

Communications Blockset

For Use with Simulink[®]

Modeling
|

Simulation
|

Implementation
|

Getting Started

Version 2



How to Contact The MathWorks:



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



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



508-647-7000	Phone
--------------	-------



508-647-7001	Fax
--------------	-----



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

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

Getting Started with the Communications Blockset

© COPYRIGHT 2001 - 2004 by The MathWorks, Inc.

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

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History:	June 2001	Online only	New for Version 2.0.2 (Release 12.1)
	July 2002	First printing	Revised for Version 2.5 (Release 13)
	October 2004	Online only	Revised for Version 2.5.2 (Release 13SP2)

Introduction

1

What Is the Communications Blockset?	1-2
Related Products List	1-3
Required Software	1-4
Using This Guide	1-5
Who Should Read This Guide?	1-5
How to Use This Guide	1-5

Building Models

2

Introduction	2-2
New Simulink Users	2-2
Building Models of Communication Systems	2-2
Running a Simulink Model	2-4
Opening the Model	2-4
Overview of the Model	2-5
Quadrature Amplitude Modulation	2-6
Running a Simulation	2-6
Displaying the Error Rate	2-7
Setting Block Parameters	2-8
Displaying a Plot of Phase Noise	2-9
More Demos	2-10
Building a Simple Model	2-12
The Basic Steps	2-12
Using commstartup to Set Simulation Parameters	2-12
Opening a New Model Window	2-13

Opening Block Libraries	2-14
Moving Blocks into the Model Window	2-15
Connecting Blocks	2-16
Setting Block Parameters	2-18
Setting Simulation Parameters	2-19
Running the Model	2-20
Adding Noise to the Model	2-21
Saving a Model	2-24
Frames and Frame-Based Processing	2-25
Discrete Signals and Sample Times	2-25
Continuous Signals	2-25
Building a Channel Noise Model	2-27
Overview of the Model	2-27
Selecting Blocks for the Channel Noise Model	2-28
Setting Parameters in the Channel Noise Model	2-29
Connecting the Blocks	2-30
Running the Channel Noise Model	2-31
Reducing the Error Rate Using a Hamming Code	2-33
Building the Hamming Code Model	2-33
Hamming Encoder and Decoder	2-35
Setting Parameters in the Hamming Code Model	2-35
Labeling the Display Block	2-36
Running the Hamming Code Model	2-36
Displaying Frame Sizes	2-37
Adding a Scope to the Model	2-38
Setting Parameters in the Expanded Model	2-39
Observing Channel Errors with the Scope	2-41
Modeling a Channel with Modulation	2-43
Building the BPSK Model	2-43
Binary Phase Shift Keying	2-44
Setting Parameters in the BPSK Model	2-45
Running the BPSK Model	2-45
Reducing the Error Rate with a Cyclic Code	2-46
Building the Cyclic Code Model	2-46
Setting Parameters in the Cyclic Code Model	2-47
Running the Cyclic Code Model	2-48

Setting the Symbol Period	2-48
Using a Probe Block to Determine Symbol Period	2-49
Building a Frequency-Shift Keying Model	2-51
Building the FSK Model	2-52
Setting Parameters in the FSK Model	2-53
Running the FSK Model	2-55
Delays in the Model	2-57
Finding the Delay in the Model	2-58
Multirate Models	2-60
Using Sample Time Colors to Check Sample Times	2-60
Building a Convolutional Code Model	2-62
Building the Convolutional Code Model	2-62
Blocks in the Model	2-63
Setting Parameters in the Convolutional Code Model	2-64
Running the Convolutional Code Model	2-65

Using the Communications Blockset with MATLAB

3

Introduction	3-2
Sending Data to the MATLAB Workspace	3-3
Using a Signal To Workspace Block	3-3
Configuring the Signal To Workspace Block	3-4
Viewing the Error Rate Data in the Workspace	3-4
Sending Signal and Error Data to the Workspace	3-4
Viewing the Signal and Error Data in the Workspace	3-6
Analyzing Signal and Error Data	3-6
Running Simulations from the Command Line	3-8
Running a Single Simulation	3-8
Running Multiple Simulations	3-9
Setting Parameters in the Phase Noise Model	3-9
Plotting the Results of Multiple Simulations	3-11

Importing Data from the MATLAB Workspace	3-12
Simulating a Signal by Importing Data	3-12
Simulating Noise with Imported Data	3-13
Simulating Noise with Specified Error Patterns	3-14
Setting Sample Times and Samples per Frame	3-15
Learning More	3-17
Online Help	3-17
Demos	3-17
The MathWorks Online	3-18

Introduction

What Is the Communications Blockset?	1-2
Related Products List	1-3
Required Software	1-4
Using This Guide	1-5

What Is the Communications Blockset?

The Communications Blockset is a collection of Simulink® blocks for designing and simulating communication systems. With the Communications Blockset, you can design models in the form of block diagrams, using simple click-and-drag mouse operations. You can run simulations on a model at the push of a button, and tune models and visualize the results.

The Communications Blockset contains ready-to-use blocks to model various processes within communication systems, including

- Signal generation
- Source coding
- Error-correction
- Interleaving
- Modulation/demodulation
- Transmission along a channel
- Synchronization

In addition, you can create specialized blocks, to implement your own algorithms.

All the power of MATLAB® is available to you when you use the Communications Blockset. You can run simulations from the command line and invoke MATLAB expressions and M-files.

Related Products List

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Communications Blockset.

For more information about any of these products, see either

- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at <http://www.mathworks.com>; see the “products” section

Note The toolboxes listed below all include functions that extend the capabilities of MATLAB. The blocksets all include blocks that extend the capabilities of Simulink.

Product	Description
CDMA Reference Blockset	Design and simulate IS-95A mobile phone equipment
Communications Toolbox	Design and analyze communication systems
DSP Blockset	Design and simulate DSP systems
Real-Time Workshop	Generate C code from Simulink models
Signal Processing Toolbox	Perform signal processing, analysis, and algorithm development
Simulink	Design and simulate continuous- and discrete-time systems
Stateflow	Design and simulate event-driven systems

Required Software

To build and run the models in this manual, the Communications Blockset and the products it requires, which are listed below, must be installed:

- MATLAB
- Simulink
- Signal Processing Toolbox
- Communications Toolbox
- DSP Blockset

You can find instructions for installing these products in the MATLAB installation documentation for your platform. To determine what products are installed on your system, type `ver` at the MATLAB prompt. This displays information about the version of MATLAB you are running, including a list of all toolboxes and blocksets installed on your system.

The Communications Blockset provides demonstration files that show how to use the blockset to model communication systems. To launch these demos, type

`demo`

at the MATLAB prompt, then select **Blocksets**, and then **Communications**.

Using This Guide

Who Should Read This Guide?

If you are a new user, this guide, *Getting Started with the Communications Blockset*, is written for you. Its purpose is to quickly get you started using the Communications Blockset. It shows you how to

- Run a Simulink model
- Build and run models of communication systems
- Display the results of a simulation
- Change simulation parameters
- Run simulations from the MATLAB command line
- Transfer data between a model and the MATLAB workspace

If you are an experienced user, the online documentation for the Communications Blockset provides more detailed and comprehensive information about the blockset.

How to Use This Guide

This guide describes specific models of communication systems that you can build with the Communications Blockset. These examples show you how to use the blockset and familiarize you with some of its features. The best way to learn about the blockset is to build these models for yourself as you read the guide.

This guide contains two main chapters, in addition to this introduction:

- Chapter 2, “Building Models” introduces you to Simulink and explains how to build models with the Communications Blockset.
- Chapter 3, “Using the Communications Blockset with MATLAB” describes how to use MATLAB to expand the capabilities of the Communications Blockset.

Building Models

Introduction	2-2
Running a Simulink Model	2-4
Building a Simple Model	2-12
Building a Channel Noise Model	2-27
Reducing the Error Rate Using a Hamming Code	2-33
Modeling a Channel with Modulation	2-43
Reducing the Error Rate with a Cyclic Code	2-46
Building a Frequency-Shift Keying Model	2-51
Building a Convolutional Code Model	2-62

Introduction

This chapter introduces you to Simulink and the Communications Blockset. The chapter begins by showing you how to run an existing Simulink model and how to build a simple model. It then explains how to build typical models of communication systems using the Communications Blockset. These models show you how to use the blockset and illustrate some of its important features. We encourage you to build and run these models for yourself, as this is the best way to learn about the Communications Blockset.

New Simulink Users

If you are new to Simulink, the first two sections get you started with building and running models:

- “Running a Simulink Model” on page 2-4 shows how to run a demo model that is included in the Communications Blockset.
- “Building a Simple Model” on page 2-12 explains how to build a Simulink model and how to use Simulink block libraries. The section also explains sample times, frames, and sample-based versus frame-based processing.

Building Models of Communication Systems

If you are already familiar with Simulink, you can skip to these later sections, which describe some typical models of communication systems:

- “Building a Channel Noise Model” on page 2-27 describes a model that simulates a channel with noise and calculates the bit error rate of data passing through the channel.
- “Reducing the Error Rate Using a Hamming Code” on page 2-33 describes how to add a Hamming code to the channel noise model to improve the error rate. The section also explains how to view errors in a scope.
- “Modeling a Channel with Modulation” on page 2-43 describes a model of a channel with binary phase shift keying (BPSK) modulation and additive white Gaussian noise (AWGN).
- “Reducing the Error Rate with a Cyclic Code” on page 2-46 shows how to add a binary cyclic code to the BPSK model to improve the error rate. The section also shows how to use a Probe block to determine frame sizes and frame periods of signals.

- “Building a Frequency-Shift Keying Model” on page 2-51 describes a model of frequency-shift keying. The section explains delays created by blocks in the model and how to compensate for them.
- “Building a Convolutional Code Model” on page 2-62 describes a model of convolutional coding.

Running a Simulink Model

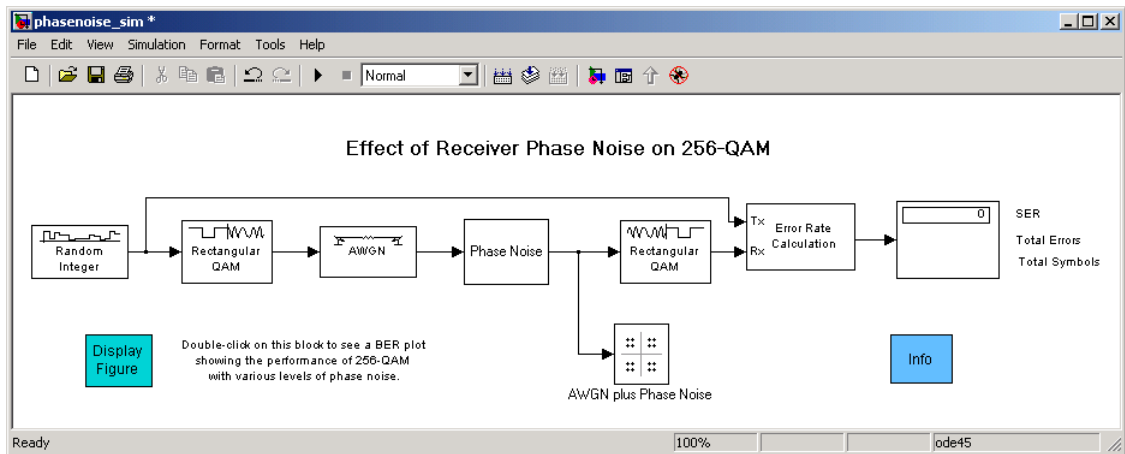
This section describes a demo model of a communications system that comes with the Communications Blockset. The model displays a scatter plot of a signal with added noise. The purpose of this section is to familiarize you with the basics of Simulink models and how they function.

The section takes you through some key elements of working with this model, including

- Opening the Model
- Overview of the Model
- Quadrature Amplitude Modulation
- Running a Simulation
- Displaying the Error Rate
- Setting Block Parameters
- Displaying a Plot of Phase Noise

Opening the Model

To open the model, first start MATLAB. In the MATLAB Command Window, type `phasenoise_sim` at the prompt. This opens the model in a new window, as shown in the following figure.



Overview of the Model

The model shown in the preceding section, “Opening the Model”, simulates the effect of phase noise on quadrature amplitude modulation (QAM) of a signal. During a simulation, the model generates a random signal and modulates it using QAM. The model then adds noise to the signal to simulate a channel. The model also displays the symbol error rate and a scatter plot of the modulated signal.

Notice that the model looks like a block diagram. The blocks generate and process data during a simulation. For example, the Random Integer Generator block, labeled “Random Integer,” generates a signal consisting of a sequence of random integers between zero and 255. This signal then moves in the direction of the arrows to other blocks in the model. The following sequence of blocks processes the signal:

- The Rectangular QAM Modulator Baseband block, to the right of the Random Integer Generator block, modulates the signal using baseband 256-ary QAM.
- The AWGN Channel block simulates a channel with noise, by adding additive white Gaussian noise (AWGN) to the modulated signal.
- The Phase Noise block adds random noise to the angle of the complex signal.
- The Rectangular QAM Demodulator Baseband block, to the right of the Phase Noise block, demodulates the signal.

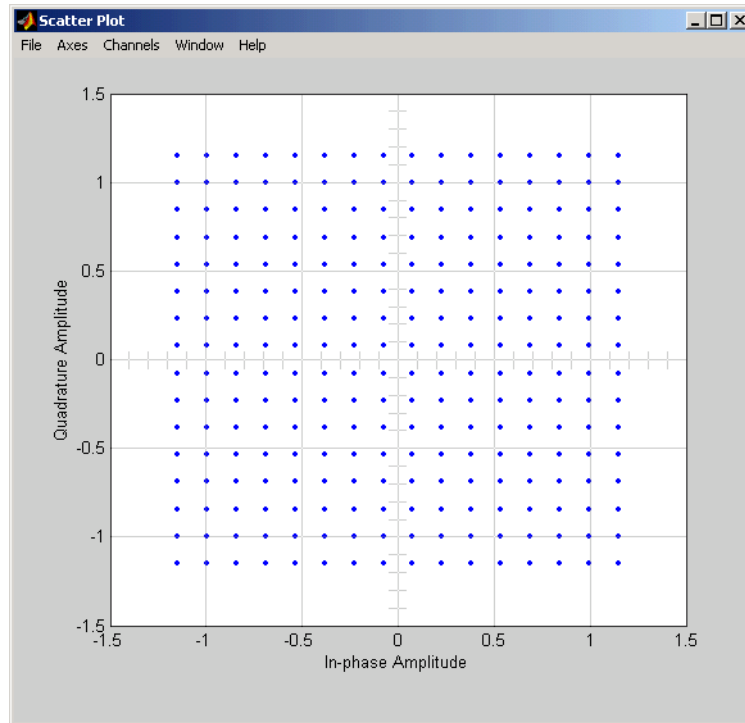
In addition, the model uses the following blocks to calculate the errors introduced by noise and displays the results:

- The Discrete-Time Scatter Plot Scope block, labeled “AWGN plus Phase Noise,” displays a scatter plot of the signal with added noise.
- The Error Rate Calculation block compares the received signal with the transmitted signal and calculates the number of symbols that differ between the two signals.
- The Display block, at the far right of the model window, displays the symbol error rate (SER), the total number of errors, and the total number of symbols processed during the simulation.

All these blocks are included in the Communications Blockset and Simulink. You can find more detailed information about any standard block by right-clicking the block and selecting **Help** from the context menu.

Quadrature Amplitude Modulation

This model simulates quadrature amplitude modulation (QAM), which is a method for converting a digital signal to a complex signal. The model modulates the signal onto a sequence of complex numbers that lie on a lattice of points in the complex plane, called the *constellation* of the signal. The constellation for baseband 256-ary QAM is shown in the following figure.



