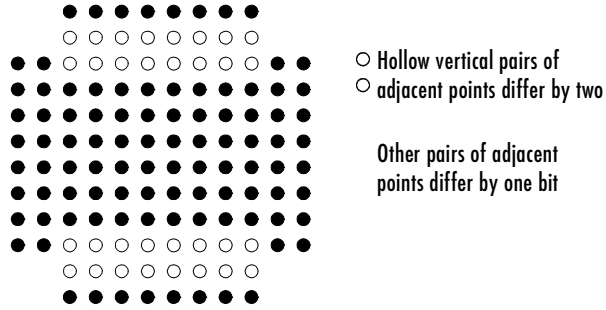
Input Signal Values

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.
- 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. 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**, then the block uses a natural binary-coded constellation.
- If **Constellation ordering** is set to **Gray** and K is even, then the block uses a Gray-coded constellation.
- If **Constellation ordering** is set to **Gray** and K is odd, then 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.

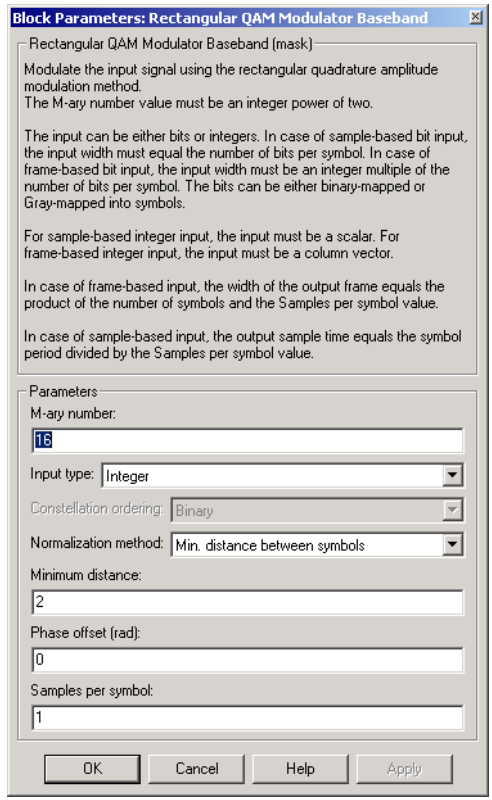


For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block and the paper listed in "References" below. Note that since the in-phase and quadrature components are assigned independently, the Gray and binary orderings coincide when M = 4.

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 Using the Communications Blockset.

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 group of input bits to a corresponding integer. This field is active only when **Input 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Phase offset (rad)

The rotation of the signal constellation, in radians.

Samples per symbol

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

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 Modulator Passband

Purpose

Modulate using M-ary quadrature amplitude modulation

Library

AM, in Digital Passband sublibrary of Modulation

Description

The Rectangular QAM Modulator Passband block modulates using M-ary quadrature amplitude modulation with a constellation on a rectangular lattice. The output is a passband representation of the modulated signal. 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.

This block uses the baseband equivalent block, Rectangular QAM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

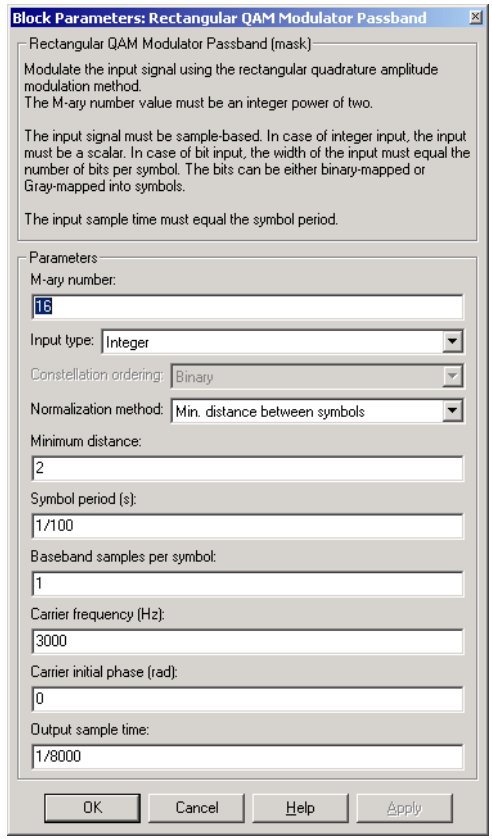
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)$^{-1}$
- **Output sample time** < [2*__Carrier frequency__ + 2/(**Symbol period**)]$^{-1}$

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Note A model containing this block must use a variable-step solver. To configure a model so that it uses a variable-step solver, select **Simulation parameters** from the model window's **Simulation** menu and then set the **Type** parameter on the **Solver** panel to **Variable-step**.

