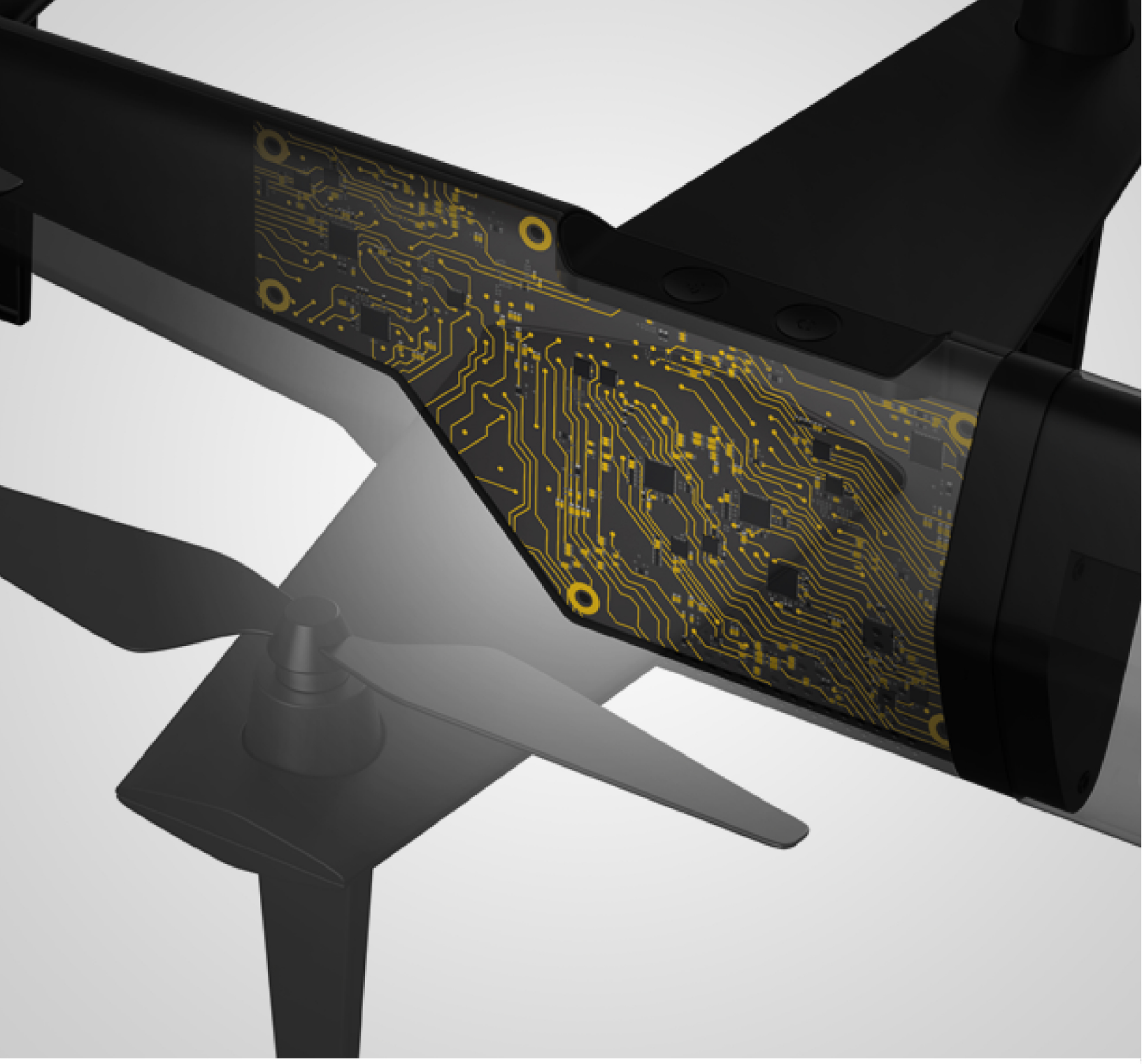


Altair Embed®

Altair Embed/Comm 2022.1
User Guide

ALTAIR

qBeam, Inc.

Altair Embed/Comm User's Guide - Version 2022

Altair Embed/Comm is an add-on from a third party designed to work with Altair Embed™

qBeam, Inc.
19490 Sandridge Way, Ste 330
Leesburg, VA 20176

Contents

Preface

Welcome to Altair Embed/Comm™.

Embed/Comm is a Windows® program for system level modeling and simulation of end-to-end communication systems at the physical layer. It provides fast and accurate solutions for analog, digital and mixed-mode communication system designs. Engineers at leading comm equipment manufacturers are using Embed/Comm to decrease design cycle time, minimize hardware prototyping, and build better products.

Embed/Comm includes an extensive library of modulators, demodulators, channel models, filters, phase locked loop (PLL) elements, distortion-true RF blocks, and forward error correction (FEC) blocks to name a few. Visualization features include time domain plots, strip charts, spectrum plots, eye diagrams, phase scatter plots, bit error rate curves, and other communications related outputs.

Embed/Comm supports the use of complex envelope notation, also known as complex baseband representation. This allows users to use *lowpass equivalent* models and significantly reduce the computational complexity required to support most communication analysis problems. This topic is further discussed in detail in Chapter 2, *Using Embed/Comm*.

This manual assumes that you are already familiar with the use of the Altair Embed SE simulation environment. Please consult the *Embed SE User's Guide* for more details on the core Embed SE environment.

Embed/Comm documentation

To help you get the most out of Embed/Comm, the following information is available:

- **Program help.** Extensive information is provided for the Embed SE simulation environment from the Help menu. For specific information on Embed/Comm and its blocks, use the Embed/Comm Help command under the Help menu. Help for a particular block can also be accessed by clicking on the Help button within the Setup dialog for each block.

You may also find it helpful to browse through the sample block diagrams included with Embed/Comm. These diagrams, which are listed in Appendix B, "Sample Block Diagrams," demonstrate how Embed/Comm can be applied to a variety of communication system problems.

Conventions used in this manual

This manual assumes that you are already familiar with the Embed SE graphical user interface. If you need to review the interface, consult your *Embed SE User's Guide.*

The following conventions are used in this manual:

- Block descriptions are arranged alphabetically within each category. Block categories are presented in alphabetical order.
- Unless specifically stated otherwise, use the left mouse button whenever you are choosing a command, selecting a block, or activating a dialog box parameter. For example, when you read *press the OK button...,* you are to position the pointer over the specified object and click the left mouse button.
- To choose a menu command, you can use the mouse or you can press a sequence of keyboard keys. Only the mouse operations are documented.

The following visual conventions are used to make this manual easier to read:

Visual convention	Where it's used
Italics	To reference a book, chapter, or section. Also used to emphasize certain keywords.
SMALL CAPS	To indicate the names of keys on the keyboard.
Shortcut key combinations	Shortcut key combinations are joined with the plus sign (+). For example, the command CTRL+C means to hold down the CTRL key while you press the C key.
ALL CAPS	To indicate directory names, filenames, and acronyms.
Initial Caps	To indicate menu names, command names, and dialog box options.

Technical support service

Altair technical support teams consist of engineers and IT specialists who are very familiar with the Embed product line. With this knowledge, they are able to provide assistance and advice for most problems you may encounter.

You can receive technical support over the telephone or via email. Typically, technical support provides help in solving specific problems, rather than providing training on how to use Embed SE or Embed /Comm.

For	Do this
Telephone support	Click on the following URL for telephone numbers of our regional offices: http://www.altairhyperworks.com/ClientCenterHWSupportProduct.aspx
Email support	Send an email to embed-support@solidthinking.com and include the following: • Briefly describe your issue • Attach the diagram in question • Specify the version of the software that you are using
Secondary Email support	Click on the following URL: https://solidthinking.com/Page.aspx?category=Community&item=Need%20Help

Altair also provides an online Forum where you can post questions or search for answers to commonly asked questions. To access the Forum, click on http://forum.altairhyperworks.com/index.php?/forum/69-solidthinking-embed/ .

Chapter 1

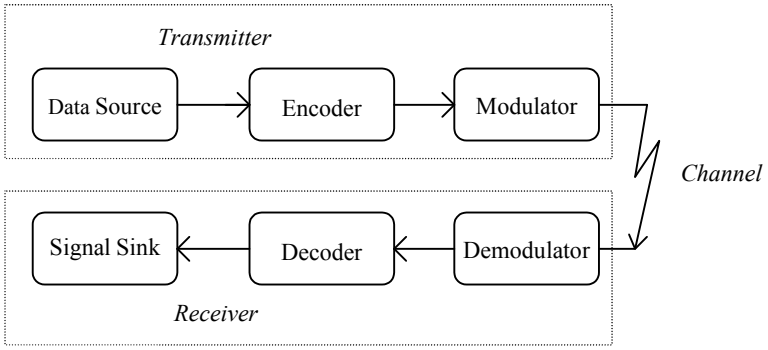
Introduction

This chapter provides information on:

- Key elements of a communication system
- Lowpass equivalent systems
- Summary of communication blocks

A typical communication system

A typical communication link includes, at a minimum, three key elements: a transmitter, a communication medium or channel, and a receiver. The ability to simulate all three of these elements is required to successfully model any end-to-end communication system.



The transmitter and receiver elements can in turn be subdivided into further sub-systems, as shown in the preceding figure. These include a *data source* (analog or digital), an optional *data encoder*, a *modulator*, a *demodulator*, an optional *data decoder*, and a *signal sink*.

Data source

The data source generates the information signal that is intended to be sent to a particular receiver. This signal can be either an analog signal such as speech, or a digital signal such as a binary data sequence. This signal is typically a baseband signal represented by a voltage level.

Encoder

For analog signals, it is often desirable to digitally encode the signal prior to transmission by undergoing a quantization process. This step converts the analog signal into a digital signal. While some information is lost in this process, the resulting digital signal is often far less susceptible to the effects of noise in the transmission channel.

An encoder can also be used to add redundancy to a digital data stream, in the form of additional data bits, in a way that provides an error correction capability at the receiver. This overall process is referred to as *forward error correction* (FEC). Among the most popular FEC schemes are

convolutional coding, block coding, and trellis coding. It is important to note that usually the output bit rate of an encoder is not equal to the input bit rate. To properly distinguish between the two bit rates, the transmitter's input rate is referred to as the information data rate, while the transmitter output rate is referred to as the channel data rate.

Modulator

Depending on the type of information signal and the particular transmission medium, different modulation techniques are employed. Modulation refers to the specific technique used to represent the information signal as it is physically transmitted to the receiver. For example, in *amplitude modulation* (AM), the information is represented by amplitude variations of the carrier signal.

Channel

Once the signal is modulated, it is sent through a transmission medium, also known as a channel, to reach the intended receiver. This may be a copper wire, coax cable, or the atmosphere in the case of a radio transmission. To some extent, all channels introduce some form of distortion to the original signal. Many different channel models have been developed to mathematically represent such distortions. A commonly used channel model is the *additive white Gaussian noise* (AWGN) channel. In this channel, noise with uniform power spectral density (hence the term *white*) is assumed to be added to the information signal. Other types of channels include fading channels and multipath channels.

Demodulator

When the transmitted signal reaches the intended receiver, it undergoes a demodulation process. This step is the opposite of modulation and refers to the process required to extract the original information signal from the modulated signal. Demodulation also includes all steps associated with signal synchronization, such as the use of phase-locked loops in achieving phase coherence between the incoming signal and the receiver's local oscillator.

Data decoder

When data encoding is included at the transmitter, a data decoding step must be performed prior to recovering the original data signal. The signal decoding process is usually more complicated than the encoding process and can be very computationally intensive. Efficient decoding schemes, however, have been developed over the years—one example is the Viterbi decoding algorithm, which is used to decode convolutionally encoded data.

Signal sink

Finally, an estimate of the original signal is produced at the output of the receiver. The receiver's output port is sometimes referred to as the signal sink. As a communications engineer, you are usually interested in knowing how well the source information was recreated at the receiver's output. Several metrics can be used to evaluate the success of the data transmission. The most common metric, in the case of digital signals, is the received *bit error rate* (BER). Other valuable performance indicators include the received *signal to noise ratio* (SNR), eye patterns, and phase scatter plots.