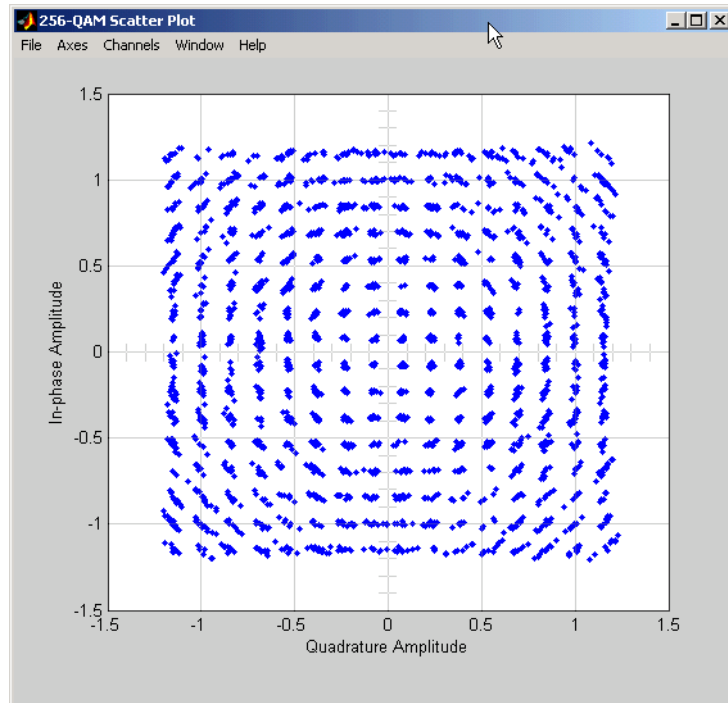
Constellation for 256-ary QAM

Running a Simulation

To run a simulation, select the **Simulation** menu from the top of the model window and then select **Start** (or, on Microsoft Windows, click the **Start** button on the toolbar). The simulation stops automatically at the **Stop time**, which is specified in the **Simulation parameters** dialog box. You can stop the simulation at any time by selecting **Stop** from the **Simulation** menu at the top

of the model window (or, on Microsoft Windows, by clicking the **Stop** button on the toolbar).

When you run the model, a new window appears displaying a scatter plot of the modulated signal with added noise, as shown in the following figure.



Scatter Plot of Signal Plus Noise

The points in the scatter plot do not lie exactly on the constellation shown in the figure Constellation for 256-ary QAM on page 2-6, because of the added noise. The radial pattern of points is due to the addition of phase noise, which alters the angle of the complex modulated signal.

Displaying the Error Rate

The Display block, labeled “Error Rate Display,” displays the number of errors introduced by the channel noise. When you run the model, three small boxes

appear in the block, as shown in the following figure, displaying the vector output from the Error Rate Calculation block.

0.007701	SER
774	Total Errors
1.005e+005	Total Symbols

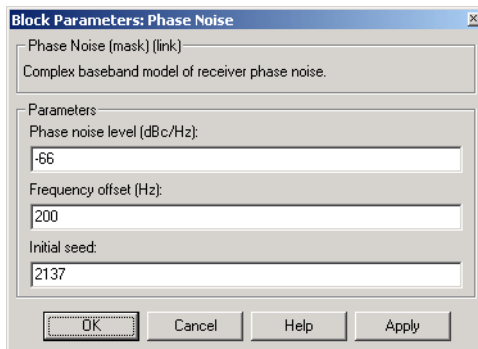
Error Rate Display

The block displays the output as follows:

- The first entry is the symbol error rate (SER).
- The second entry is the total number of errors.
- The third entry is the total number of comparisons made. The notation $1e+004$ is shorthand for 10^4 .

Setting Block Parameters

You can control the way a Simulink block functions by setting its parameters. To view or change a block's parameters, double-click the block. This opens a dialog box called the block's *mask*. For example, the mask for the Phase Noise block is shown in the following figure.



Mask for the Phase Noise Block

To change the amount of phase noise, click in the **Phase noise level (dBc/Hz)** field and enter a new value. Then click **OK**.

Alternatively, you can enter a variable name, such as `phasenoise`, in the field. You can then set a value for that variable at the MATLAB prompt in the Command Window, for example by typing `phasenoise=2`. Setting parameters at the command line is convenient if you need to run multiple simulations with different parameter values. See the section “Running Multiple Simulations” on page 3-9.

You can also change the amount of noise in the AWGN Channel block. Double-click the block to bring up its mask, and change the value in the **Es/No** parameter field. This changes the signal to noise ratio, in decibels. Decreasing the value of **Es/No** increases the noise level.

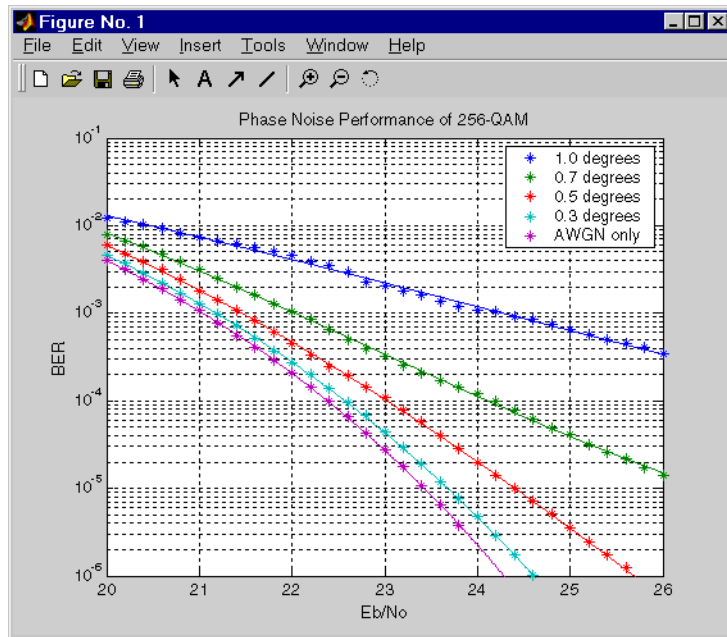
You can experiment with the model by changing these or other parameters and then running a simulation. For example,

- Change **Phase noise level (dBc/Hz)** to 0 in the mask for the Phase Noise block.
- Change **Es/No** to 100 in the mask for the AWGN Channel block.

This essentially removes all noise from the model. When you now run a simulation, the scatter plot appears as in the figure Constellation for 256-ary QAM on page 2-6.

Displaying a Plot of Phase Noise

Double-click the block labeled “Display Figure” at the bottom left of the model window. This displays a plot showing the results of multiple simulations.



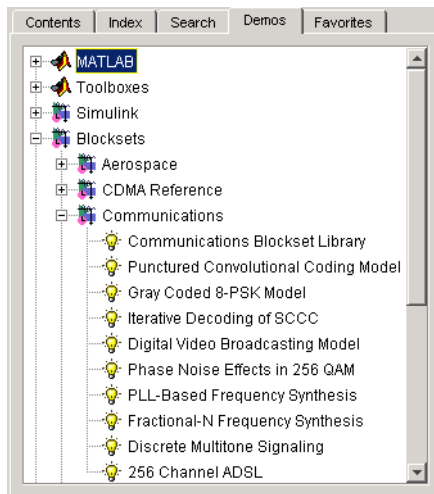
Plot of BER at Different Noise Levels

Each curve is a plot of bit error rate as a function of signal to noise ratio for a fixed amount of phase noise.

You can create plots like this by running multiple simulations with different values for the **Phase noise level (dBc/Hz)** and **Es/No** parameters. “Running Multiple Simulations” on page 3-9 describes how to do this with a MATLAB script, using variables for the parameters.

More Demos

You can find more demos for the Communications Blockset by typing **demo** at the MATLAB prompt or selecting the **Demos** tab in the Help browser. Click the **+** sign next to **Blocksets** in the left pane, and then select **Communications**. This displays a list of the Communications Blockset demos. To run a demo, double-click its name.



Building a Simple Model

In the previous section, you ran a model that was already built. This section explains how to build a simple Simulink model that displays a sine wave in a scope. For more detailed information on building models, see the Simulink documentation.

The Basic Steps

This section describes the basic steps in building a model. It explains how to

- Set simulation parameters with `commstartup`
- Open a new model window
- Open block libraries
- Move blocks into a model window
- Connect the blocks
- Set block parameters
- Set simulation parameters
- Run the model

Building a model usually involves several iterations, as you decide which blocks to include and what parameter settings to make. In the example in this section, you will refine the model by adding noise. The section explains how to

- Add noise to the model
- Save the model

The section also explains

- Frames and frame-based processing
- Discrete signals and sample times
- Continuous signals

Using `commstartup` to Set Simulation Parameters

Before starting to build the model, enter

```
commstartup
```

at the MATLAB prompt. This

- Sets the Simulink **Boolean logic signals** parameter to **Off**
- Sets default simulation parameters that are optimal for communications models

The Communications Blockset does not support signals with boolean data types. If you want to use Simulink blocks that output boolean data types, such as the Logical Operator block, in a model with blocks from the Communications Blockset, enter `commstartup` before building the model. The `commstartup` settings apply to any models you create during the current MATLAB session. You must enter `commstartup` at the beginning of each MATLAB session to establish these settings.

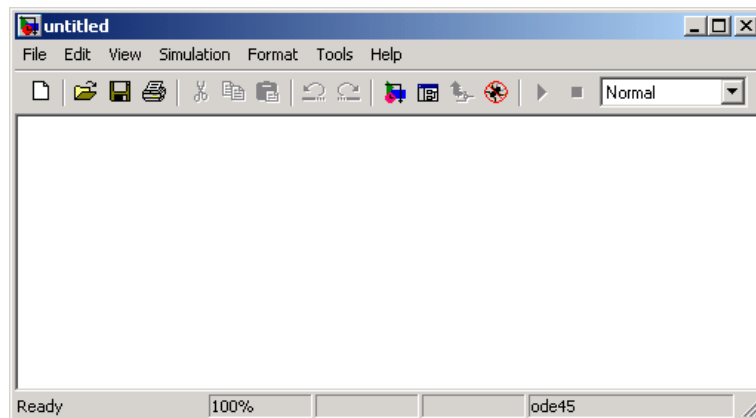
If you build a model without entering `commstartup` and subsequently decide to use Simulink blocks that output signals with boolean data types, turn off the model's **Boolean logic signals** parameter by entering

```
set_param('my_model', 'BooleanDataType', 'off')
```


where `my_model.mdl` is the name of the model.

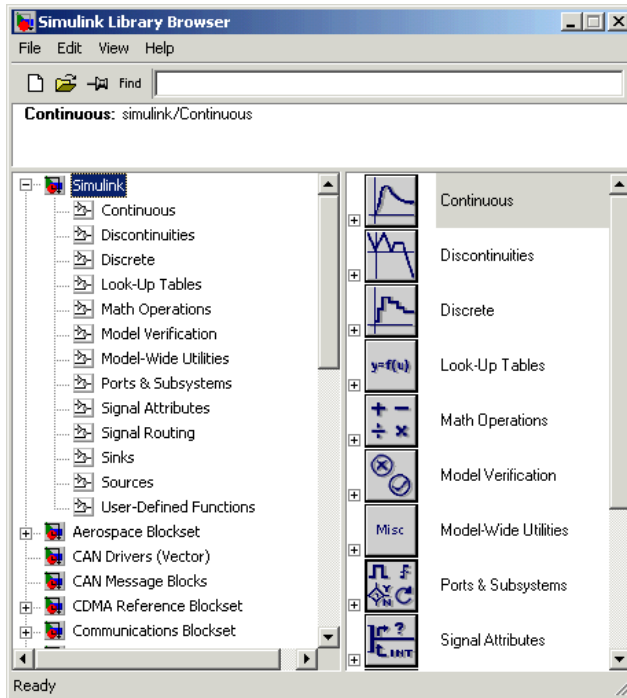
Opening a New Model Window

The first step in building a model is to open a new model window. To do so, select **New** from the **File** menu, and then select **Model**. This opens an empty model window, as shown in the following figure.



Opening Block Libraries

The next step is to select the blocks for the model. These blocks are contained in libraries. To view the libraries for the products you have installed, type `simulink` at the MATLAB prompt (or, on Microsoft Windows, click the Simulink button  on the MATLAB toolbar). If you are using Microsoft Windows, this displays the Simulink Library Browser, as shown in the following figure.



Simulink Library Browser

The left pane displays the installed products, each of which has its own library of blocks. To open a library, click the + sign next to the name of the blockset in the left pane. This displays the contents of the library in the right-hand pane.

You can find the blocks you need to build models of communication systems in the libraries of the Communications Blockset, the DSP Blockset, and Simulink.

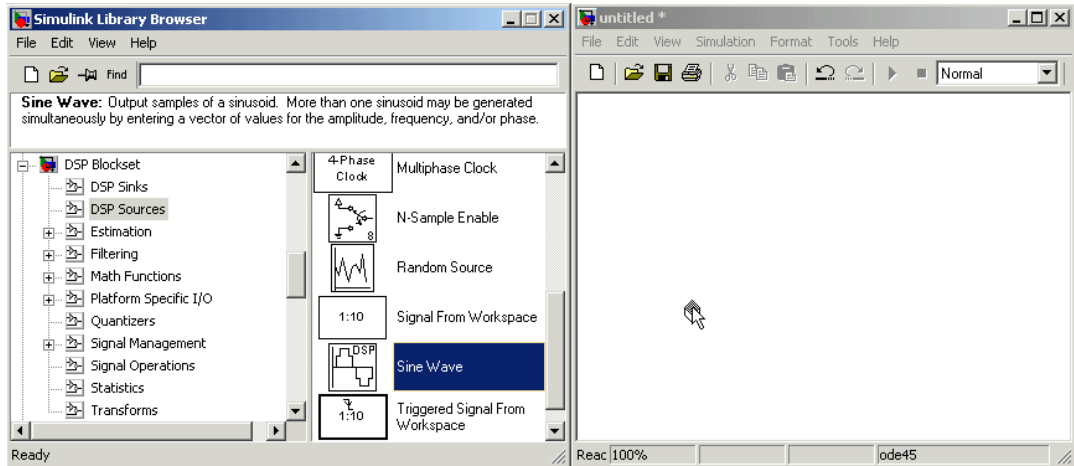
Note To search for a block in the Simulink Library Browser, enter a keyword in the field next to **Find**, at the top of the Browser window, and click **Find**. This displays the next block that contains the keyword. To continue the search, click **Find** again.

Moving Blocks into the Model Window

The next step in building the model is to move blocks from the Simulink Library Browser into the model window. To do so,

- 1 Click the + sign next to **DSP Blockset** in the left pane of the Library Browser. This displays a list of the DSP Blockset libraries, as shown in the following figure.
- 2 Click **DSP Sources** in the left pane. This displays a list of the DSP Sources library blocks in the right pane. If you do not see the Sine Wave block, scroll down the list until it is visible.
- 3 Click the Sine Wave block and drag it into the model window. To drag a block, position the mouse pointer over the block, then press and hold down the mouse button. Move the pointer into the model window, and then release the mouse button.
- 4 Click **DSP Sinks** in the left pane of the Library Browser.
- 5 Scroll down in the right pane of the Library Browser until you see the Vector Scope block, and drag the block into the model window to the right of the Sine Wave block.

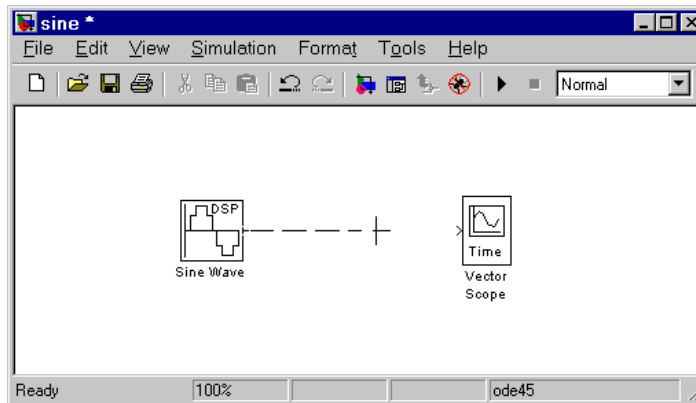
Once a block is in the model window, you can move it to another position by clicking and dragging the block while pressing the mouse button.