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 group of input bits to a corresponding integer. This field is active only when **Input 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 (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

Rectangular QAM Demodulator Passband

See Also

General QAM Modulator Passband, Rectangular 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.

Rician Fading Channel

Purpose

Simulate a Rician fading propagation channel

Library

Channels

Description

The Rician Fading Channel block implements a baseband simulation of a Rician fading propagation channel. This block is useful for modeling mobile wireless communication systems when the transmitted signal can travel to the receiver along a dominant line-of-sight or direct path. If the signal can travel along a line-of-sight path and also along other fading paths, then you can use this block in parallel with the Multipath Rayleigh Fading Channel block. For details about fading channels, see the works listed in "References" on page 2-506.

The input can be either a scalar or a frame-based column vector. The input is a complex signal.

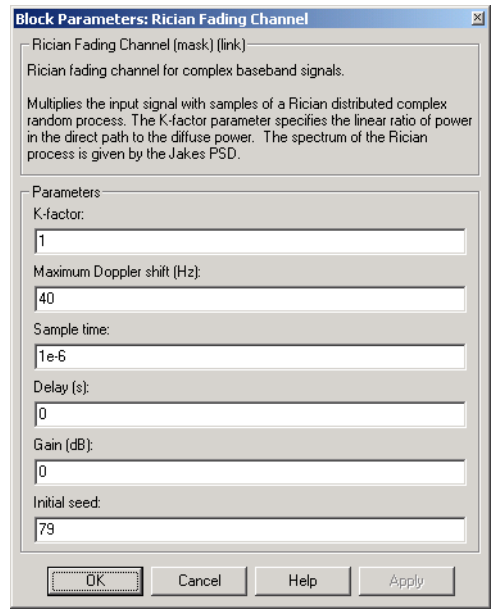
Fading causes the signal to spread and become diffuse. The **K-factor** parameter, which is part of the statistical description of the Rician distribution, represents the ratio between direct-path (unspread) power and diffuse power. The ratio is expressed linearly, not in decibels. While the **Gain** parameter controls the overall gain through the channel, the **K-factor** parameter controls the gain's partition into direct and diffuse components.

Relative motion between the transmitter and receiver causes Doppler shifts in the signal frequency. The Jakes PSD (power spectral density) determines the spectrum of the Rician process.

The **Sample time** parameter is the time between successive elements of the input signal. Note that if the input is a frame-based column vector of length n, then the frame period (as Simulink's Probe block reports, for example) is n*__Sample time__.

The **Delay** parameter specifies a time delay in seconds and the **Gain** parameter specifies a gain that applies to the input signal. Both parameters are scalars.

Dialog Box



K-factor

The ratio of power in the direct path to diffuse power. The ratio is expressed linearly, not in decibels.

Maximum Doppler shift (Hz)

A positive scalar that indicates the maximum Doppler shift.

Sample time

The period of each element of the input signal.

Delay (s)

A scalar that specifies the propagation delay.

Gain (dB)

A scalar that specifies the gain.

Initial seed

The scalar seed for the Gaussian noise generator.

See Also

Rician Noise Generator, Multipath Rayleigh Fading Channel

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.

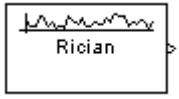
Purpose

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

$$f(x) = \begin{cases} \frac{x}{\sigma^2} I_0\left(\frac{mx}{\sigma^2}\right) e^{-\frac{x^2+m^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

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's `randseed` function. At the MATLAB prompt, type the command

```
randseed
```

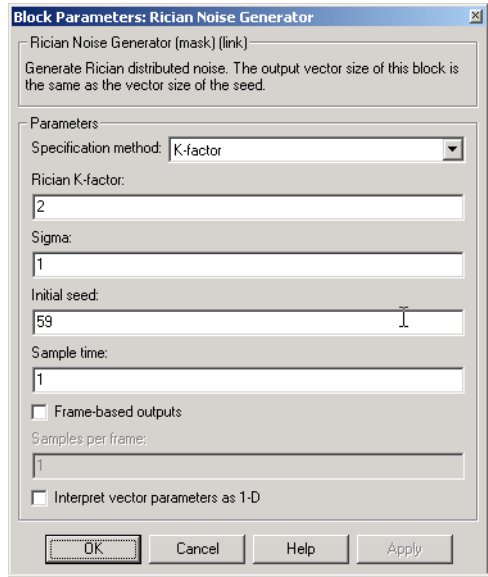
This returns a random prime number greater than 30. Typing `randseed` again produces a different prime number. If you add an integer argument, `randseed` always returns the same prime for that integer. For example, `randseed(5)` always returns the same answer.

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 Using the Communications Blockset 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.

Dialog Box



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.

See Also

Rician Fading Channel

References

[1] Proakis, John G. *Digital Communications*, Third edition. New York: McGraw Hill, 1995.