Lowpass equivalent systems

The sampling requirements of a given simulation can be reduced through the use of *complex envelope notation*. When a data signal is modulated, a sampling rate of at least eight samples per carrier cycle is usually selected in order for the simulation to accurately represent the carrier signal. This means that the simulation step size is dictated by the carrier frequency and not the data rate, which may be several orders of magnitude lower. When a simulation is transformed to a lowpass (or baseband) equivalent, its carrier frequency is mathematically translated (shifted) to 0

Hz, and the simulation sampling interval can then based on the sampling requirements of the data signal. The result is a reduction in execution time without loss of simulation accuracy. A brief description follows.

A modulated bandpass signal $s(t)$ usually occupies a narrow band of frequencies near the carrier ω_c, and can be represented as

$$s(t) = a(t)\cos[\omega_c t + \phi(t)] \tag{2.1}$$

where $a(t)$ represents the amplitude and $\phi(t)$ the phase of $s(t)$. After expanding, the above equation becomes

$$\begin{aligned} s(t) &= a(t)\cos\phi(t)\cos\omega_c t - a(t)\sin\phi(t)\sin\omega_c t \\ s(t) &= x(t)\cos\omega_c t - y(t)\sin\omega_c t \\ x(t) &= a(t)\cos\phi(t) \\ y(t) &= a(t)\sin\phi(t) \end{aligned} \tag{2.2}$$

where $x(t)$ and $y(t)$ are respectively the *inphase* and *quadrature* components of $s(t)$. Using complex exponentials, you can also write (2.1) as

$$s(t) = \mathrm{Re}[a(t)e^{j(\omega_c t+\phi(t))}] = \mathrm{Re}[a(t)e^{j\phi(t)}e^{j\omega_c t}]$$

and

$$\begin{aligned} s(t) &= \mathrm{Re}[u(t)e^{j\omega_c t}] \\ u(t) &= a(t)e^{j\phi(t)} = x(t) + jy(t) \end{aligned} \tag{2.3}$$

The signal $u(t)$ above is defined as the complex envelope of $s(t)$. Note that $u(t)$ and $s(t)$ are directly related by a simple frequency translation. When simulating linear systems, it can be shown that as long as the bandwidth B of the complex envelope signal is much smaller than the carrier frequency, that is $B << \omega_c$, then the signal $s(t)$ may be equivalently represented by $u(t)$. The primary advantage of using $u(t)$ is that a much lower simulation sampling rate can be used. For more information on lowpass equivalent signals, see *Digital Communications* (J. Proakis, McGraw-Hill, 1989).

There are some instances when the use of complex baseband notation is not recommended. One instance is when wanting to model the effects of inter-modulation (IM) distortion. IM products are related to the absolute frequency of the carrier, and thus it's recommended that such systems be simulated at the intended operating frequency, or a uniformly scaled-down frequency (i.e. all frequencies in the system are scaled by a common factor as opposed to being translated).

Communication blocks

To allow you to easily simulate a communication link, Altair Embed/Comm provides a vast selection of communication elements, called Comm blocks. These blocks free you from the task of building such elements using the standard block set and allow you to concentrate on modeling systems at a higher hierarchical level.

Below is a brief summary of the blocks available within each category. Blocks marked with a single asterisk (*) are not available in certain versions of the software, and those marked with a double asterisk (**) have limited capability.

Channels

`AWGN (Complex & Real)`; `Binary Symmetric Channel`; `Jakes Mobile*`; `Mobile Fading Channel*`; `Multipath`; `Propagation Loss*`; `Rayleigh/Rice Fading`; `Rummler Multipath`; `Saleh-Valenzuela (Complex & Real)*`; `TWTA`; `Vector AWGN`.

Complex Math

Addition; Conjugate; Division; Inverse; Multiplication; Power; Square Root; Complex to Mag/Phase; Complex to Real/Imag; Mag/Phase to Complex; Real/Imag to Complex.

Decoders and Encoders

Block Interleaver; Convolutional Encoder; Convolutional Interleaver*; Depuncture*; Gray Decoder; Gray Encoder; Hamming Decoder; Hamming Encoder; Manchester Encoder; Puncture*; Reed-Solomon Decoder; Reed-Solomon Encoder; Trellis Decoder*; Trellis Encoder*; Viterbi Decoder (Hard); Viterbi Decoder (Soft).

Demodulators

DPSK Detector**; FM Demodulator; IQ Detector*; PPM Demodulator; PSK Detector**; QAM/PAM Detector**.

Digital Elements

Accumulate & Dump; Binary Counter; Bits to Symbol; Buffer, Divide by N; D Flip Flop; JK Flip Flop; Mux/Demux*; Packet Timing*; Parallel to Serial; Pulse Extend; Queue*; Serial to Parallel; State Machine*; Symbol to Bits; Toggle; Unbuffer.

Estimators

Average Power (Complex & Real); BER Control (#errors); BER Curve Control; Bit/Symbol Error Rate; Complex Correlation*; Correlation*; Delay Estimator; Event Time; File Correlation (Complex & Real)*; Frequency Counter; Mean; Median*; MinMax; Variance; Vector Correlation*; Weighted Mean*.

Filters

Adaptive Equalizer (Complex & Real); Discrete Equalizer (Complex & Real)*; File FIR; FIR; IIR; MagPhase*; Pulse Shaping Filter; Sampled FIR*; Sampled File FIR*; Variable Spaced Equalizer (Complex & Real)*.

Fixed Point

Fixed Point FIR; Fixed Point IIR; Fixed Point VCO (Complex & Real).

Instruments

BER Curve Display; Oscilloscope Display; Spectrum Analyzer Display (Complex & Real).

Modulators (Complex and Real)

AM; ASK; Differential PSK**; FM; FSK; IQ*; MSK; PM; PPM (Real only); PSK**; QAM/PAM**; SQPSK.

Multirate Support

Clock Edge*; Clock Extend*; Interpolator*.

Operators

A/D Converter; Compander; Complex Exponential; Complex FFT,IFFT; Conversions; D/A Converter; Delay (Complex); Delay (Real); Gain (dB); Integrate & Dump (Complex); Integrate & Dump (Real); I/Q Mapper*; Max Index; Modulo; Oscilloscope (Core); Phase Rotate; Phase Unwrap*; Polynomial; Subsample; Spectrum (Complex); Spectrum (Real); Vector FFT.

Phase-Locked Loops

Charge Pump; Loop Filter (2nd Order PLL); Loop Filter (3rd Order PLL)*; Type-2 Phase Detector; Type-3 Phase/Freq Detector; Type-4 Phase/Freq Detector*.

RF Elements

Amplifier; Antenna*; Attenuator; Cable*; Coupler; Double Balanced Mixer*; Feko Far Field; RF Conversions; RF Gain; Splitter/Combiner; Switch; Variable Attenuator*.

Signal Sinks

File Write; Final Value; Wave Write.

Signal Sources

Complex Tone; File Data; Frequency Sweep; Impulse; Impulse Train; Noise; PN Sequence; Poisson Arrivals*; Random Distribution; Rectangular Pulses; Random Seed; Random Signals; Sinusoid; Spectral Mask*; VCO (Complex); VCO (Real); Vector Constant; Walsh Sequence*; Wave Data; Waveform Generator.

Turbo Codes

LTE Turbo Decoder*; LTE Turbo Encoder*; TC Interleaver Generator*; Turbo Code Decoder*; Turbo Code Encoder*; UMTS Turbo Decoder*; UMTS Turbo Encoder*.

Vector Operators

Matrix to Vector; SubVector; Vector Bits to Symbols; Vector Demux; Vector Mux; Vector Merge; Vector Symbols to Bits; Vector to Matrix.

In addition, using the `plot` block (Blocks/Signal Consumer/Plot), you can view simulation results through BER curves, eye diagrams, spectral plots, and phase scatter plots.

Note: Some elements described in this manual are pre-configured compound blocks that perform more complex communication functions. These compound blocks (described in Appendix A) and are typically read-only but can be modified by first creating a new copy and saving under a new name.

Chapter 2

Embed/Comm Overview

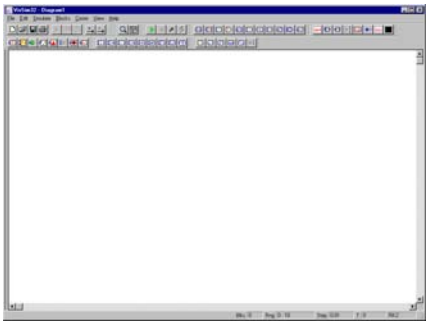
This chapter provides information on:

- Starting Embed/Comm
- Learning your way around the Embed/Comm simulation environment
- Generating BER curves
- Running a sample communication system

Starting Embed/Comm

To start Embed/Comm from the desktop

1. Click on the Start button and choose Programs; then choose Altair Embed/Comm.
2. Embed/Comm starts up and displays the following window:



The Embed/Comm environment

The Embed/Comm environment provides a user-friendly interface for creating and simulating communication system models. Its intent is to allow you to rapidly prototype system designs and carry out performance trade-offs.

The Comm block set, which is summarized in Chapter 1, “The Basics,” was developed using efficient algorithms and coding techniques to provide a fast simulation capability. Individual block parameters were selected to provide sufficient flexibility to meet most modeling situations. In

addition, features such as complex envelope representation were incorporated to allow the use of lowpass equivalent models and the associated savings in diagram execution speed.

Inserting blocks

All communication blocks are accessed via the Comm menu located on the menu bar. All other blocks are listed under the Blocks menu.

♋ To insert a block into a diagram

1. Click on the Comm or Blocks menu and select a particular block category.
2. Click on the desired block.
3. Position the pointer where you want the block to appear in the diagram and click the mouse.

As the mouse passes over a block, either in the menu or diagram, a brief description of it is presented in the status bar at the bottom of the window.

Connecting blocks

Connecting blocks is very easy. Just point to a block's connector tab and hold down the mouse button. The pointer changes into an upward pointing arrow. Then drag the pointer to the destination block's connector tab and then release the mouse button.

Block connectors

There are two types of block connectors: scalar and vector. Vector connectors are colored green in Normal view mode and appear with a thicker connector stub in Presentation mode. Vector connectors are used by many Comm blocks to identify complex signals or vector data.

Embed/Comm will only allow a connection between connectors of the same type.

Optional connectors

The connector count of all Comm blocks is usually fixed, with the exception of connectors identified as *optional* in the block descriptions. In such cases, these optional connectors may be added or deleted using the Add Connector and Remove Connector commands in the Edit menu or toolbar.

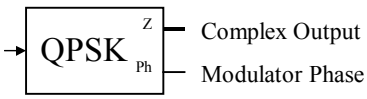
Unconnected connectors

It is not uncommon for several connectors to remain unused within a simulation. For example, many blocks with a clock input can also operate with an internal clock.

Embed/Comm will issue a warning when it detects unconnected block inputs. To disable the warning message, de-activate the Check Connections options in the Preferences property sheet for the Simulate menu's Simulation Properties command.

Connector labels

To more easily identify each connector, connector labels are displayed for most Comm blocks. Connector labels may be activated or de-activated using the View menu's **Connector Labels** command. In the following example, Connector Labels option has been activated; the "Z" label indicates a complex value.



Some Comm blocks may operate using either an internal clock or an external clock. When such blocks are configured for the *internal* clock setting, the external clock connector label will appear

in brackets "[ck]". When configured for an *external* clock, the connector label will appear without brackets "ck."

Diagram timing

Many Comm blocks require "clock" signals to control bit or symbol timing in a diagram. A value of "0" is considered low, while a "1" is considered high. To accommodate floating-point values, the internal threshold of most blocks (but not all) is usually set to 0.5. For these blocks, a clock input signal greater than 0.5 is considered high. Please check individual block descriptions for more details.

Note: In Embed/Comm, clock signals are typically represented by an impulse train. For proper block operation, clock signals should typically be high for a single simulation step.

In simulations that include filters or other sources of delay, you may be required to synchronize the clock signal and other diagram delays in order to achieve proper timing within the simulation. The `Delay Estimator` block can be useful in determining unknown signal delays.

Important: While the Altair Embed SE modeling environment allows for simulation start times other than zero seconds, most Comm blocks assume a simulation start time of t= 0 sec. Unexpected results may occur when using a non-zero setting.

Random numbers

All Comm blocks that generate random values, for example the `Random Symbols` or `Noise` blocks, use a separate seed for controlling the random process than that used by core Altair Embed SE blocks.