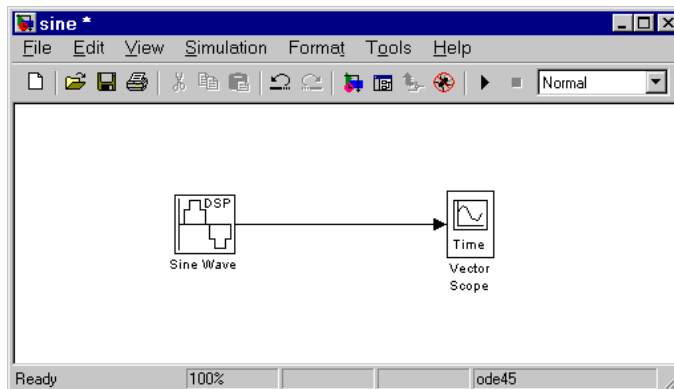
Dragging a Block into a Model Window

Connecting Blocks

The small arrowhead pointing outward from the right side of the Sine Wave block is an output port for the data the block generates. The arrowhead pointing inward on the Vector Scope block is an input port. To connect the two blocks, click the output port of the Sine Wave block and move the mouse toward the input port of the Vector Scope block, while pressing the mouse button, as shown in the following figure.



When the pointer is on the input port of the Vector Scope block, release the mouse button. You should see a solid arrow appear, as in the following figure.



Sine Wave Model

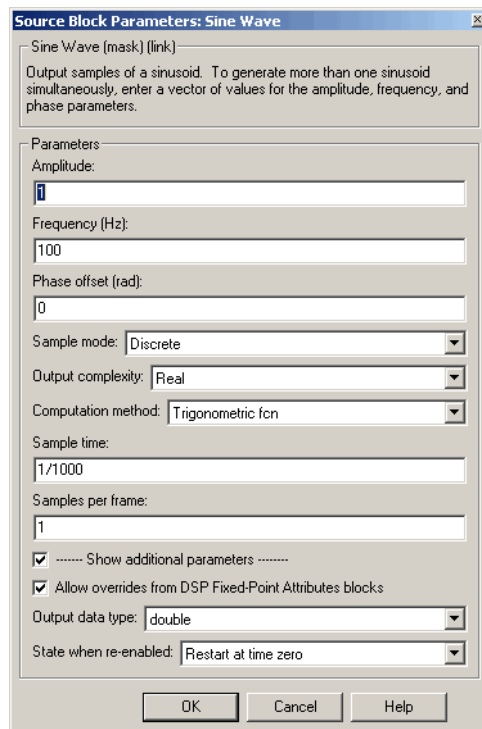
If you do not see a solid arrowhead, you have not made a good connection, and data will not pass from the Sine Wave block to the Vector Scope block. In this case, click the arrowhead again, drag it all the way to the Vector Scope's input port, and release the mouse button.

Setting Block Parameters

To set parameters for the Sine Wave block, double-click the block to bring up its mask, as shown in the following figure. Change the following parameters by clicking in the field next to the parameter, deleting the default setting, and entering the new setting in its place:

- 1 Set **Amplitude** to 5.
- 2 Set **Frequency** to 30.
- 3 Set **Samples per frame** to 100.
- 4 Click **OK**.

Note You must set **Samples per frame** to a value larger than 1 to see an image of the sine wave in the Scope block.

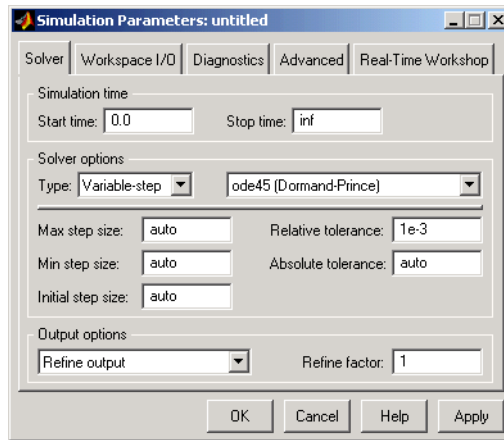


Mask for the Sine Wave Block

Setting Simulation Parameters

Besides individual block parameters, the model also has overall simulation parameters. To view the current settings,

- 1 Select the **Simulation** menu at the top of the model window.
- 2 Select **Simulation parameters** to open the **Simulation Parameters** dialog box, as shown in the following figure.



Simulation Parameters Dialog Box

If you typed `commstartup` before creating the model, the **Stop time** should be set to `inf`. The **Stop time** determines the time at which the simulation ends. Setting **Stop time** to `inf` causes the simulation to run indefinitely, until you stop it by selecting **Stop** from the **Simulation** menu.

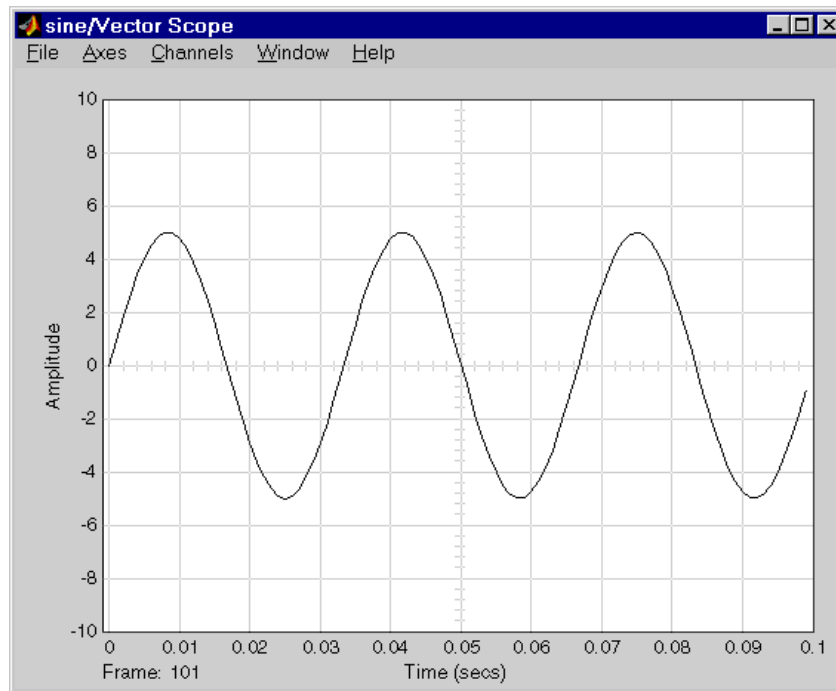
The **Stop time** is not the actual time it takes to run a simulation. The actual run-time for a simulation depends on factors such as the model's complexity and your computer's clock speed.

The settings in the **Simulation parameters** dialog box affect only the parameters of the current model.

Note To conserve memory in a model, click the **Workspace I/O** tab and make sure the boxes next to **Time** and **Output** under **Save to workspace** are not selected.

Running the Model

Run the model by selecting **Start** from the **Simulation** menu. When you do so, a scope window appears, displaying a sine wave as shown in the following figure.



Sine Wave Displayed in a Scope

Note If you do not see the sine wave in the scope, make sure that the **Samples per frame** parameter for the Sine Wave block is set to 100.

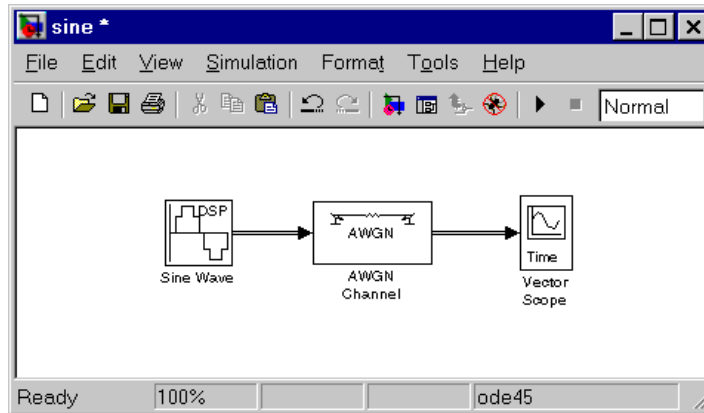
When you are finished observing the simulation, select **Stop** from the **Simulation** menu.

Adding Noise to the Model

You can add noise to the model using the AWGN Channel block, from the Channels library of the Communications Blockset. The block adds white Gaussian noise to the sine wave. Move the block from the Simulink Library

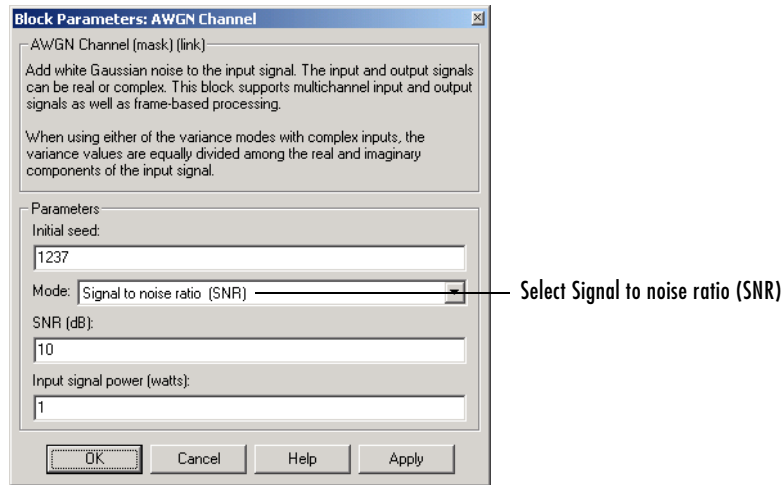
Browser into the model window, as described in “Moving Blocks into the Model Window” on page 2-15. You can add the block to the model as follows:

- 1 Extend the line between the Sine Wave block and the Vector Scope block by dragging the Vector Scope block to the right, to make room for the AWGN Channel block.
- 2 Click the AWGN block and drag it onto the line. This automatically connects the Sine Wave block and the Vector Scope block to the AWGN Channel block.



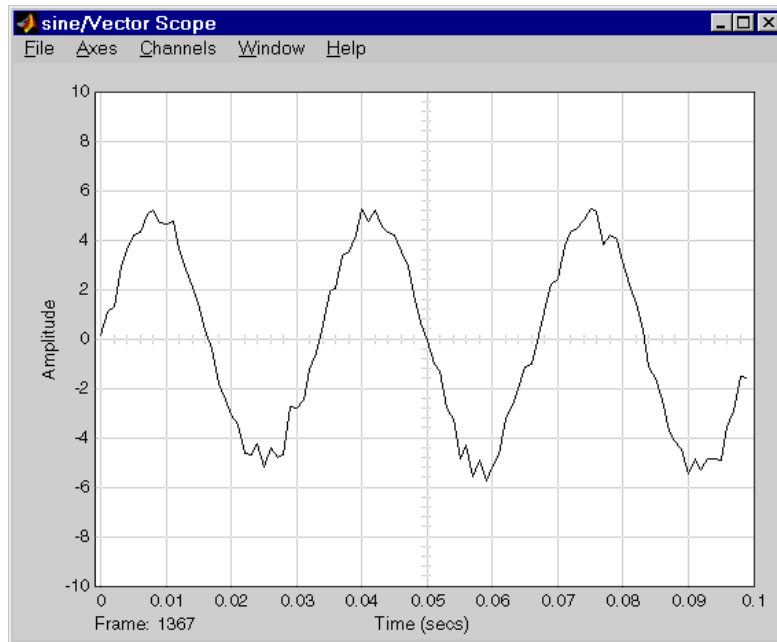
Sine Wave Plus Noise

Double-click the AWGN Channel block to bring up its mask, as shown in the following figure. Then, click the down arrow in the **Mode** field and select **Signal to noise ratio (SNR)**.



Mask for the AWGN Channel Block

Now when you run the model, the scope clearly shows the added noise.



Sine Wave with Noise Added

When you are finished observing the simulation, stop the model by selecting **Stop** from the **Simulation** menu.

Saving a Model

To save your model for future use, select **Save** from the **File** menu. The first time you save the model, this displays the **Save As** dialog box. In the **Save in** field, select the directory where you want to save the model. It is best to keep your work files in a separate directory from the files shipped with the product. In the **File name** field, enter a name for the model, such as `sine.mdl`, and click **Save**.

To load the model in a future MATLAB session, first change your working directory to the one where you saved the file. You can do this by selecting the directory in the **Current Directory** field on the MATLAB toolbar. Then type `sine` at the MATLAB prompt.

Frames and Frame-Based Processing

A *frame* is a sequence of samples combined into a single vector. By setting **Samples per frame** to 100 in the Sine Wave block, you set the frame size to 100, so that each frame contains 100 samples. This enables the Vector Scope block to display enough data for a good picture of the sine wave.

Another important reason to set the frame size is that many Communications Blockset blocks require their inputs to be vectors of specific sizes. If you connect a source block, such as the Sine Wave block, to one of these blocks, you can set the input size correctly by setting **Samples per frame** to the required value. The model described in “Reducing the Error Rate Using a Hamming Code” on page 2-33 shows how to do this.

In *frame-based processing* all the samples in a frame are processed simultaneously. In *sample-based processing*, on the other hand, samples are processed one at a time. The advantage of frame-based processing is that it can greatly increase the speed of a simulation. If you see double lines between blocks, the model uses frame-based processing.

Discrete Signals and Sample Times

The Sine Wave block in the DSP Blockset generates a *discrete* signal. This means that it updates the signal at integer multiples of a fixed time interval, called the *sample time*. You can set the length of this time interval in the **Sample time** parameter in the block’s mask. In the example shown in the figure Sine Wave Model on page 2-17, the **Sample time** has the default value of 1/1000. All the sources in the Communications Blockset and the DSP Blockset generate discrete signals exclusively. These sources are primarily designed for modeling digital communication systems.

To learn more about sample times, see “Building a Frequency-Shift Keying Model” on page 2-51.

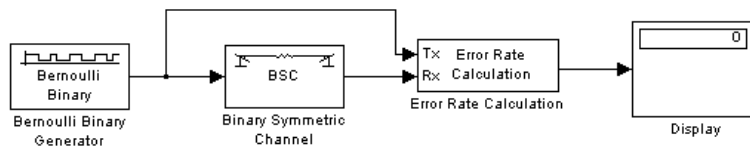
Continuous Signals

The Simulink libraries also contain blocks that generate *continuous* signals. This means that they update the signal at variable time intervals, whose length is determined by the numerical solver the simulation uses. For example, the Sine Wave block in the Simulink Sources library can generate a continuous sine wave.

Note Many blocks in the Communications Blockset accept only discrete signals. To find out whether a block accepts continuous signals, consult the reference page for the block.

Building a Channel Noise Model

This section shows how to build a simple model of a communication system. The model, shown in the following figure, contains the most basic elements of a communication system: a source for the signal, a channel with noise, and means of detecting errors caused by noise.



Channel Noise Model

We encourage you to build the model for yourself, as this is the best way to learn how to use the Communications Blockset.

This section gives an overview of the model, and explains how to

- Select blocks for the model
- Set parameters in the model
- Connect the blocks
- Run the model

Overview of the Model

The channel noise model generates a random binary signal, and then switches the symbols 0 and 1 in the signal, according to a specified error probability, to simulate a channel with noise. The model then calculates the error rate and displays the result. The model contains the following components.

Source

The source for the signal in this model is the Bernoulli Binary Generator block, which generates a random binary sequence.

The Channel

The Binary Symmetric Channel block simulates a channel with noise. The block introduces random errors to the signal by changing a 0 to a 1 or the reverse, with a probability specified by the **Error probability** parameter in the block's mask.

Error Rate Calculation

The Error Rate Calculation block calculates the error rate of the channel. The block has two input ports, labeled Tx , for the transmitted signal, and Rx , for the received signal. The block compares the two signals and checks for errors. The output of the block is a vector with three entries:

- Bit error rate, which you expect to be approximately .01, since this is the probability of error in the channel
- Number of errors
- Total number of bits that are transmitted

Display

The Display block displays the output of the Error Rate Calculation block, as described in “Displaying the Error Rate” on page 2-7.