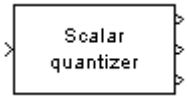
Purpose

Quantize a signal, indicating quantization index, coded signal, and distortion

Library

Source Coding

Description



The Sampled Quantizer Encode block encodes an input signal using scalar quantization. The block outputs the quantization levels (or quantization index) of the input signal, the encoded signal, and the mean square distortion.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the input signal.

The **Quantization partition** parameter is a length-n real vector whose entries are in strictly ascending order. The first output signal corresponding to an input signal of x is:

- 0 if $x \le$ **Quantization partition**(1)
- m if **Quantization partition**(m) $< x \le$ **Quantization partition**(m+1)
- n if **Quantization partition**(n) $< x$

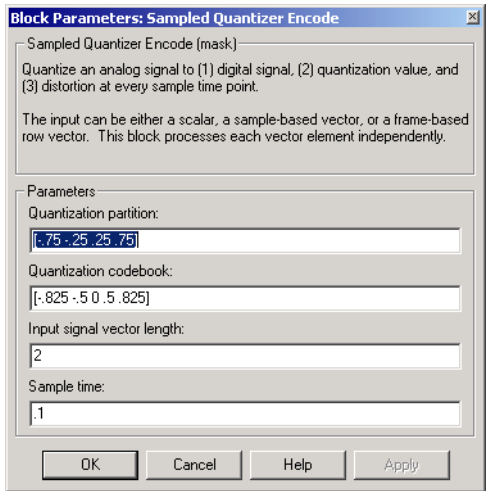
The **Quantization codebook** parameter, whose length exceeds the length of **Quantization partition** by one, prescribes a value for each partition in the quantization. The first element of **Quantization codebook** is the value for the interval between negative infinity and the first element of **Quantization partition**. The second output signal from this block contains the quantization of the input based on the quantization levels and prescribed values.

At a given time, the third output signal measures the mean square distortion between the input and the second output, considering the stream of data up through that time.

You can use the function `lloyds` in the Communications Toolbox with a representative sample of your data as training data, to obtain appropriate partition and codebook parameters.

Sampled Quantizer Encode

Dialog Box



Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values assigned to each partition.

Input signal vector length

The length of the input signal.

Sample time

The output sample time.

Pair Block

Quantizer Decode

See Also

Enabled Quantizer Encode; `lloyds` (Communications Toolbox)

Purpose

Scrambler

Purpose

Scramble the input signal

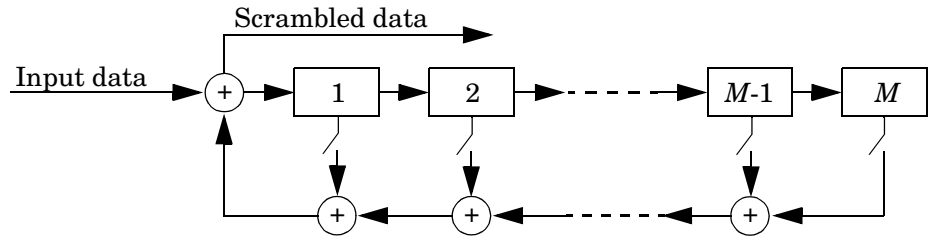
Library

Sequence Operations, in Basic Comm Functions

Description

The Scrambler block scrambles the scalar input signal. If the **Calculation base** parameter is N, then the input values must be integers between 0 and N-1.

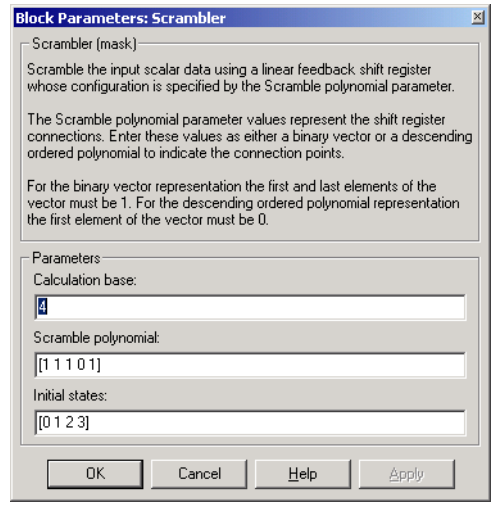
One purpose of scrambling is to reduce the length of strings of 0s or 1s in a transmitted signal, since a long string of 0s or 1s may cause transmission synchronization problems. Below is a schematic of the scrambler. All adders perform addition modulo N.