The value of the seed is controlled by accessing the **Comm Module Random Seed…** command in the Comm menu.

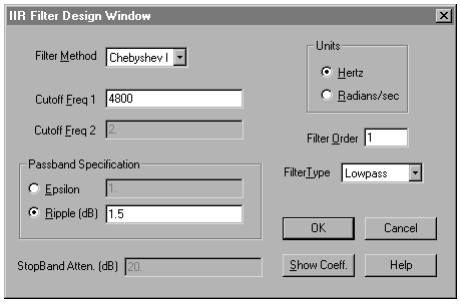
By enabling the Random Seed setting, simulation runs will use the same sequence of pseudo-random values from run to run, based on the seed value. If the selection is disabled, random numbers generated by Comm blocks will differ from run to run. At your discretion, the random number generator can also be reset upon an automatic restart condition; otherwise, the generator continues from its last state. The entries in the associated Setup dialog box are described below:

Entry	Description
Enable DLL Random Seed	Controls whether the Comm DLL random number generator is initialized with the Seed value at the start of a simulation run. This causes the output of all Comm random sources to repeat exactly each time the simulation is run.
Seed	Specifies the random number generator seed to be used by the Comm DLL.
Reset Seed on Auto Restart	Forces the Comm random number generator to reset itself (using the Seed value) at the start of each run when in Auto Restart mode. This causes the outputs of all Comm DLL random sources to repeat exactly for all run iterations. Otherwise, the random number generator begins the next run from where the random number sequence was left at the end of the previous run.

Setting up block parameters

To view and modify block parameters, you access the block's Setup dialog box by clicking the right mouse button over the block. Alternatively, you can also choose the Block Properties command from the Edit menu, point to the block, and click the left mouse button.

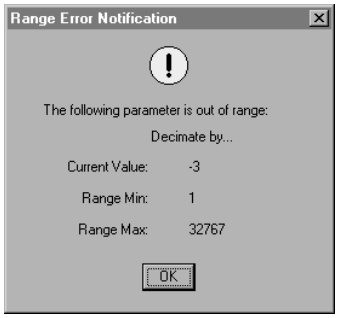
The changes you make in the dialog box do not take effect until you close the dialog box by pressing the OK button. Some parameters in the dialog box may at times be grayed out, indicating that they are unavailable for use given the particular block configuration. An example of a Setup dialog box is shown below.



Range checking

Range checking is performed on the majority of Comm block parameters to reduce the incidence of domain type errors (such as divide by zero). Range checking is performed either upon pressing the OK button to close the Setup dialog box, or at simulation start (the latter usually being the case).

When a range error is detected, a warning message is displayed similar to the one shown below.



A range error message describes the parameter in error, its current setting, and its allowed range. When the error is detected at simulation start time, the name of the offending block is also specified, and its instance in the simulation diagram is turned red.

Global variables

Most block parameters fields can accept numeric input, string variables (global or local), or user specified formulas including standard arithmetic operators, numerals, and string variables. Whenever a particular field is restricted to numeric entry only, an indication to this effect is provided in the Setup dialog box.

An sample parameter field entry using global variables is shown below:

offset + 2**pi***freq*

where *offset* and *freq* are user-defined global variables and *pi* is a Altair Embed SE constant.

All Embed/Comm parameter entries are evaluated once at the start of the simulation during the very first time step.

Beyond the initial time step, changes to global variables that are referenced by a block will have no effect on the block's behavior. Global variable values may themselves be the result of simple arithmetic operations involving other constants.

Note: Since the Filter Viewer does not actually start a simulation, it may be necessary to single step a simulation once in order for the value of a recently added or modified global variable to be updated.

Choosing an integration method

It is recommended that Embed/Comm diagrams be used with the **Euler** integration method. This mode provides the fastest diagram execution and does not involve the use of any sub-steps. Comm blocks were also designed to be compatible with the Runge Kutta 2nd Order integration method, which includes the use of sub-steps when integrator blocks are present in the diagram (if no integration blocks are present, it runs as Euler). This option is a less preferred choice since the presence of sub-steps can complicate diagram execution, especially in the presence of impulse-type signals. Finally, the Trapezoidal method may also be used in most cases, although not all Comm blocks are compatible with this mode.

When blocks that were not designed to operate under a specific integration method are detected in the active diagram, a warning message is issued to the user. All Comm blocks are fully compatible with the **Euler** integration method, which is the recommended setting to be used.

Note: The Altair Embed SE `transferFunction` block and Comm Filter blocks are not affected by the integration method you choose.

Multirate diagrams

The Embed/Comm environment now supports multirate diagrams, greatly enhancing the ability to model systems that require multiple sample rates. This feature is accessed through the use of "locally stepped" compound blocks. A local step compound block has its own sample rate, which may or may not be related to the main simulation time step, i.e. it need not be related to the base simulation rate by an integer multiple.

All blocks internal to a compound block with its own local time step behave as if the simulation time step were set at the local rate. The local time step must always be greater or equal than the main simulation time step (inverse of the simulation rate).

This feature can be used to create decimation in diagrams. For example, by setting the local time step to four times (4x) the main simulation step, all blocks inside the compound block, including output plots, will execute at a 4x decimated rate.

The output of a local rate compound block is sampled at the rate of the diagram level containing the compound block. This will often result in an over-sampled, staircase-like, output signal. When such an effect is undesirable the `Interpolator` block can be used to mitigate the effect by providing a linear interpolation capability, although it introduces a slight extra delay into the simulation.

The use of nested compound blocks that make use of local time steps is supported by Embed/Comm. Care should be taken when implementing such diagrams to ensure proper synchronization within the simulation. Another topic that requires special attention is the propagation of pulsed clock signals across differing sample rate regions in a diagram. To facilitate such tasks, a `Clock Edge` and `Clock Extend` block are provided under the Multirate Support category. Please refer to their respective block descriptions for more details.

Configuring compound blocks for local rate mode

A compound block can be configured to use a local time step by accessing its preferences dialog box.

♋ To specify a local time step

1. While holding down the CRTL key, right click on the desired compound block.
2. Select the "Local Time Step" check box.
3. In the appropriate field, specify the desired local time step (inverse of desired local sample rate). Formulas are allowed, so it's OK to specify "1.0 / desired rate".

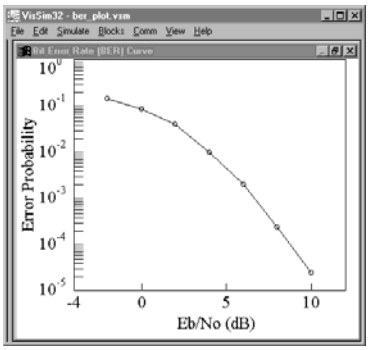
It's also possible to configure a compound block to use an externally supplied clock when in local time step mode. This option is accessed by selecting "Enabled Execution" in the same preferences dialog box as described above. The result is a compound block that tries to run at the specified local rate but is also gated by the external clock.

Output plots

The `plot` block can be used in a variety of ways to view simulation signals. In its basic mode, it displays up to eight time domain signals. By making use of its many display options, however, you can generate many of the plots used in communications. For detailed information on the `plot` block, refer to the *Altair Embed SE User's Guide*.

BER curves

BER curves are often the final indicator of performance in communication systems. They display the required SNR, usually expressed in E_b/N_o, to achieve a specified bit error rate at the receiver. An example is shown below.



♋ To configure a plot block for a BER curve

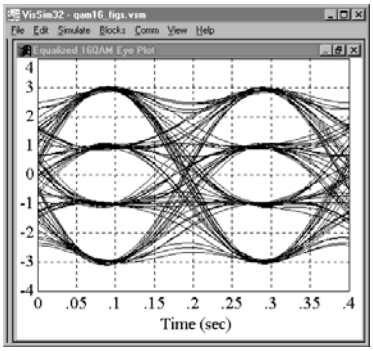
1. Click the right mouse button over the `plot` block to invoke its Plot Properties dialog box.
2. Click on the Options tab and do the following:
   - Activate the XY Plot option.
   - Indicate which input is the *x*-axis (E_b/N_o).
   - Activate Log Y mode.
   - Activate External Trigger.
   - Optionally, Select Over Plot. This will allow viewing of the BER curve as it is being generated point by point. You must press Clear Overplot to later clear the plot.
3. Click on the OK button, or press ENTER.

4. Connect the three outputs from the `BER Curve Control` block or from the `BER Control (# Errors)` block to the `plot` block.
5. The first output is the data trigger, the second the y-axis (Error Rate), and the third the *x*-axis (E_b/N_o).

The topic of generating BER curves is discussed in detail later in this chapter.

Eye plots

The eye plot simulates the use of an oscilloscope and is particularly useful for analyzing digital data waveforms. Its applications include estimating SNR, detecting amplitude compression and observing the effects of *inter-symbol interference* (ISI). The following example illustrates an I-channel eye plot of a 16-QAM modulated signal. An eye plot span of two symbol periods is shown.



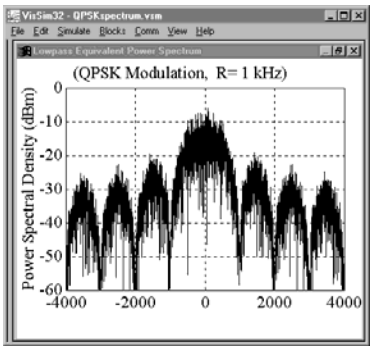
To configure a plot block as an eye diagram

1. Click the right mouse button over the `plot` block to invoke its Plot Properties dialog box.
2. Click on the Axis tab and do the following:
   - Activate the Retrace Enabled option.
   - In the Interval box, enter the desired interval.
   - In the Start and End boxes, enter the desired start and end times.
3. Click on the OK button, or press ENTER.

Frequency domain plots

Frequency domain plots are generally used to view the power spectrum associated with a time domain signal. They can also be used to view the magnitude and phase response (transfer function) of a filter or system.

There are two ways to view frequency domain plots. All `plot` blocks have the basic ability to show a frequency domain representation (FFT) of a time domain signal after a simulation run has been completed. In addition, a `Complex FFT,IFFT` block and a `Spectrum` block are provided, which can calculate complex frequency spectra during a run. The following figure illustrates a lowpass equivalent power spectrum of a QPSK modulated signal.



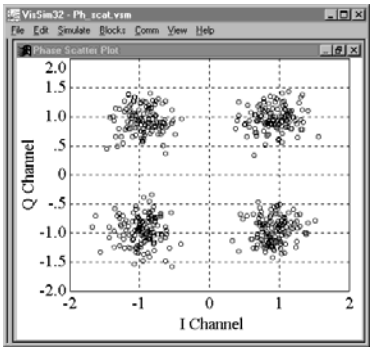
To configure a plot block for displaying the output from a Complex FFT/IFFT block or Spectrum Analyzer block

1. Click the right mouse button over the `plot` block to invoke its Plot Properties dialog box.
2. Click on the Options tab and do the following:
   - Activate the XY Plot option.
   - Indicate which input is the frequency axis.
   - Activate External Trigger.
3. Click on the OK button, or press ENTER.
4. Connect the `Complex FFT/IFFT` (or `Spectrum Analyzer`) trigger output to the trigger on the plot block, the frequency output to the input specified as the x-axis, and connect the remaining outputs (magnitude or phase) to any of the remaining plot block inputs.

For additional information on the use of the `Complex FFT/IFFT` block or the `Spectrum Analyzer` block, see Chapter 3, “Comm Block Set.”

Phase scatter plots

Phase scatter plots are used to view a received signal constellation in (I, Q) space and observe the impacts of channel noise and/or distortions. Below is an example of a QPSK constellation phase scatter plot. The effects of noise are evident in the *cloud* observed at each of the four constellation point locations.