Selecting Blocks for the Channel Noise Model

To build the model, first move its blocks into a new model window, as follows:

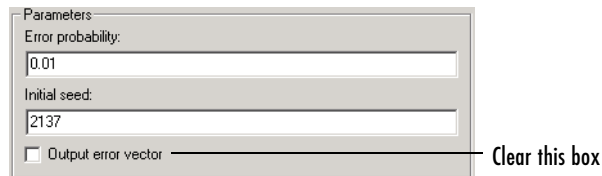
- 1 Type `commstartup` at the MATLAB prompt to set simulation parameters for the model.
- 2 Type `simulink` at the MATLAB prompt to open the Simulink Library Browser.
- 3 From the **File** menu, select **New**, and then **Model**. This opens a new model window.
- 4 Drag the following blocks from the Simulink Library Browser into the model window:
 - Bernoulli Binary Generator block, from the Data Sources sublibrary of the Comm Sources library
 - Binary Symmetric Channel block, from the Channels library

- Error Rate Calculation block, from the Comm Sinks library
- Display block, from the Simulink Sinks library

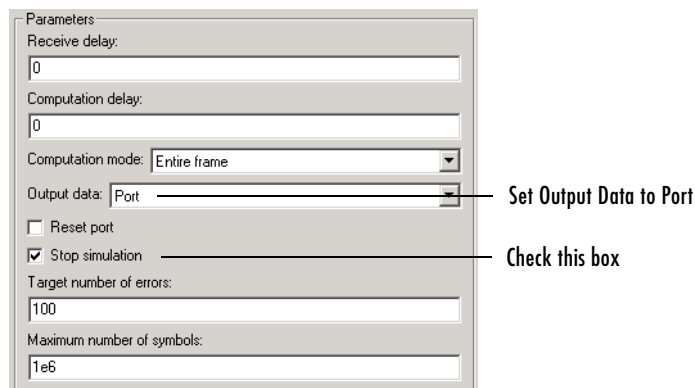
Setting Parameters in the Channel Noise Model

To set block parameters in the channel noise model, do the following:

- 1 Double-click the Binary Symmetric Channel block and make the following changes to the default parameters in the block's mask:
 - Set **Error probability** to 0.01.
 - Clear the box next to **Output error vector**. This removes the block's lower output port, which is not needed for this model.



- 2 Double-click the Error Rate Calculation block and make the following changes to the default parameters in the block's mask:
 - Set **Output data** to **Port** to create an output port for the block.
 - Select the box next to **Stop simulation**.



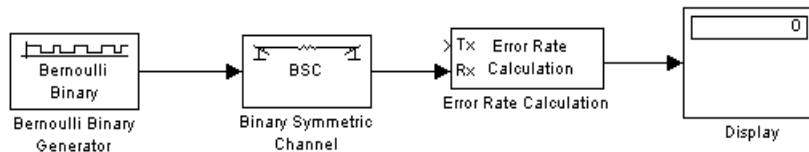
Selecting the box next to **Stop simulation** causes the simulation to stop after the target number of errors occurs or the maximum number of symbols is reached.

Initial Seeds

The Bernoulli Binary Generator block and the Binary Symmetric Channel block both use a random number generator to generate random sequences of bits. In both blocks, the **Initial seed** parameter initializes the random sequence. The initial seeds in the two blocks should have different values to ensure that the source signal and the channel noise are statistically independent. In general, initial seeds should have a different values in all blocks that have an **Initial seed** parameter.

Connecting the Blocks

Next, connect the blocks as shown in the following figure. Make sure to connect the arrow from the Binary Symmetric Channel block to the input port labeled Rx on the Error Rate Calculation block. To learn how to do this, see “Connecting Blocks” on page 2-16.



The next section explains how to draw the upper branch line in the model.

Drawing a Branch Line

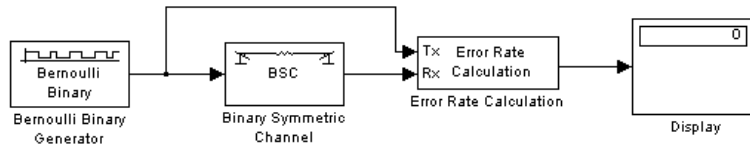
The upper line leading from the Bernoulli Binary Generator block to the Error Rate Calculation block, shown in the following figure, is called a *branch line*. Branch lines carry the same signal to more than one block.

To draw the branch line, follow these steps:

- 1 Right-click the line between the Bernoulli Random Generator block and the Binary Symmetric Channel block.

- 2 Move the mouse pointer to the input port labeled Tx on the Error Rate Calculation block, while pressing the right mouse button.
- 3 Release the mouse button. The end of the branch line should connect to the input port of the Error Rate Calculation block.
- 4 Click the horizontal section of the branch line and drag it upward until the line is above the Binary Symmetric Channel block.

The model should now appear as in the following figure.

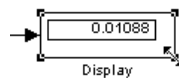


Running the Channel Noise Model

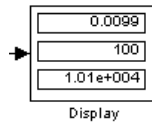
To run the model, select **Start** from the **Simulation** menu. After a few seconds, the model will stop automatically.

To see all three boxes in the Display block, you must enlarge the block slightly, as follows:

- 1 Select the Display block and move the mouse pointer to one of the lower corners of the block, so that a diagonal arrow appears on the corner, as shown.



- 2 Drag the corner of the block down with the mouse until three windows appear, as shown.



The Display block displays the following information:

- The bit error rate
- The number of errors
- The total number of bits that are transmitted

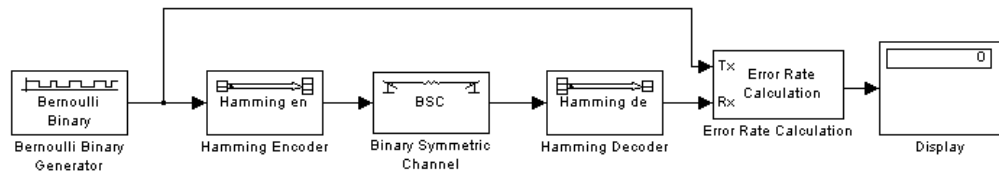
Note that the exact values that appear will vary, depending on the **Initial seed** parameters in the Bernoulli Binary Generator block and the Binary Symmetric Channel block.

Since the **Target number of errors** in the mask for the Error Rate Calculation block is set to 100, the simulation stops when 100 errors have been detected.

To save the model, select **Save** from the **File** menu, type a name for the model, such as channelnoise, in the **File name** field, and click **Save**.

Reducing the Error Rate Using a Hamming Code

This section describes how to reduce the error rate in the model shown in the figure Channel Noise Model on page 2-27 by adding an error-correcting code. The following figure shows an example that uses a Hamming code.



Hamming Code Model

The section explains how to

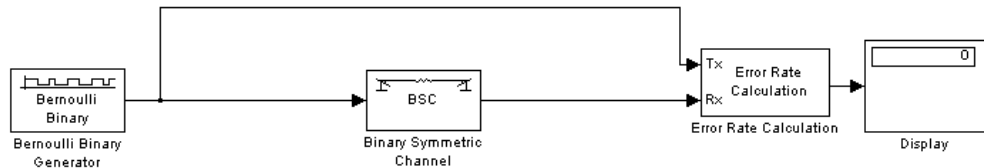
- Build the model
- Use the Hamming Encoder and Decoder blocks
- Set parameters in the model
- Label the Display block
- Run the model
- Display frame sizes
- Add a scope to the model
- Set parameters in the new model
- Observe channel errors with the scope

We encourage you to build the model for yourself. Alternatively, to open a completed version of the model, type `hammingdoc` at the MATLAB prompt.

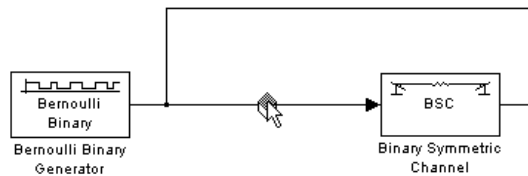
Building the Hamming Code Model

You can build the Hamming code model by adding blocks to the model shown in the figure Channel Noise Model on page 2-27. To do so, follow these steps:

- 1 Type `channeldoc` at the MATLAB prompt to open the channel noise model. Then save the model as `my_hamming` in the directory where you keep your work files. See “Saving a Model” on page 2-24.
- 2 Drag the following two Communications Blockset blocks from the Simulink Library Browser into the model window:
 - Hamming Encoder block, from the Block sublibrary of the Error Detection and Correction library
 - Hamming Decoder block, from the Block sublibrary of the Error Detection and Correction library
- 3 Click the right border of the model and drag it to the right to widen the model window.
- 4 Move the Binary Symmetric Channel block, the Error Rate Calculation block, and the Display block to the right by clicking and dragging. This creates more space between the Binary Symmetric Channel block and the blocks next to it. The model should now look like the following figure.



- 5 Click the Hamming Encoder block and drag it on top of the line between the Bernoulli Binary Generator block and the Binary Symmetric Channel block, to the right of the branch point, as shown in the following figure. Then release the mouse button. The Hamming Encoder block should automatically connect to the line from the Bernoulli Binary Generator block to the Binary Symmetric Channel block.



- Click the Hamming Decoder block and drag it on top of the line between the Binary Symmetric Channel block and the Error Rate Calculation block.

Hamming Encoder and Decoder

The Hamming Encoder block encodes the data before it is sent through the channel. The default code is the [7,4] Hamming code, which encodes message words of length 4 into code words of length 7. As a result, the block converts frames of size 4 into frames of size 7. The code can correct one error in each transmitted code word.

For an $[n,k]$ code, the input to the Hamming Encoder block must consist of vectors of size k . In this example, $k=4$.

The Hamming Decoder block decodes the data after it is sent through the channel. If at most one error is created in a code word by the channel, the block decodes the word correctly. However, if more than one error occurs, the Hamming Decoder block might decode incorrectly.

To learn more about the block coding features of the Communications Blockset, see the section “Block Coding” in the online documentation for the Communications Blockset.

Setting Parameters in the Hamming Code Model

Double-click the Bernoulli Binary Generator block and make the following changes to the parameter settings in the block’s mask, as shown in the following figure:

- Select the box next to **Frame-based outputs** in the mask for the Bernoulli Binary Generator block.

- 2 Set **Samples per frame** to 4. This converts the output of the block into frames of size 4, in order to meet the input requirement of the Hamming Encoder Block. See “Frames and Frame-Based Processing” on page 2-25 for more information about frames.

Parameters

Probability of a zero:
0.5

Initial seed:
61

Sample time:
1

Frame-based outputs

Samples per frame:
4

Interpret vector parameters as 1-D

Note Many Communications Blockset blocks, such as the Hamming Encoder block, require their input to be a vector of a specific size. If you connect a source block, such as the Bernoulli Binary Generator block, to one of these blocks, you should select the box next to **Frame-based outputs** in the mask for the source, and set **Samples per frame** to the required value.

Labeling the Display Block

You can change the label that appears below a block to make it more informative. For example, to change the label below the Display block to “Error Rate Display,” first select the label with the mouse. This causes a box to appear around the text. Then enter the changes to the text in the box.

Running the Hamming Code Model

To run the model, select **Start** from the **Simulation** model. The model terminates after 100 errors occur. The error rate, displayed in the top window of the Display block, is approximately .001. Note that you get slightly different results if you change the **Initial seed** parameters in the model or run a simulation for a different length of time.

You expect an error rate of approximately .001 for the following reason. The probability of two or more errors occurring in a code word of length 7 is

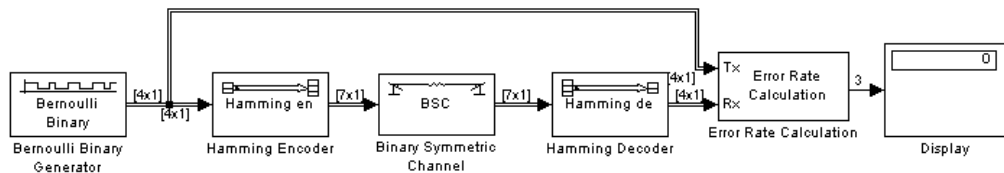
$$1 - (0.99)^7 - 7(0.99)^6(0.01) = 0.002$$

If the code words with two or more errors were decoded randomly, you would expect about half the bits in the decoded message words to be incorrect. This indicates that .001 is a reasonable value for the bit error rate.

To obtain a lower error rate for the same probability of error, you can try using a Hamming code with larger parameters. To do this, change the parameters **Code word length** and **Message length** in the Hamming Encoder and Decoder block masks. You also have to make the appropriate changes to the parameters of the Bernoulli Binary Generator block and the Binary Symmetric Channel block.

Displaying Frame Sizes

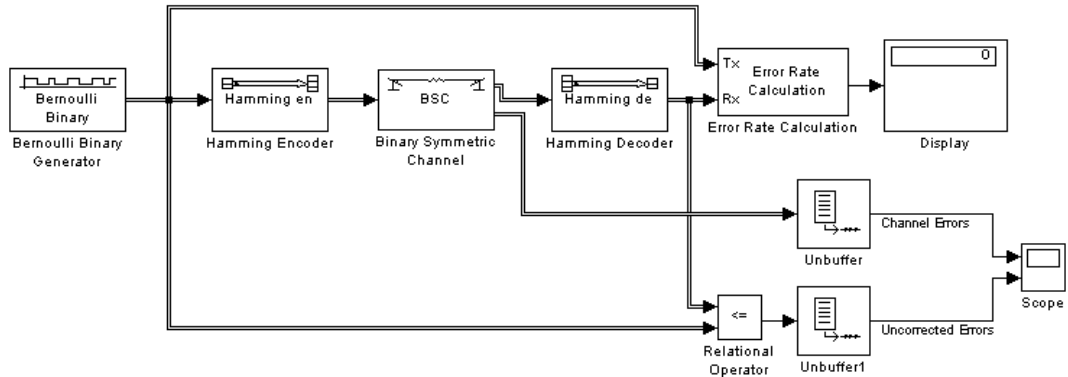
You can display the sizes of data frames in different parts of the model by selecting **Signal dimensions** from the **Format** menu at the top of the model window. This is shown in the following figure. Note that the line leading out of the Bernoulli Binary Generator block is labeled [4x1], indicating that its output consists of column vectors of size 4. Since the Hamming Encoder block uses a [7,4] code, it converts frames of size 4 into frames of size 7, so its output is labeled [7x1].



Displaying Frame Sizes

Adding a Scope to the Model

To display the channel errors produced by the Binary Symmetric Channel block, you can add a Scope block to the model. This is a good way to see whether your model is functioning correctly. The example shown in the following figure shows where to insert the Scope block into the model.



To build this model from the one shown in the figure Hamming Code Model on page 2-33, follow these steps:

- 1 Drag the following blocks from the Simulink Library Browser into the model window:
 - Relational Operator block from the Simulink Math Operations library
 - Scope block from the Simulink Sinks library
 - Two copies of the Unbuffer block from the Buffers sublibrary of the DSP Signal Management library
- 2 Double-click the Binary Symmetric Channel block to open its mask, and check the box next to **Output error vector**. This creates a second output port for the block, which carries the error vector.
- 3 Connect the blocks as shown in the preceding figure. To learn how to do this, see “Connecting Blocks” on page 2-16 and “Drawing a Branch Line” on page 2-30.

Setting Parameters in the Expanded Model


Make the following changes to the parameters for the blocks you added to the model.

Error Rate Calculation Block

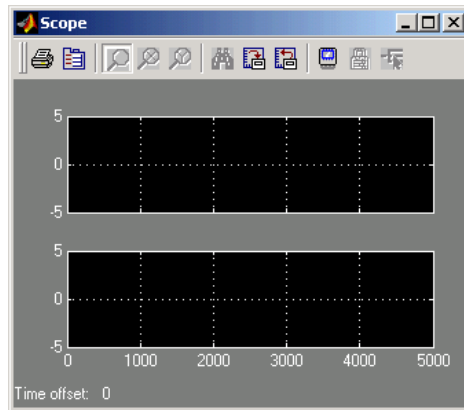
Double-click the Error Rate Calculation block and clear the box next to **Stop simulation** in the block's mask.

Scope Block

The Scope block displays the channel errors and uncorrected errors. To configure the block:

- 1 Double-click the block to open the scope.
- 2 Click the **Parameters** button  on the toolbar.
- 3 Set **Number of axes** to 2.
- 4 Set **Time range** to 5000.
- 5 Click the **Data history** tab.
- 6 Type 30000 in the **Limit data points to last** field.
- 7 Click **OK**.

The scope should now appear as shown.



