Communications Toolbox

For Use with MATLAB®

Computation

Visualization

Programming

The MATH WORKS Inc.

New Features Guide

Version 1.4 Release 11

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`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
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

Communications Toolbox New Features Guide

Printing History: January 1998 First printing New for Version 1.3
January 1999 Second printing Updated for Version 1.4 (Release 11)

Contents

1 New in the Communications Toolbox

2 Using New Simulink Blocks

3 Simulink Block Library Reference

A Corrections to User's Guide

1

New in the Communications Toolbox

Introduction

Version 1.4 (Release 11) is the newest release of the Communications Toolbox, a collection of MATLAB® functions and Simulink® blocks for research, development, system design, analysis, and simulation in the communications area.

Note: This release does not include new MATLAB functions.

Requirements

This release of the Communications Toolbox requires MATLAB Version 5.3. Simulink Version 3.0 and DSP Blockset Version 3.0 are required to use the Simulink blocks contained in the Communications Toolbox.

About This Document

This guide documents the functionality new to the Communications Toolbox, since Version 1.2. It also supplements the *Communications Toolbox User's Guide* for Version 1.2, which is available online via the Help Desk, and supersedes the *Communications Toolbox New Features Guide Version 1.3*.

This chapter provides an overview of new features and additions to the Communications Toolbox for Version 1.4. Chapter 2 provides examples of how you can use the new blocks. Chapter 3 describes all of the blocks added since Version 1.2. Appendix A contains corrections to the *Communications Toolbox User's Guide* for Version 1.2.

Feedback

Your input is very valuable to us. Expanded functionality and improvements to the Communications Toolbox are the result of your feedback. Please send your suggestions to: `suggest@mathworks.com`.

Accessing the Communications Toolbox Version 1.4

To access the Communications Toolbox from MATLAB, type the following command.

```
commlib
```

The Communications Toolbox Library 1.4 (main library) window appears.

Compatibility with Version 1.3

Some Communications Toolbox Version 1.4 blocks are not compatible with earlier versions. This results from a difference in how the Version 1.4 blocks handle complex signals. Simulink Version 3.0 and the Communications Toolbox Version 1.4 now support intrinsic complex data types. This means that Version 1.4 blocks handle complex inputs as single-width inputs, while earlier version blocks require double-width inputs for complex signals. Therefore, you cannot use blocks from Communications Toolbox Version 1.4 libraries that have complex inputs or outputs in models that you created with earlier versions of the Communications Toolbox.

You can continue to modify Version 1.3 models, since the Communications Toolbox Version 1.4 comes with both Version 1.4 and Version 1.3 block libraries. However, we recommend that you create new models using Version 1.4 block libraries.

The Version 1.4 blocks that now process complex input signals as one signal include: baseband analog and digital modulation blocks, and channel blocks.

To access the Version 1.3 block libraries, type the following command in the MATLAB window.

```
commlib 1
```

The Communications Toolbox Simulink Block Library (the Communications Toolbox Version 1.3) window appears.

What's New in Version 1.4

The Communications Toolbox Version 1.4 contains the following changes and new features:

- Reorganized and streamlined Simulink block libraries
- The following new Simulink blocks:
  - Convolutional Encoder
  - Viterbi Decoder
  - Data Mapper
  - Error Rate Calculation
- Support for the new Simulink complex data type
- Other Simulink block enhancements

Reorganized and Streamlined Block Libraries

The Communications Toolbox Version 1.4 reorganizes and streamlines many of the block libraries. This section describes the reorganization and streamlining of the block libraries as follows:

- Table 1-1 lists the changes to the organization of the libraries from Version 1.3 to Version 1.4.
- Figure 1-1 shows the block libraries and sublibraries for the Communications Toolbox Version 1.4.

To display the Communications Toolbox Simulink Block library tree structure in the MATLAB window, type

```
help commlib
```

Communications Toolbox Block Library Changes from Version 1.3 to 1.4

This section specifies the block library changes from Version 1.3 and earlier to Version 1.4.

Table 1-1: Changes to the Block Libraries from Version 1.3 to 1.4

Block Library in Version 1.3	Description of Change in Version 1.4
Source/Sink	Replaced by the following two separate libraries: Comm Sources library Comm Sinks library
Interleave and Scrambler	This library no longer exists. The Interleave and Scrambler blocks now reside in the Utility Functions library. The PN Sequence Generator block now resides in the Comm Sources library.
Error Control	This library has been renamed Channel Coding. Five of the Error Control sublibraries—Hamming, BCH, Linear, Cyclic, and Reed-Solomon—are now combined into the Block Coding sublibrary.
Multiple Access	This library no longer exists. You can create the functions formerly available in this library from the Utility Functions library.
Filters	This library no longer exists. You can create the functions formerly available in this library from the Utility Functions library and the DSP Blockset.
Modulation/ Demodulation	This library has been renamed Modulation. The modulation sublibrary Separate Versions for Digital Modem no longer exists. To access the separate components of a modulator or demodulator, click on a block, select **Edit** menu, then select **Look under mask**.

Communications Toolbox Version 1.4 Libraries and Sublibraries

The Communications Toolbox Version 1.4 Simulink Block Libraries and Sublibraries are now organized as shown in Figure 1-1 below. To access the main library, type the following command.

```
commlib
```

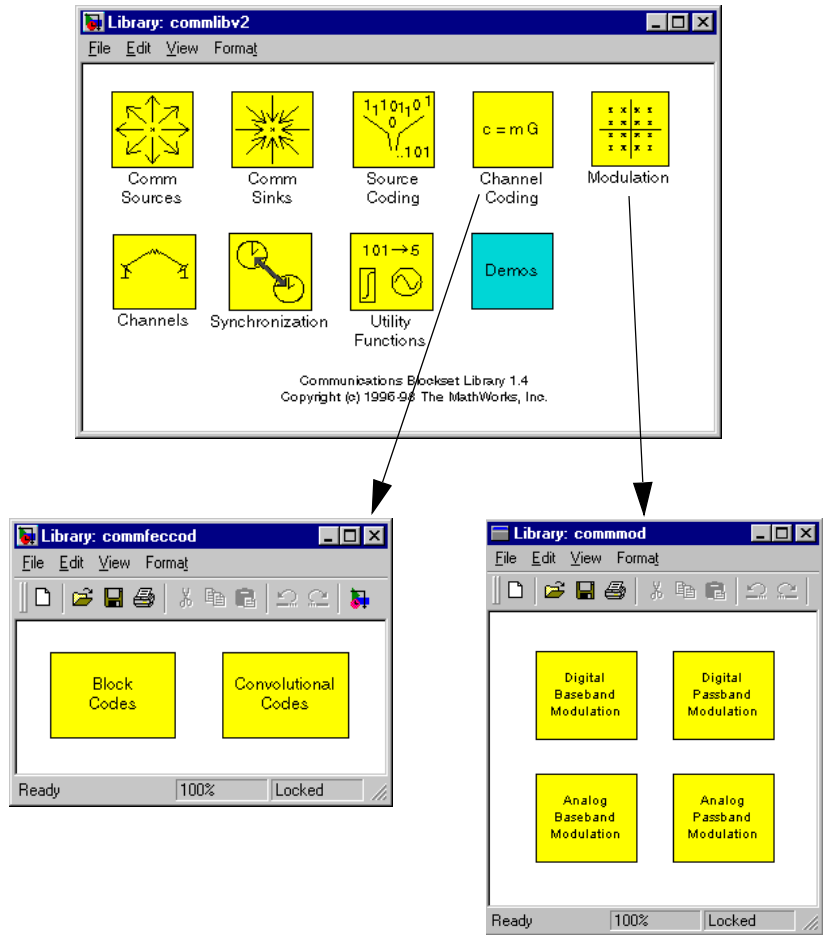
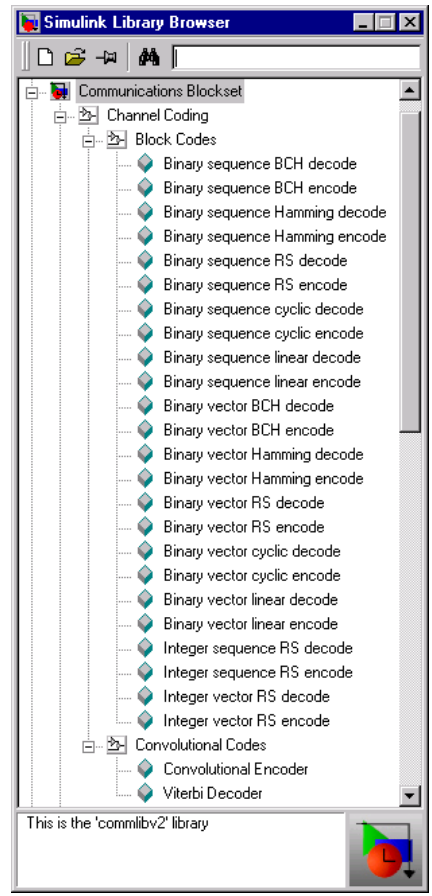


Figure 1-1: Communications Toolbox Version 1.4 Simulink Block Libraries

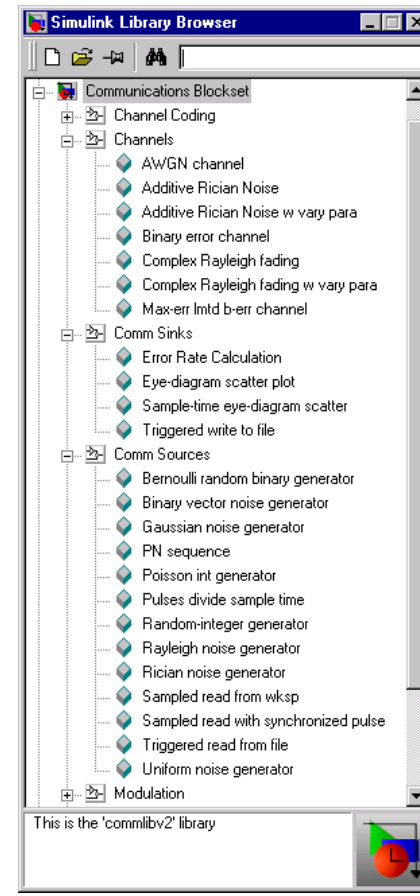
Contents of the Communications Toolbox Version 1.4 Block Libraries

The following figures display the contents of each of the Communications Toolbox sublibraries.

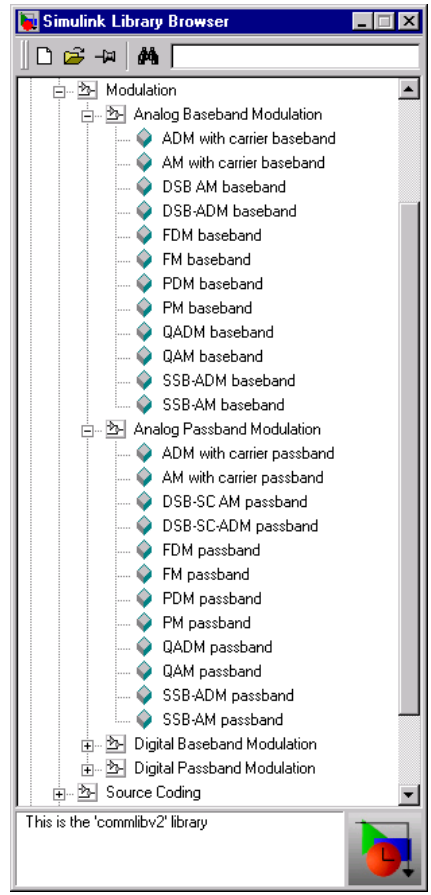
Channel Coding Library



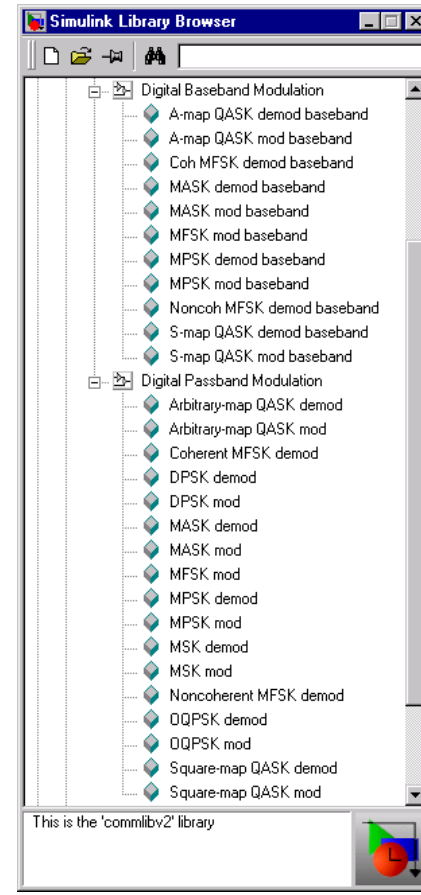
Channels, Sinks, and Sources Libraries



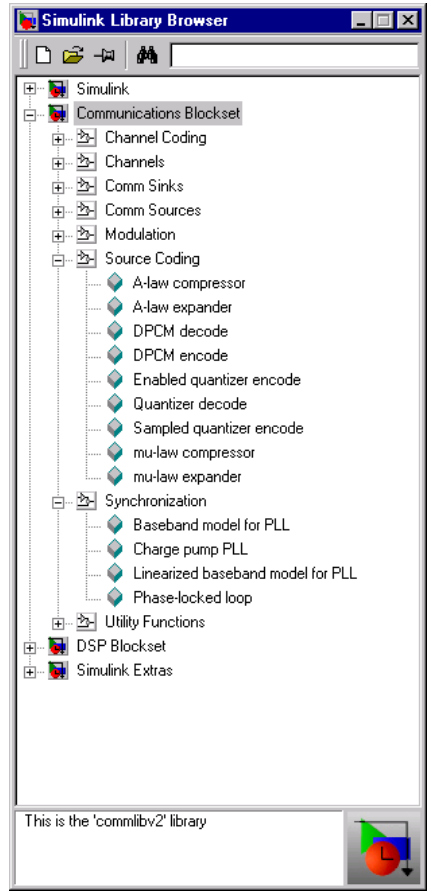
Analog Modulation Sublibraries



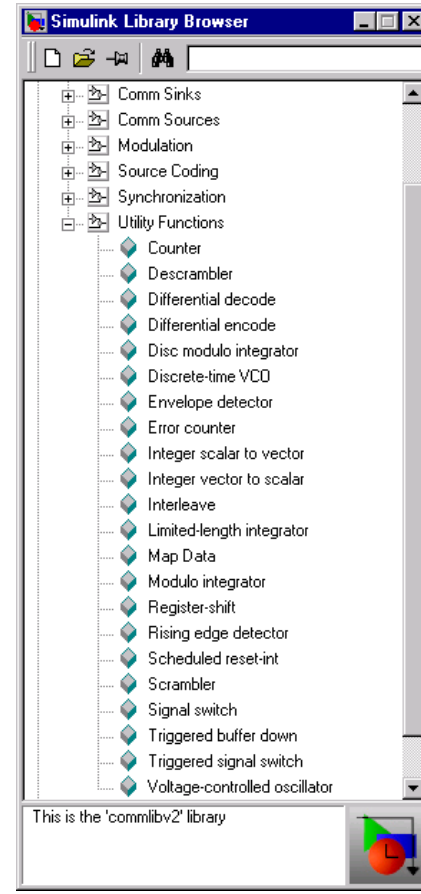
Digital Modulation Sublibraries



Source Coding and Synchronization Libraries



Utility Functions Library



New Blocks

The Communications Toolbox Version 1.4 contains the following new Simulink blocks:

- Convolutional Encoder
- Viterbi Decoder
- Data Mapper
- Error Rate Calculation

Chapter 2 contains examples of how to use the new blocks. Chapter 3 contains reference pages, in alphabetical order, for each of the blocks added to the Communications Toolbox since Version 1.2. See these reference pages for detailed descriptions of how to use the new Simulink blocks.