To configure a plot block as an IQ phase scatter plot

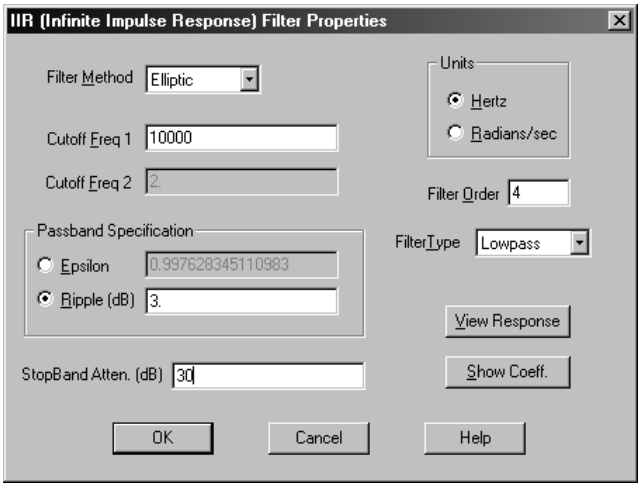
1. Click the right mouse button over the `plot` block to invoke its Plot Properties dialog box.
2. Click on the Options tab and do the following:
   - Activate the XY Plot option.
   - Indicate which input is the *x*-axis.

- Under Line Type, select Point.
- Activate External Trigger.
- Activate the Geometric Markers option. The Marker Count value should exceed the expected number of data points.

3. Click on the OK button, or press ENTER.
4. Connect the data clock to the trigger input, the I signal to the input specified as the *x*-axis, and the Q signal to any of the remaining inputs.

Filter Viewer

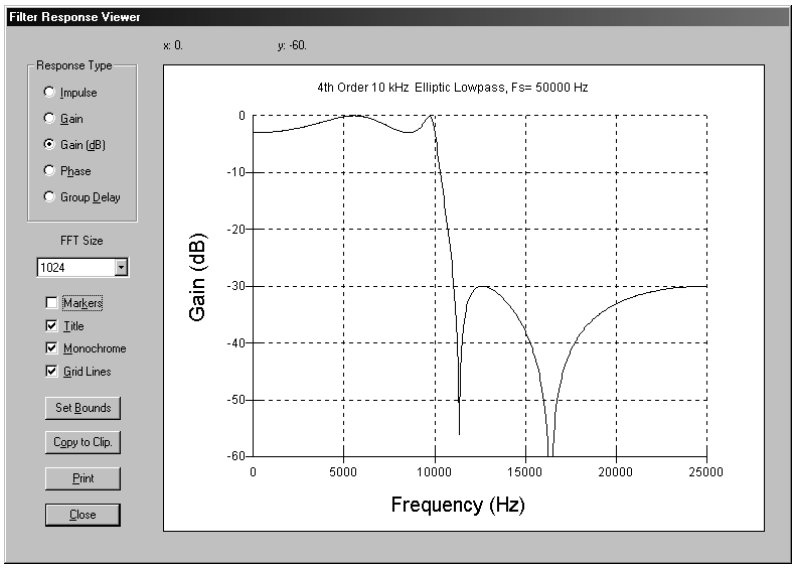
A filter response viewer is available from within the `FIR` and `IIR` filter blocks to aid the User in the filter design process. Once the desired filter parameters have been selected, the viewer can be accessed by pressing the "View Response" button in the filter properties dialog as shown in the following figure.



The following filter response details are available within the viewer:

- Impulse Response
- Gain Response (either plain or in dB)
- Phase Response
- Group Delay Response
- Cumulative Area (gain)

The appropriate selection is made by pressing the desired radio button within the viewer. To zoom within the plot, click and drag the left mouse button over the desired graph region. To zoom back out, click the right mouse button over the graph. A specific plot range can be specified along with the number of desired tick marks by clicking the "Set Bounds" button. The FFT size parameter controls the resolution of the displayed result (default FFT size is 1024).



The "Copy to Clipboard" button can be used to export the filter response plot to other Windows applications. The filter response can also be sent directly to the printer by pressing the "Print" button.

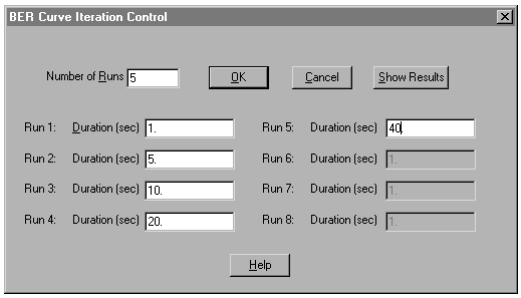
When selecting the Gain (dB) mode, the viewer defaults to a range of [0,-60] dB to avoid large negative values due to potential deep nulls in the filter response. To view the entire filter response should the plot be clipped, simply click the right mouse button over the graph.

Note: Since the Filter Viewer does not actually start a simulation, it may be necessary to single step a simulation once in order for the value of a recently added or modified global variable to be updated.

Generating BER curves

As discussed earlier in this chapter, BER curves are easily generated using the `BER Curve Control` or the `BER Control (# Errors)` block. These blocks make it possible to execute up to ten consecutive runs of the same simulation, each with a different SNR and simulation duration (or observed number of errors). The `BER Curve Control` block has two inputs, one for an error rate signal and the other for an SNR signal; the `BER Control (# Errors)` block requires a third input representing the number of observed errors. The BER curve is displayed at the end of each simulation run.

The `BER Curve Control` and `BER Control (# Errors)` blocks are located in the Estimators category under the Comm menu. Parameters for this block include the overall number of runs and the duration of each run in seconds (or number of errors). The block's three outputs drive a `plot` block configured for XY mode and external trigger, as described earlier in this chapter in the *BER curves* discussion of the *Output Plots* section.

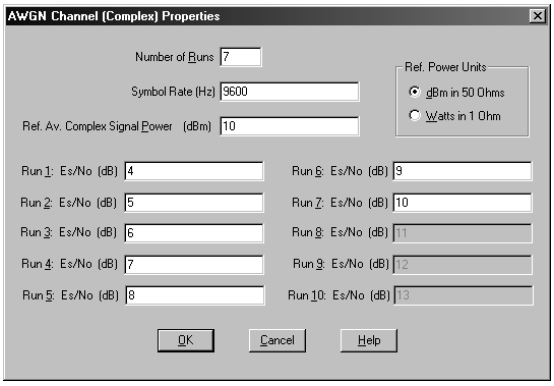


Enabling multiple consecutive runs

To enable multiple consecutive runs, you must activate the Auto Restart parameter in the Simulation Properties dialog box. To access this parameter, choose the Simulate menu's Simulation Properties command and click on the Range tab. You should also enter a value to the Range End parameter that is equal to, or exceeds, the longest duration specified in the `BER Curve Control` block Setup dialog box.

Varying the SNR for each run

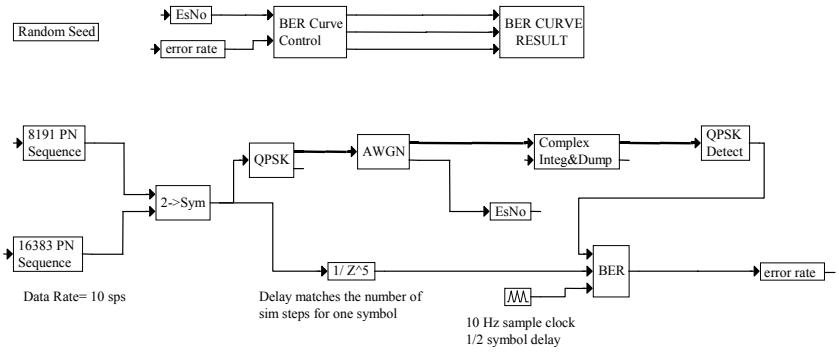
Varying the SNR for each run is done using the `AWGN` block. Its SNR output is connected to the bottom input of the `BER Curve Control` block. As shown below, the `AWGN` Setup dialog box allows for the specification of up to ten individual SNRs, corresponding to ten separate runs. The value of the Number of Runs parameter must match that specified in the `BER Curve Control` block.



Calculating BER measurements

The actual BER measurement for each run is calculated by using a `Bit/Symbol Error Rate` block, located in the Estimators category under the Comm menu. This block compares an original data sequence to a received data stream. It is very important that the original sequence be appropriately time shifted to compensate for any delays introduced by filters or an integrate and dump operation. The error rate output of the `Bit/Symbol Error Rate` block is connected to the top input of the `BER Curve Control` block.

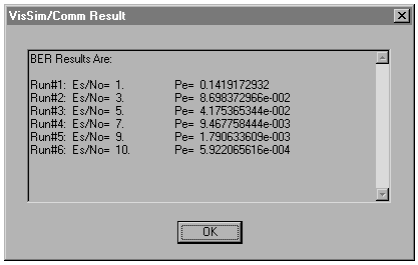
The following example illustrates the generation of a BER curve in a simplified end-to-end QPSK communication link. To run this model, open the BERCURVE.VSM diagram in the COMM_EX subdirectory.

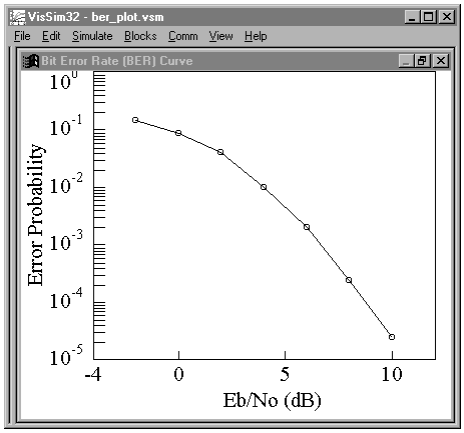


In the above model, the symbol rate has been arbitrarily set at 10 symbol/sec (20 bps), and because this is a lowpass equivalent simulation, the carrier frequency has been set to 0 Hz.

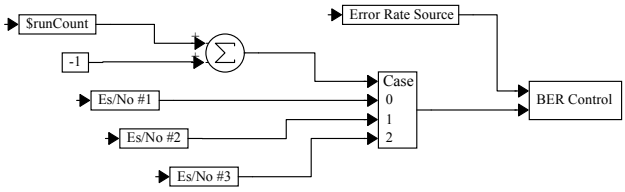
Two binary pseudo-random data sequences are combined to form a two-bit symbol, which is then modulated using QPSK. After adding noise, an integrate and dump operation is performed to obtain the received I and Q values. These are in turn passed to a detector, which determines the closest constellation point. This final output is then compared in the `BER Curve Control` block to the value originally sent. Note that the `BER Curve Control` block's input clock signal is offset by ½ a symbol to sample at the data symbol's midpoint. The additional 1 symbol delay in the source data stream is required for synchronization with the received output data. Had any filtering been included in the simulation, an additional signal delay would have been necessary.

The `Bit/Symbol Error Rate` block's output is connected to the `BER Curve Control` block, as is the current signal to noise ratio (EsNo) obtained from the `AWGN` block. At the end of the simulation, the `BER Curve Control` block creates a message box summarizing the results of all runs and outputs the BER curve result.



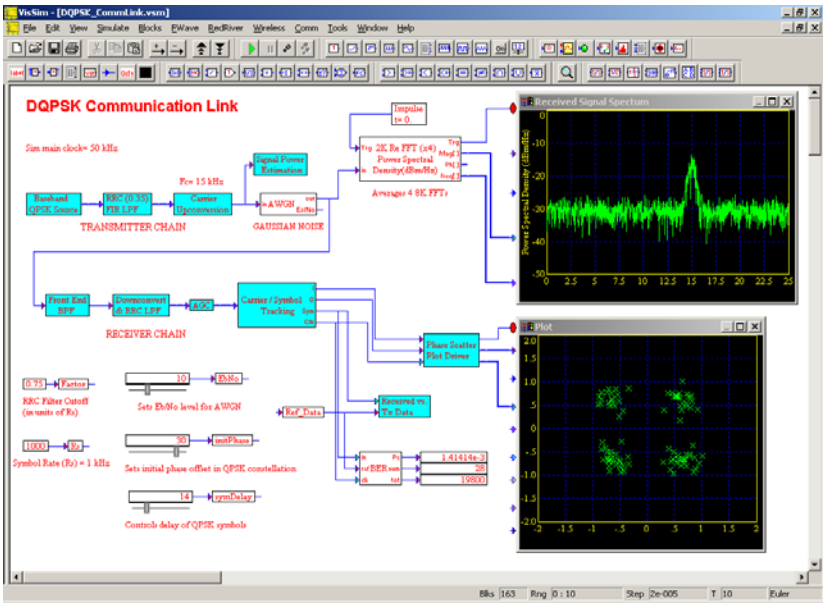


If you are not using an `AWGN` block to specify an SNR, then a custom solution may be created using the $runCount variable along with a `case` block, as shown in the following illustration. Note that the `BER Curve Control` block reads the input signals (error rate and SNR) only on the very last step of each individual run.