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}, 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

SSB AM Demodulator Baseband

Purpose

Demodulate SSB-AM-modulated data

Library

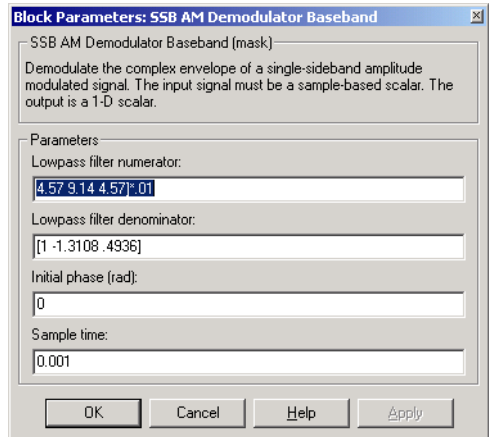
Analog Baseband Modulation, in Modulation

Description

The SSB AM Demodulator Baseband block demodulates a signal that was modulated using single-sideband amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Initial phase (rad)

The initial phase in the corresponding SSB AM Modulator Baseband block.

Sample time

The sample time of the output signal.

Pair Block

SSB AM Modulator Baseband

See Also

DSB AM Demodulator Baseband, DSBSC AM Demodulator Baseband

SSB AM Demodulator Passband

Purpose

Demodulate SSB-AM-modulated data

Library

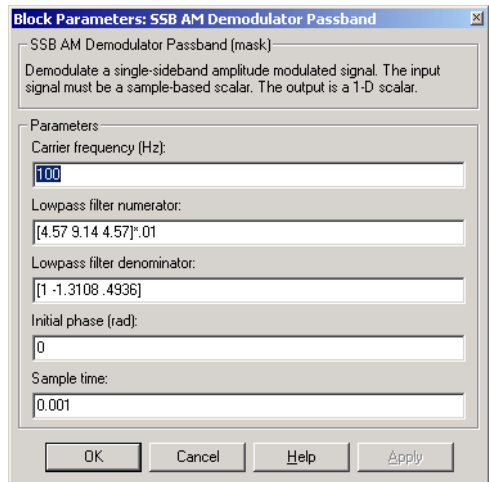
Analog Passband Modulation, in Modulation

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.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Carrier frequency (Hz)

The carrier frequency in the corresponding SSB AM Modulator Passband block.

Lowpass filter numerator

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

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of *s*. For an FIR filter, set this parameter to `1`.

Initial phase (rad)

The initial phase of the carrier in radians.

Sample time

The sample time of the output signal.

Pair Block

SSB AM Modulator Passband

See Also

SSB AM Demodulator Baseband, DSB AM Demodulator Passband, DSBSC AM Demodulator Passband

SSB AM Modulator Baseband

Purpose

Modulate using single-sideband amplitude modulation

Library

Analog Baseband Modulation, in Modulation

Description

The SSB AM Modulator Baseband block modulates using single-sideband amplitude modulation with a Hilbert transform filter. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

SSB AM Modulator Baseband transmits either the lower or upper sideband signal, but not both. To control which sideband it transmits, use the **"upper" sideband or "lower" sideband** parameter.

If the input is $u(t)$ as a function of time t, then the output is

$$(u(t) \pm j\hat{u}(t))e^{j\theta}$$

where θ is the **Initial phase** parameter and $\hat{u}(t)$ is the Hilbert transform of the input $u(t)$. The plus sign indicates the upper sideband and the minus sign indicates the lower sideband.

Hilbert Tranform Filter Parameters

This block uses a Hilbert transform filter, possibly with a compensator. The filter produces a Hilbert transform of its input signal. These mask parameters relate to the Hilbert transform filter:

- The **Time delay for Hilbert transform filter** parameter specifies the delay in the filter design. You should choose a value of the form

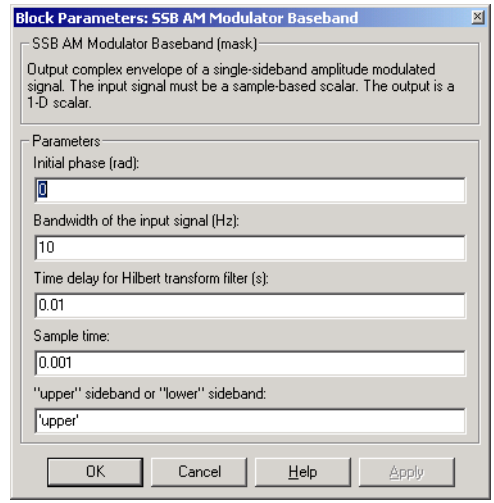
  (N+1/2)*(**Sample time** parameter)

  where N is a positive integer.
- The **Bandwidth of the input signal** parameter is the estimated highest frequency component in the input message signal. This parameter is used to design a compensator for the Hilbert transform filter, which would force the message signal amplitude to remain within the assigned range.

  If this parameter is either 0 or larger than 1/(2***Sample time**), then the block does not generate a compensator.