New Simulink Complex Data Type

Simulink Version 3.0 now supports complex signals as an intrinsic data type. All of the Communications Toolbox Version 1.4 Simulink blocks that accept complex signals as inputs or produce complex signals as outputs use this new data type. See “Compatibility with Version 1.3” on page 1-3 for information about compatibility between Communications Toolbox Version 1.4 blocks and earlier versions.

Enhancements to Existing Blocks

Some existing Simulink blocks in the Communications Toolbox have been enhanced or modified in Version 1.4 as follows:

- You can now use the AWGN Channel block with either real or complex inputs. You can also supply the value of the noise variance either as a block parameter or as an input to the block. With these two enhancements, the functionality of the AWGN Channel block now encompasses that of the following four Version 1.3 blocks:
  - The AWGN Channel block
  - The AWGN w/ Varying Parameters block
  - The Rayleigh Noise CE Channel block
  - The Rayleigh Noise CE Channel w/ Varying Variance block

- You also now have the option to specify the noise variance in the parameter mask for the AWGN Channel block in terms of the signal to noise ratio.
- You can choose to reverse the order of the vector elements in the Integer Scalar to Vector and Integer Vector to Scalar blocks.
- You can now set the modulation constant in the parameter mask for the following eight phase and frequency modulation blocks:
  - FDM Baseband
  - FDM Passband
  - FM Baseband
  - FM Passband
  - PDM Baseband
  - PDM Passband
  - PM Baseband
  - PM Passband

  In the FM (Frequency Modulator) and FDM (Frequency Demodulator) blocks, the modulation constant is specified in units of Hertz per volt. In the PM (Phase Modulator) and PDM (Phase Demodulator) blocks, the modulation constant is specified in units of radians per volt.

  Both of the PDM blocks, Baseband and Passband, are implemented with a phase locked loop containing a voltage controlled oscillator (VCO). For these two blocks you can also specify a value for the VCO gain in the parameter mask, in units of Hertz per volt.

2

Using New Simulink Blocks

Introduction

This chapter presents selected examples of the use of new Simulink blocks added to the Communications Toolbox Version 1.4. Two examples illustrate the use of the four new blocks as well as the modified AWGN Channel block.

The first example is a simulation of a communications link using 8-PSK modulation with Gray coding. It includes the modified AWGN Channel block and the new Data Mapper and Error Rate Calculation blocks.

The second example is a simulation of a coded communications link using a punctured convolutional code with Viterbi decoding. It includes the modified AWGN Channel block and the new Convolutional Encoder, Viterbi Decoder, and Error Rate Calculation blocks.

Gray Coded 8-PSK Modulation

Gray coding is a technique often used in multilevel modulation schemes to minimize the bit error rate by ordering modulation symbols so that the binary representations of adjacent symbols differ by only one bit. This example shows how to use the new Data Mapper block to achieve this ordering.

The model shown below simulates the operation of a Gray coded 8-PSK modulation system. To load this model from MATLAB, type

```
tstgraycod
```

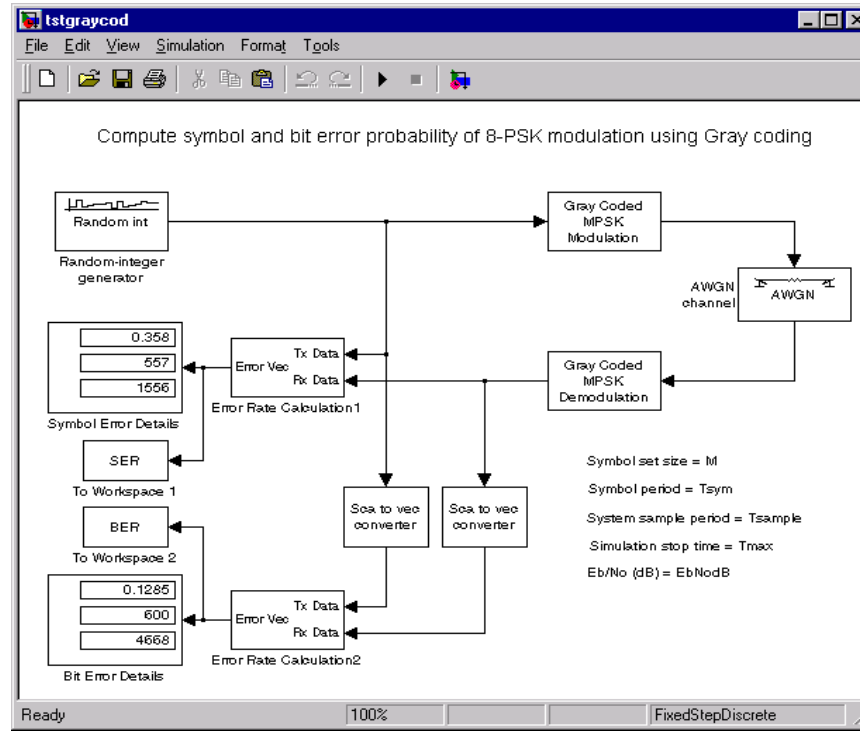


Figure 2-1: Simulink Model of a Gray Coded 8-PSK Modulation System

Data flows through this model in the following sequence:

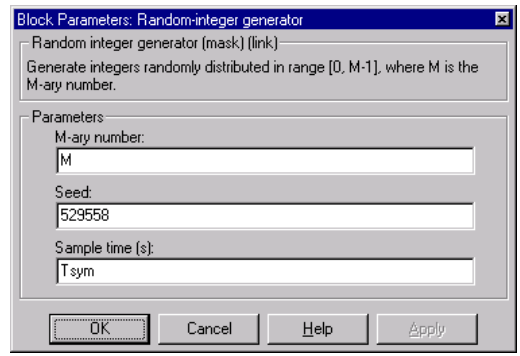
1. The Random-Integer Generator block serves as the source, producing a sequence of integers.
2. The Gray Coded MPSK Modulator subsystem modulates the data in complex envelope format.
3. The AWGN Channel block adds white Gaussian noise to the modulated data.
4. The corrupted data is demodulated in the Gray Coded MPSK Demodulator subsystem.
5. The demodulated integer data is compared to the original source data in the Error Rate Calculation1 block, yielding symbol error statistics.
6. Integer Scalar to Vector blocks convert the original and demodulated data to binary vectors.
7. The Error Rate Calculation2 block produces bit error statistics from the two binary vectors.

The values of a number of variables that are used in multiple blocks and subsystems are set in the workspace when the model is loaded. To clarify the discussion of the individual blocks and subsystems, their values are listed below:

- `M`, symbol set size (8)
- `Tsym`, symbol period (0.2 sec)
- `Tsample`, sample period (0.01 sec)
- `Tmax`, simulation stop time (10000 sec)
- `EbNodB`, the ratio of energy per bit to noise power spectral density, (E_b/N_0) , in decibels (0 dB)

Random Source Generation

The Random Integer Generator block produces random data that is used as the information in this simulation. Double-click on this block to open the parameter mask window shown below.

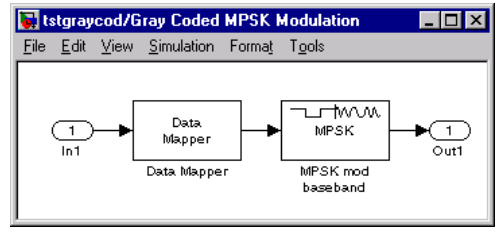


This block generates one integer symbol in the range 0 to `M-1` every `Tsym` seconds.

Gray Coded MPSK Modulation

The Gray Coded MPSK Modulation block, shown below, is a masked subsystem consisting of two Communications Toolbox library blocks in series:

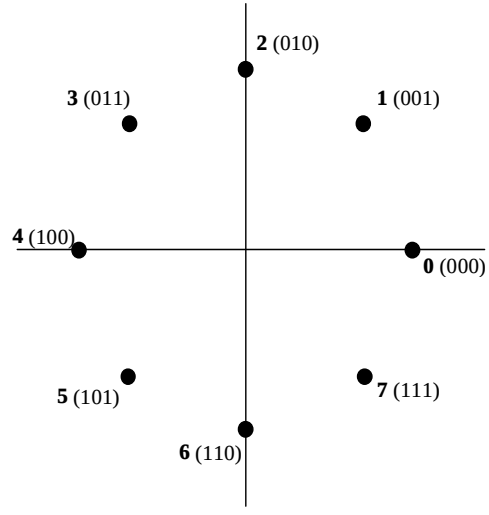
- The Data Mapper block
- The MPSK Mod Baseband block



Once you set the parameter M in its mask, the MPSK Mod Baseband block:

- Accepts integer inputs in the range 0 to M-1
- Generates unit-magnitude complex phasor outputs with evenly spaced phases in the range 0 to $2\pi(M-1)/M$

The mapping of input integers to output phases in this library block is as follows: 0 to 0, 1 to $2\pi/M$, 2 to $4\pi M$, ..., M-1 to $2\pi(M-1)/M$. The mapping from integers (and their binary equivalents) to phases is shown below for $M = 8$.



This mapping from integers to phases does not reflect a Gray code ordering. To map the input symbols onto the M phasors using a Gray code ordering, the MPSK Mod Baseband block must be preceded by a Data Mapper block.

The Data Mapper block accepts integer inputs and produces integer outputs. The mapping of inputs to outputs follows one of four mapping modes:

- **Binary to Gray**
- **Gray to Binary**

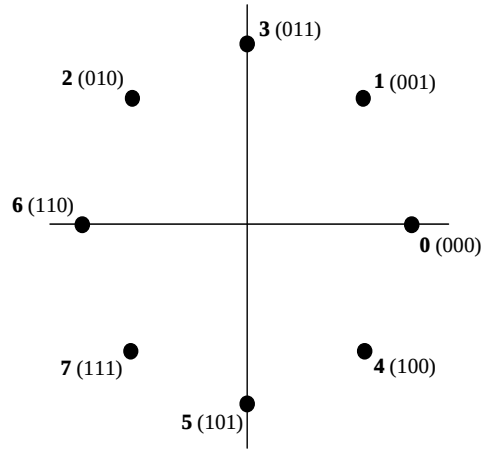
- **User Defined**
- **Straight Through**

You can select these modes from the parameter mask for the Data Mapper block. To achieve a Gray code ordering of integer inputs to complex phasor outputs in this subsystem, select the **Gray to Binary** option. This converts the desired Gray code ordering at the input to the subsystem to the binary order used by the MPSK Mod Baseband block.

Table 2-1: Gray Coded MPSK Modulation Subsystem I/O Mapping

Data Mapper Input	Data Mapper Output	MPSK Mod Output
0	0	e^{0}
1	1	$e^{j\pi/4}$
2	3	$e^{j3\pi/4}$
3	2	$e^{j\pi/2}$
4	7	$e^{j7\pi/4}$
5	6	$e^{j3\pi/2}$
6	4	$e^{j\pi}$
7	5	$e^{j5\pi/4}$

The overall effect of this subsystem is a Gray code mapping as shown in the diagram below.

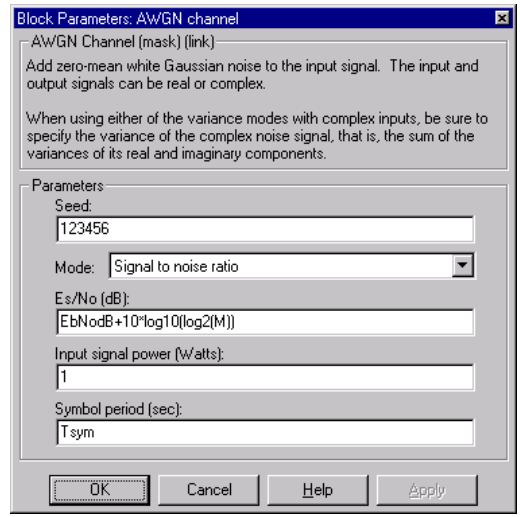


Signal Transmission

The AWGN Channel block is used to simulate transmission over a noisy channel. The parameter mask for this block has three modes: **Signal to noise ratio**, **Variance from mask**, and **Variance from port**. Selecting the **Signal to noise ratio** mode requires the entry of the following quantities to determine the variance:

- E_s/N_0 **,** the ratio of energy per symbol to noise power spectral density
- The input signal power
- The symbol period

For information on the variance modes, see the reference page for AWGN Channel on page 3-3.