Sample communication simulation

A moderately complex example of an end-to-end communication system simulation is shown below. The block diagram, DQPSK_COMMLINK.VSM, is located in the "Applications" folder within the "Comm Examples" directory.



This example illustrates an end-to-end simulation of a *differential quadrature phase shift keying* (DQPSK) communications link. The diagram includes a transmitter stage, Gaussian noise channel, and features a complete software receiver inclusive of the following operations: automatic gain control (AGC), carrier tracking, symbol tracking, and data demodulation. The output plots illustrate the modulated signal spectrum and the received signal constellation after receiver tracking.

In this model, a random data sequence at 1.0 kHz is used as the data source. The four-level input sequence is passed to a DPSK block to be modulated. The baseband modulated data is then passed through a root raised cosine filter to shape the output spectrum and is then unconverted to a 15 kHz IF carrier frequency. The simulation sample rate is 50 kHz.

The channel is modeled as a simple Additive White Gaussian Noise (AWGN) process, where the noise level is set to achieve a 10 dB Eb/No signal level. The signal is then passed to the receiver chain, where the first stage includes a front end Butterworth bandpass filter. This is followed by an open loop downconversion to complex baseband, a root raised cosine filtering operation, and an AGC block. The purpose of the AGC is to provide a normalized signal level to the receiver's tracking loops, irrespective of the actual input signal level.

Carrier tracking is performed using a decision directed Costas Loop, which includes a standard phase locked loop (PLL) and soft-decision based phase error estimation. Symbol tracking is provided by the data transition tracking loop (DTTL) compound block. This block recovers the data's timing signal from the received signal and drives the timing of the integrate and dump operation in the receiver. In particular, the DDTL operates by comparing half symbol integrations and adjusting the timing signal until these two values are about equal across a symbol transition. By running the simulation from within the carrier tracking loop compound block, and using a shorter simulation end-time (e.g. 1 sec), the response of the various loops can be observed in more detail.

The tracked DQPSK constellation is plotted in the top level IQ Phase Scatter plot. A buffer stage is added to emulate persistence in the IQ scatter plot. The resulting plot displays 100 IQ samples at a time with a 20% replacement rate.

Finally, the received data decisions are compared to a delayed version of the original data stream to arrive at a bit error rate (BER) measurement. The outputs from the BER block provide the cumulative error probability, total number of observed errors, and total number of processed bits. The BER block does not start counting errors until the first 100 data symbols have passed. This is to allow the tracking loops to stabilize before making the BER measurement.

Chapter 3

Comm Block Set

The Comm menu lists the communication blocks provided with Embed/Comm. When you click on the Comm menu, most of the items that appear have a filled triangle (♦) next to them. These items are block categories. Click on a block category and a cascading menu appears listing the additional blocks.

To make it easier to find blocks in this chapter, they are presented in alphabetical order by category. The categories are: Channels, Complex Math, Demodulators, Digital, Encode / Decode, Estimators, Filters, Modulators - Complex, Modulators - Real, Multirate Support, Operators, PLL, RF, Signal Sources, Turbo Codes, Vector Ops, and Wireless.

If a block has parameters, they are listed after the block description.

Channels category

Blocks in the Channel category include `AWGN (Real and Complex)`, `Binary Symmetric Channel`, `Jakes Mobile`, `Multipath`, `Propagation Loss`, `Rice/Rayleigh Fading`, `Rummler Multipath`, `Saleh-Valenzuela (Real and Complex)` and `TWTA`.

AWGN (Complex or Real)

These blocks implement an AWGN channel in which Gaussian noise is added to the input signal. Two versions of this block exist: one block is complex and the other is real. The appropriate noise variance is automatically computed based on the desired SNR, simulation sampling frequency, symbol rate, and reference signal power. The block supports multiple run simulations by allowing up to ten different SNR values to be specified. The SNR is specified as E_s/N_o, as opposed to E_b/N_o.

The `AWGN` blocks can be used in conjunction with the `BER Curve Control` block or the `BER Control (# Errors)` block to generate Bit Error Rate (BER) curves. The information signal's power is specified as a parameter, as is the single-sided noise bandwidth. The simulation step size is also taken into account in computing the noise variance. In the case of the complex version of the block, the two noise samples (real and imaginary) are independent.

The `AWGN` blocks can also be used as a source of Gaussian noise by simply leaving the inputs disconnected or using a 0 input.

x = Input signal ([Re, Im] for complex)

y_1 = Output signal ([Re, Im] for complex)

y_2 = Current run's E_s/N_o value

Number of Runs

Specifies the number of simulation iterations. The maximum is ten.

Symbol Rate

Specifies the symbol rate R in hertz. This value is used internally to determine the energy per symbol as a fraction of the total specified signal power.

Real Average Complex Signal Power
Real Average Signal Power

Specifies the complex or average power of the information bearing signal and is used in calculating the appropriate noise variance. The units for this parameter are specified by the Ref. Power Units selection. You can specify power as either watts into 1 Ohm or dBm into 50 Ohms.

Ref. Power Units

dBm in 50 Ohms

Indicates that the average power reference is specified as dBm into a 50 Ohms impedance.

Watts in 1 Ohm

Indicates that the average power reference is specified as watts into a 1 Ohm impedance.

Run *x* E_s/N_o

Specifies the symbol SNR in decibels for each run. E_b can be easily converted to E_s by knowing the number of bits/symbol.

Binary Symmetric Channel

This block implements a *binary symmetric channel* (BSC). It expects a binary data stream {0, 1}, and periodically flips the output bits according to the specified error probability. Any input value larger than 0.5 is considered a 1.

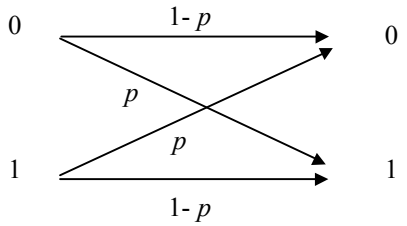
The `Binary Symmetric Channel` block accepts a real signal as input and outputs a real signal.

x_1 = Input signal (binary)

x_2 = Input data clock (impulse train)

y_1 = Output signal (0, 1)

y_2 = Error output (0= no error ; 1= error)



Channel Error Prob.

Indicates the probability of a bit flip *p*. The range for this value is from 0.9999 to $3.1x10^{-8}$.

Jakes Mobile

This block implements the Jakes fading channel model, which is commonly used in the mobile communications industry. This model approximates a Rayleigh fading process via summation of

multiple complex sinusoids, as indicated below. Block parameters include the maximum Doppler frequency and the desired number of additional terms (N_0) to be used.

Because the Jakes process is fully deterministic, two such blocks in the same diagram will output the same fading profile if the blocks' parameters are identical. Because of this, different parameter values should be used if somewhat independent fading processes are desired (also see the `Mobile Fading` block). It is also recommended to wait a short period of time before using the block's output, as the output waveform transitions to a more random state after a short while. A suitable time period appears to be about 1.5 x #terms x $1/f_m$ seconds.

The `Jakes Mobile` block accepts a complex signal as input and outputs a complex signal.

x = Complex input signal [Re, Im]

y = Complex output Signal [Re, Im]

$$y(t) = \frac{1}{\sqrt{2N_0 + 1}} \cdot [z_c + jz_s] \cdot x \qquad N = 2(2N_0 + 1)$$

$$z_c(t) = 2\sum_{k=1}^{N_0} \cos\phi_k \cos\omega_k t + \sqrt{2}\cos\phi_m \cos\omega_m t$$

$$z_s(t) = 2\sum_{k=1}^{N_0} \sin\phi_k \cos\omega_k t + \sqrt{2}\sin\phi_m \cos\omega_m t$$

where $\omega_k = \omega_m \cos(2\pi k / N)$ and $\phi_k = \pi k / (N_0 + 1),\ \phi_m = 0$

Number of Terms

Indicates the number of terms N_0 to be used in the Jakes approximation in addition to the primary Doppler frequency term.

Doppler Frequency

Indicates the maximum Doppler frequency (ω_m) for the channel. This value is specified in hertz. Usually, this term is usually based on the speed of the mobile.

Mobile Fading

This block implements a mobile Raleigh fading channel suitable for modeling mobile communications systems. This block is similar to the `Jakes Mobile` block, but uses a different approach for shaping the spectrum of the fading process. While the `Jakes Mobile` block approximates a Rayleigh fading process via the summation of multiple complex sinusoids, the `Mobile Fading` block does so by passing a uniform fading spectrum through an appropriate FIR shaping filter. The output process is then multiplied by the input signal. This technique is preferred in general, and especially when multiple instances of a fading block are present in the same diagram and need to have independent fading.

The impulse response of the internal fading FIR filter is obtained by computing the inverse FFT of the desired fading spectrum in the frequency domain. To keep the implementation efficient, the fading process is internally generated at a sampling rate ~8x the fading cutoff frequency. The output of the fading process is then interpolated to achieve the actual simulation sampling frequency. The linear interpolation process does introduce some faint spectral lines, but considerably improves the efficiency of the block.

Block parameters include the Doppler shift frequency and the desired number of taps for the FIR fading filter. This block takes a complex signal as its input, and outputs a complex signal. The internal fading process is fully initialized at simulation start and there is no need for a waiting period before the output is valid.

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

$$y(t) = h(t) * x(t)$$

$$H(f) = \frac{1}{2\pi f_m} \frac{1}{\sqrt{1 - \left(\frac{f}{f_m}\right)^2}} \quad |f| \le f_m \; ; \; H(f) = 0 \text{ otherwise}$$

where f_m is the doppler shift frequency

Since the above equation is undefined for $|f| = f_m$, the value of *H(f)* at this point is computed by linearly extrapolating the slope at the previous sample point in the frequency domain.

Number of Taps

Indicates the tap length of the internal FIR filter used to produce the fading process.

Doppler Shift

Indicates the maximum Doppler frequency for the channel in hertz. This term is usually based on the speed of the mobile.

Multipath

This block implements a multipath channel, in which multiple time and phase shifted versions of a signal are modeled as arriving simultaneously at a receiver. Multipath channels are commonly used to model the interaction between a direct signal and multiple reflected path signals. The reflected signals affect both the amplitude and phase of the received signal. Block parameters include the number of total paths, and the individual path's delay, relative gain, and phase rotation. This block takes a complex signal as its input, and outputs a complex signal.

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

$$y(t) = \sum_{k=0}^{N-1} \alpha_k x(t - \tau_k) e^{j\phi_k}$$

where τ_k = delay of path k

α_k = relative weight for path k

ϕ_k = phase rotation for path k

N = number of paths

Number of Paths

Specifies the number of paths in the model. Up to four paths may be specified.

Initial Condition (Real)

Specifies the Real component initial condition for the internal shift register used by the model.

Initial Condition (Imag)

Specifies the Imaginary component initial condition for the internal shift register used by the model.

Delay Mode

Sim Steps

Indicates the path delays are specified in simulation steps.

Seconds

Indicates the path delays are specified in seconds.

Path Delay

Specifies the delay in *seconds* or simulation steps associated with each path in the channel model.

Weight

Specifies a relative weight for each of the model paths. This value is not in dB.

Phase Rotation

Specifies the phase rotation in degrees associated with each path.

Propagation Loss

This block implements free space path attenuation assuming isotropic antennas. The loss is related to both the path distance and the specified signal frequency. The block can be configured for a fixed propagation path, or to accept an externally provided value.

The formula used by this block can result in a gain (rather than a loss) when the distance value is very small. To avoid this problem, the block will not allow the gain to exceed unity. The `Propagation Loss` block also outputs the instantaneous attenuation value being applied in dB.

x_1 = Input signal

x_2 = Path distance (in External mode)

y_1 = Attenuated output signal

y_2 = Attenuation in dB

$$y(t) = x(t) \cdot \frac{c}{4\pi f d} \qquad \text{where } c = \text{speed of light}$$

Path Distance

Indicates the path distance *d* to be used in computing the path loss. Units are either kilometers or miles, depending on the selected mode. This parameter is only used when in Internal Distance Mode.

Frequency

Indicates the frequency *f* in MHz to be used in computing the path loss.