To configure the axes, follow these steps:

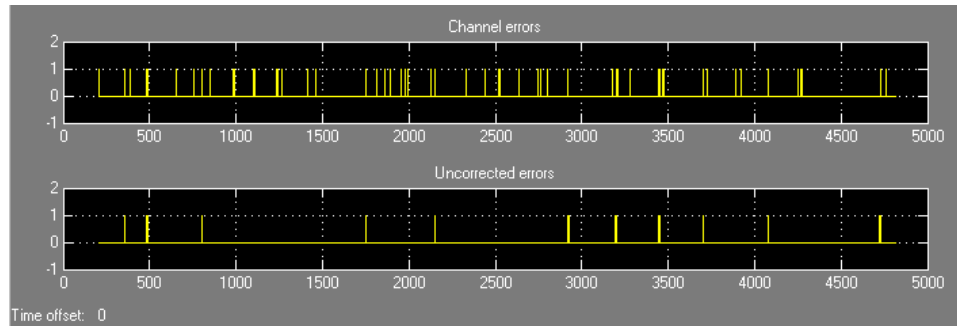
- 1 Right-click the vertical axis at the left side of the upper scope.
- 2 In the context menu, select **Axes properties**.
- 3 In the **Y-min** field type -1.
- 4 In the **Y-max** field type 2.
- 5 Click **OK**.
- 6 Repeat the same steps for the vertical axis of the lower scope.
- 7 Widen the scope window until it is roughly three times as wide as it is high. You can do this by clicking the right border of the window and dragging the border to the right, while pressing the mouse button.

Relational Operator

Set **Relational Operator** to $\sim =$ in the block's mask. The Relational Operator block compares the transmitted signal, coming from the Bernoulli Random Generator block, with the received signal, coming from the Hamming Decoder block. The block outputs a 0 when the two signals agree and a 1 when they disagree.

Observing Channel Errors with the Scope

When you run the model, the Scope block displays the error data. At the end of each five thousand time steps, the scope appears as shown in the following figure. The scope then clears the displayed data and displays the next five thousand data points.

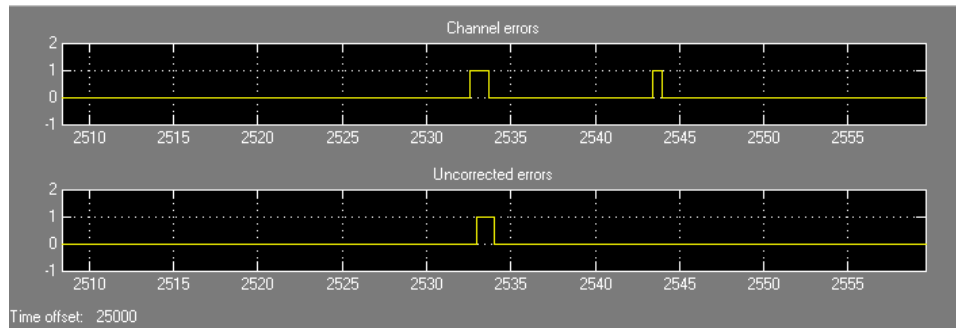


Scope with Model Running

The upper scope shows the channel errors generated by the Binary Symmetric Channel block. The lower scope shows errors that are not corrected by channel coding.

Click the **Stop** button on the toolbar at the top of the model window to stop the scope.

To zoom in on the scope so that you can see individual errors, first click the middle magnifying glass button at the top left of the Scope window. Then click one of the lines in the lower scope. This zooms in horizontally on the line. Continue clicking the lines in the lower scope until the horizontal scale is fine enough to detect individual errors. A typical example of what you might see is shown in the figure below.



Zooming in on the Scope

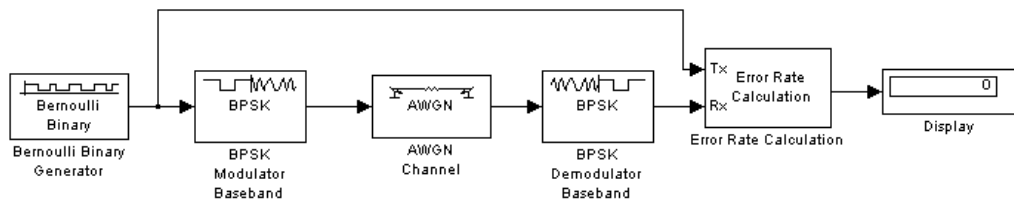
The wider rectangular pulse in the middle of the upper scope represents two 1s. These two errors, which occur in a single code word, are not corrected. This accounts for the uncorrected errors in the lower scope. The narrower rectangular pulse to the right of the upper scope represents a single error, which is corrected.

When you are done observing the errors, selecting **Stop** from the **Simulation** menu.

“Sending Signal and Error Data to the Workspace” on page 3-4 explains how to send the error data to the MATLAB workspace for more detailed analysis.

Modeling a Channel with Modulation

The Binary Symmetric Channel block, which simulates a channel with noise, is useful for building models of channel coding. For other types of applications, you might want to construct a more realistic model of a channel. For example, you can add modulation and demodulation, and replace the Binary Symmetric Channel block with an AWGN Channel block, which adds white Gaussian noise to the channel. This following figure shows an example that uses binary phase shift keying (BPSK).



BPSK Modulation Model

This section explains how to

- Build the model
- Set parameters in the model
- Run the model

The section also explains binary phase shift keying.

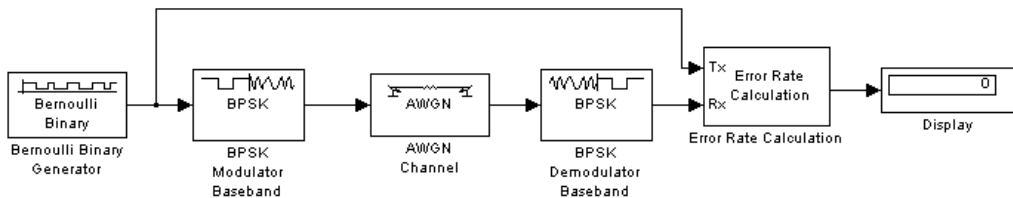
We encourage you to build the model for yourself. Alternatively, to open a completed version of the model, type `bpskdoc` at the MATLAB prompt.

Building the BPSK Model

You can build the BPSK model from the one shown in the figure Channel Noise Model on page 2-27. To build the model, follow these steps:

- 1 Type `channeldoc` at the MATLAB prompt to open the channel noise model, and then save the model as `my_bpsk` in the directory where you keep your work files.

- 2 Delete the Binary Symmetric Channel block from the model by right-clicking the block and selecting **Clear**.
 - 3 Move the following blocks from the Simulink Library Browser into the model window, and insert them into the model as shown in the following figure:
 - BPSK Modulator Baseband block, from PM in the Digital Baseband Modulation sublibrary of the Modulation library
 - AWGN Channel block, from the Channels library
 - BPSK Demodulator Baseband block, from PM in the Digital Baseband Modulation sublibrary of the Modulation library
- The model should now appear as in the figure below.



Binary Phase Shift Keying

The BPSK Modulator and Demodulator Baseband blocks implement binary phase shift keying (BPSK) modulation. BPSK is a method for modulating a binary signal onto a complex waveform by shifting the phase of the complex signal. In digital baseband BPSK, the symbols 0 and 1 are modulated to the complex numbers $\exp(jt)$ and $-\exp(jt)$, respectively, where t is a fixed angle. In this example, $t = 0$, so these numbers are just 1 and -1.

You can set the value of t in the **Phase offset** parameter in the masks for the BPSK Modulator Baseband block and the BPSK Demodulator Baseband block. The default value is 0.

To learn more about the digital modulation features of the Communications Blockset, see “Digital Modulation” in the online Communications Blockset documentation.

Setting Parameters in the BPSK Model

To set block parameters in the BPSK model, do the following:

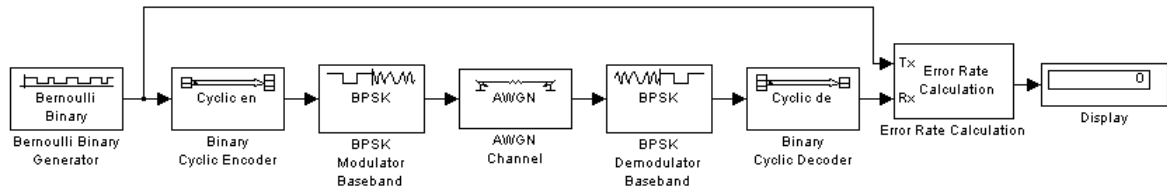
- 1 Double-click the AWGN Channel block and set **Es/No** to 4 . 2.
- 2 Double-click the Error Rate Calculation block and make the following changes to the default parameters in the block's mask:
 - Set **Output data** to **Port**.
 - Check the box next to **Stop simulation**.

Running the BPSK Model

When you run the model, the Display block shows an error rate of approximately 0.01, the same as in the channel noise model. The BPSK model uses the BPSK Modulator Baseband, the AWGN Channel, and the BPSK Demodulator Baseband blocks to simulate a channel with noise. This provides a more realistic model of a channel than using just the Binary Symmetric Channel block. You can also model other types of channel noise using blocks from the Communications Blockset Channels library.

Reducing the Error Rate with a Cyclic Code

You can improve the error rate in the model shown in the figure BPSK Modulation Model on page 2-43 by adding channel coding. An example that uses a binary cyclic code is shown in the following figure.



Cyclic Code Model

This section explains how to

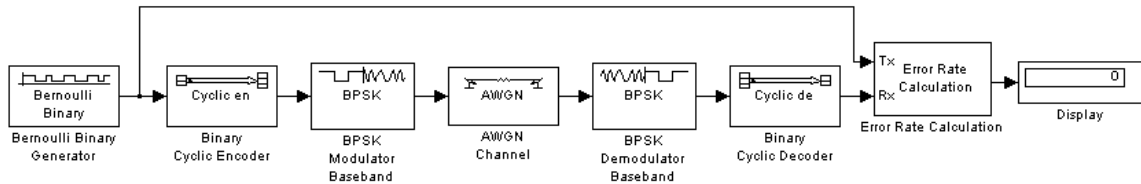
- Build the model
- Set parameters in the model
- Run the model
- Set the symbol period
- Display frame sizes and frame periods

Building the Cyclic Code Model

You can build the cyclic code model by adding blocks to the model shown in the figure BPSK Modulation Model on page 2-43. To do so, follow these steps:

- 1 Type `bpskdoc` at the MATLAB prompt to open the BPSK model. Then save the model as `my_cyclic` in the directory where you keep your work files.
- 2 Drag the following two Communications Blockset blocks from the Simulink Library Browser into the model window:
 - Binary Cyclic Encoder block, from the Block sublibrary of the Error Detection and Correction library
 - Binary Cyclic Decoder block, from the Block sublibrary of the Error Detection and Correction library

3 Widen the model window and connect the blocks as in the following figure.



Binary Cyclic Encoder and Decoder

The Binary Cyclic Encoder block implements a binary cyclic code. The block has the following parameter settings:

- **Codeword length 4**
- **Message length 7**

The code rate is given by

$$\text{Code rate} = \frac{\text{Codeword length}}{\text{Message length}}$$

This example uses a rate 4 / 7 code. The codeword length N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The input to the block must have the same size as the setting for **Message length**.

The Binary Cyclic Decoder block decodes the signal after it passes through the channel. The block should have the same parameter settings as the Binary Cyclic Encoder block.

Setting Parameters in the Cyclic Code Model

Make the following changes to the parameter settings for the BPSK model, described in “Setting Parameters in the BPSK Model” on page 2-45:

- 1 Double-click the Bernoulli Binary Generator block and make the following changes to the default parameters in the block’s mask.
 - Check the box next to **Frame-based outputs**.

- Set **Samples per frame** to 4 to match the input requirement of the Binary Cyclic Encoder Block.
- 2 Double-click the Binary Cyclic Encoder block and set **Codeword Length N** to 7. Make the same change in the Binary Cyclic Decoder block.
 - 3 Double-click the AWGN Channel block and make the following changes to the default parameters in the block's mask.
 - Set **Symbol period** to $4/7$. With this setting, the AWGN channel block produce the same amount of noise per symbol as in the BPSK model without coding. This enables you to evaluate the improvement in bit error rate due to channel coding. For more information on setting **Symbol period**, see "Setting the Symbol Period" on page 2-48.
 - Set **Es/No** to 4.2.

Running the Cyclic Code Model

When you run the model, the bit error rate is approximately one tenth the error rate in the model shown in the figure BPSK Modulation Model on page 2-43, which does not have channel coding.

Double-click the AWGN Channel block and change Es/No to 2.05. Now when you run the model, the bit error rate is approximately the same as that in the BPSK model. This means that the coding gain from the cyclic code at this bit error rate is $4.2 - 2.05 = 2.15$ dB.

Setting the Symbol Period

To compare the cyclic code model to the BPSK model, which does not have channel coding, the ratio of energy per information symbol to noise spectral density, E_b/N_0 , should be the same in both models. You can use the **Symbol period** parameter in the AWGN Channel block to adjust the amount of channel noise so that E_b/N_0 is the same as in the model without coding. Since the cyclic code has rate $4/7$, you set the **Symbol period** to $4/7$.

In general, set the **Symbol period** to k/n if there are k information symbols for each n channel symbols. Both channel coding and modulation with more than two channel symbols require a change to the **Symbol period**.

Note that **Es/No**, the ratio of energy per channel symbol to noise spectral density, is not the same as E_b/N_0 . To convert between the two, use the formula

$$E_b/N_0 = E_s/N_0 + 10\log(k/n)$$

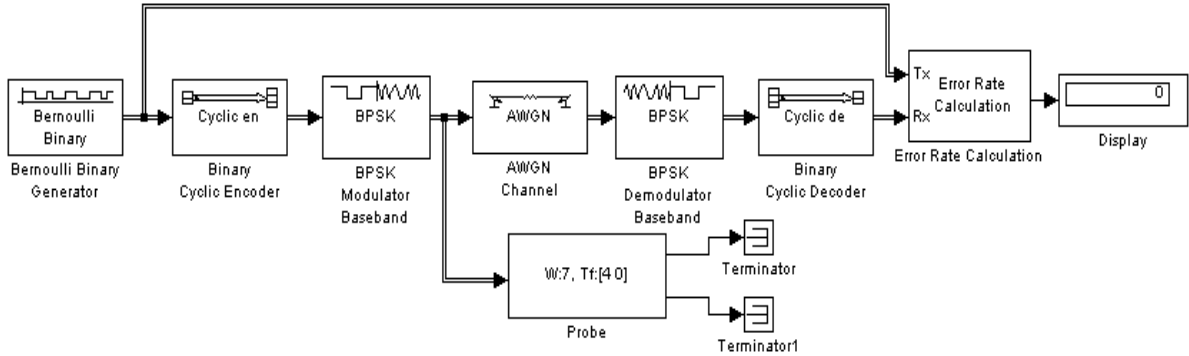
where k/n is the ratio of information symbols to channel symbols. Changing the **Symbol period** to k/n has the same effect as subtracting $10\log k/n$ from the Es/No parameter.

Using a Probe Block to Determine Symbol Period

If you are unsure how to set the **Symbol period**, you can find the correct value using a Probe block. To do so, follow these steps:

- 1 Drag the following blocks from the Simulink Library Browser into the cyclic code model window:
 - Probe block, from the Simulink Signal Attributes library
 - Two copies of the Terminator block, from the Simulink Sinks library
- 2 Double-click the Probe block to display its mask.
- 3 Clear the boxes next to **Probe complex signal** and **Probe signal dimensions** and check the boxes next to **Probe width** and **Probe sample time**. This reduces the number of output ports in the block to two.
- 4 Connect the blocks as shown in the following figure.
- 5 Select **Update diagram** from the **Edit** menu.

The model should now appear as shown in the following diagram.



The number after W in the Probe block tells you the frame size, which in this case is 7. The first number after T_f tells you the frame period, which is 4. You can determine the symbol period by the following formula.