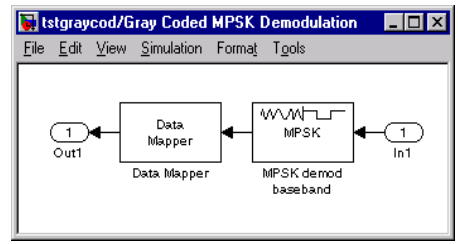
The values for these parameters are chosen as follows:

- The E_s/N_0 parameter is computed from the workspace variables `EbNodB` and `M`. The conversion from bit energy to symbol energy reflects the fact that each symbol carries `log2(M)` bits of information.
- The signal power is set to 1 watt because the M-PSK baseband modulation block produces unit power signals.
- The symbol period of the channel is set to `Tsym`.

Gray Coded MPSK Demodulation

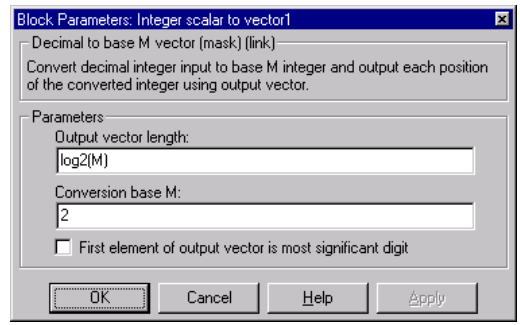
To construct the demodulation subsystem to correctly mirror the Gray coded modulation process:

- Follow the MPSK Demod Baseband block by a Data Mapper block as shown below.
- Set the mapping mode for the Data Mapper block to **Binary to Gray** in the parameter mask.



Symbol to Bit Conversion

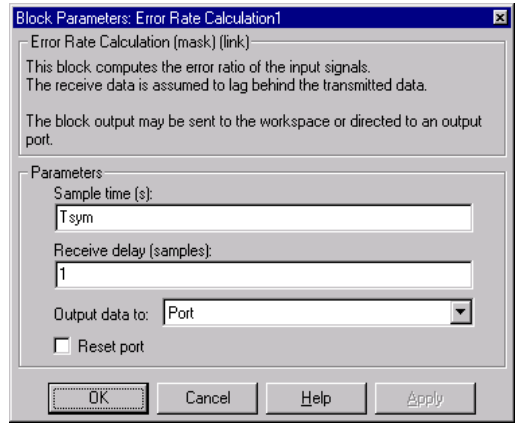
The Integer Scalar to Vector block converts integer symbols to their binary equivalent. The two parameters are the output vector length and the conversion base. The ordering of the output vector can be selected via a check box.



The Simulink block diagram for this example contains two converter blocks, one for the demodulated symbols and the other for the original source symbols. The symbols are converted to their binary equivalents to measure the bit error rate of the system.

Error Rate Calculation

The Error Rate Calculation block compares demodulated symbols to original source symbols to compute the error rate. Two Error Rate Calculation blocks are used in this model, one to compute the symbol error rate and the other to compute the bit error rate. The parameter mask for the Error Rate Calculation block is shown below.



The parameters required for both symbol and bit error calculations are identical. Because the symbols are decoded into vectors of bits, the **Sample time** for the symbol error calculation and bit error calculation are the same. In both cases, the demodulation of the data introduces a single sample delay which is reflected in the **Receive delay** field.

Theoretical Performance

The theoretical symbol error probability of MPSK is given by

$$P_E(M) = \operatorname{erfc}\left(\sqrt{\frac{E_s}{N_0}}\sin\left(\frac{\pi}{M}\right)\right)$$

where erfc is the complementary error function, E_s/N_0 is the ratio of energy in a symbol to noise power spectral density, and M is the number of symbols.

To determine the bit error probability, the symbol error probability, P_E, needs to be converted to its bit error equivalent. There is no general formula for the symbol to bit error conversion. Upper and lower limits are nevertheless easy to establish. The actual bit error probability, P_b, can be shown to be bounded by

$$\frac{P_E(M)}{\log_2 M} \le P_b \le \frac{M/2}{M-1} P_E(M)$$

The lower limit corresponds to the case where the symbols have undergone Gray coding. The upper limit corresponds to the case where pure binary coding is used.

Simulation Results

To test the Gray code modulation scheme in this model, simulate the `testmodmap` model for a range of E_b/N_0 values. Because increasing the value of E_b/N_0 lowers the number of errors produced, the length of each simulation must be increased to ensure that the statistics of the errors remain stable.

Using the `sim` command to run a Simulink simulation from MATLAB, the following code generates the data for symbol error rate and bit error rate curves for E_b/N_0 values in the range 0 dB to 12 dB in steps of 2 dB.

```
M = 8;
Tsym = 0.2;
Tsample = 0.01;
BERVec = [];
SERVec = [];
EbNoVec = [0:2:12];
TVec = [1000 1000 1000 15000 20000 100000 100000]*Tsym;
for n=1:length(EbNoVec);
   Tmax = TVec(n);
   EbNodB = EbNoVec(n);
   sim('testmodmap');
   SERVec(n,:) = SER;
   BERVec(n,:) = BER;
end;
```

After simulating for the full set of E_b/N_0 values, you can plot the theoretical and simulated results. These plots are shown in Figure 2-2.

Figure 2-2 also shows results for an 8-PSK modulation system *without* Gray coding. To modify the model to generate this data, you must either:

- Replace the modulation and demodulation subsystems with the MPSK modulator and demodulator blocks.
- Change the mode for each of the Data Mapper blocks in the modulation and demodulation subsystems to **Straight Through.**

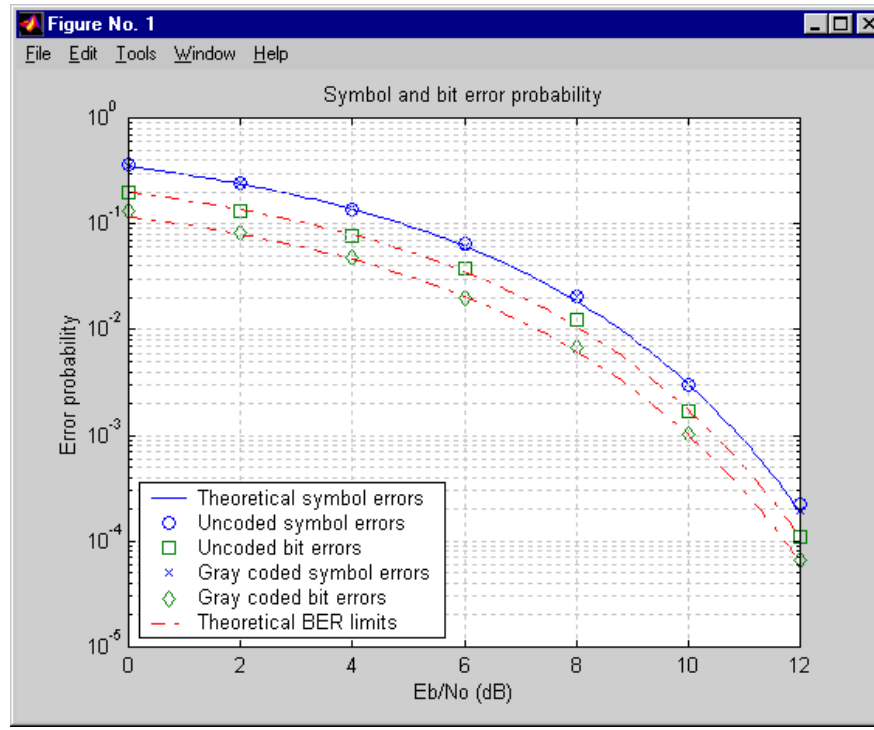


Figure 2-2: Symbol and Bit Error Rates for 8-PSK

The simulation results agree well with the theoretical bounds for the symbol and bit error probabilities. Since the Gray coding only affects the mapping of symbol errors to bit errors, the symbol error probability is the same in both cases.

Punctured Convolutional Coding

The complexity of a Viterbi decoder increases rapidly with the code rate. Puncturing is a technique that allows the encoding and decoding of higher rate codes using standard rate 1/2 encoders and decoders. This example demonstrates how to use the new Convolutional Encoder and Viterbi Decoder blocks in the simulation of a punctured coding system.

The Simulink block diagram constructed for this example, shown below, contains six blocks from the libraries of the Communications Toolbox:

- Bernoulli Random Binary Generator
- Convolutional Encoder
- MPSK Mod Baseband
- AWGN Channel
- Viterbi Decoder
- Error Rate Calculation

To open this diagram from MATLAB, type

```
tstconvcod
```

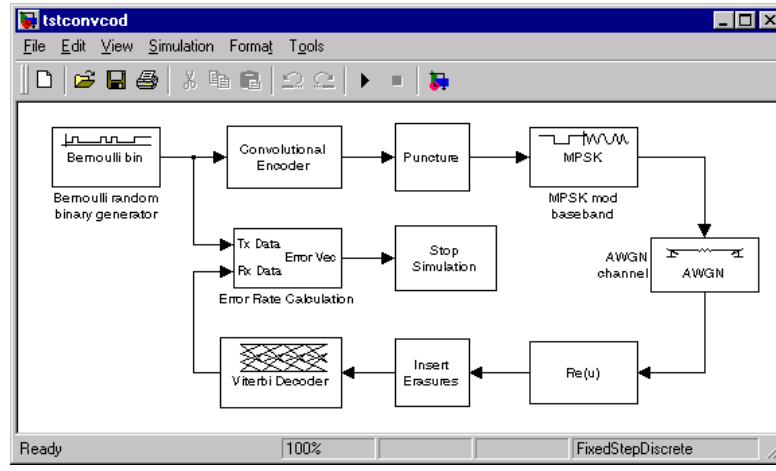


Figure 2-3: The Punctured Convolutional Coding Simulink Demo

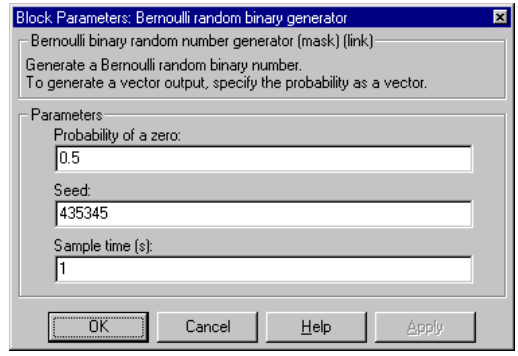
Three additional subsystems have been constructed for the purpose of this example:

- The Puncture block periodically removes bits from the encoded bit stream, thereby increasing the code rate.
- The Insert Erasures block restores the encoded bit stream to the original lower rate, allowing the use of a simpler decoder.
- The Stop Simulation block ends the simulation once a designated number of errors are observed or when a preset number of bits are processed.

Note: These three subsystems are presented as examples and are not part of any library in the Communications Toolbox.

Random Source Generation

From the Comm Sources library, the Bernoulli Random Binary Generator block produces the information source for this simulation. Double-click on this block to open the parameter mask window shown below.



One bit is generated by this block at each sample time. The bits are produced randomly, in an equiprobable fashion. The sample time is arbitrarily set to 1 second.

Convolutional Encoding

The rate 1/2 convolutional code used in this example is the industry standard constraint length 7 code defined by the encoder diagram below.

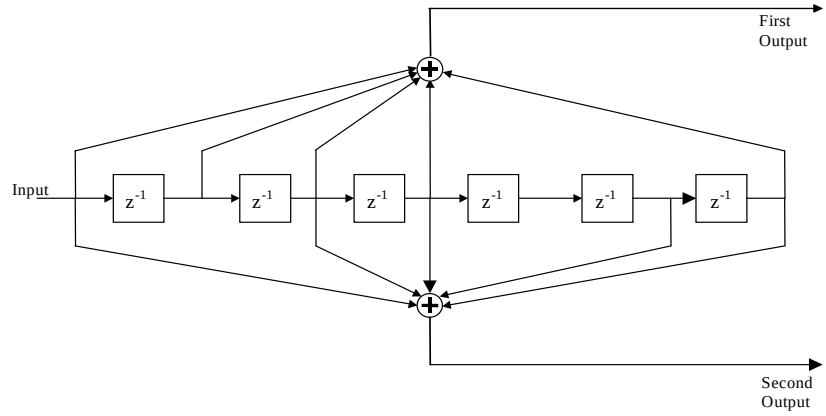
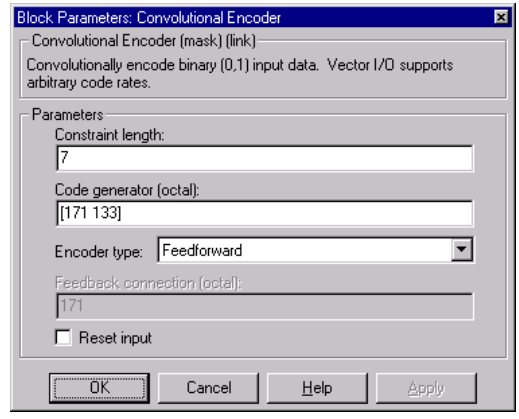


Figure 2-4: Convolutional Encoder Schematic Block Diagram

Typically, this encoder structure is specified by a pair of octal numbers indicating the connections from the delay cells to the modulo-2 summing nodes. You can set both the constraint length and the encoder structure in the mask for the Convolutional Encoder block shown below.

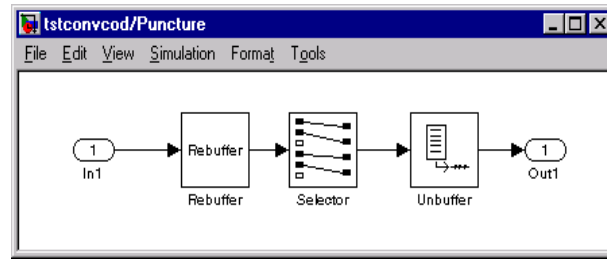


In this case, the octal pair `[171 133]` is entered in the **Code generator** field of the Convolutional Encoder parameter mask. This octal pair

represents the shift register connections for the constraint length 7 code shown in Figure 2-4.

Puncturing

Puncturing is accomplished using blocks from the DSP Blockset and Simulink libraries. As shown in the accompanying figure for the Puncture subsystem, a selector block is used to periodically remove designated bits.