Distance Mode

External

Indicates the path distance is specified via the Path Distance parameter.

Internal

Indicates the path distance is specified externally via the x_2 connector.

Distance Units

kilometers

Indicates that the path distance is specified in kilometers.

miles

Indicates that the path distance is specified in miles.

Rice/Rayleigh Fading

This block implements a *frequency non-selective* (flat) Rice or Rayleigh fading channel, which is commonly used to model tropospheric or ionospheric scattering in a communications link. The fading process is considered frequency-nonselective when the signal bandwidth is much smaller than the coherence bandwidth of the channel, that is, when all spectral components within the signal bandwidth are affected equally by the channel. In the Rayleigh channel, the received signal consists solely of uncorrelated scattered components. In the Rice channel, a direct signal path, or a fixed reflector in the medium, is also present.

In the `Rice/Rayleigh Fading` block, the input signal is multiplied by a single complex random variable having a Rayleigh amplitude distribution and uniform phase. The fading process is spectrally shaped using a two-pole Butterworth lowpass filter of cutoff frequency equal to the specified rms Doppler spread. The inverse of this value denotes the approximate coherence time of the fading process. Note that in this case the Doppler spread (spectral broadening) is due to the time domain amplitude fluctuations of the signal and not to any relative motion between the transmitter and receiver. For scenarios involving the latter, please refer to the `Jakes` and `Mobile Fading` blocks.

The `Rice/Rayleigh Fading` block accepts a complex signal as its input and outputs a complex signal. Block parameters include the Rice Factor, the RMS Doppler Spread Bandwidth, and the RMS Fading Loss. The internal fading process is fully initialized at simulation start and there is no need for a waiting period before the output is valid.

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

$$y(t) = Ax(t) \cdot \left[\alpha + \beta(t)e^{-j\Phi(t)}\right] \qquad \beta(t) \text{ is Rayleigh distributed}$$

$$\beta(t) = \sqrt{u_1(t)^2 + u_2(t)^2} \text{ and } u_1(t), u_2(t) \text{ are } N(0, \sigma^2)$$

$$\Phi(t) \text{ is uniform over } (-\pi, \pi)$$

$$\alpha^2 + 2\sigma^2 = 1 \qquad r = \frac{\alpha^2}{2\sigma^2} \qquad A = (10)^{\frac{-L_{dB}}{20}}$$

Rice Factor (*r*)

Determines the proportion of direct to scattered signal power in the channel. When $r = 0$, the model becomes Rayleigh (pure scattering).

RMS Doppler Spread (B_d)

Indicates the rms Doppler spread associated with the Rayleigh or Rice fading channel. This value is specified in hertz. The inverse of this value denotes the approximate coherence time of the fading process.

RMS Fade Loss (L_{dB})

Specifies the rms fading loss of the channel. This value is specified in decibels. The default is 0 dB, which yields a normalized unity gain channel. Positive values indicate a loss.

Rummler Multipath

This block implements the Rummler multipath fading channel, which is commonly used to model digital microwave links. The Rummler multipath channel models the interaction between a direct path signal and a reflected path signal. The reflected signal affects both the amplitude and phase of the received signal. Block parameters include the reflected path delay, reflected path relative gain, overall path attenuation, and the frequency of the fade null.

The `Rummler Multipath` block accepts a complex signal as input and outputs a complex signal.

x = Complex input signal [Re, Im]

y = Complex output Signal [Re, Im]

$$y(t) = \alpha x(t) - \alpha\beta e^{j2\pi f_0 \tau} x(t-\tau)$$

where τ = reflected path delay

α = overall path attenuation

β = reflected path relative gain

f_0 = fade null frequency

Reflected Path Delay

Indicates the delay, in seconds, of the reflected path relative to the direct path. This delay value is rounded to the closest number of simulation steps within the block. This value was experimentally found to be about 6.3 ns in the original Rummler model.

Shape Parameter

Indicates the amplitude gain of the reflected path relative to the direct path, also referred to as the shape parameter β. This value is not in decibels.

Fade Null Frequency

Indicates the frequency of the fade null. This value is specified in hertz.

Overall Gain Term

Indicates the amplitude gain α of the direct path. This value is not in decibels.

Saleh-Valenzuela

This block implements the Saleh-Valenzuela channel model, which is suitable for indoor multipath propagation. This model is used extensively in the ultrawideband arena, where short time domain pulses are used for signaling purposes. The model is implemented as a tap delay line structure as described below.

Two versions of this block are provided, one Real and one Complex. The two versions are identical, except that the Real version does not implement any phase rotation in the channel output.

The model assumes that multipath components arrive in clusters, where the cluster arrival rate is described by a Poisson process. The average power of subsequent clusters is assumed to decay exponentially. Each cluster is composed of many multipath rays, whose arrival times are also described by a Poisson process. Furthermore, the average ray power in any given cluster is assumed to decay exponentially, and its phase is uniformly distributed over [0, 2pi].

This block generates its internal tap delay structure at simulation start according to the various random processes specified by the model and the parameter values provided by the user. Once the tap delay structure has been generated, it is kept fixed for the duration of the simulation run. The particular tap delay structure used during a run can be viewed using the "View Response" button within the block's setup dialog.

x = Input signal ([Re, Im] for Complex version of block)

y = Output signal ([Re, Im] for Complex version of block)

$$h(t) = \sum_{l=0}^{\infty} \sum_{k=0}^{\infty} \beta_{kl} e^{j\theta_{kl}} \delta(t - T_l - \tau_{kl})$$

where T_l represents the arrival time of the l^{th} cluster,

τ_{kl} represents the arrival time of the k^{th} ray within the l^{th} cluster,

β_{kl} represents the ray's gain, and θ_{kl} its phase (complex version only)

$$p(T_l \mid T_{l-1}) = \Lambda \exp[-\Lambda(T_l - T_{l-1})], \quad l > 0$$

$$p(\tau_{kl} \mid \tau_{(k-1)l}) = \lambda \exp[-\lambda(\tau_{kl} - \tau_{(k-1)l})], \quad k > 0$$

$$\overline{\beta_{kl}^2} = \overline{\beta_{00}^2} \exp[-T_l / \Gamma] \cdot \exp[-\tau_{kl} / \gamma]$$

where $\overline{\beta_{00}^2}$ is the average power gain of the first ray of the first cluster

Time Units

Specifies the time units to be applied to the values of the next four parameters. Available choices are sec, msec, usec and nsec.

1 / Cluster Arrival Rate

Specifies the inverse of the average Poisson arrival rate Λ for the clusters. The default value is 300 nsec.

1 / Ray Arrival Rate

Specifies the inverse of the average Poisson arrival rate λ for the rays within each cluster. The default value is 5 nsec.

Cluster Decay Time Constant

Specifies the decay time constant Γ to be applied to subsequent arriving clusters. The default value is 60 nsec.

Ray Decay Time Constant

Specifies the decay time constant γ to be applied to the rays within a cluster. The default value is 20 nsec.

Number of Taps

Specifies the number of taps to be used in the model's tap delay line structure. This parameter specifies a truncation point for the channel model.

Average First Ray Power Gain

Specifies the average gain (not in dB) to be applied to the first arriving ray (first tap).

Generate Taps

Allows the user to generate a set of channel taps according to the selected block parameters, which can then be viewed using the View Response button. This function is useful when wanting to generate and examine the random tap pattern before running a simulation. The user can then select to use the generated taps (instead of generating new values at simulation start) by selecting the "Use Generated Taps" option.

Use Generated Taps

This option forces the simulation to use the most recently generated tap values obtained from pressing the Generate Taps button. If not selected, the simulation will generate a new random set of taps upon starting.

Show/Save Taps

Displays the current tap delay line values. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by clicking on the Save to File button. This allows the user to save a set of random taps for re-use at a later date.

By selecting the Use FIR File Format option, the values are saved in a format compatible with the `File FIR` block.

View Response

Invokes the Embed/Comm Response Viewer, which displays the generated channel impulse response.

Use Taps From File

This option forces the simulation to use the tap values as specified in an external file. The file format is the same as that of the File FIR block.

Select File

Opens the Select File dialog box for selecting the desired channel taps file.

Browse File

Opens the selected channel taps file using Notepad.

Taps File Path

Specifies the DOS path to the desired channel taps file. The format of the channel tap file is described below:

File header (anything)

number of taps

tap value #1

tap value #2, tap value #3

...

Multiple tap values can be specified on a given line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 128 characters.

TWTA (Analytical)

This block provides an analytical model of a *traveling wave tube amplifier* (TWTA) channel, which is frequently used to simulate satellite links. The `TWTA` block simulates a nonlinear amplifier and exhibits both AM/AM conversion and AM/PM conversion. AM/AM conversion maps signal envelope power fluctuations into gain variations, while AM/PM conversion maps signal envelope power fluctuations into carrier phase rotation.

The `TWTA` block accepts a complex signal as input and outputs a complex signal. Block parameters include the tube operating point and the average input signal power. The AM/AM and AM/PM equations are shown below and were obtained from "Frequency-Independent and Frequency-Dependent Nonlinear Models of TWT Amplifiers," Adel A. M. Saleh, *IEEE transactions on Communications*, pp. 1715-1720, 1981.

To implement a specific frequency response for the TWTA tube, please cascade this block with a Filter block, such as the `MagPhase` block.

x_1 = Complex input signal [Re, Im]

x_2 = Reference average power level

y = Complex output signal [Re, Im]

$$y(t) = AG(r)e^{j\Phi(r)}x(t)$$

where $G(r) = \text{am/am function}$ $\quad G(r) = \dfrac{\alpha_a r}{1+\beta_a r^2}$

$\Phi(r) = \text{am/pm function}$ $\quad \Phi(r) = \dfrac{\alpha_\phi r^2}{1+\beta_\phi r^2}$

$$r = \sqrt{\frac{|x_1|^2}{P_{av}}} \qquad A = \text{scaling factor}$$

Operating Point

Indicates the operating point of the TWTA in decibels. This is the location relative to saturation (0 dB) where the average input power lies.

Saturation Gain

Indicates the signal gain of the tube at the saturation point. This value is specified in decibels. The value of $G(r)$ is appropriately scaled so that $G(1)$ equals the desired gain.

Average Input Power

Indicates the average input complex power in watts. This parameter is only available when you select Internal Average Power Mode.

Average Power Mode

External
Internal

Indicates that the average power reference is provided either externally or internally.

Alpha_a

Indicates the amplitude gain coefficient. See equation above.

Beta_a

Indicates the amplitude gain coefficient. See equation above.

Alpha_phi

Indicates the phase rotation coefficient. See equation above.

Beta_phi

Indicates the phase rotation coefficient. See equation above.

Some example values from the above referenced paper by Saleh are shown below. The values were obtained by fitting the above equations to experimental TWT data.

Case	α_a	β_a	α_ϕ	β_ϕ
Example #1	1.9638	0.9945	2.5293	2.8168
Example #2	1.6623	0.0552	0.1533	0.3456
Example #3	2.1587	1.1517	4.0033	9.1040

TWTA (Table Lookup)

This compound block provides a table lookup version of a *traveling wave tube amplifier* (TWTA) channel. This block models a nonlinear amplifier and implements both AM/AM and AM/PM conversion via lookup tables. AM/AM conversion maps signal envelope power fluctuations into gain variations, while AM/PM conversion maps signal envelope power fluctuations into carrier phase rotation.

The `TWTA (Table Lookup)` block accepts a complex signal as input and outputs a complex signal. Internal compound block parameters include the AM/AM and AM/PM lookup tables, which are referenced in dBs and are implemented using the Altair Embed SE `map` block. The user must also provide an external reference power level indicating the tube's saturation level (not in dB).