This block uses the `hilbiir` function in the Communications Toolbox to design the Hilbert transform filter.

Dialog Box



Initial phase (rad)

The phase offset, θ, of the modulated signal.

Bandwidth of the input signal (Hz)

The highest frequency component of the message signal. To avoid using a compensator in the Hilbert transform filter design, set this to `0`.

Time delay for Hilbert transform filter (s)

The time delay in the design of the Hilbert transform filter.

Sample time

The sample time of the Hilbert transform filtering.

“upper” sideband or “lower” sideband

A string that specifies whether to transmit the upper or lower sideband. Choices are `'upper'` and `'lower'`.

Pair Block

SSB AM Demodulator Baseband

See Also

DSB AM Modulator Baseband, DSBSC AM Modulator Baseband

References

[1] Peebles, Peyton Z, Jr. *Communication System Principles*. Reading, Mass.: Addison-Wesley, 1976.

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 **"upper" sideband or "lower" sideband** 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 Parameters

This block uses a Hilbert transform filter, possibly with a compensator. These mask parameters relate to the Hilbert transform filter:

- The **Time delay for Hilbert transform filter** parameter specifies the delay in the filter design. You should choose a value of the form

  (N+1/2)*(**Sample time** parameter)

  where N is a positive integer.
- The **Bandwidth of the input signal** parameter is the estimated highest frequency component in the input message signal.

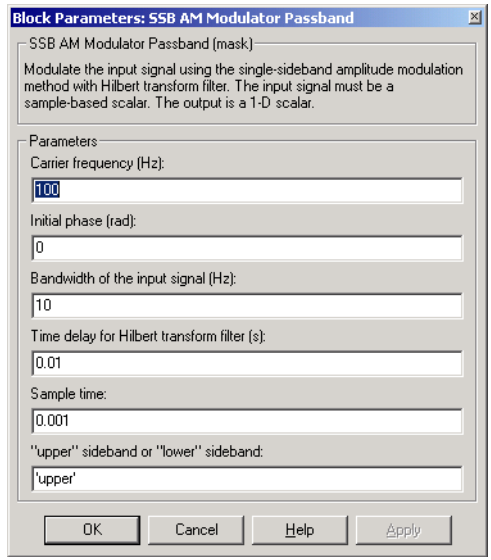
  This parameter is used to design a compensator for the Hilbert transform filter, which would force the message signal amplitude to remain within the

assigned range. If this parameter is either 0 or larger than 1/(2***Sample time**), then the block does not generate a compensator.

This block uses the `hilbiir` function in the Communications Toolbox to design the Hilbert transform filter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (SSB AM Modulator Baseband block) instead of passband simulation.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The phase offset, θ, of the modulated signal.

Bandwidth of the input signal (Hz)

The highest frequency component of the message signal. To avoid using a compensator in the Hilbert transform filter design, set this to `0`.

Time delay for Hilbert transform filter (s)

The time delay in the design of the Hilbert transform filter.

Sample time

The sample time of the Hilbert transform filtering.

“upper” sideband or “lower sideband”

A string that specifies whether to transmit the upper or lower sideband. Choices are `'upper'` and `'lower'`.

Pair Block

SSB AM Demodulator Passband

See Also

SSB AM Modulator Baseband, 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.

Tanh Nonlinearity

Purpose

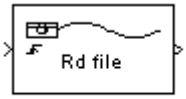
Purpose

Read from a file, refreshing the output at rising edges of an input signal

Library

Controlled Sources sublibrary of Comm Sources

Description



The Triggered Read From File block reads a new record from a file *only* at the rising edge of the input trigger signal. The output is a sample-based signal.

Note The triggered behavior of this block is one difference between this block and Simulink's From File block. However, the From File block is useful for reading platform-independent MAT-files.

The file can be an ASCII text file, a file containing integer or floating point numbers, or a binary file (in the format of the C `fwrite` function). The file must be either in the current working directory on the MATLAB path.

When a rising edge of the input trigger signal is detected, this block reads from the file a record whose length is specified in the parameter **Output vector length**. The first reading always occurs at the first rising edge. After that, if the **Decimation** parameter is a positive integer k, then the block reads at every kth rising edge. If **Decimation** is `1`, then the block reads at every rising edge.