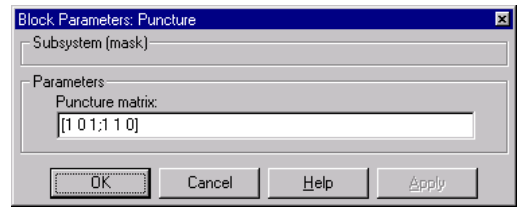
The puncture pattern is specified using standard matrix notation. Each column indicates a pair of output bits from the encoder:

- Ones indicate the bits that are transmitted.
- Zeros indicate the bits to be punctured.

For example, the optimal puncture matrix for creating a rate 3/4 code from the rate 1/2, constraint length 7 code is

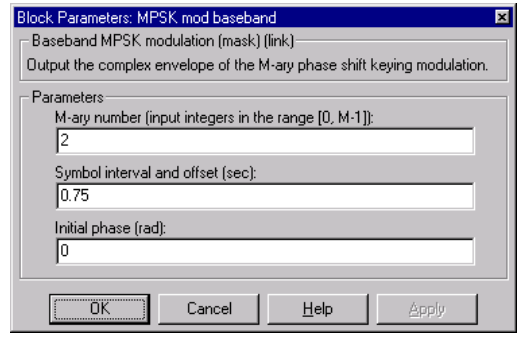
$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

This matrix is the only parameter necessary for the Puncture subsystem. You can enter it into the mask as shown below.



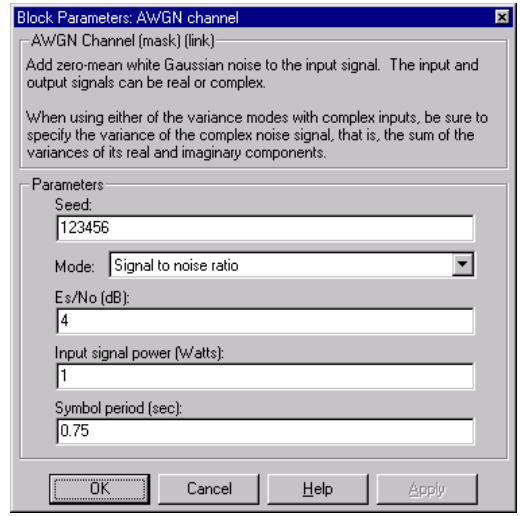
Modulation

Binary Phase Shift Keying (BPSK) modulation is simulated using the MPSK Mod Baseband block, with the **M-ary number** parameter value set to 2. The symbol interval is reduced to 0.75 seconds to match the rate 3/4 encoding.



Signal Transmission

The AWGN Channel block is used to simulate transmission over a noisy channel. The parameter mask for this block is set for the **Signal to noise ratio** mode.



In this mask:

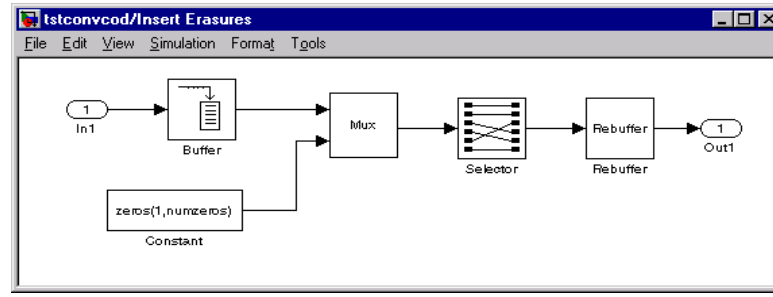
- The **Es/No** parameter is set to 4 dB in the mask parameters window. This value typically is changed from one simulation run to the next.
- The preceding modulation block generates unit power signals, so the **Input signal power** is set to 1 watt.
- The **Symbol period** is set to 0.75 seconds to match the symbol period of the modulator.

Demodulation

In this simulation, the Viterbi Decoder block is set to accept unquantized inputs. The MPSK Demodulator block produces hard decisions, so it cannot be used for demodulation in this model. Instead, the simulation passes the channel output through a Simulink Complex to Real-Imag block that extracts the real part of the complex samples.

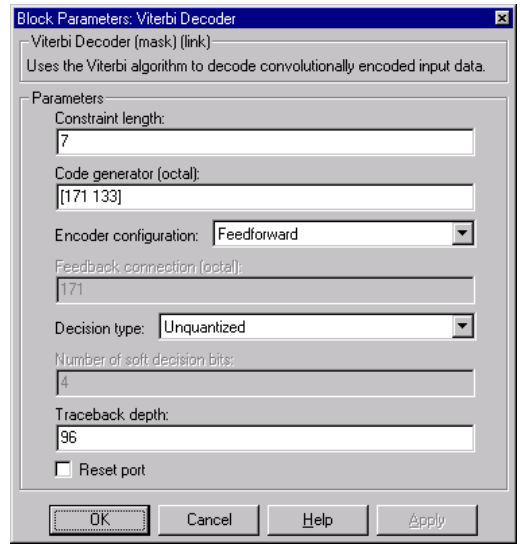
Erasure Insertion

The Insert Erasures block performs the inverse operation of the Puncture block. Because the punctured bits are not transmitted, there is no information to indicate their values. Therefore, since BPSK is an antipodal modulation format, zeros must be inserted into the punctured bit positions. Similarly to the Puncture block, this subsystem uses blocks from the DSP Blockset and Simulink libraries as shown below. The Selector block is used to insert zeros into the correct positions.



Viterbi Decoding

The Viterbi Decoder block is configured to decode the same rate 1/2 code specified in the Convolutional Encoder block.

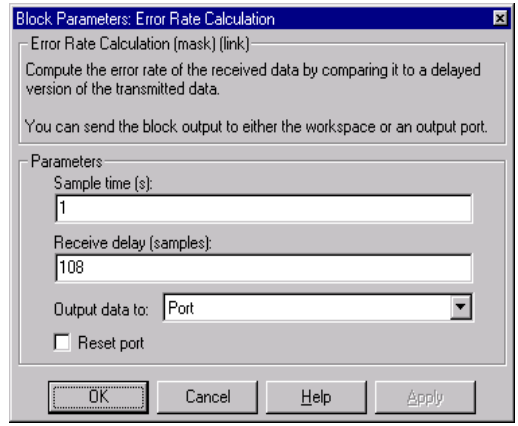


For this example, the decision type is set to **Unquantized**. You would normally set the **Traceback depth** for this code to something close to 40. However, for decoding punctured codes, a higher value is required to give the decoder time to resolve the ambiguities introduced by the inserted erasures.

Error Rate Calculation

The decoded bits are compared to the original source bits in the Error Rate Calculation block. In the mask for this block, shown below, the decoder sample time is set to 1 second. The puncturing and erasure insertion operations both introduce delay due to the use of buffers. Each of these two blocks incurs 6 seconds of delay: 3 seconds each from the

input and output buffers. Adding these delays to the decoder delay of 96 seconds, produces a total **Receive delay** of 108 seconds.



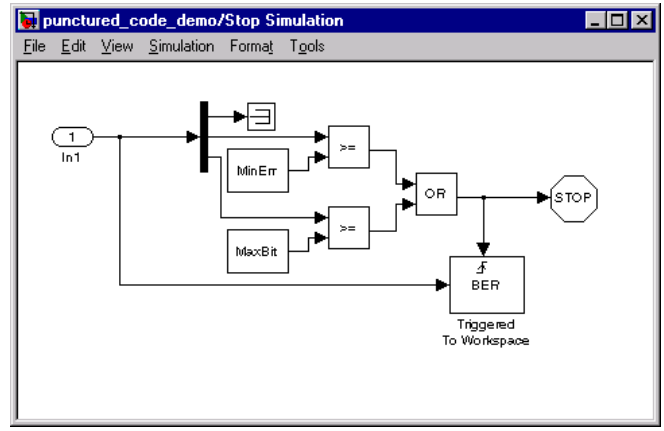
Stopping the Simulation

The output of the Error Rate Calculation block is a three-element vector containing the calculated bit error rate (BER), the number of errors observed, and the number of bits processed. BER simulations are typically set to run until a minimum number of errors have been observed, or until a maximum number of bits have been processed.

You can use the Stop Simulation block to set these limits and to write the final BER data to the MATLAB workspace. To see how the Stop Simulation block works:

- Select the Stop Simulation block.
- Select **Look Under Mask** in the **Edit** menu.

The following diagram will appear.



Evaluating Results

Generating a bit error rate (BER) curve requires multiple simulations. You can perform multiple simulations from the command line using the `sim` command. To do this:

- Change the value of the **Es/No** parameter in the AWGN Channel block mask from a constant to the variable `EsNodB`.
- Run the following code to generate the data for plotting the BER curve.

```
CodeRate = 0.75;
EbNoVec = [2:.2:10];
EsNoVec = EbNoVec + 10*log10(CodeRate);
BERVec = zeros(length(EsNoVec),3);
for n=1:length(EsNoVec),
   EsNodB = EsNoVec(n);
   sim('tstconvcod');
   BERVec(n,:) = BER;
end
```

To confirm the validity of the results, we compare them to an established performance bound. The bit error rate performance of a rate $r = (n-1)/n$ punctured code is upper bounded by the expression

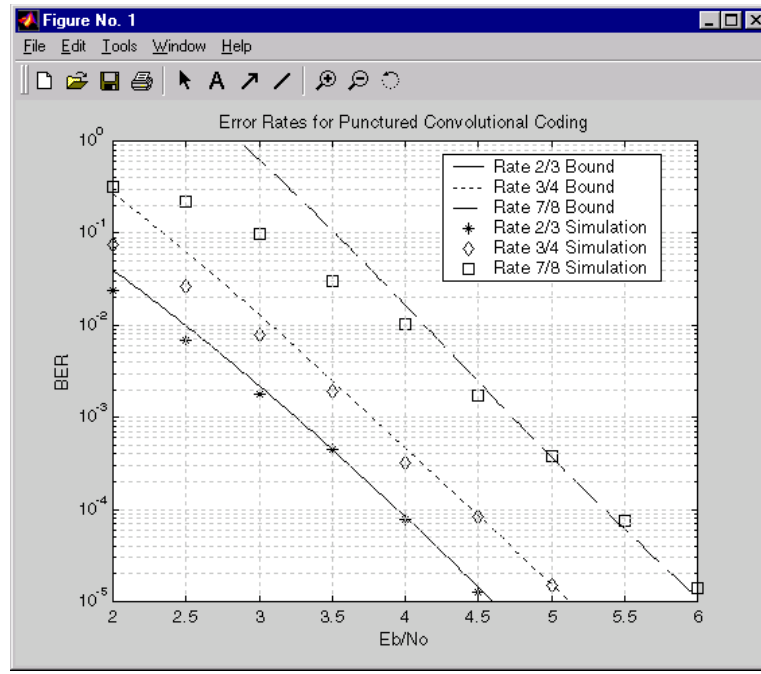
$$P_b \le \frac{1}{2(n-1)} \sum_{d = d_{free}}^{\infty} w_d \operatorname{erfc}(\sqrt{rd(E_b/N_0)})$$

In this expression, erfc denotes the complementary error function, r is the code rate, and both d_{free} and w_d are dependent on the particular code. For the rate 3/4 code of this example, $d_{free} = 5$, $w_5 = 42$, $w_6 = 201$, $w_7 = 1492$, and so on. See reference [1] for more details.

The following commands compute an approximation to this bound in MATLAB using the first seven terms of the summation:

```
dist = [5:11];
nerr = [42 201 1492 10469 62935 379644 2253373];
CodeRate = 3/4;
EbNo_dB = [2:.02:10];
EbNo = 10.0.^(EbNo_dB/10);
arg = sqrt(CodeRate*ebno'*dist);
bound = nerr*(1/6)*erfc(arg)';
```

The figure below shows simulation results and bounds for the rate 3/4 punctured code in this example, as well as other punctured codes of rates 2/3 and 7/8 derived from the same original constraint length 7 rate 1/2 code. The puncture patterns for these other rates are listed in reference [1]. The simulations used to generate the data for this plot were set to stop after 1000 errors or 40 million bits, whichever came first.



In each case the results agree well with the theoretical bounds. In some cases, at the lower bit error rates, the simulation results appear to indicate error rates slightly above the bound. This is not a result of simulation variance, since over 500 bit errors were observed at even the lowest bit error rate value. Rather, this is a result of the finite traceback depth in the decoder.

Bibliography

[1] Yasuda, Y., K. Kashiki, and Y. Hirata, "High Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding," *IEEE Transactions on Communications*, Vol. COM-32, pp. 315-319, March 1984.

3

Simulink Block Library Reference

Overview

This chapter describes the new function blocks for the Communications Toolbox added since Version 1.2.

New Function Blocks in Version 1.4

The following table lists the function blocks new to the Communications Toolbox in Version 1.4.

Block Name	Purpose
Convolutional Encoder	Convolutionally encode binary input data.
Data Mapper	Map integer data from one scheme to another.
Error Rate Calculation	Compute the bit error rate or symbol error rate of input data.
Viterbi Decoder	Decode convolutionally encoded data using the Viterbi algorithm.

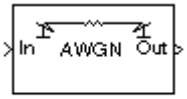
Purpose

Add zero-mean white Gaussian noise to the input signal.

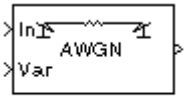
Library

Channels

Description



AWGN
channel



AWGN
channel

You can use the AWGN Channel block with either real or complex input signals. When the input signal is real, this block generates real Gaussian noise and produces a real output signal. When the input signal is complex, this block generates complex Gaussian noise and produces a complex output signal.

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

- **Signal to noise ratio**
- **Variance from mask**
- **Variance from port**

In the **Signal to noise ratio** mode, the variance is calculated from the following quantities you specify in the parameter mask:

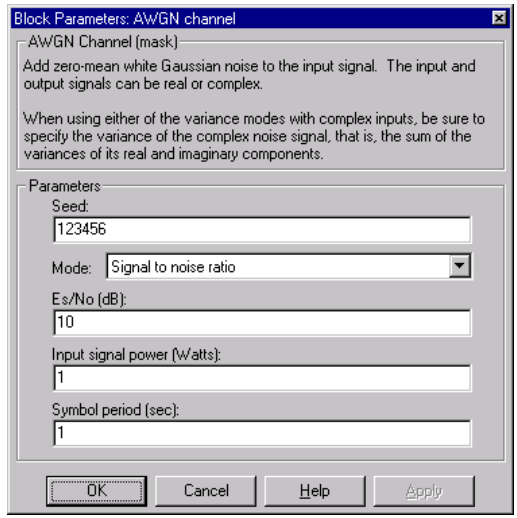
- E_s/N_0 **,** the ratio of energy per symbol to noise power spectral density
- The input signal power
- The symbol period

In the **Variance from mask** mode, you directly specify a value for the variance in the parameter mask.