To implement a specific frequency response for the TWTA tube, please cascade this block with a Filter block, such as the `MagPhase` block.

x_1 = Complex input signal [Re, Im]

x_2 = Saturation reference point power level (not in dB)

y = Complex output signal [Re, Im]

$$y = G(r)e^{j\Phi(r)}x_1$$

$$\text{where : } \quad G(r) = \text{am/am function} \qquad \Phi(r) = \text{am/pm function}$$

$$r = \text{instantaneous complex power scaled by } x_2$$

Vector AWGN

This block implements a vector version of the `Additive White Gaussian Noise` (`AWGN`) block. Gaussian noise of the specified level is added to each element of the input vector. The appropriate noise variance is automatically computed based on the simulation sampling frequency, specified noise bandwidth, and reference signal power. The `Vector AWGN` block supports multiple run simulations by allowing up to ten different Signal to Noise Ratio (SNR) values to be specified. The signal to noise ratio is specified as E_s/N_o, as opposed to E_b/N_o. The `Vector AWGN` block can be used in conjunction with the `BER Curve Control` or the `BER Control (# Errors)` block. The information signal's power is specified as a parameter, and must be supplied. This block can also be used as a vector source of Gaussian noise by simply leaving the input disconnected or using an all zero vector input.

x_1 = Input signal vector (Real elements)

x_2 = Frame clock (impulse)

y_1 = Output signal vector

y_1 = Frame clock (impulse)

y_3 = Current run's E_s/N_o value

Number of Runs

Specifies the number of simulation iterations. The maximum number is 10.

Vector Size

Specifies the size of the input and output vectors.

Ref. Average Signal Power

Specifies the average signal power for each element of the input vector. The units for this parameter are specified by the Ref. Power Units selection. Power may be specified either as watts into 1 Ohm, or dBm into 50 Ohms.

Ref. Power Units

dBm in 50 Ohms

Indicates that the average power reference is specified as dBm into a 50 Ohms impedance.

Watts in 1 Ohm

Indicates that the average power reference is specified as watts into a 1 Ohm impedance.

Run *x* E_s/N_o

Specifies the symbol Signal to Noise Ratio (SNR) in decibels (dB) for each run. By knowing the number of bits per symbol, a desired bit level SNR (E_b) can be easily converted to E_s.

WinProp Multipath

This block implements a multipath channel using path-delay data generated using the Altair WinProp application and stored in a .str file. This block operates using a complex signal input and only supports static scenarios, although the block can select the path rays associated with a specific time step in a "dynamic" .str file.

The user can either specify to use the first block of path data present in the file, or search for path data associated with a specific receiver location's (x, y, z) coordinates. Each path ray entry from the .str file provides a time delay and intensity associated with the ray.

The block operates by creating an internal delay shift register for the input signal and using appropriately spaced taps and weights to produce a complex superposition of all the rays in the calculation. The taps fall on the closest time step to the actual specified delay. This can introduce some error if the simulation time step is not a small fraction of the wavelength period. To overcome this issue, when the simulation time step is not a small fraction, the user is also provided with the option to "interpolate" the path delay, which will allocate the ray to the two closest taps with appropriate weights.

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

Max Path Rays

Specifies the maximum number of path rays to include in the simulation, up to the number of actual paths contained in the .str file. The maximum number is 128.

Rays Used

Read only field. Displays the actual number of path rays used in the simulation. The text box will populate after clicking on the "Check File" button or after a simulation run.

WinProp File Type

Static

Indicates that a "static" .str file is being used.

Dynamic

Indicates that a "dynamic" .str file is being used. The user must also specify the desired time step from the .str file from which to read the path data.

Interpolate Path Delay

When selected, provides complex phasor interpolation by allocating each ray to the two closest taps in the internal buffer. For this technique to work, the simulation rate should be at least three (3) times larger than the frequency of the signal of interest.

Time Offset

Allows the user to trim the internal size of the delay buffer by the stated time in nsec. This results in memory resource savings. The time offset should be less than the shortest path in the .str file.

Frequency

Read only field. Displays the frequency of the signal specified in the .str file for reference. The text box will populate after clicking on the "Check File" button or after a simulation run.

Ref Power

Read only field. Displays the transmit power of the signal specified in the .str file for reference. The text box will populate after clicking on the "Check File" button or after a simulation run.

Sample Time

Allows the user to specify the desired time step for the path rays when the "Dynamic" WinProp file type is selected.

Point Coordinates

Allows the user to specify the desired (x, y, z) coordinates for the receiver location in the .str file.

Tolerance

Specifies a tolerance value associated with the (x, y, z) coordinates above.

WinProp File Path

Specifies the Windows file path and name of the desired WinProp .str file.

Use First PATH Data Found in File

Allows the user to simply specify to use the first block of path rays found in the .str file without having to enter the associated (x, y, z) coordinates or time step.

Complex Math category

Blocks in the Complex Math category include `Addition`, `Conjugate`, `Division`, `Inverse`, `Multiplication`, `Power`, `Square Root`, `Complex to Mag/Phase`, `Complex to Real/Imag`, `Mag/Phase to Complex`, and `Real/Imag to Complex`.

Complex Addition

This block performs complex addition.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 + x_2$$

Complex Conjugate

This block outputs the complex conjugate of the input.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = x^*$$

Complex Division

This block performs complex division.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 \div x_2$$

Complex Inverse

This block outputs the inverse of the complex input.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = \frac{1}{x}$$

Complex Multiplication

This block performs complex multiplication.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 \cdot x_2$$

Complex Power

This block raises the complex input to the specified power. The result is obtained by converting the input value to magnitude and phase, raising to the power, and then mapping back to complex. For exponent values less than 1, the output value is one of many possible solutions.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = x^{\alpha}$$

Exponent

Specifies the exponent α to which the input complex value is raised. The exponent can be any real value.

Complex Square Root

This block produces the square root of the complex input. The result is obtained by converting the input value to magnitude and phase, taking the square root, and then mapping back to complex. The output value is one of many possible solutions.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = \sqrt{x}$$

Complex to Mag/Phase

This block converts a complex vector input into magnitude and phase components. A four-quadrant arctangent function is used, which returns values in the range [-π, π]. If both the real and imaginary parts of the input are 0, zero phase is returned.

x = Complex vector input [Re, Im]

y_1 = Complex magnitude

y_2 = Complex Phase

$$y_1 = \sqrt{\mathrm{Re}(x)^2 + \mathrm{Im}(x)^2} \qquad y_2 = \mathrm{atan}(\mathrm{Im}(x), \mathrm{Re}(x))$$

Complex to Real/Imag

This block splits a complex vector input into its real and imaginary parts.

x = Complex vector input [Re, Im]

y_1 = Real part

y_2 = Imaginary part

$$y_1 = \mathrm{Re}(x) \qquad y_2 = \mathrm{lm}(x)$$

Mag/Phase to Complex

This block converts magnitude and phase inputs into a complex vector output.

x_1 = Complex magnitude

x_2 = Complex phase

y = Complex vector output [Re, Im]

$$y = \left[x_1 \cos(x_2), x_1 \sin(x_2)\right]$$

Real/Imag to Complex

This block combines real and imaginary inputs into a complex vector output.

x_1 = Real part

x_2 = Imaginary part

y = Complex vector output [Re, Im]

$$y = \left[x_1, x_2\right]$$

Demodulators category

Blocks in the Demodulators category include `DPSK Detector`, `Integrate & Dump (Complex)`, `FM Demodulator`, `Integrate & Dump (Real)`, `PPM Demodulator`, `PSK Detector`, and `QAM/PAM Detector`.

Differential PSK Detector

This block implements a differential phase shift keying (DPSK) detector, and is used to map a baseband DPSK constellation value back to a symbol number.

This block accepts a complex value, representing a point in the (I, Q) plane, and determines the phase change since the previous detection whenever the input clock is high. It then converts the observed phase change to a symbol number according to the specified phase transition map file. Its input is typically the output of an `Integrate & Dump` block or of a filter matched to the symbol rate.

Supported DPSK modes include DBPSK, DQPSK, Pi/4-DQPSK, D8PSK, D16PSK and D32PSK.

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y_1 = Output symbol number

x_2 = Output clock (impulse train)

DPSK Type

Specifies the desired DPSK modulation type. Supported modes include DBPSK, DQPSK, Pi/4-DQPSK, D8PSK, D16PSK and D32PSK.

Initial Phase

Specifies the initial phase of the detector in *degrees*.

Select File

Opens the Select File dialog box for selecting a DPSK phase transition map file.

Browse File

Opens the selected DPSK phase transition map file using Notepad.

DPSK File Path

Specifies the DOS path to the desired DPSK phase transition map file. For a description of the file format, please refer to the `Differential PSK Modulator` block description.

FM Demodulator

This block demodulates a Frequency Modulated (FM) signal. The block operates by estimating the derivative of the input signal, as shown by the formula below. The `FM Demodulator` assumes a baseband complex input, but will work at IF frequency as well. For IF operation, a bias is introduced in the output, but this can be compensated using the Offset parameter.

To demodulate a Real (as opposed to Complex) FM signal, a Hilbert transform (see the `FIR` block) can be used to create a complex version of the signal.

Note: The delay through an FIR filter block is (N-1)/2 samples, where N is the number of taps.

$$y = \frac{1}{2\pi\beta} \frac{\mathrm{Re}(x) \cdot \frac{d}{dt}\mathrm{Im}(x) - \mathrm{Im}(x) \cdot \frac{d}{dt}\mathrm{Re}(x)}{\mathrm{Re}(x)^2 + \mathrm{Im}(x)^2}$$

x = Complex FM modulated signal

y = Demodulated signal

FM Deviation Index

Specifies the FM deviation index β used by the transmitter in units of hertz/volt. This value controls the extent of carrier frequency deviation as a function of modulator input drive level.

Offset

Specifies the offset to be added to the output in volts. The default is zero. This parameter is provided to compensate for the fixed bias that occurs at the block's output when the input to demodulator is NOT at baseband. The proper compensation value can be computed as follows.

Offset = Carrier Freq / FM deviation index

Overflow Value

Specifies the value to be output when an internal overflow (divide by zero) is detected. A divide by zero can occur since the output signal is normalized by the complex magnitude of the input signal. The default value is zero.

IQ Detector

This block is used to map an arbitrary baseband IQ constellation point back to its corresponding symbol value. It takes as input a complex value — representing a point in the (I, Q) plane — and determines the closest constellation point per the user provided look-up table. It then outputs the corresponding symbol number from the file. This block typically follows an `Integrate & Dump` block.

The `IQ Detector` block operates by measuring the Euclidian distance between the received point and all constellation points, and then selecting the closest point. The simulation speed of the block is inversely proportional to the constellation size. For this reason it is recommended that this block not be used with very large constellations unless absolutely necessary.

x_1 = IQ Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y = Output symbol number

Constellation Size

Specifies the size of the IQ constellation. This value should match the information provided in the external IQ Constellation File.

Select File

Opens the Select File dialog box for selecting the IQ constellation map file.

Browse File

Opens the selected constellation map file using Notepad.

File Path

Specifies the DOS path to the desired IQ constellation map file. This entry should indicate the same file used with the associated `IQ Mapper` block at the transmitter. The format of the mapping file is described below:

File header (can be anything)

Symbol number I output Q output

...

The *symbol number* values need not be entered in increasing order, although it is highly recommended to do so for clarity. The table must contain a total of *N* entries, where *N* is the constellation size. Table entries may be separated by blank spaces, tabs, or commas. Blank lines are also acceptable. An example of an IQ map file is shown below:

Arbitrary IQ Constellation Map File (*Header line)*
0 1.2 1

```
1   -0.9      1.1
2   -1        -1.15
                                        (blank lines are OK)
3   -1.05     -0.95
```

The above example illustrates a QPSK constellation with slight imbalance.

PPM Demodulator

This block demodulates a *pulse position modulated* (PPM) signal. The `PPM Demodulator` block accepts a baseband real signal as input and outputs a symbol number. The values of the block's parameters should match the associated `PPM` block values where applicable. For more details, see the `PPM Modulator` block description.

x = Baseband modulated signal

y = Output symbol value (0, 1, ..., N-1) N = number of levels

Number of Levels

Indicates the number N of possible symbol values.

Pulse Width

Indicates the width of the rectangular pulse in seconds.

Symbol Rate

Indicates the symbol, or pulse, rate in symbols/second.

Frame Start Delay

Indicates the initial start delay of the symbol frame in seconds.

PSK Detector

This block is used to map a baseband PSK constellation point back to its corresponding symbol number. An external clock has been added to this block for improved efficiency.

The `PSK Detector` block accepts a complex value representing a point in the (I, Q) plane, and determines the closest PSK constellation point whenever the input clock is high. It then outputs the corresponding symbol number from the specified PSK constellation map file. Its input is typically the output of an `Integrate & Dump` block or of a filter matched to the symbol rate.