When the block reaches the end-of-file marker, it either:

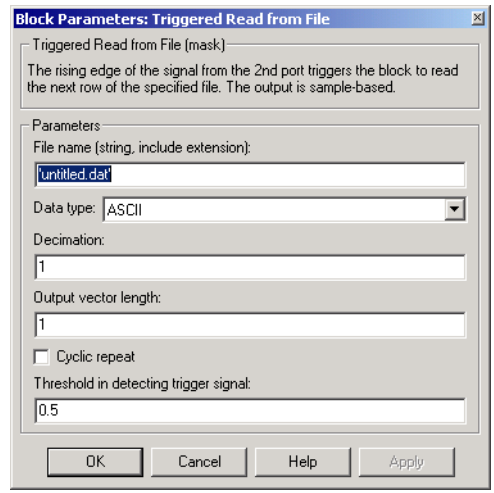
- Rereads from the beginning of the file, if the **Cyclic repeat** box is checked, or
- Outputs zeros, if the **Cyclic repeat** box is not checked

If the **Data type** parameter is **ASCII**, then the output is an integer. The mapping between decimal integers and ASCII characters is shown below.

Integer	ASCII	Integer	ASCII	Integer	ASCII	Integer	ASCII
0	NUL	32	SP	64	@	96	\
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d

Integer	ASCII	Integer	ASCII	Integer	ASCII	Integer	ASCII
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Dialog Box



File name

The filename, including its extension, as a string.

Data type

The data type. Choices are **ASCII**, **binary**, **float**, and **integer**.

Decimation

A decimation factor. If it is 1, then the block reads at every rising edge.

Output vector length

The vector length of the block's output.

Cyclic repeat

Specifies whether to cycle continuously through the contents of **File name** or to output only zeros after reaching the end-of-file marker.

Threshold in detecting trigger signal

The threshold for the rising edge of the trigger signal.

Pair Block

Triggered Write to File

See Also

To File (Simulink)

Triggered Write to File

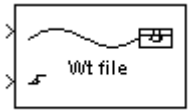
Purpose

Write to a file at each rising edge of an input signal

Library

Comm Sinks

Description



The Triggered Write to File block creates a file containing selected data from the input signal. Unlike Simulink's To File block, the Triggered Write to File block writes new data to the file *only* at the rising edge of the input trigger signal. However, the To File block is useful for creating platform-independent MAT-files.

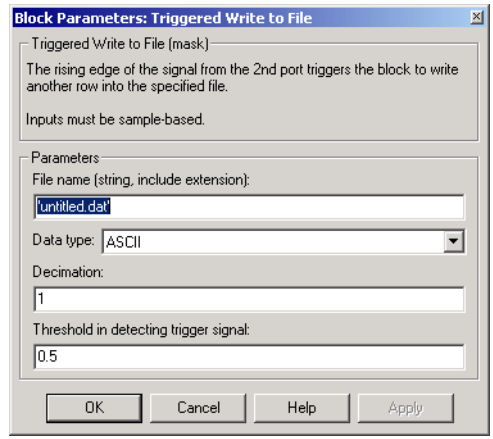
The file can be an ASCII text file, a file containing integer or floating-point numbers, or a binary file (in the format of the C `fwrite` function). You specify the file type using the **Data type** parameter. If **Data type** is **ASCII** then this block converts the data into ASCII characters before writing, using the mapping shown on the reference page for the Triggered Read From File block. For example, an input of 65 would cause the block to write the character "A" to the file. Other file types receive the data directly.

Caution If the destination file already exists, then this block *overwrites* it.

The first input signal contains the data to write. This input must be sample-based. The second input signal is a sample-based scalar trigger signal that controls the timing of writing. When a rising edge of the input trigger signal is detected, this block writes the elements of the data signal to the file. The file does not contain information about the dimension or orientation of the data input, however.

The first write always occurs at the first rising edge. After that, the **Decimation** parameter determines how many triggers the block receives between successive file writes. Setting this parameter to 1 causes the block to write at every rising edge.

Dialog Box



File name

The filename, including its extension, as a string.

Data type

The data type. Choices are **ASCII**, **binary**, **float**, and **integer**.

Decimation

A decimation factor. If it is 1, then the block writes at every rising edge.

Threshold in detecting trigger signal

The threshold for the rising edge of the trigger signal.

Pair Block

Triggered Read From File

See Also

To File (Simulink)

Uniform Noise Generator

Purpose

Generate uniformly distributed noise between the 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.

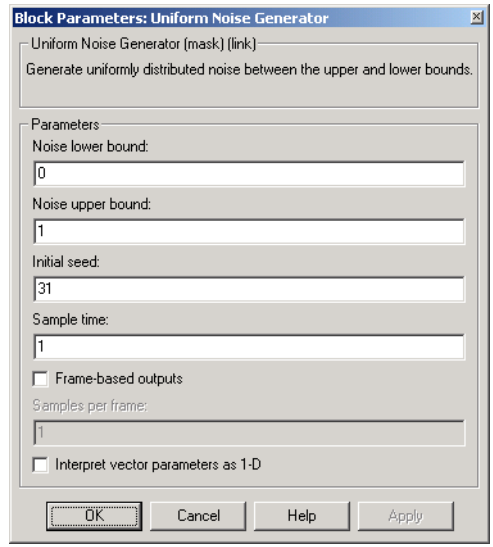
If all the elements of the output vector are to be independent and identically distributed (i.i.d.), then you can use a scalar for the **Noise lower bound** and **Noise upper bound** parameters. Alternatively, you can specify the range for each element of the output vector individually, by using vectors 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 Using the Communications Blockset 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



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.