In the **Variance from port** mode, you provide the variance as an input to the block.

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

Dialog Box and Parameters



Seed

The initial seed value for the random number generator.

Mode

The mode used to specify the noise variance: **Signal to noise ratio, Variance from mask** or **Variance from port**.

Es/No (**Signal to noise ratio** mode only)

The ratio of energy per symbol to noise power spectral density.

Input signal power (**Signal to noise ratio** mode only)

The average power of the input signal, in watts.

Symbol period (**Signal to noise ratio** mode only)

The duration of a channel symbol, in seconds.

Variance (**Variance from mask** mode only) Not shown

The variance of the Gaussian noise.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete, Inherited
Scalar Expansion	N/A
Vectorized	No
Complex	Yes

Block Interleave

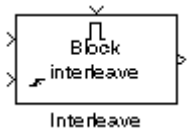
Purpose

Compute block interleaving or block deinterleaving.

Library

Utility Functions

Description



Block interleaving is accomplished in two steps.

- The input sequence is written row by row into an M-row by N-column (M-by-N) array.
- The output sequence is read column by column from this M-by-N array.

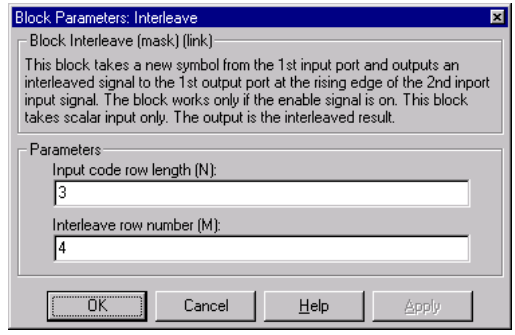
Deinterleaving switches the directions of the operations: the sequence is written into the array column by column and read out row by row.

Block interleaving is implemented using the Register Shift, Triggered Buffer Down, and Triggered Signal Switch (vector re-arrangement) utility blocks.

Interleaving is often used with error-control coding. Typically, you select the interleave parameters so that the number of columns, N, is greater than the expected burst lengths. The choice of the number of rows depends on the type of error-control coding scheme used. For block coding, the number of rows should be greater than the codeword length; thus, a burst of length N can cause at most a single error in any block codeword.

The timing for starting to fill the frame is very important. A block deinterleave should be synchronized with the block interleave in order to be able to recover the interleaved symbols. The delay is M*N symbols for each block (interleaver and deinterleaver).

Dialog Box and Parameters



Input code row length (N)

The number of columns in the array.

Interleave row number (M)

The number of rows in the array.

Note: To use Block Interleave for deinterleaving, reverse the N and M parameter settings.

Characteristics

Direct Feedthrough	Yes
Sample Time	Triggered action. Inherits the sample time from the blocks that input to this block.
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

BPSK Demod

Purpose

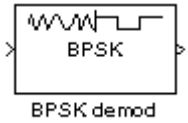
Demodulate BPSK modulated signal.

Library

Version 1.3 Passband Digital Modulation/Demodulation

Description

The BPSK (binary phase shift keying) Demodulation block demodulates the BPSK Mod signal. This block uses the Corr BPSK Demod block.

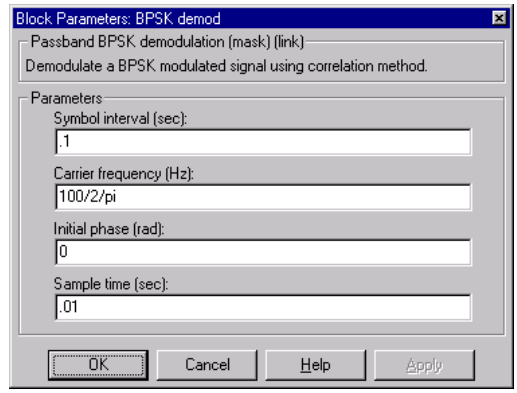


BPSK Demod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

This block receives a modulated analog signal as input. It outputs a demodulated binary signal.

Dialog Box and Parameters



Symbol interval, Carrier frequency, Initial phase

Match these parameters to the ones used in the corresponding BPSK Mod block. The offset value in **Symbol interval** can be different.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	No
Complex	No

Pair Block

BPSK Mod

BPSK Map

Purpose

Map a binary signal to phase shift for phase modulation.

Library

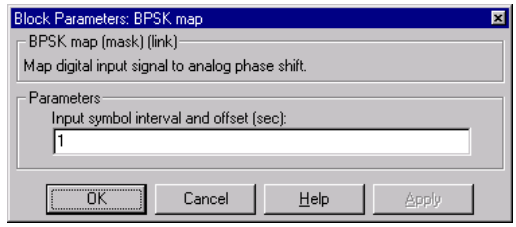
Version 1.3 Passband Digital Modulation/Demodulation

Description

BPSK map

The BPSK (binary phase shift keying) Map block maps the digital input signal to an analog signal for Phase Modulation (PM). If the input signal is 0, the phase shift is 0. If the input signal is 1, the phase shift is π. This block is a special case of the m-ary phase shift keying (MPSK) block in which M equals 2.

Dialog Box and Parameters



Input symbol interval and offset

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	No
Complex	No

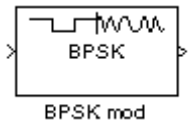
Purpose

Modulate the input signal using binary phase shift keying method.

Library

Version 1.3 Passband Digital Modulation/Demodulation

Description



BPSK mod

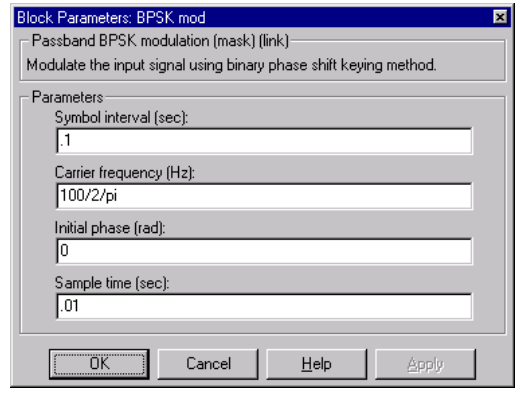
The BPSK (binary phase shift keying) Modulation block modulates the input signal. This block uses the BPSK Map block before feeding the signal to the Digital Phase Modulation (PM) block. Refer to their individual reference pages for descriptions of the techniques involved.

BPSK Mod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a binary signal. The output is a modulated analog signal with a maximum amplitude equal to 1.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Carrier frequency

The frequency of the carrier signal.

Initial Phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	No
Complex	No

Pair Block

BPSK Demod

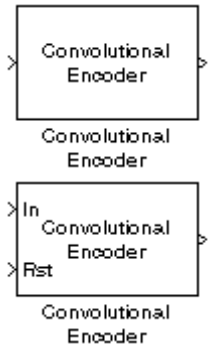
Purpose

Convolutionally encode binary (0,1) input data.

Library

Convolutional Codes, in Channel Coding

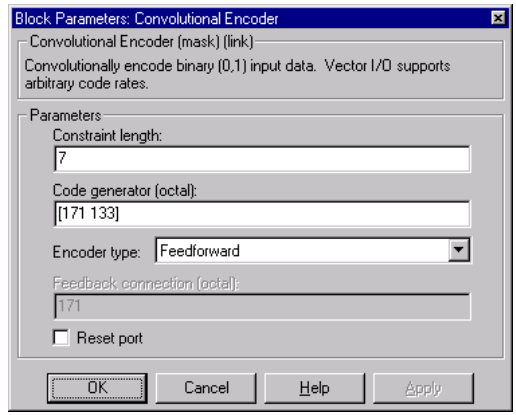
Description



The Convolutional Encoder block encodes a sequence of binary input vectors to produce a sequence of binary output vectors. The size of the input and output vectors depends on the code rate. When you specify a rate k/n code, the input is a length k vector, and the output is a length n vector.

You can configure the Convolutional Encoder block to implement either a feedforward encoder or a feedback encoder. To specify a particular encoder, you must also supply values for the parameters shown in the block mask below.

Dialog Box and Parameters



Constraint length

1-by-k vector specifying the delay for each of the k input bit streams.

Code generator

k-by-n matrix of octal numbers specifying the n output connections for each of the k inputs.

Encoder type

Feedforward or **Feedback**.

Feedback connection (**Feedback** configuration only)

1-by-*k* vector of octal numbers specifying the feedback connection for each of the *k* inputs.

Reset input

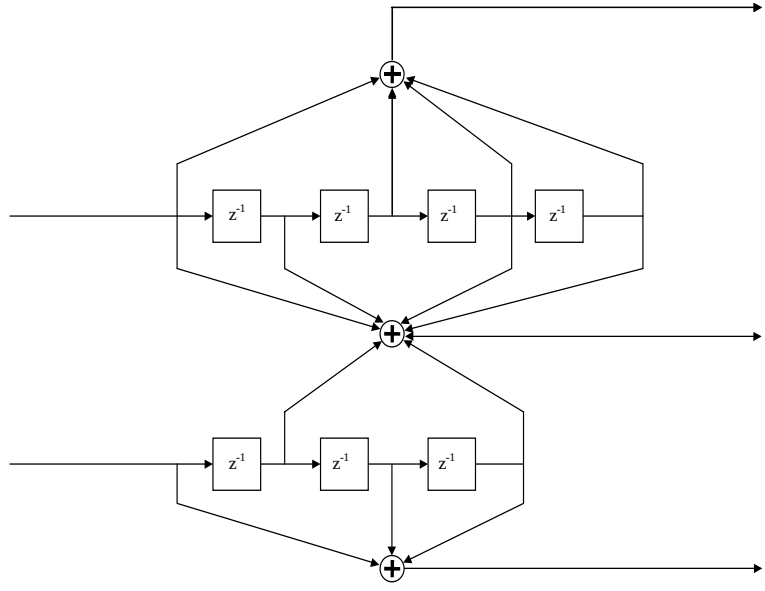
When you check this box, the encoder has a second input port labeled Rst. A nonzero input value at this port causes the internal memory to be set to its initial state prior to processing the input data.

Note: Although the constraint length could be derived from the code generator and feedback connection parameters, you must enter this parameter as a cross check for the code generator and feedback parameters.

Examples

Example 1: Rate 2/3 Feedforward Encoder

The diagram below shows an example of a typical rate 2/3 feedforward encoder.



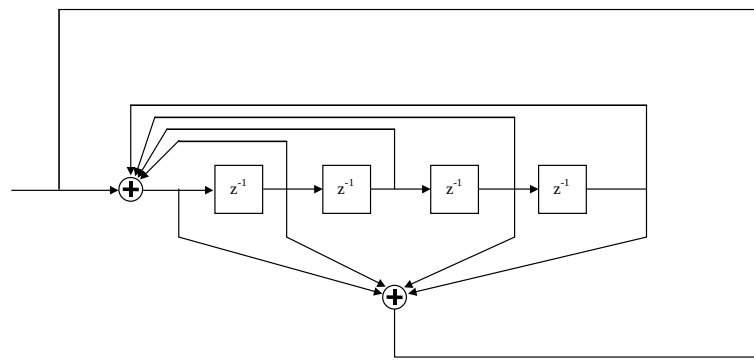
You can implement this encoder using the Convolutional Encoder block by supplying the following values for the parameters in the block mask.

You specify the **Constraint length** with a 1-by-2 vector, since there are two input bits. The elements of this vector indicate the number of bits stored in each shift register (including the current input bits). You must enter `[5 4]` for the code shown above.

To specify the **Code Generator** for the rate 2/3 encoder shown above, you must enter it as a 2-by-3 matrix of octal numbers, `[27 33 0;0 5 13]`.

Example 2: Rate 1/2 Feedback Encoder

An example of a rate 1/2 systematic encoder with feedback is shown below.



To specify this encoder, you must change the **Encoder type** parameter to **Feedback**. The other parameters are specified as follows.

You specify the **Constraint length** in this case by a scalar value, since there is only one input bit. Enter the value `5` for this encoder.

For systematic encoders, the **Code generator** and **Feedback connection** parameters corresponding to the systematic bits must have the same values. In this example, you must enter `[37 33]` for the **Code generator** and `37` for the **Feedback connection**.

Convolutional Encoder

Characteristics	Direct Feedthrough	Yes
	Sample Time	Discrete, Inherited
	Scalar Expansion	N/A
	Vectorized	Yes
	Complex	No

See Also Viterbi Decoder

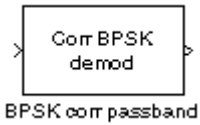
Purpose

Calculate binary phase shift keying demodulation correlation.

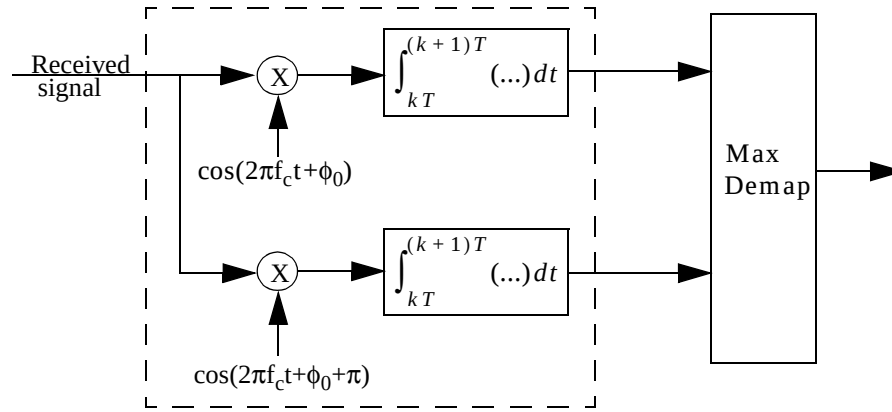
Library

Version 1.3 Passband Digital Modulation/Demodulation

Description



The correlation BPSK (binary phase shift keying) Demodulation block calculates the correlation between an input signal and a two-element vector of carrier frequency sinusoidal signals. The difference between the phases of the two carrier frequency signals is π. The following figure shows the block's operation.