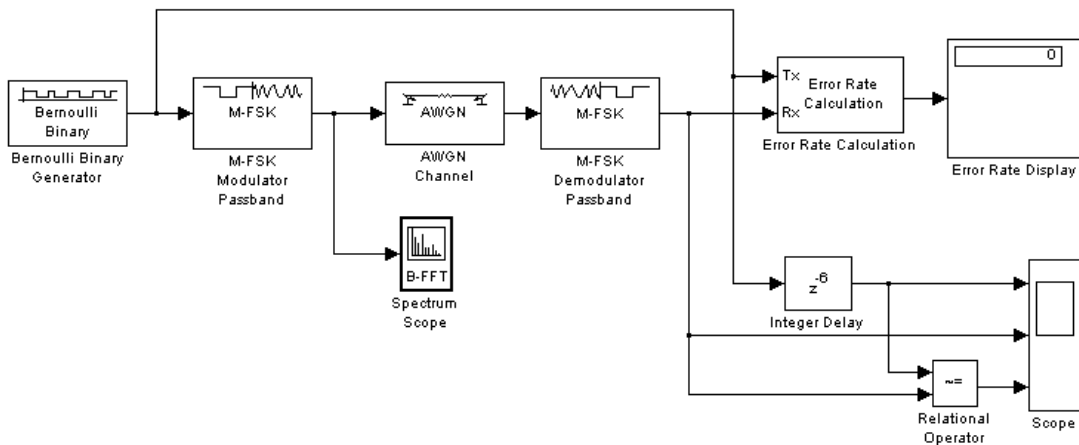
$$\text{symbol period} = \frac{\text{frame period}}{\text{frame size}}$$

In this example, the symbol period is $4 / 7$.

When the signal is sample-based, the second value in the Probe block is the sample time, rather than the frame period.

Building a Frequency-Shift Keying Model

Frequency-shift keying (FSK) is a standard modulation technique in which a digital signal is modulated onto a sinusoidal carrier whose frequency shifts between different values. The Bell Telephone System first used this technique in their Model 103 modem. The model shown in the following figure is an example of passband FSK.



FSK Model

This section explains how to

- Build the model
- Set parameters in the model
- Run the model

The section also explains

- Delays in the model and how to find the delay
- Multirate models
- Using sample time colors to check sample times

To open a completed version of the model, type `fskdoc` at the MATLAB prompt.

Building the FSK Model

You can build the FSK model by adding blocks to the model shown in the figure Channel Noise Model on page 2-27. To open the channel noise model, type `channeldoc` at the MATLAB prompt. Then save the model as `my_fsk` in the directory where you keep your work files. See “Saving a Model” on page 2-24.

You need to add the following blocks to the model.

M-FSK Modulator and Demodulator Passband

The M-FSK Modulator Passband block, from FM in the Digital Passband sublibrary of the Modulation library, modulates the binary signal onto a sinusoidal carrier of 10000 Hz. The block modulates a 0 by shifting the frequency of the carrier up to 10500 Hz, and a 1 by shifting the frequency down to 9500 Hz.

The M-FSK Demodulator Passband block, from FM in the Digital Passband sublibrary of the Modulation library, demodulates the signal.

Relational Operator


The Relational Operator block, from the Simulink Math Operations library, compares the transmitted signal, from the Bernoulli Binary Generator block, with the received signal, from the M-FSK Demodulator Passband block. The block outputs a 0 when the two signals agree, and a 1 when they differ.

Spectrum Scope

The Spectrum Scope block, from the DSP Sinks library, calculates the fast Fourier transform (FFT) of the signal and displays its spectrum. When you run the model, a scope appears, as in the figure FSK Spectrum on page 2-56.

Scope

The Scope block, from the Simulink Sinks library, displays the transmitted signal, the received signal, and the output of the Relational Operator block. To create three input ports for the block, follow these steps:

- 1 Double-click the block to open the scope.
- 2 Click the **Parameters** button  on the toolbar.
- 3 Set **Number of axes** to 3.

4 Set **Time range** to 1.

5 Click **OK**.

To set the limits on the vertical axes:

1 Right-click the vertical axis at the left side of the upper scope.

2 In the context menu, select **Axes properties**.

3 In the **Y-min** field type -1.

4 In the **Y-max** field type 2.

5 Click **OK**.

Repeat these steps for the middle and lower vertical axes.

Integer Delay

The Integer Delay Block, from the DSP Blockset Signal Operations library, delays the transmitted signal so that it can be accurately compared with the received signal. Its purpose is explained further in “Delays in the Model” on page 2-57.

Drag these blocks into the model window and connect them as shown in the figure FSK Model on page 2-51. The next section explains how to set the parameters for these blocks.

Setting Parameters in the FSK Model

Make the following changes to the default parameter settings in the masks for the blocks:

- 1** Double-click the Bernoulli Binary Generator block and make the following changes to the default parameters in the block’s mask:
 - Set **Probability of a zero** to 0.5.
 - Set **Sample time** to 1/1200.
- 2** Double-click the M-FSK Modulator Passband block and make the following changes to the default parameters in the block’s mask:

- Set **M-ary number** to 2. This specifies the number of frequencies in the modulated signal.
 - Set **Frequency separation** to 1000. This specifies the separation between the two frequencies of the modulated signal.
 - Set **Symbol period** to 1/1200 to match the **Sample time** in the Bernoulli Binary Generator block.
 - Set **Baseband samples per symbol** to 5. This causes the block to oversample the incoming signal. Oversampling increases the sampling rate by a factor of 5, thereby raising the sampling rate above the Nyquist rate. This enables the Spectrum Scope block to generate a more accurate picture.
 - Set **Carrier frequency** to 10000. This specifies the frequency of the carrier.
 - Set **Output sample time** to 1/30000.
- 3** Double-click the M-FSK Demodulator Passband block and make the same changes to the block's default parameters as for the M-FSK Modulator Passband block. Set **Input sample time** to 1/30000 to match the **Output sample time** of the M-FSK Modulator Passband block.
- 4** Double-click the AWGN Channel block and set **Symbol period** to 1/1200, to match the **Symbol period** in the M-FSK Modulator Passband block.
- 5** Double-click the Spectrum Scope block and make the following changes to the default parameters in the block's mask:
- Check the box next to **Buffer input**. This converts the input into frames so that the block can calculate the FFT.
 - Set **Buffer size** to 1024.
 - Set **Buffer overlap** to 256.
 - Set **Number of spectral averages** to 20.
 - Check the box next to **Axis properties**.
 - Set **Minimum Y-limit** to -40.
 - Set **Maximum Y-limit** to 20.
- These settings rescale the y-axis.

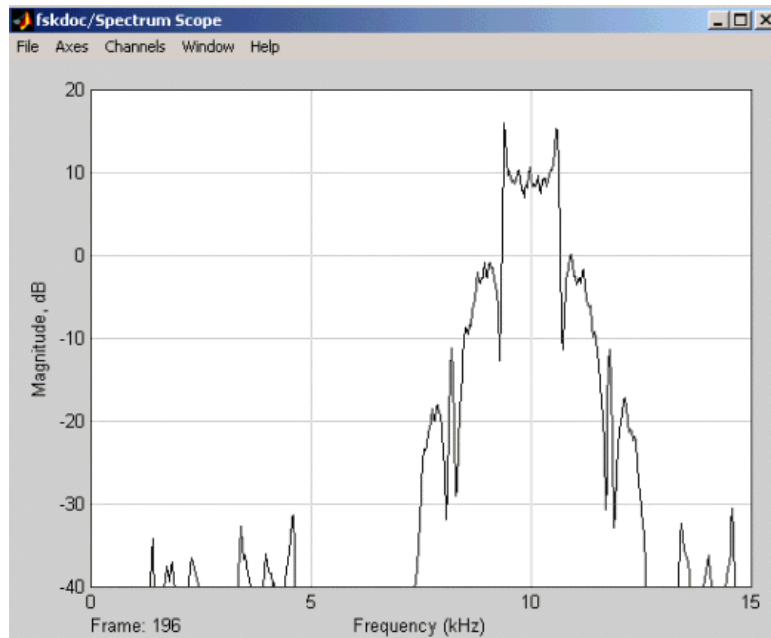
- 6 Double-click the Integer Delay block and change **Delay** to 6 in the block's mask. The Integer Delay block delays the transmitted signal by the number of sample periods specified by the **Delay** parameter. For more information about delays, see the section "Delays in the Model" on page 2-57.
- 7 Double-click the Error Rate Calculation block and make the following changes to the default parameters in the block's mask:
 - Set **Receive delay** to 6.
 - Set **Output data** to **Port**.

Running the FSK Model

Set the **Stop time** parameter to 15 and run the model.

Displaying the Signal's Spectrum in the Spectrum Scope

The Spectrum Scope displays the spectrum of the modulated signal, as shown in the following figure.

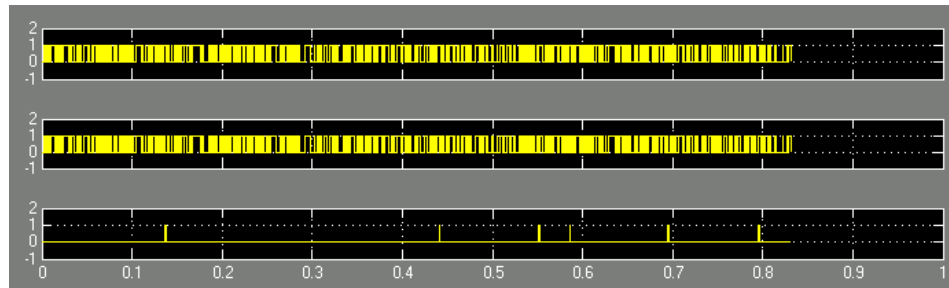


FSK Spectrum

Note the peaks at 9500 Hz and 10500 Hz, representing the integers 1 and 0, respectively. The peaks outside the range 9500 to 10500 Hz are side lobes, which are not relevant to the display.

Displaying the Errors in the Scope

Double-click the Scope block to open the scope, as shown in the following figure.



Displaying Errors in the Scope

The top window displays the transmitted signal. The middle window displays the received signal. The bottom scope displays a 0 where the two signals agree and a 1 where they differ.

Delays in the Model

Due to the way they process data, some blocks cause a signal to be delayed as it passes through a model. For example, there is a delay of 6 symbol periods between the input signal to the M-FSK Modulator Passband block and the output signal from the M-FSK Demodulator Passband block. As a result, there is a delay of 6 symbol periods between the transmitted and received signals in the FSK model.

To calculate the bit error rate correctly, you need to introduce an additional delay of 6 seconds to the transmitted signal to synchronize it with the received signal. You can do this directly in the mask for the Error Rate Calculation block by setting the **Receive delay** to 6.

For the same reason, you need to synchronize the transmitted and received signals before they enter the Relational Operator block. In this case, you must use an Integer Delay block, which delays a signal by the number of sample periods specified by the **Delay** parameter. Set the **Delay** to 6. This is indicated by the exponent -6 on the block. The delay synchronizes the transmitted signal with the received signal so that the Relational Operator block can compare them correctly.

Note If the Error Rate Calculation block in a model gives an error rate close to .5 for a random binary signal, you might not have taken into account delays in the model. The block is probably comparing two unsynchronized signals.

Several blocks in the Communications Blockset Modulation library produce delays. A list of these is given in “Delays in Digital Modulation” in Using the Communications Blockset. The Viterbi Decoder block, from the Convolutional sublibrary of the Error Detection and Correction library, also produces a delay equal to its **Traceback depth** parameter – see the section “Viterbi Decoder” on page 2-64.

Finding the Delay in the Model

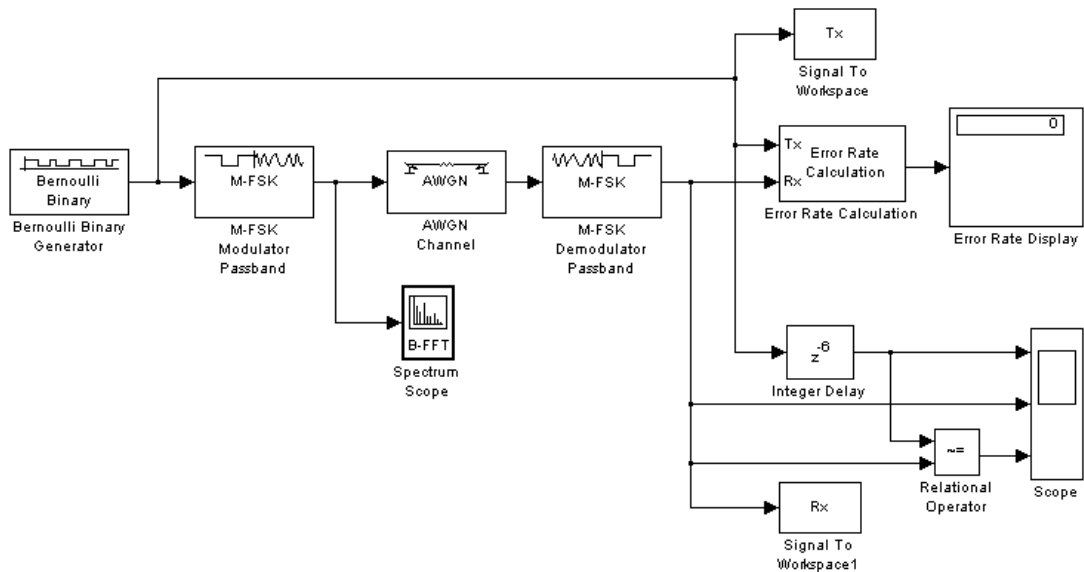
In constructing this model, to set the parameters for the Error Rate Calculation block and Delay block correctly, you need to know the delay between the transmitted and received signals. You can sometimes determine this delay from the parameters of the blocks between the transmitted and received signals. But if you are unable to determine the delay in this way, you can do so with the `xcorr` function, from the Signal Processing Toolbox, which finds the cross correlation between the signal and shifts of its delayed version.

To use the `xcorr` function, you must modify the model slightly as follows:

- 1 Set the **Es/No** parameter of the AWGN Channel block to 100. This essentially removes all noise from the model.
- 2 Drag a Signal To Workspace block, from the DSP Sinks library, into the model window.
- 3 Connect the line leading out of the Bernoulli Binary Generator block to the Signal To Workspace block. You can do this by right-clicking the line and moving the mouse pointer to the input port of the Signal To Workspace block while pressing the mouse button.
- 4 Double-click the Signal To Workspace block to open its mask, and change the **Variable name** parameter to Tx.

- 5 Drag another Signal To Workspace block, from the DSP Sinks library, into the model window and connect it to the line leading out of the M-FSK Demodulator Passband block.
- 6 Double-click the second Signal To Workspace block into the model window and change the **Variable name** parameter to Rx.
- 7 Pull down the **Simulation** menu and select **Simulation parameters**. Set **Stop time** to 1.

When you are done, the model should appear as in the following figure.



Determining the Delay in the FSK Model

Running the model sends the transmitted and received signals to the workspace as vectors called Tx and Rx, respectively. To find the delay between Tx and Rx, type the following commands at the MATLAB prompt.

```
[m, index]=max(xcorr(Rx, Tx));
L=length(Tx);
```

```
delay=index-L
```

MATLAB should return `delay=6`, which is the correct value of the delay.

The `xcorr` function calculates the correlations between Tx and Rx when the two vectors are shifted in all possible ways that overlap. The maximum correlation occurs at the shifted distance corresponding to the true delay.

You can test whether you have found the correct value for the delay by using the MATLAB `isequal` command, which returns a 1 when two vectors are equal, and a 0 when they differ. To do so, at the MATLAB prompt type

```
isequal(Tx(1:L-delay),Rx(delay+1:L))
```

MATLAB returns a 1 if the number of symbols Tx and Rx are offset by `delay`.

If you use this procedure and MATLAB returns a 0 to the `isequal` command, it might indicate that there is still noise present in the model.

Multirate Models

The model shown in the figure FSK Model on page 2-51 differs from the earlier models in that it contains signals with different sample times. The Bernoulli Binary Generator block has a sample time of 1/1200. The M-FSK Modulator Passband block receives this signal and upsamples it at a rate of five samples per symbol. As a result, the sample time of the block's output signal is 1/6000. A model that contains signals with different sample times is called a *multirate* model.

The multiple sample times present in this model do not affect the bit error rate of a simulation. But you should be aware that in other models, sample times can affect the results of a simulation. This is usually the case when different signals are combined. See “Setting Sample Times and Samples per Frame” on page 3-15 for an example of this.

Using Sample Time Colors to Check Sample Times

You can easily check whether there are different sample times in a model by selecting **Sample time colors** from the **Format** menu. Then select **Update diagram** from the **Edit** menu. When you do this, blocks and lines in the model are colored according to their sample times.