See Also

Random Source (DSP Blockset); `rand` (built-in MATLAB function)

Purpose

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

Library

Utility Functions

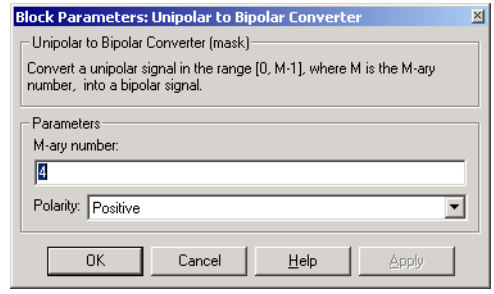
Description

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, and vice-versa.

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.

Polarity

A value of **Positive** (respectively, **Negative**) causes the block to maintain (respectively, reverse) the relative ordering of symbols in the alphabets.

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].

Unipolar to Bipolar Converter

Pair Block Bipolar to Unipolar Converter

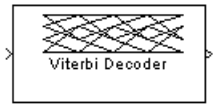
Purpose

Decode convolutionally encoded data using the 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, then 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, then 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.

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	+1: logical zero -1: logical one

Decision type Parameter	Possible Entries in Decoder Input	Interpretation of Values
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, then 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 **Reset** parameter set to **On each 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 encoder. If the encoder has `k` input streams and constraint length vector `constr` (using the polynomial description), then "enough" means `k*max(constr-1)`.

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, then the decoding delay consists of D zero symbols
- If the input signal is frame-based and the **Operation mode** parameter is set to **Continuous**, then the decoding delay consists of D zero symbols
- If the **Operation mode** parameter is set to **Truncated** or **Terminated**, then 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, then a typical **Traceback depth** value is about five times the constraint length of the code.

Reset Port

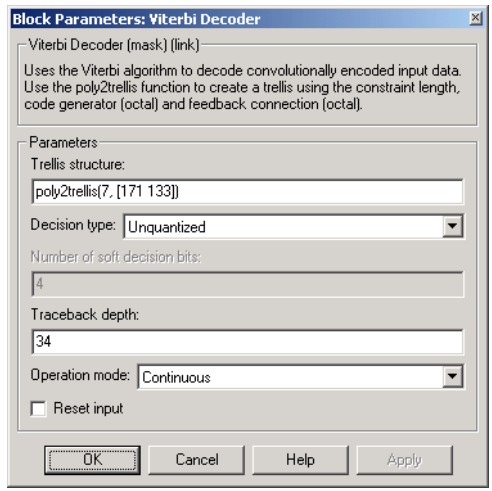
The reset port is usable only when the **Operation mode** parameter is set to **Continuous**. Checking the **Reset input** check box causes the block to have an

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 the **Reset** parameter in the Convolutional Encoder block to **On nonzero Rst input**.

Dialog Box



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.

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**.

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.

Reset input

When you check this box, the decoder has a second input port labeled `Rst`. Providing a nonzero input value to this port causes the internal memory to be set to its initial state prior to processing the input data.

See Also

Convolutional Encoder, 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] Heller, Jerrold A. and Irwin Mark Jacobs. "Viterbi Decoding for Satellite and Space Communication." *IEEE Transactions on Communication Technology*, vol. COM-19, October 1971. 835-848.

Voltage-Controlled Oscillator

Purpose

Implement a voltage-controlled oscillator

Library

Controlled Sources sublibrary of Comm Sources

Description

The Voltage-Controlled Oscillator (VCO) block generates a signal whose frequency shift from the **Oscillation 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

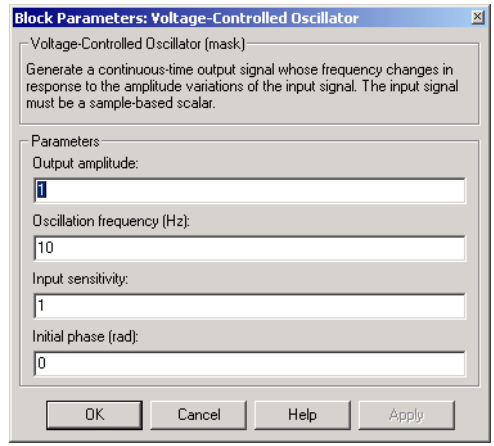
$$y(t) = A_c \cos(2\pi f_c t + 2\pi k_c \int_0^t u(\tau) d\tau + \varphi)$$

where A_c is the **Output amplitude** parameter, f_c is the **Oscillation 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



Output amplitude

The amplitude of the output.