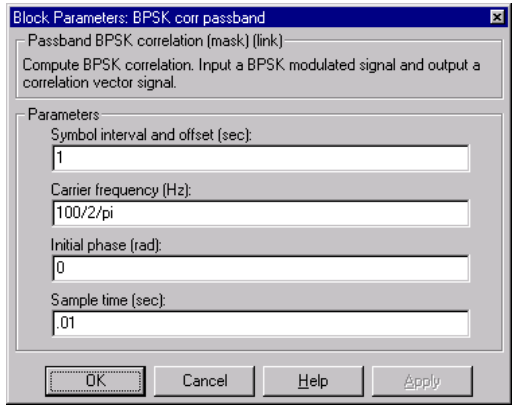


Corr BPSK Demod

In the figure, f_c is the carrier frequency, T is the symbol time interval, and ϕ_0 is the initial phase offset. This block accepts a scalar signal and it outputs a two-element correlation vector signal.

Corr BPSK Demod

Dialog Box and Parameters



Symbol interval

The duration of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Carrier frequency

The frequency of the carrier signal.

Initial Phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

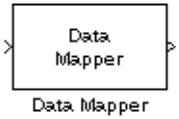
Purpose

Map integer symbols from one coding scheme to another.

Library

Utility Functions

Description



Data Mapper

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**.

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.

Table 3-1: Example of Binary to Gray and Gray to Binary Mappings

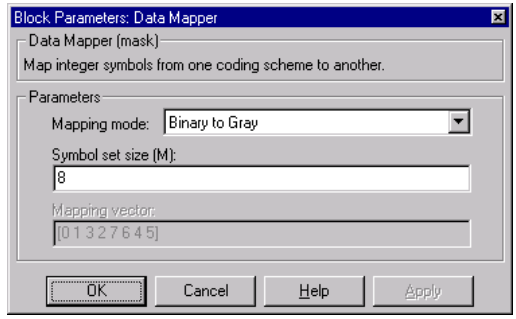
Binary to Gray Mode		Gray to Binary Mode	
Input	**Output**	**Input**	**Output**
0	0 *(000)*	0 *(000)*	0
1	1 *(001)*	1 *(001)*	1
2	3 *(011)*	2 *(010)*	3
3	2 *(010)*	3 *(011)*	2
4	6 *(110)*	4 *(100)*	7
5	7 *(111)*	5 *(101)*	6
6	5 *(101)*	6 *(110)*	4
7	4 *(100)*	7 *(111)*	5

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

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

When you select the **Straight Through** mode, the output is equal to the input.

Dialog Box and Parameters



Mapping mode

Binary to Gray, **Gray to Binary**, **User Defined**, **Straight Through**

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 (**User Defined** mode only)

A length *M* vector containing the integers 0 to *M*-1. The order of the elements of this vector specifies the mapping of inputs to outputs.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Descrambler

Purpose

Descramble the input signal.

Library

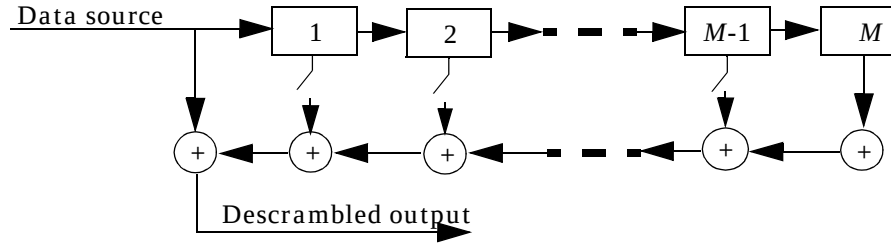
Utility Functions

Description

The Descrambler block is used in pair with the Scrambler block. When you use the Scrambler block on the transmitting side, you must use the Descrambler block on the receiving side.

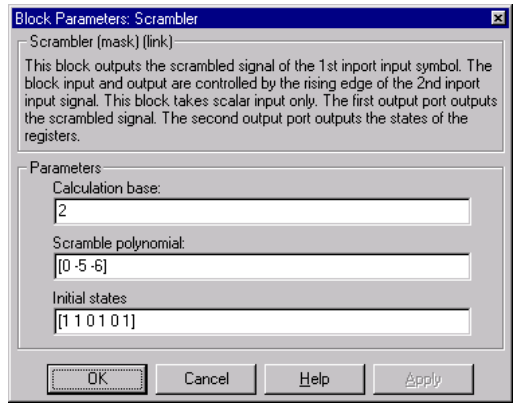
The Descrambler block has two input ports and two output ports. The first input port is the signal to be descrambled. The second input port is the synchronization pulse signal. The Descrambler block takes the first input port signal at the rising edge of the synchronization pulse (when the pulse crosses 0). The first output port outputs the descrambled signal. The second output port outputs the current register state vector. The Descrambler block inherits the sample time from the block that inputs to it.

The figure below illustrates the working process of the descrambler.



At the rising edge of the second input port synchronization pulse, the data input and the register shift to the next register. The switch is on or off as defined by the descrambler polynomial. To obtain the same output, the descrambler polynomial and the initial condition of the descrambler should be exactly the same as the one defined in the scrambler. See Scrambler on page 3–52 for the detailed definition of the scrambler polynomial. The initial state of at least one of the registers must be nonzero in order to generate a nonzero sequence.

Dialog Box and Parameters



Calculation base, Scramble polynomial, Initial states

Match these parameters to the ones used in the corresponding Scrambler block. The **Initial states** may be different, considering the transmitting and receiving filter delay.

Characteristics

Direct Feedthrough	Yes
Sample Time	Triggered action. Inherits the sample time of the blocks that input to this block.
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

Scrambler

Differential Decoder

Purpose

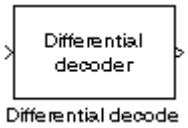
Decode a binary signal using differential coding technique.

Library

Utility Functions

Description

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

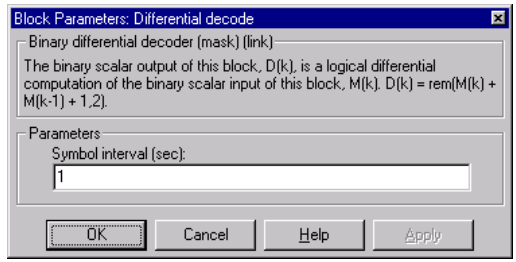


The input/output relationship of this block is given by

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

where m denotes the input and d denotes the output.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	Yes
Complex	No

Pair Block

Differential Encoder

Purpose

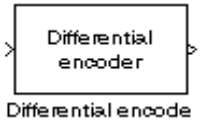
Encode a binary signal using differential coding technique.

Library

Utility Functions

Description

The Differential Encoder block encodes the binary input signal. The output of this block is the encoded binary signal.

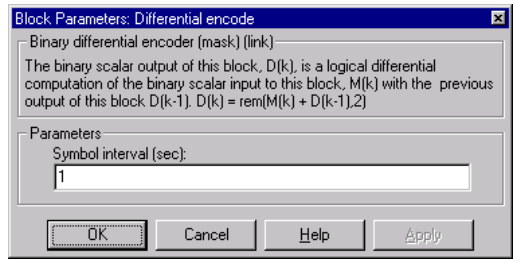


The input/output relationship of this block is given by

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

where m denotes the input and d denotes the output.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	Yes
Complex	No

Pair Block

Differential Decoder

DPSK Demod

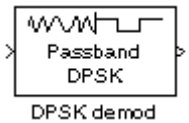
Purpose

Demodulate DPSK modulated signal.

Library

Digital Passband Modulation, in Modulation

Description



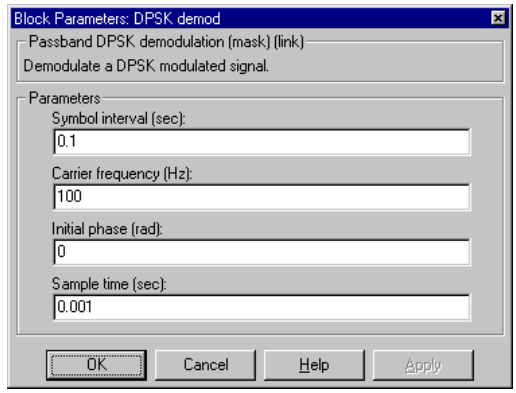
The DPSK (differential phase shift keying) Demodulation block demodulates the DPSK Mod signal. The block uses the digital MPSK Demod block and the Differential Decoder block. Refer to their individual reference pages for descriptions of the techniques involved.

DPSK Demod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a modulated analog signal. The output is a demodulated binary signal.

Dialog Box and Parameters



Symbol interval, Carrier frequency, Initial phase

Match these parameters to the ones used in the corresponding DPSK Mod block. The offset value in **Symbol interval** can be different.

Sample time (sec)

The block's sample time in seconds.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

DPSK Mod

DPSK Mod

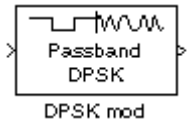
Purpose

Modulate the input signal using differential phase shift keying method.

Library

Digital Passband Modulation, in Modulation

Description



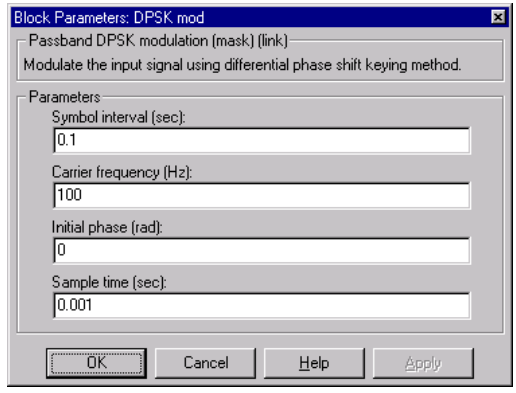
The DPSK (differential phase shift keying) Modulation block modulates the input signal. The block uses the Differential Encoder block before feeding the signal to the digital MPSK Mod block. Refer to their individual reference pages for descriptions of the techniques involved.

DPSK Mod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to the DPSK Mod block is a binary signal. The output is a modulated analog signal with a maximum amplitude equal to 1.

Dialog Box and Parameters



Symbol interval

The duration of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Carrier frequency

The frequency of the carrier signal.

Initial phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

DPSK Demod

Error Rate Calculation

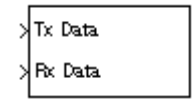
Purpose

Compute the bit error rate or symbol error rate of input data.

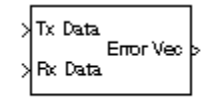
Library

Comm Sinks

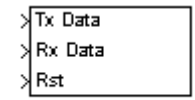
Description



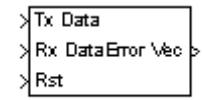
Error Rate Calculation



Error Rate Calculation



Error Rate Calculation



Error Rate Calculation

The Error Rate Calculation block takes input data from a transmitter and compares it with input data from a receiver. It calculates the error rate as a running statistic, by dividing the total number of data elements that are not equal by the total number of input data elements from one source.

This block adapts to either scalar or vector input. For vector inputs, the count of the number of elements compared at each sample time increases by the length of the vector.

You can easily compute symbol or bit error rates because the Error Rate Calculation block does not consider the magnitude of the difference between input data elements. If the inputs are bits, this block computes the bit error rate. If the inputs are symbols, the result is the symbol error rate.

The data produced at the output of the Error Rate Calculation block is a vector whose entries correspond to:

- The error rate
- The number of error events
- The total number of input events

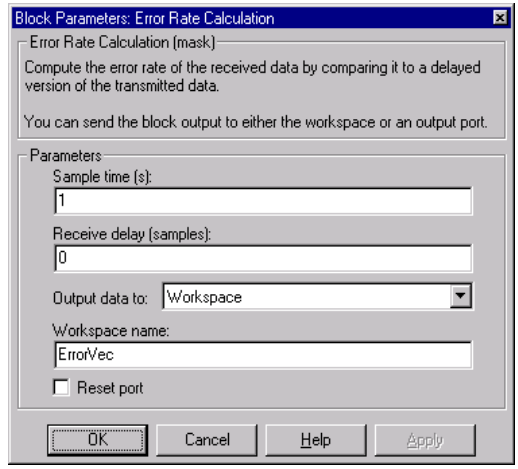
You can configure this block to provide this output data to the workspace or to a port. When you write the data to the workspace, the values that are stored are the values that are current at the time you stop the simulation. To observe the running error statistics, you must provide the data to a port.

You can optionally configure this block with a reset port. When the reset port input is nonzero, the Error Rate Calculation block clears the error statistics.

Combining these two options, you can configure the Error Rate Calculation block in one of the following four modes:

- No reset, output to workspace (first block shown)
- External reset, output to workspace (second block shown)
- No reset, output to port (third block shown)
- External reset, output to port (fourth block shown)

Dialog Box and Parameters



Sample time

Sample time of the transmit and receive input data.

Receive delay

Number of samples by which the received data lags behind the transmitted data.

Output data to

Workspace or **Port**, depending on where you want to send the output data.

Workspace name (only if the **Output data** option **Workspace** is selected)

Name of workspace variable for output data vector.

Reset port

When you check this box, this block has a second input port labeled Rst.

Characteristics

Direct Feedthrough	No
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	Yes
Complex	No

MSK Demod

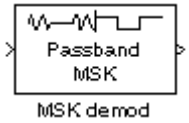
Purpose

Demodulate MSK modulated signal.

Library

Digital Passband Modulation, in Modulation

Description



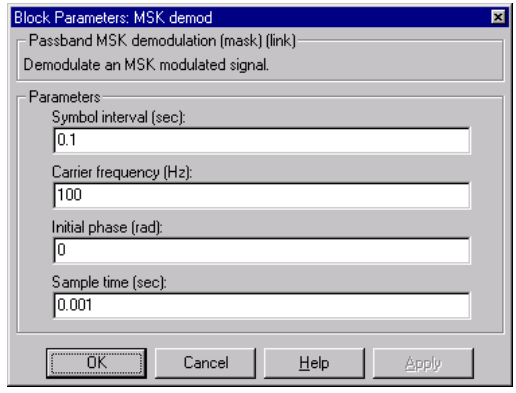
The MSK (minimum shift keying) Demodulation block demodulates the MSK Mod signal. This block uses a matched filter to process the input signal and detect the in-phase and quadrature components of the signal. Then the filtered output is fed to an OQPSK Demap block resulting in a binary signal.