Red blocks and lines indicate the fastest sample time in the model. Green indicates the second fastest sample time. Yellow blocks contain signals with

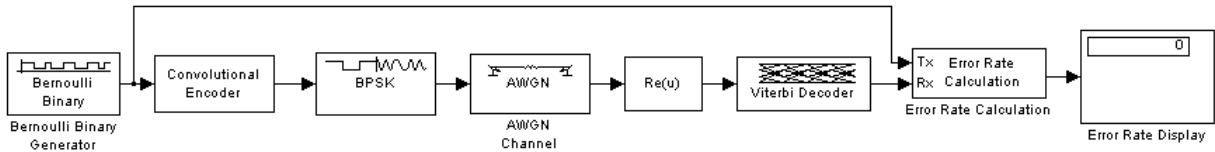
different sample times. If all sample times in the model are the same, all blocks and lines are colored red. For more information on sample time colors, see the Simulink documentation.

For frame-based signals, the colors correspond to the frame periods of the signals rather than sample times.

If you need to determine the actual sample time of a signal, you can use a Probe block as described in “Building a Frequency-Shift Keying Model” on page 2-51.

Building a Convolutional Code Model

The following model simulates the use of convolutional coding to send a signal through a channel with noise.



Convolutional Code Model

This section explains how to

- Build the model
- Set block parameters
- Run the model

It also explains the new blocks in the model.

To open a completed version of the model, type `convdoc` at the MATLAB prompt.

Building the Convolutional Code Model

You can build the convolutional code model by adding blocks to the model shown in the figure Channel Noise Model on page 2-27.

To build the model, follow these steps:

- 1 Type `channeldoc` at the MATLAB Help browser to open the channel noise model. Then save the model as `my_conv` in the directory where you keep your work files.
- 2 Delete the Binary Symmetric Channel block.

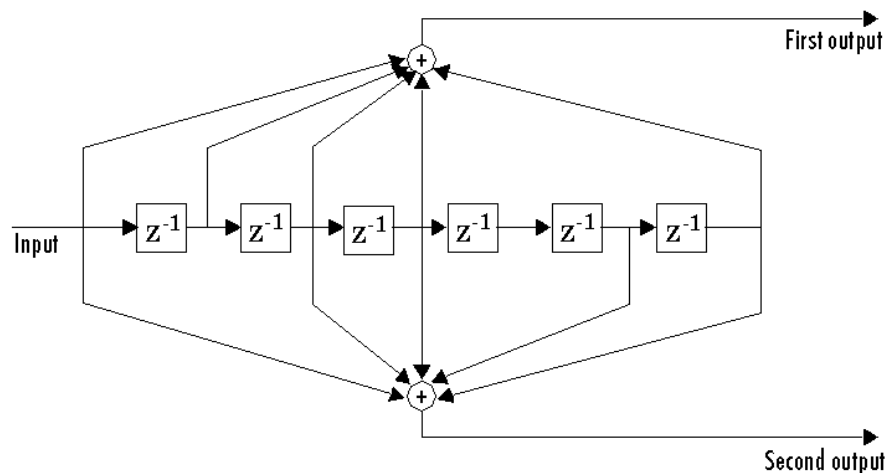
- 3** Drag the following blocks from the Simulink Library Browser into the model window, and connect them as shown in the figure Convolutional Code Model on page 2-62:
- Convolutional Encoder, from the Convolutional sublibrary of the Error Detection and Correction library
 - BPSK Modulator Baseband, from PM in the Digital Baseband Modulation sublibrary of the Modulation library
 - Complex to Real-Imag, from the Simulink Math Operations library
 - Viterbi Decoder, from the Convolutional sublibrary of the Error Detection and Correction library

Blocks in the Model

The model contains the following new blocks.

Convolutional Encoder

The Convolutional Encoder block encodes the signal from the Bernoulli Binary Generator. The example uses the industry standard rate 1/2 convolutional code, with constraint length 7, defined by the following diagram.



Convolutional Encoder Schematic Block Diagram

The encoder structure is described by a pair of binary numbers, having the same length as the code's constraint length, that specify the connections from the delay cells to modulo-2 addition nodes. The binary number for the upper addition node is 1111001. A 1 indicates that the bit in the corresponding delay cell (reading from left to right) is sent to the addition node, and a 0 indicates that the bit is not sent. The binary number for the lower addition node is 1011011. Converting these two binary numbers to octal gives the pair [171,133]. You can enter this pair into the block's mask by typing `poly2trellis(7, [171 133])` in the field for **Trellis Structure**.

To learn more about the convolutional coding features of the Communications Blockset, see "Convolutional Coding" in the online Communications Blockset documentation.

Complex to Real-Imag

The Complex to Real-Imag block, labeled $\text{Re}(u)$, receives the complex signal and outputs its real part. Since the output BPSK Modulator Baseband block has zero complex part, all of the signal is carried by the real part. You can set this option by selecting **Real** in the **Output** parameter field in the mask. It is not necessary to demodulate the signal, because the Viterbi Decoder block can accept unquantized inputs.

Viterbi Decoder

The Viterbi Decoder block decodes the signal using the Viterbi algorithm. The **Decision Type** parameter is set to **Unquantized** so that the block can accept real numbers from the Complex to Real-Imag block. The **Traceback depth** parameter, which is set to 96, is the number of branches in the trellis that the block uses to construct each traceback path. This produces a delay of 96 between the input and output of the block. For more information on delays, see "Finding the Delay in the Model" on page 2-58.

For an example of a convolutional coding model that uses soft-decision decoding, see the section "Example: Soft-Decision Decoding" in the online Communications Blockset documentation.

Setting Parameters in the Convolutional Code Model

To set parameters in the convolutional code model, do the following:

- 1 Double-click the Bernoulli Binary Generator block and check the box next to **Frame-based outputs** in the block's mask.

- 2 Double-click the AWGN Channel block and make the following changes to the default parameters in the block's mask:
 - Set **Es/No** to -1.
 - Set **Symbol period** to 1/2. Since the code rate is 1/2, this setting causes the block to produce the same amount of noise per channel symbol as it would without channel coding. For more information, see "Setting the Symbol Period" on page 2-48.
- 3 Double-click the Error Rate Calculation block and make the following changes to the default parameters in the block's mask:
 - Set **Receive delay** to 96. The Viterbi Decoder block creates a delay of 96, due to its **Traceback depth** setting.
 - Check the box next to **Stop simulation**.
 - Set **Target number of errors** to 100.

Running the Convolutional Code Model

When you run the model, you observe an error rate of approximately .003.

You can compare this model with the one shown in the figure Cyclic Code Model on page 2-46. To do so, change the Es/No parameter in the AWGN channel block to 4.2, so that the two models have the same amount of channel noise. Now when you run the convolutional code model, the error rate is 0.

Using the Communications Blockset with MATLAB

Introduction	3-2
Sending Data to the MATLAB Workspace	3-3
Running Simulations from the Command Line	3-8
Importing Data from the MATLAB Workspace	3-12
Learning More	3-17

Introduction

This chapter describes how to use MATLAB to extend the capabilities of the Communications Blockset. The chapter explains how to run simulations from the command line, and how to run multiple simulations. It also explains how to transfer data between a model and the MATLAB workspace.

This chapter covers the following topics:

- “Sending Data to the MATLAB Workspace” on page 3-3 shows how to send the results of simulations from a model to the MATLAB workspace. The section also explains how you can use MATLAB to analyze the data.
- “Running Simulations from the Command Line” on page 3-8 shows how to run models from the MATLAB command line, and how to run multiple simulations with varying parameters. The section also shows how to plot the results of multiple simulations.
- “Importing Data from the MATLAB Workspace” on page 3-12 shows how to import data from the workspace into a model. This enables you to run simulations on data that you create in the workspace or import from outside MATLAB. You can also create specific error patterns, such as burst errors, and import them into a model, in order to test error correction codes.
- “Learning More” on page 3-17 describes other sources of information about the Communications Blockset.

Sending Data to the MATLAB Workspace

This section explains how to send data from a Simulink model to the MATLAB workspace so you can analyze the results of simulations in greater detail.

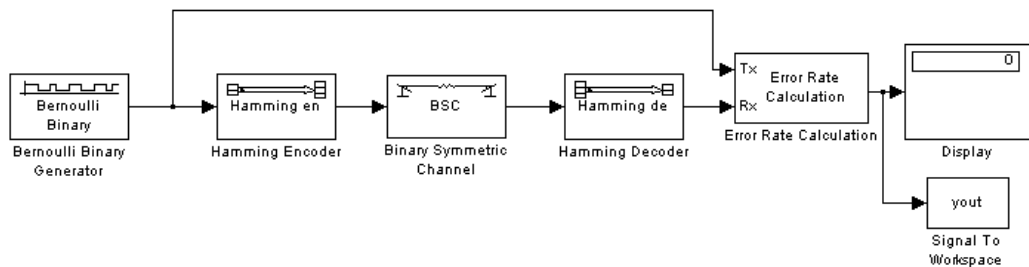
This section explains how to

- Use a Signal To Workspace block
- Configure the Signal To Workspace block
- View the error rate data in the workspace
- Send the signal and error data to the workspace
- View the signal and error data in the workspace
- Analyze the data

Using a Signal To Workspace Block

You can use a Signal To Workspace block, from the DSP Sinks library, to send data to the MATLAB workspace as a vector. For example, you can send the error rate data from the Hamming code model, described in the section “Reducing the Error Rate Using a Hamming Code” on page 2-33. To insert a Signal to Workspace block into the model, follow these steps:

- 1 Type `hammingdoc` at the MATLAB Help browser to open the model.
- 2 Drag a Signal To Workspace block from the DSP Sinks library into the model window and connect it as shown in the following figure.



Hamming Code Model with a Signal To Workspace Block

Configuring the Signal To Workspace Block

To configure the Signal to Workspace block, follow these steps:

- 1 Double-click the block to display its mask.
- 2 Type `hammcode_BER` in the **Variable name** field.
- 3 Type 1 in the **Limit data points to last** field. This limits the output vector to the values at the final time step of the simulation.
- 4 Click **OK**.

When you run a simulation, the model sends the output of the Error Rate Calculation block to the workspace as a vector of size 3, called `hamming_BER`. The entries of this vector are the same as those shown by the Error Rate Display block.

Viewing the Error Rate Data in the Workspace

After running a simulation, you can view the output of the Signal to Workspace block by typing the following commands at the MATLAB prompt.

```
format short e  
hammcode_BER
```

The vector output is the following.

```
hammcode_BER =  
5.4066e-003  1.0000e+002  1.8496e+004
```

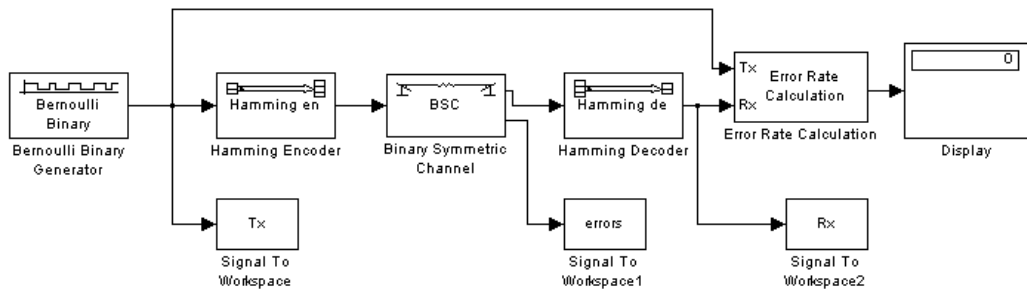
The command `format short e` displays the entries of the vector in exponential form. The entries are as follows:

- The first entry is the error rate.
- The second entry is the total number of errors.
- The third entry is the total number of comparisons made.

Sending Signal and Error Data to the Workspace

To analyze the error correction performance of the Hamming code, you can send the transmitted signal, the received signal, and the error vectors, created

by the Binary Symmetric Channel block, to the workspace. An example of this is shown in the following figure.



Sending Signal and Error Data to the Workspace

You can build this model from the one shown in the figure Hamming Code Model on page 2-33. To build the model, follow these steps:

- 1 Type `hammingdoc` to open the model.
- 2 Double-click the Binary Symmetric Channel block to open its mask, and check the box next to **Output error vector**. This creates an output port for the error data.
- 3 Drag three Signal To Workspace blocks from the DSP Sinks library into the model window and connect them as shown in the preceding figure.
- 4 Double-click the left Signal To Workspace block.
 - Type `Tx` in the **Variable name** field in the block's mask. The block sends the transmitted signal to the workspace as an array called `Tx`.
 - In the **Frames** field, select **Log frames separately (3-D array)**. This preserves each frame as a separate column of the array `Tx`.
 - Click **OK**.
- 5 Double-click the middle Signal To Workspace block:
 - Type `errors` in the **Variable name** field.

- In the **Frames** field, select **Log frames separately (3-D array)**.
 - Click **OK**.
- 6 Double-click the right Signal To Workspace block:
- Type Rx in the **Variable name** field.
 - In the **Frames** field, select **Log frames separately (3-D array)**.
 - Click **OK**.

Viewing the Signal and Error Data in the Workspace

After running a simulation, you can display individual frames of data. For example, to display the tenth frame of Tx, at the MATLAB prompt type

```
Tx(:, :, 10)
```

This returns a column vector of length 4, corresponding to the length of a message word. Usually, you should not type Tx by itself, as this displays the entire transmitted signal, which is very large.

To display the corresponding frame of errors, type

```
errors(:, :, 10)
```

This returns a column vector of length 7, corresponding to the length of a code word.

To display frames one through five of the transmitted signal, type

```
Tx(:, :, 1:5)
```

Analyzing Signal and Error Data

You can use MATLAB to analyze the data from a simulation. For example, to identify the differences between the transmitted and received signals, type

```
diffs=Tx~=Rx;
```

The vector `diffs` is the XOR of the vectors Tx and Rx. A 1 in `diffs` indicates that Tx and Rx differ at that position.

You can determine the indices of frames corresponding to message words that are incorrectly decoded with the following MATLAB command.

```
error_indices=find(diffs);
```

A 1 in the vector `not_equal` indicates that there is at least one difference between the corresponding frame of Tx and Rx. The vector `error_indices` records the indices where Tx and Rx differ. To view the first incorrectly decoded word, type

```
Tx(:, :, error_indices(1))
```

To view the corresponding frame of errors, type

```
errors(:, :, error_indices(1))
```

You can analyze this data to determine the error patterns that lead to incorrect decoding.

Running Simulations from the Command Line

This section describes how to run simulations from the command line using the `sim` command. This is especially useful for running multiple simulations on a model, as described in the next section.

The section explains how to

- Run a single simulation
- Run multiple simulations
- Set parameters in the model
- Plot the results of multiple simulations

Running a Single Simulation

As an example, to run the phase noise model, described in “Running a Simulink Model” on page 2-4, from the command line, type

```
sim('phasenoise_sim')
```

at the MATLAB prompt. This runs the model in the background without opening the model window. While the simulation is running, the MATLAB prompt is unavailable and you cannot enter another MATLAB command.

Note To stop a simulation that you are running from the command line, press **Ctrl+C** on the keyboard when the MATLAB window is active.

After the simulation stops, the prompt reappears. You can then view the results of the simulation by typing `phasenoise_sim` to open the model.

It is not necessary to open the model window when running a simulation from the command line. Usually you want to send the results of the simulation to the MATLAB workspace, for example, if you are running multiple simulations on a single model.

You can also specify simulation parameters from the command line. For example, the command

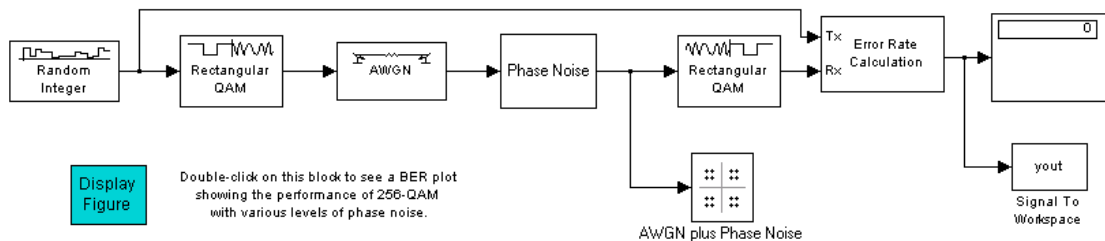
```
sim('phasenoise_sim', 1000);
```

runs the model with a **Stop time** of 1000. This overrides the **Stop time** setting in the **Simulation parameters** dialog box. For more information on running simulations, see the Simulink documentation.