Oscillation 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 **Oscillation frequency** value. The units of **Input sensitivity** are Hertz per volt.

Initial phase (rad)

The initial phase of the oscillator in radians.

See Also

Discrete-Time VCO

Walsh Code Generator

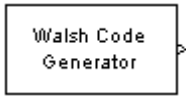
Purpose

Generate a Walsh code from an orthogonal set of codes

Library

Sequence Generators sublibrary of Comm Sources

Description



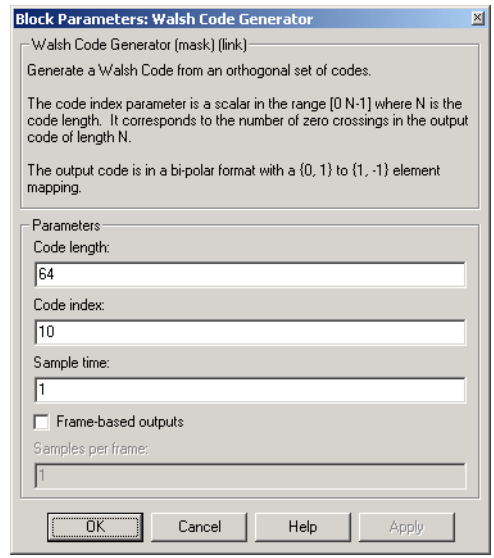
Walsh codes are defined as a set of N codes, denoted W_j, for $j = 0, 1, \ldots, N - 1$, which have the following properties:

- 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



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.

See also

Hadamard Code Generator, OVSF Code Generator

Windowed Integrator

Purpose

Integrate over a time window of fixed length

Library

Integrators, in Basic Comm Functions

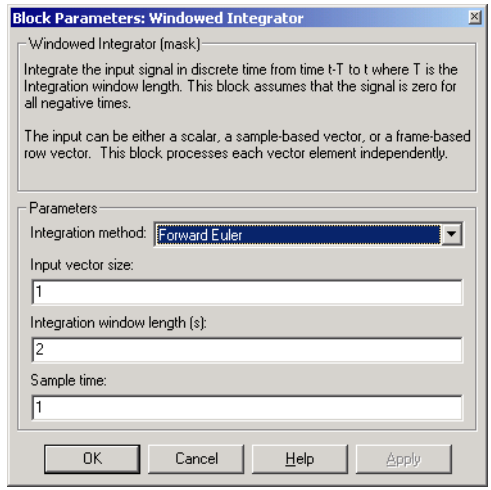
Description

The Windowed Integrator block integrates the input signal in discrete time, over a sliding time window of fixed length. If the **Integration window length** parameter is T, then the output at time t is the result of integrating the input signal from t-T to t. The block assumes that the input signal is zero for all negative t.

You can choose one of three integration methods: **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. The block processes each vector element independently. If the input signal is a vector, then the output is a vector of the same length. This length appears as the **Input vector size** parameter.

Dialog Box



Integration method

The integration method. Choices are **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

Input vector size

The length of the input vector.

Integration window length (s)

The length of the interval of integration, in seconds.

Sample time

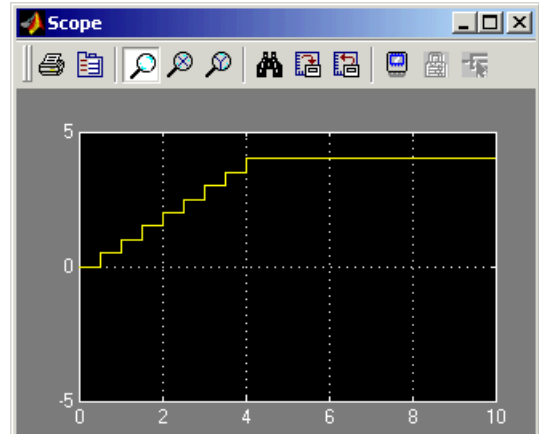
The integration sample time. This must not exceed the **Integration window length** parameter.

Examples

Integrate a scalar constant signal whose value is 1, for 10 seconds, using these parameters:

- **Integration method** = **Forward Euler**
- **Input vector size** = 1
- **Integration window length** = 4
- **Sample time** = .5

You can use a Simulink Constant block for the input signal. The Simulink Scope block shows the output below.



Notice that the output from time 0 to time 4 is a discrete approximation of a ramp. During this period, the interval of integration increases with time. Also notice that the output after time 4 is a constant value of 4. This is the result of integrating the value 1 over the full integration window length of 4 seconds.

See Also Discrete Modulo Integrator, Integrate and Dump, Discrete-Time Integrator (Simulink)

Index

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W