MSK Demod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a modulated analog signal. The output is a demodulated binary signal.

Dialog Box and Parameters



Symbol interval, Carrier frequency, Initial phase

Match these parameters to the ones used in the corresponding MSK Mod block. The offset value in **Symbol interval** can be different.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

MSK Mod

MSK Mod

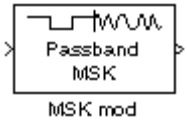
Purpose

Modulate the input signal using minimum shift keying method.

Library

Digital Passband Modulation, in Modulation

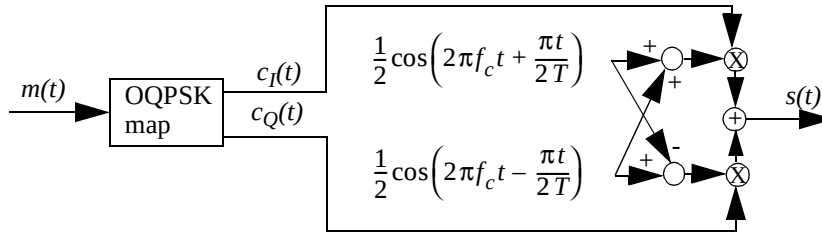
Description



The MSK (minimum shift keying) Modulation block modulates the binary input signal. The block uses the OQPSK Map block to map the binary input signal to the in-phase component $c_I(t)$ and quadrature component $c_Q(t)$. The output of the MSK Mod is $s(t)$ where

$$s(t) = c_I(t)\cos\left(\frac{\pi t}{2T}\right)\cos 2\pi f_c t + c_Q(t)\sin\left(\frac{\pi t}{2T}\right)\sin 2\pi f_c t$$

MSK Mod is implemented based on the block diagram structure in the following figure.

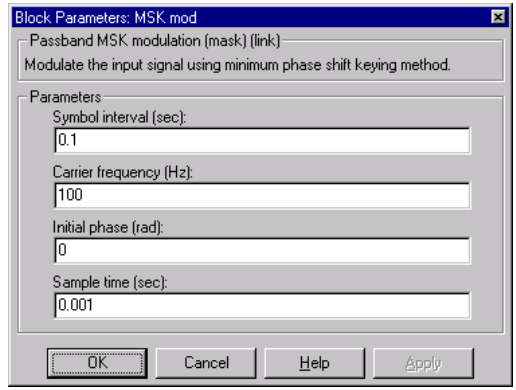


MSK Mod is a passband simulation block. There are three time related variables in this block: symbol interval (T_d), carrier frequency (f_c), and simulation sample time (T_s). You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a binary signal. The output is a modulated analog signal with a maximum amplitude equal to 1.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol.

Carrier frequency

The frequency of the carrier signal.

Initial phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

MSK Demod

OQPSK Demap

Purpose

Reverse OQPSK map converted signal.

Library

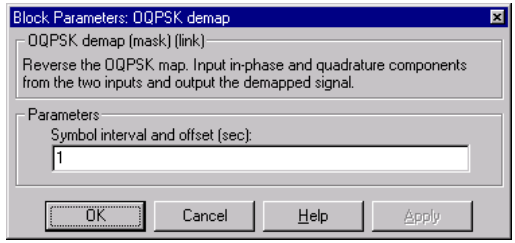
Digital Passband Modulation, in Modulation

Description

OQPSK demap

The OQPSK (offset quadrature phase shift keying) Demap block reverses an OQPSK Map converted signal. The inputs to this block are in-phase and quadrature components of the QADM (passband analog quadrature amplitude demodulation) signal. The output of this block is a scalar binary signal, with sample time *T*. Refer to the OQPSK Map block description for the conversion technique.

Dialog Box and Parameters



Symbol interval and offset

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

OQPSK Map

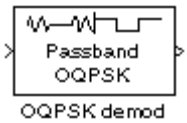
Purpose

Demodulate OQPSK modulated signal.

Library

Digital Passband Modulation, in Modulation

Description



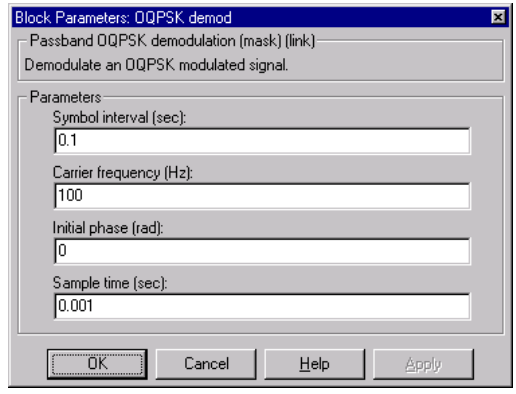
The OQPSK (offset quadrature phase shift keying) Demodulation block demodulates the OQPSK Mod signal. The block uses the QADM (passband analog quadrature amplitude demodulation) block and the OQPSK Demap block. Refer to their individual reference pages for descriptions of the techniques involved.

OQPSK Demod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a modulated analog signal. The output is a demodulated binary signal.

Dialog Box and Parameters



Symbol interval, Carrier frequency, Initial phase

Match these parameters to the ones used in the corresponding OQPSK Mod block. The offset value in **Symbol interval** can be different.

Sample time

The block's sample time.

OQPSK Demod

Characteristics	Direct Feedthrough	Yes
	Sample Time	Discrete
	Scalar Expansion	N/A
	Vectorized	N/A
	Complex	No
Pair Block	OQPSK Mod	

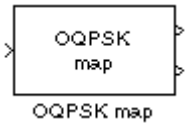
Purpose

Convert a binary scalar input signal to a binary vector output signal, which includes in-phase and quadrature components.

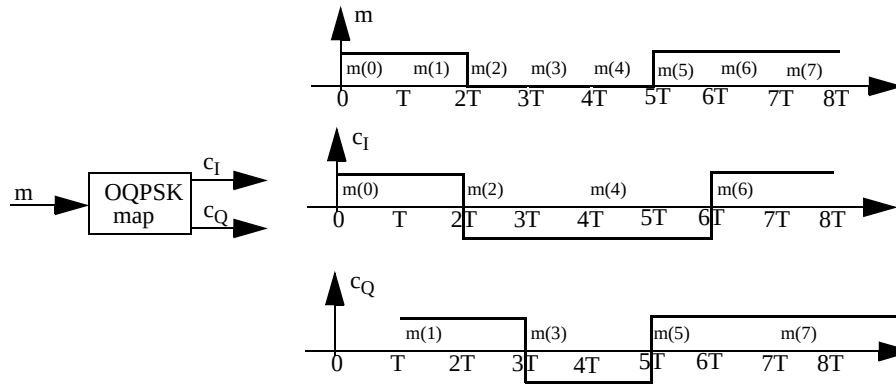
Library

Digital Passband Modulation, in Modulation

Description

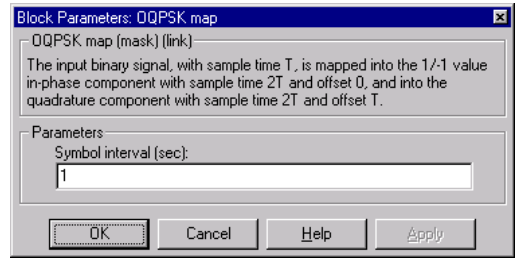


For OQPSK (offset quadrature phase shift keying) Mapping, an OQPSK Modulation maps its binary input signal to a two-component vector. The converted output contains in-phase and quadrature components. The mapped signal is then input to the QAM (passband analog quadrature amplitude modulation) block. The mapping method is explained in the following figure.



In the figure, $m(kT)$ is the input signal, and $c_I(kT)$ and $c_Q(kT)$ are the output in-phase and quadrature signals respectively. Compare the differences between this block and the QPSK Map block to see the time shifting difference.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol.

OQPSK Map

Characteristics	Direct Feedthrough	Yes
	Sample Time	Discrete
	Scalar Expansion	N/A
	Vectorized	No
	Complex	No
Pair Block	OQPSK Demap	

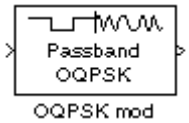
Purpose

Modulate the input signal using offset quadrature phase shift keying method.

Library

Digital Passband Modulation, in Modulation

Description



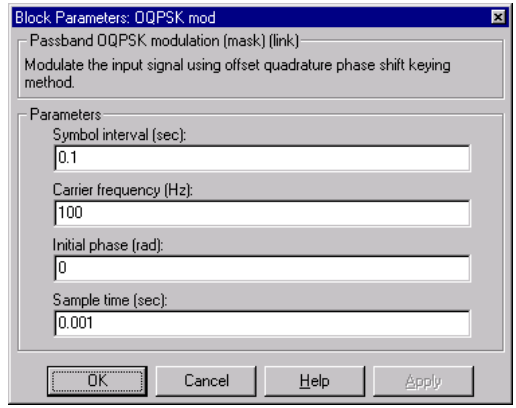
The OQPSK (offset quadrature phase shift keying) Modulation block modulates the input signal. The block uses the OQPSK Map block before feeding the signal to the QAM (passband analog quadrature amplitude modulation) block. Refer to their individual reference pages for descriptions of the techniques involved.

OQPSK Mod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a binary signal. The output is a modulated analog signal with a maximum amplitude equal to 1.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol.

Carrier frequency

The frequency of the carrier signal.

Initial phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	No
Complex	No

Pair Block

OQPSK Demod

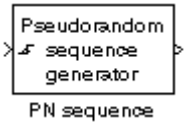
Purpose

Generate pseudonoise sequence.

Library

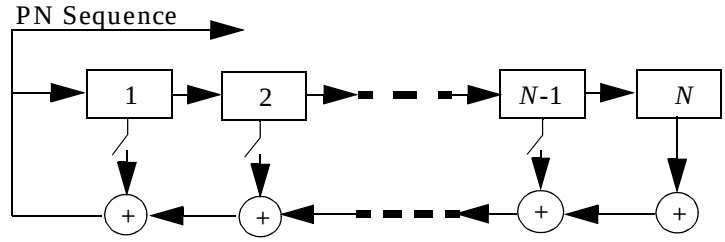
Comm Sources

Description



The pseudonoise sequence generator (PN Sequence) block generates a sequence of pseudorandom symbols. 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 block has one input port and one output port. The input port inputs the synchronization pulses. The output port outputs the generated pseudonoise sequence values.

The pseudorandom sequence generator block diagram is:

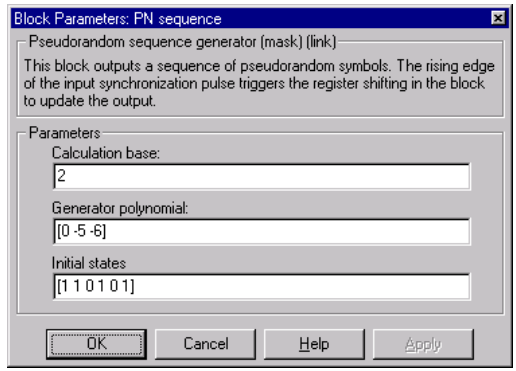


All *N* registers in the generator update their values at the rising edge of the input port synchronization pulse. The state of each switch is defined by the generator polynomial. You can specify the generator either by the coefficients of the polynomial, or by the terms for which the coefficient is 1; for the other terms, the coefficient is the default, 0. For example p = [1 0 0 0 0 0 1 0 1] and p = [0 -6 -8] represent the same polynomial $p(z) = 1 + z^{-6} + z^{-8}$.

It is very important that proper initial values be assigned to the pseudonoise sequence generator. The initial state of at least one of the registers must be nonzero in order to generate a nonzero sequence.

PN Sequence

Dialog Box and Parameters



Calculation base

The base used by the block for calculation.

Generator polynomial

Generator polynomial. Determines the shift register feedback connections.

Initial states

Initial states of the registers. The vector length of this entry must equal the order of the generator polynomial. The elements of the vector are restricted by the **Calculation base**.

Characteristics

Direct Feedthrough	Yes
Sample Time	Triggered action. Inherits the sample time of the blocks that input to this block.
Scalar Expansion	N/A
Vectorized	No
Complex	No

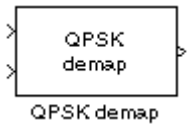
Purpose

Reverse QPSK map converted signal.

Library

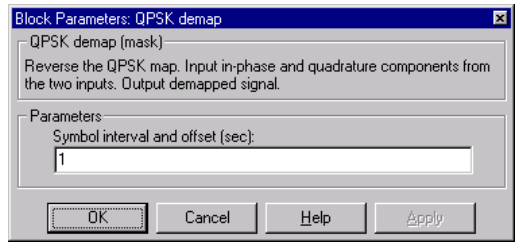
Version 1.3 Passband Digital Modulation/Demodulation

Description



The QPSK (quadrature phase shift keying) Demap block reverses a QPSK Map converted signal. The inputs to this block are in-phase component and quadrature component signals. The output of this block is a scalar binary signal, with sample time *T*. Refer to the QPSK Map block description for the conversion technique.

Dialog Box and Parameters



Symbol interval and offset

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

QPSK Map

QPSK Demod

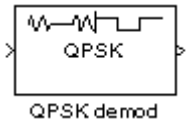
Purpose

Demodulate QPSK modulated signal.

Library

Version 1.3 Passband Digital Modulation/Demodulation

Description



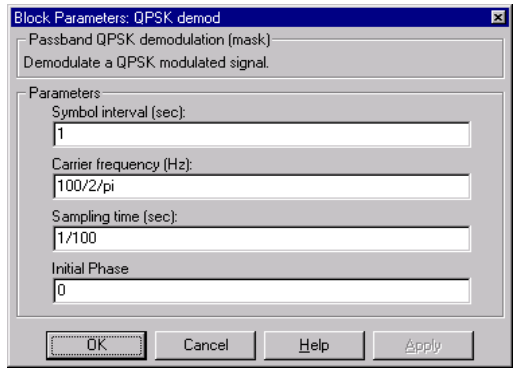
The quadrature phase shift keying (QPSK) Demodulation block demodulates the QPSK Mod signal. The block uses the QADM (passband analog quadrature amplitude demodulation) block and the QPSK Demap block. Refer to their individual reference pages for descriptions of the techniques involved.