Running Multiple Simulations

You can run multiple simulations, with different parameters, from the command line using a MATLAB script. For example, you might want to run the phase noise model with varying amounts of channel noise. To do this, first type `phasenoise_sim` to open the model.

Drag a Signal To Workspace block from the DSP Sinks library into the model window and connect it as shown in the following figure.



Phase Noise Model with a Signal To Workspace Block

Setting Parameters in the Phase Noise Model

To set parameters in the model, do the following:

- 1 Select **Simulation parameters** in the **Simulation** menu and set **Stop time** to `inf`.
- 2 Double-click the AWGN Channel block and set E_s/N_0 to $E_bNodB + 10 * \log_{10}(8)$ in the block's mask. The variable `E_bNodB` represents bit energy to noise ratio. The term $10 * \log_{10}(8)$ converts `E_sNodB` to E_s/N_0 , which is the symbol energy to noise ratio. The term 8 is present because there are eight bits per channel symbol in 256-QAM.
- 3 Double-click the Phase Noise block and set **Phase noise level (dBc/Hz)** to `-66`.

- 4 Double-click the Error Rate Calculation block and make the following changes to the default parameters in the block's mask:
 - Check the box next to **Stop simulation**.
 - Type 100 in the **Target number of errors** field.
 - Set **Maximum number of symbols** to 10^5 .
- 5 Double-click the Signal To Workspace block and make the following changes to the default parameters in the block's mask:
 - Set **Variable name** to `phase_BER`.
 - Set **Limit data points to last** to 1.
- 6 Type the following command at the MATLAB prompt.

```
EbNoVec=20:.2:21;
```

This creates the vector

```
EbNoVec =  
20.0000  20.2000  20.4000  20.6000  20.8000  21.0000
```

You can now run six simulations, with **Es/No** parameter values given by these vector entries plus $10 \log(8)$. The following script runs the simulations in a loop and stores the results in a matrix called `BER_Vec`. The simulations might take several minutes to run.

```
BER_Vec=[];  
for n=1:length(EbNoVec);  
    EbNodB=EbNoVec(n);  
    sim('phasenoise_sim');  
    BER_Vec(n,:)=phase_BER;  
end;
```

When the simulations have ended and the prompt reappears, type the following at the MATLAB prompt.

```
format short e  
BER_Vec
```

This displays the results in a matrix. Each row is the output of the Error Calculation Block for a single simulation. Notice that the first column, which gives the error rates for the six simulations, decreases. This is because the

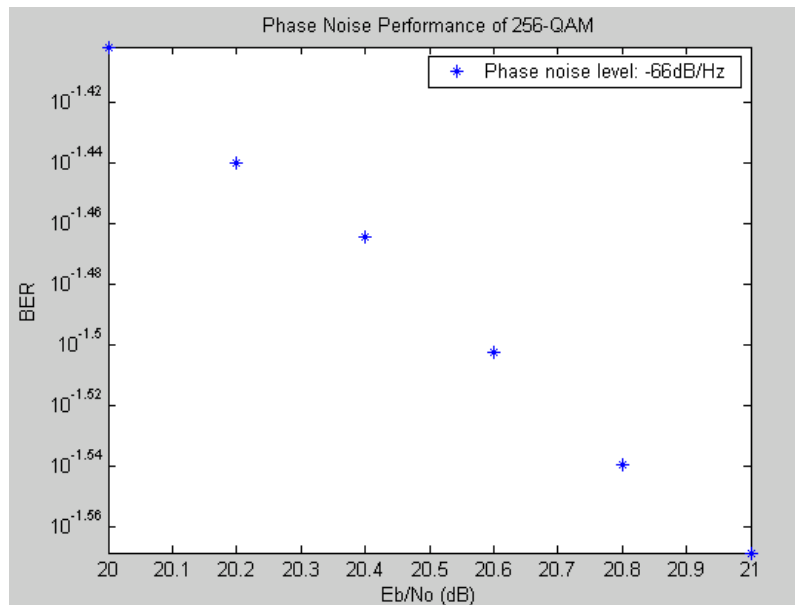
noise level decreases as EbNo dB increases. The last column gives the number of symbols processed in each simulation.

Plotting the Results of Multiple Simulations

You can plot the results of both sets of simulations by entering the following commands.

```
semilogy(EbNoVec,BER_Vec(:,1),'*');
xlabel('Eb/No (dB)');ylabel('BER');
title('Phase Noise Performance of 256-QAM');
legend('Phase noise level: -66 dBc/Hz');
```

The resulting plot is shown in the following figure.



This is a part of the plot shown in “Displaying a Plot of Phase Noise” on page 2-9, corresponding to phase noise of degree 1 and **Eb/No** values between 20 and 21.

Importing Data from the MATLAB Workspace

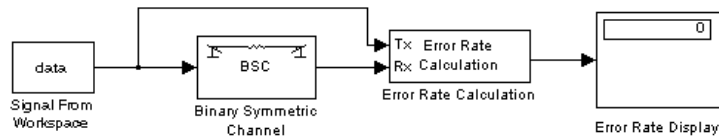
This section explains how to import data into a model directly from the MATLAB workspace using the Signal From Workspace block, from the DSP Sources library. This is useful for simulating different kinds of signals and noise.

The section explains how to

- Simulate a signal by importing data
- Simulate noise by importing data
- Create error patterns in the workspace
- Set sample times and samples-per-frame

Simulating a Signal by Importing Data

To import a signal that you create in the workspace, you can use a Signal From Workspace block as a source. An example is the model shown in the following figure. This model is the same as the one shown in the figure Channel Noise Model on page 2-27, except that the Bernoulli Binary Generator block has been replaced with a Signal From Workspace block.



Importing a Signal from the Workspace

To build this model, follow these steps:

- 1 Type `channe1doc` at the MATLAB prompt to open the channel noise model. Then save it under a different name in the directory where you keep your work files.
- 2 Replace the Bernoulli Binary Generator block with a Signal From Workspace block from the DSP Sources library.

- 3 In the Signal from Workspace block's mask, change the **Signal** parameter to `data` (or another variable name).

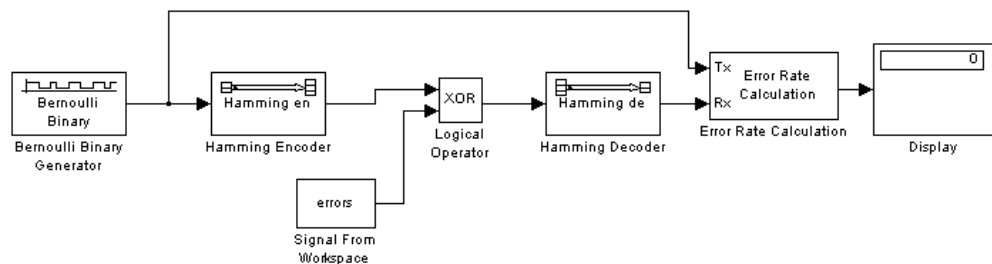
Before using the model, you must define the vector data in the MATLAB workspace. For example, type `data=randint(1,10^4)`; at the MATLAB prompt. This defines `data` as a random binary vector of length 10^4 .

Next, select **Simulation parameters** from the **Simulation** menu and set the **Stop time** parameter to `length(data)`. When you run a simulation, the model imports the random vector data into the model.

You can change the vector data in the MATLAB workspace to simulate a less random signal, or import a signal from outside MATLAB.

Simulating Noise with Imported Data

You can also use the Signal From Workspace block to simulate channel noise with specific error patterns, in order to test the performance of an error correcting code. An example of this is shown in the following figure.



Using Data from the Workspace to Simulate Errors

This example is similar to the model shown in the figure Hamming Code Model on page 2-33, but instead of using the Binary Symmetric Channel block to simulate noise, the model imports error data from the workspace.

To build this model, follow these steps:

- 1 Type `hammingdoc` at the MATLAB prompt to open the Hamming code model. Then save the model under a different name in the directory where you store your work files.
- 2 Delete the Binary Symmetric Channel block from the model.
- 3 Drag the following blocks into the model window:
 - A Signal From Workspace block, from the DSP Sources library
 - A Logical Operator block, from the Simulink Math Operations library
- 4 Connect these blocks as shown in the preceding figure.
- 5 In the **Simulation** menu, select **Simulation Parameters** to open the **Simulation Parameters** dialog box.
- 6 In the **Solver** pane, type `length(errors)` in the **Stop time** field.
- 7 Click **OK**.
- 8 Double-click the Signal From Workspace block and make the following changes to the default parameters in the block's mask:
 - Set **Signal** to `errors`.
 - Set **Sample time** to `4/7`.
 - Set **Samples per frame** to `7`, to match the frame size of the signal coming out of the Hamming Encoder block.
- 9 Double-click the Logical Operator block and set **Operator** to **XOR** in the block's mask.

Simulating Noise with Specified Error Patterns

To use the model in the figure Using Data from the Workspace to Simulate Errors on page 3-13, you must first create a binary vector called `errors` in the workspace to represent errors in the channel. A 1 in the vector represents an error in the channel, while a 0 represents no error. When you run a simulation, the Logical Operator block performs the XOR operation on the vector `errors` and the signal.

For example, to create a vector of length 7×10^4 that contains exactly one 1 in each sequence of entries from $7n + 1$ to $7(n + 1)$, enter the following at the MATLAB prompt.

```
errors=[];
for n=1:10^4
    p=randperm(7);
    v=[1 0 0 0 0 0 0];
    errors=[errors v(p)];
end
```

The function `randperm` generates a random permutation of the numbers one through seven. The vector `v(p)` applies the permutation to the entries of the vector `v`, which has exactly one entry that is 1. The result is that the vector `errors` contains exactly one entry that is 1 in each sequence from $7n + 1$ to $7(n+1)$.

Running a Simulation with Imported Error Data

When you run a simulation, the bit error rate is zero because the Hamming code can correct one error in each code word.

To test the code with an error vector that creates two errors in each code word, just change the vector `v` to `v=[1 1 0 0 0 0 0]` in the preceding code.

Setting Sample Times and Samples per Frame

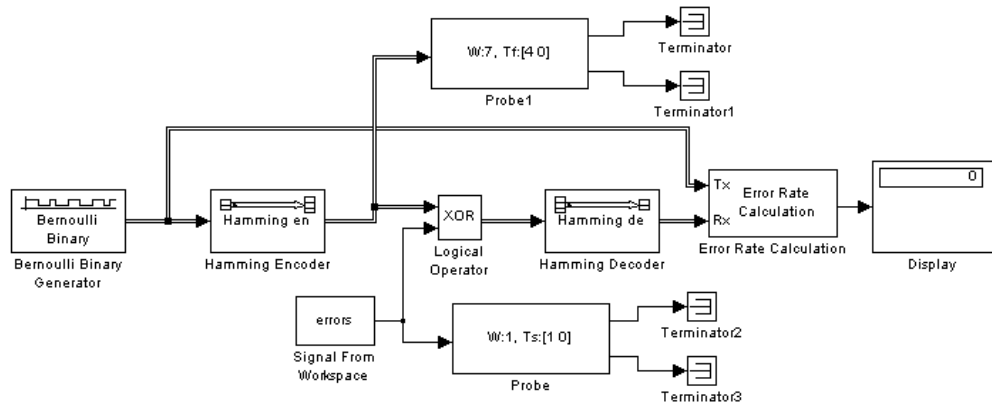
It is important to set the **Sample time** and **Samples per frame** parameters correctly in the Signal From Workspace block, so that the block has the same frame size and frame period as the Hamming Encoder block. This ensures that errors coming from the Signal From Workspace block are synchronized with channel symbols coming from the Hamming Encoder block. To determine the correct sample time, use the following relationship.

$$\text{Sample time} = \frac{\text{Frame period}}{\text{Samples per frame}}$$

The frame size of the signal coming from the Hamming Encoder block is 7. You set the frame size by the **Codeword length** parameter in the block's mask. So you should set the **Samples per frame** parameter in the Signal From Workspace block to 7.

The frame period of the Hamming Encoder block is 4, because the Bernoulli Binary Generator block has a **Sample time** of 1, and a **Samples per frame** of 4. So you should set the **Sample time** parameter in the Signal From Workspace block to 4/7 so that the frame period of the block is 4.

If you are not sure what the frame sizes and frame periods of signals in the model are, you can display them using a Probe block, as described in “Building a Frequency-Shift Keying Model” on page 2-51. To do this, attach two Probe blocks, one to the line leading out of the Hamming Encoder block and one to the line leading out of the Signal From Workspace block. In the masks for both Probe blocks, clear the boxes next to **Probe complex signal** and **Probe signal dimensions**. Next, attach Terminator blocks to the output ports of the Probe blocks. Then select **Update diagram** from the **Edit** menu. The model should appear as in the following figure.



Hamming Code Model with Probe Blocks

Both Probe blocks should display a frame size of 7 and a frame period of 4.

A quick way to check whether the frame periods of two signals are the same is to select **Sample time colors** from the **Format** menu. See “Using Sample Time Colors to Check Sample Times” on page 2-60.

Learning More

You can learn more about the Communications Blockset from the following sources.

Online Help

To find online documentation, select **Full Product Family Help** from the **Help** menu in the MATLAB desktop. This launches the Help browser. For a more detailed explanation of any of the topics covered in this book, see the documentation listed under Communications Blockset in the left pane of the Help browser.

Besides this book, *Getting Started with the Communications Blockset*, the online documentation contains the following topics:

- Using the Libraries describes each of the core libraries of the blockset.
- Modeling Communication Systems illustrates techniques for modeling a full communication system.
- Blocks - By Category—Provides descriptions of the Communications Blockset libraries and lists the blocks in them
- Blocks - Alphabetical List—Provides a detailed description of the blocks in the Communications Blockset in alphabetical order.

The Help Navigator, in the left pane of the Help browser, supports string searches. You can specify strings and the online manuals that you want to search. To begin a search, click the **Search** tab. To view the index for the documentation, click the **Index** tab.

Demos

To see more Communications Blockset examples, type

```
demo
```

at the MATLAB prompt. This opens the MATLAB Demo window. Double-click **Blocksets** and then select **Communications** to list the available demos.

The MathWorks Online

To read the documentation for the Communications Blockset on the MathWorks Web site, point your Web browser to

`www.mathworks.com/access/helpdesk/help/helpdesk.shtml`

A

adding noise to models 2-21

B

binary phase shift keying (BPSK) 2-44

block libraries 2-14

block masks 2-8

block parameters 2-8

blocks

connecting 2-16

labeling 2-36

moving into model 2-15

BPSK modulation example 2-43

building models 2-12

C

channel coding 2-27

channel noise example 2-27

commstartup 2-12

connecting blocks 2-16

constellation 2-6

continuous signals 2-25

convolutional code example 2-62

convolutional encoder 2-63

cyclic code example 2-46

D

delays 2-57

calculating 2-58

demos 2-10

discrete signals 2-25

drawing a branch line 2-30

E

error rate

displaying 2-7

errors

displaying in a scope 2-38

simulating by importing data 3-14

F

frame-based processing 2-25

frames 2-25

displaying sizes of 2-37

frequency-shift keying (FSK) 2-51

FSK model 2-51

H

Hamming code example 2-33

Help Navigator 3-17

I

importing data from MATLAB workspace 3-12

L

labeling

blocks 2-36

learning more 3-17

Library Browser 2-14

M

MathWorks Web site 3-18

models

building 2-12

multirate 2-60

- running 2-20
- saving 2-24
- moving blocks into a model 2-15
- multirate models 2-60

O

- online help 3-17

P

- plotting results of multiple simulations 3-11

Q

- quadrature amplitude modulation (QAM) 2-6

R

- running simulations 2-6

S

- sample time colors 2-60
- sample times 2-25
 - setting 3-15
- sample-based processing 2-25
- samples per frame 2-25
 - setting 3-15
- saving models 2-24
- sending data to MATLAB workspace 3-3
- signals
 - continuous 2-25
 - discrete 2-25
 - displaying in a scope 2-21
 - simulating by importing data 3-12
- simulation parameters 2-19
- simulations

- plotting results of 3-11
- running 2-6
 - running from command line 3-8
- Simulink libraries 2-14
- Simulink Library Browser 2-14
- symbol periods 2-48