QPSK Demod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a modulated analog signal. The output is a demodulated binary signal.

Dialog Box and Parameters



Symbol interval, Carrier frequency, Sampling time

Match these parameters to the ones used in the corresponding QPSK Mod block. The offset value in **Symbol interval** can be different.

Initial phase (rad)

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

QPSK Mod

QPSK Map

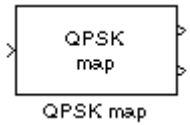
Purpose

Convert a binary scalar input signal to a binary vector output signal, which includes in-phase and quadrature components.

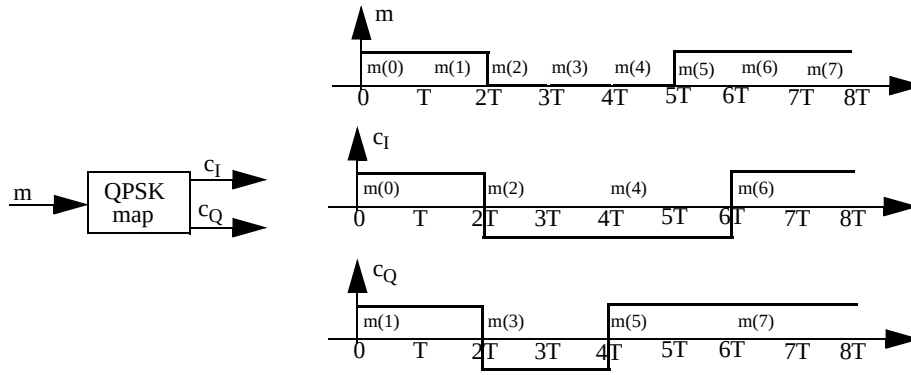
Library

Version 1.3 Passband Digital Modulation/Demodulation

Description

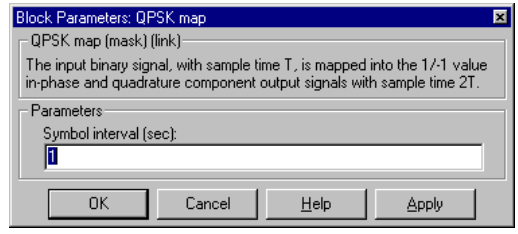


For quadrature phase shift keying (QPSK) mapping, a QPSK modulation maps its binary input signal to a two-component vector. The converted output contains in-phase and quadrature components. The mapped signal is then input to the QAM (passband analog quadrature amplitude modulation) block. The mapping method is explained in the following figure.



In the figure, $m(kT)$ is the input signal, and $c_I(kT)$ and $c_Q(kT)$ are the output in-phase and quadrature signals respectively. As illustrated in this figure, $c((k+1)T)$ outputs the input signal of $c((k+1)T)$. This makes the system noncausal. To implement the block, the time interval delay, T, is introduced.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

QPSK Demap

QPSK Mod

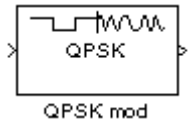
Purpose

Modulate the input signal using quadrature phase shift keying method.

Library

Version 1.3 Passband Digital Modulation/Demodulation

Description



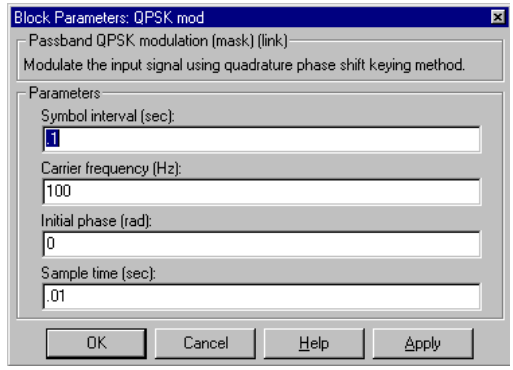
The quadrature phase shift keying (QPSK) Mod block modulates the input signal. This block uses the QPSK Map block before feeding the signal to the QAM (passband quadrature amplitude modulation) block. Refer to their individual reference pages for descriptions of the techniques involved.

QPSK Mod is a passband simulation block. There are three time related variables in this block: symbol interval *(T_d)*, carrier frequency *(f_c)*, and simulation sample time *(T_s)*. You must select values for these variables so that they satisfy the mathematical relations below.

$$T_d > 1/f_c > 2T_s$$

The input signal to this block is a binary signal. The output is a modulated analog signal with a maximum amplitude equal to 1.

Dialog Box and Parameters



Symbol interval

The sample time of the input symbol. When this parameter is a two-element vector, the second element is the offset value.

Carrier frequency

The frequency of the carrier signal.

Initial phase

The initial phase of the carrier signal.

Sample time

The block's sample time.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

QPSK Demod

Scrambler

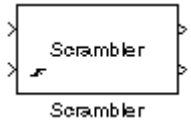
Purpose

Scramble the input signal.

Library

Utility functions

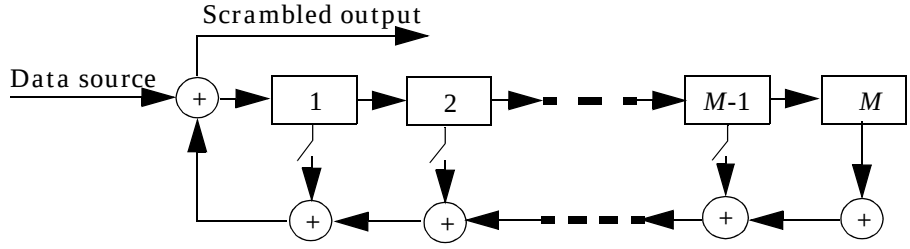
Description



Scrambling is used to minimize a long string of 0's or 1's in a transmitted signal. A long string of 0's or 1's may cause transmission synchronization problems.

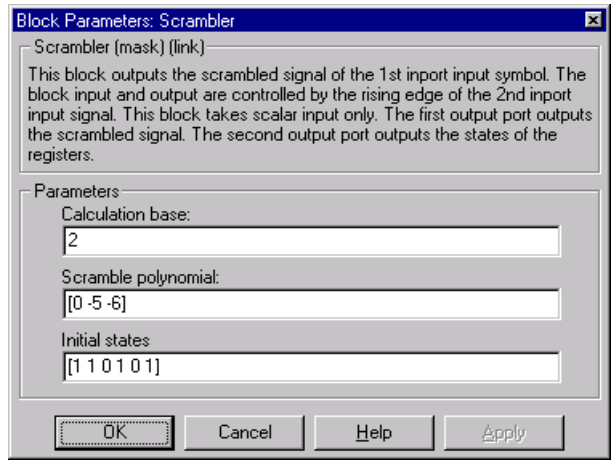
The Scrambler block has two input ports and two output ports. The first input port is the signal to be scrambled. The second input port is a synchronization pulse signal. The Scrambler block takes the first input port signal at the rising edge of the synchronization pulse (when the pulse crosses 0). The first output port outputs the scrambled signal. The second output port outputs the current register state vector. The Scrambler block inherits the sample time from the block that inputs to it.

The figure below illustrates the working process of the scrambler.



At the rising edge of the second input port synchronization pulse, the data input and the register shift to the next register. The switch is on or off as defined by the scrambler polynomial. You can specify the polynomial by using the coefficients of the polynomial, or by specifying the terms for which the coefficient is 1; for the other terms, the coefficient is the default, 0. For example p = [1 0 0 0 0 0 1 0 1] and p = [0 -6 -8] represent the same polynomial, $p(z) = 1 + z^{-6} + z^{-8}$.

Dialog Box and Parameters



Calculation base

Specify the calculation base. Let the base be N. The input and output of this block should be integers in the range [0, N-1].

Scramble polynomial

Scramble polynomial. You can define the polynomial using a descending ordered polynomial or by specifying the nonzero terms. For example [1 0 0 0 1 1] and [0 -4 -5] are the same, which means the polynomial is $1 + z^{-4} + z^{-5}$.

Initial states

Initial states of the registers. The vector length of this entry must equal the order of the generator polynomial. The elements of the initial state must be in the range [0, N-1].

Characteristics

Direct Feedthrough	Yes
Sample Time	Triggered action. Inherits the sample time of the blocks that input to this block.
Scalar Expansion	N/A
Vectorized	N/A
Complex	No

Pair Block

Descrambler

Viterbi Decoder

Purpose

Decode convolutionally encoded data using the Viterbi algorithm.

Library

Convolutional Codes, in Channel Coding

Description

Viterbi Decoder

Viterbi Decoder

The Viterbi Decoder block decodes a sequence of input symbols to produce a sequence of binary output symbols.

This block has two sets of parameters:

- Parameters that specify the corresponding rate k/n convolutional encoder (constraint length, code generator, encoder type, and feedback connection)
- Parameters that specify the operation of the decoder (soft decision type, number of soft decision bits, traceback depth, and reset input)

For explanations of the parameters corresponding to the convolutional encoder, see Convolutional Encoder on page 3–13.

Decision Types

When you set the decision type to unquantized, the decoder expects signed real input values. It uses +1 to represent a logical zero and -1 to represent a logical one.

When you set the decision type to hard decision, the decoder expects binary input values. It uses 0 to represent a logical zero and 1 to represent a logical one.

When you set the decision type to soft decision, the decoder requires input values in the range 0 to 2^b-1, where b is the value of the number of soft decision bits. For soft decisions, 0 represents the most confident decision for a logical zero and 2^b-1 represents the most confident decision for a logical one. The values in between these extremes represent less confident decisions. As an example, see the following table for information on how to interpret 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

Traceback Depth

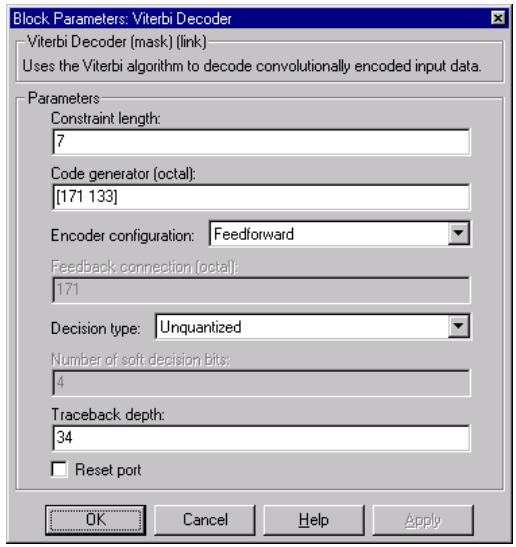
The traceback depth parameter specifies the decoding delay. Typically, you set the value of this parameter to about five times the constraint length.

Reset Port

The reset port functions similarly to the Convolutional Encoder block's reset port, except the Viterbi Decoder block's reset does not set all of its internal memory to zero. Rather, when the input to the reset port is nonzero, the decoder returns to its initial state by configuring its internal memory as follows:

- The all-zeros state metric is set to zero.
- All other state metrics are set to the maximum value.
- The traceback memory is set to zero.

Dialog Box and Parameters



Constraint length

1-by-*k* vector specifying the delay for each of the *k* input bit streams.

Code generator

k-by-*n* matrix of octal numbers specifying the *n* output connections for each of the *k* inputs.

Encoder configuration

Feedforward or **Feedback**.

Feedback connection (**Feedback** configuration only)

1-by-*k* vector of octal numbers specifying the feedback connection for each of the *k* inputs.

Decision type

Unquantized, **Hard decision**, or **Soft decision**.

Number of soft decision bits (**Soft decision** type only)

The number of soft decision bits used to represent each input.

Traceback depth

The amount of delay before the decoder releases an output decoded bit decision.

Reset port

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.

Characteristics

Direct Feedthrough	Yes
Sample Time	Discrete, Inherited
Scalar Expansion	N/A
Vectorized	Yes
Complex	No

See Also Convolutional Encoder

A

Corrections to User's Guide

This appendix documents corrections for errors in the *Communications Toolbox User's Guide* for Version 1.2. In the next version of the *Communications Toolbox User's Guide*, these errors will be corrected.

- The "Data Flow Simulation Example" on page 2–10 contains mistakes. The corrected example is:

```
% Define the multiple number, codeword length, and message
% length
M = 16; N = 15; K = 11;

% Define the carrier frequency, digital transfer frequency,
% and simulation frequency
Fc = 10000; Fd = 1000; Fs = 100000;

% Generate a 1100-by-1 matrix that contains the digital
% message to be transferred
msg = randint(K*100, 1);

% Code msg using BCH coding with codeword length N, message
% length K
code = encode(msg, N, K, 'bch');
code_M = bi2de(vec2mat(code, log(M)/log(2)));

% Use M-ary QASK method for modulation
modu = dmod(code_M, Fc, Fd, Fs, 'qask', M);
% Assume the std of the AWGN is 0.1
std_value = 0.1;

% Add AWGN with std of the noise equal to std_noise
modu_noise = modu + randn(length(modu), 1) * std_value;

% Demodulation process
demo_M = ddemod(modu_noise, Fc, Fd, Fs, 'qask', M);
demo= de2bi(demo_M)'; demo = demo(:);

% BCH decode
msg_r = decode(demo, N, K, 'bch');

% Calculate the bit-error rate
rate = biterr(msg, msg_r);
```

- In the “Source Coding” tutorial on page 3-21, the following line appears in the example:

  ```
  [p_trans, partition, codebook] = dpcmopt(sig, 1, N)
  ```

  The line should be:

  ```
  [p_trans, codebook, partition] = dpcmopt(sig, 1, N)
  ```

- In the “Example of a Linear Block Code” on page 3-32, the following line appears in the example:

  ```
  rcv = decode(code_noise,N,K 'block', gen2par(G));
  ```

  The line should be:

  ```
  rcv= decode(code_noise,N,K,'linear', gen2par(G));
  ```

- In the example on page 6-74 (the reference page for the Reed-Solomon Encode Integer Vector In/Out block), the message vector [6 7 4 0 1] should be [6 7 4 0 4]. The codeword [3 2 6 7 4 0 1] should be [3 2 6 7 4 0 4].

- In the example on page 6-76 (the reference page for the Reed-Solomon Decode Integer Vector In/Out block), the codeword [3 2 6 7 4 0 1] should be [3 2 6 7 4 0 4]. The message vector [6 7 4 0 1] should be [6 7 4 0 4].

Index

S

V