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y = Output symbol number

PSK Type

Indicates the PSK modulation type. Available choices are BPSK, QPSK, 8-PSK, 16-PSK, and 32-PSK. For SQPSK, select QPSK and delay the I channel input by ½ symbol duration. For more details on the above constellations, see the `PSK Modulator` description.

Constellation Rotation

Indicates the amount of modulator constellation rotation from the constellation default. This value is specified in degrees.

Channel Rotation

Indicates the amount of phase rotation introduced by the communication channel. This value is specified in radians.

Select File

Opens the Select File dialog box for selecting a PSK constellation map file.

Browse File

Opens the selected PSK constellation map file using Notepad.

PSK File Path

Specifies the DOS path to the desired PSK constellation map file. This entry should indicate the same file used with the associated PSK modulator. Refer to the description of the `PSK Modulator` block for an explanation of the map file format.

QAM/PAM Detector

This block is used to map a baseband QAM or PAM constellation point back to its corresponding symbol number. An external clock has been added to this block for improved efficiency.

The `QAM/PAM Detector` block accepts a complex value, representing a point in the (I, Q) plane, and determines the closest QAM or PAM constellation point given whenever the input clock is high. It then outputs the corresponding symbol number from the specified QAM/PAM constellation map file. Its input is typically the output of an `Integrate & Dump` block or of a filter matched to the symbol rate.

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

x_3 = External constellation spacing reference [Optional]

y = Output symbol number

QAM Type

Indicates the selected modulation scheme. The available choices are 16-QAM, 32-QAM, 64-QAM, 128-QAM, 256-QAM, 4-PAM, 8-PAM and 16-PAM. For more details on the above constellations, see the `QAM/PAM Modulator` description.

Constellation Rotation

Indicates the amount of modulator constellation rotation, in degrees, from the constellation default.

Channel Rotation

Indicates the amount of phase rotation, in radians, introduced by the communication channel.

Constellation Spacing

Indicates the amplitude spacing between adjacent constellation points. This value is specified in volts. It is only available when you activate the Internal Amplitude Reference parameter.

Amplitude Reference

Internal

Indicates that the constellation spacing reference is provided internally. This value is specified in volts.

External

Indicates that the constellation spacing reference is provided externally through the x_3 input connection. This value is specified in volts.

Select File

Opens the Select File dialog box for selecting a QAM constellation map file.

Browse File

Opens the selected QAM constellation map file using Notepad.

QAM File Path

Specifies the path to the desired QAM constellation map file. This entry should indicate the same file used with the associated QAM modulator. Refer to the description of the `QAM/PAM Modulator` block for an explanation of the map file format.

Digital category

Blocks in the Digital category include `Accumulate & Dump`, `Binary Counter`, `Bits to Symbol`, `Buffer`, `D Flip Flop`, `Divide by N`, `JK Flip Flop`, `Mux/Demux`, `Parallel to Serial`, `Queue`, `Serial to Parallel`, `State Machine`, `Symbol to Bits`, and `Unbuffer`.

Accumulate & Dump

This block provides a clocked Accumulate and Dump function. Two versions of this block are available; one Real and the other Complex. At each input clock pulse, the block will add the current input value to its internal accumulator register. This value is then "dumped" (i.e. output) when a clock pulse is presented on the dump input. Once dumped, the accumulator is reset to zero.

The block can be configured to either dump first and then accumulate, or vice versa. The output can also be optionally averaged by dividing by the number of input values read.

x_1 = Input value

x_2 = Sample clock (pulses)

x_3 = Dump (pulse)

y_1 = Accumulated value

y_2 = Output clock (pulses)

Dump Mode

Accumulate First

Specifies that the internal sum register will be accumulated first and then dumped and reset when a sample clock and dump clock occur at the same simulation step.

Dump First

Specifies that the internal sum register will be dumped, reset, and then accumulated when a sample clock and dump clock occur at the same simulation step.

Average by the Input Count

The accumulated value is divided by the number of values read before being output.

Binary Counter

This block implements a binary counter with optional edge triggering . The internal counter will increment each time a rising (or falling) edge is detected, or whenever the input is above a set threshold. Once the counter has reached an internal "all ones" state, the next event will reset the counter to zero and produce an output pulse on the Carry flag output. Block parameters include the number of bits for the counter, clock edge mode, the counter initial value, and edge threshold voltage.

x_1 = Input clock (impulse train)

x_2 = Reset

y_1 = Counter value

y_2 = Carry flag

Number of Bits

Specifies the number of bits for the counter. Valid range is 1 to 31.

Counter Initial Value

Specifies the initial value of the counter at simulation start.

Threshold

Specifies the voltage threshold above which the input is considered *high*.

Edge Mode

Rising Edge

Specifies that the counter will increment upon detecting a rising pulse. The input must fall back below the threshold before the counter will be incremented again.

Falling Edge

Specifies that the counter will increment upon detecting a falling pulse. The input must rise back above the threshold before the counter will be incremented again.

None

Specifies that the counter will increment upon detecting an above threshold level. There is no need for the input to fall back below threshold to re-arm the counter (i.e. the counter will keep incrementing at each simulation step as long as the input is above the threshold value).

Bits to Symbol

This block accepts inputs from *n* parallel binary bit streams and outputs the corresponding symbol number. You can specify the number of input data streams. The mapping is simply the decimal equivalent of the binary number formed by combining the input bit streams. Rounding is performed on the input data. Any value $x > 0.5$ is considered a 1, otherwise it is considered a 0.

x_1 = Input bit stream #1

.. ...

x_n = Input bit stream #*n*

y = Output symbol number (0, 1, .., 2^n-1)

Bit Order

LSB First

Indicates that x_1 is considered the *least significant bit* (LSB).

MSB First

Indicates that x_1 is considered the *most significant bit* (MSB).

Number of Input Bits

Indicates the number *n* of incoming binary data streams. Valid range is 2 to 16.

Buffer

This block is used to pack elements of a serial data stream into a vector frame of the specified size. The internal buffer is NOT a sliding buffer but rather a circular buffer (i.e. once *N* elements have been read, any new serial data will begin overwriting the buffer). The reading of the input data is controlled via an external clock. Data can be written to the output vector in either ascending or descending order.

The output vector can be made to update following each input data clock pulse, or wait until an entire frame of data is available. Once the specified number of values (*N*) has been read, the block outputs a write strobe (frame clock) to signify that the output vector is fully updated. Once this

occurs, new input values will start to overwrite the oldest values in the internal buffer in cyclic fashion. The buffer is initialized to all zeros at the start of a simulation.

To revert a data vector back to a serial data stream, please refer to the `Unbuffer` block later in this section. If the use of a sliding buffer is desired instead, please refer to the "Blocks/Matrix Operations/buffer" block.

x_1 = Input serial data

x_2 = Input clock (impulse train)

x_3 = Reset (resets vector index to first [or last] position)

y_1 = Frame Clock (impulse)

y_2 = Output data vector (size N)

Buffer Ordering

FIFO

Indicates that the first input data symbol is to occupy the first element of the output vector.

LIFO

Indicates that the last input data symbol is to occupy the first element of the output vector.

Output Mode

Sequential

In this mode the output vector is incrementally updated at each new input value in round robin fashion.

Post All at Once

In this mode the input values are internally buffered and then presented all at once to the output vector. When in this mode, the posting process can optionally be delayed by one clock cycle via the Output Delay setting.

Output Delay

This setting only applies when in Post-at-Once Output Mode.

None

When None is selected, the output vector and frame clock are posted as soon as the last buffer element (the Nth value) is read in.

Extra Clock Cycle

When this optional mode is selected, the output vector and frame clock are posted one input clock pulse after the last buffer element (the Nth value) is read in. This mode is useful when trying to keep the delay through the block equal to one block size instead of this value less one clock cycle.

Buffer Size

Specifies the size N of the output buffer. Valid range is 1 to 8192.

D Flip Flop

This block implements an edge-triggered D type flip-flop. Block parameters include the initial flip flop state, clock edge mode, and edge threshold voltage. The clock input for this block, unlike most Comm blocks, is not a pulse train but rather a rectangular type waveform.

x_1 = D input

x_2 = Clock (rectangular)

y_1 = Flip flop output

y_2 = Complemented output

Edge Mode

Rising Edge

Specifies that the flip-flop will clock upon detecting a rising clock edge.

Falling Edge

Specifies that the flip-flop will clock upon detecting a falling clock edge.

Initial State

Specifies the initial state of the flip-flop.

Threshold

Specifies the voltage threshold above which the inputs are considered high.

Divide by N

This block implements a digital divide by N function. A typical use of this block is to reduce the pulse rate of a clock signal by an integer factor. Block parameters include the divide ratio, initial delay, and signal threshold voltage.

The `Divide by N` block works by counting the rising and falling edges of the input waveform, and can accept either an impulse train or a rectangular pulse train for its input. Similarly, the output mode may be specified as producing either unity impulses or rectangular pulses. The input and output modes need not be the same. Note that if the divide factor is ODD and the input is an impulse train, then the rectangular output mode will not generate a symmetric waveform. The output of the `Divide by N` block is always either 0 or 1.

x = Input clock

y = Divided output clock [0, 1]

Divide by

Specifies the divide down ratio *N*. This value must be a positive integer.

Initial Delay

Specifies an initial delay, entered as a number of pulses, to be counted prior to commencing the divide down process. This parameter is intended to be used for clock synchronization purposes.

Threshold

Specifies the voltage threshold above which the input clock is considered high.

Output Mode

Impulse Train

The block outputs an impulse train.

Rectangular Pulses

The block outputs a rectangular pulsed waveform.

Edge Mode

This mode only applies when the Output Mode is set to *Impulse Train*.

Off

The block will consider an input that remains “high” across multiple time steps as multiple pulses.

On

The block will only count pulses that include an edge (i.e. a transition from low to high). A constant “high’ input does not get counted as multiple pulses.

JK Flip Flop

This block implements an edge-triggered JK type flip-flop. Block parameters include the initial flip flop state, clock edge mode, and edge threshold voltage. The clock input for this block, unlike most Comm blocks, is not a pulse train but rather a rectangular type waveform.

x_1 = J input

x_2 = K input

x_3 = Clock (rectangular)

y_1 = Flip flop output

y_2 = Complemented output

Edge Mode

Rising Edge

Specifies that the flip-flop will clock upon detecting a rising clock edge.

Falling Edge

Specifies that the flip-flop will clock upon detecting a falling clock edge.

Initial State

Specifies the initial state of the flip-flop.

Threshold

Specifies the voltage threshold above which the inputs are considered high.

Mux/Demux

This block implements a digital multiplexer or demultiplexer. A multiplexer combines several low speed data streams into a single high-speed stream. A demultiplexer reverses the operation.

The `Mux/Demux` block can be controlled by an internal or external clock. At each clock pulse, the current active input (output) of the multiplexer (demultiplexer) is advanced by 1 in round robin fashion. Typically the block's clock rate (switch rate) should equal the number of inputs (outputs) multiplied by the individual line rate.

Multiplexer Mode

x_1 = External clock (impulse train)

x_2 = Input #1

x_{n+1} = Input #

y_1 = Clock output (switch rate)

y_2 = Multiplexed output

Demultiplexer Mode

x_1 = External clock (impulse train)

x_2 = Multiplexed input

y_1 = Clock output (switch rate)

y_2 = Output #1

y_{n+1} = Output #*n*

Number of Lines

Specifies the number of inputs when in multiplexer mode and the number of outputs when in demultiplexer mode.

Initial Position

Specifies the initial state of multiplexer or demultiplexer. The selected input (output) will be the active connection until the first clock pulse.

Mode

Multiplexer

Specifies that the block operates as a multiplexer.

Demultiplexer

Specifies that the block operates as a demultiplexer.

Timing

External

Specifies that the block is controlled by an external clock.

Internal

Specifies internal timing control. See the Switch Rate and Start Time parameters.

Switch Rate

Specifies the block's switch rate when in internal timing mode. This is the rate at which the block changes from one input (output) to the next. Specify this value in hertz.

Start Time

Specifies the start time for the internal clock in seconds. This is the time of the first clock pulse.

Packet Timing

This block produces packet-timing signals for use in generating fixed length or variable length packet structures. Up to eight different sections can be strung together using this block, each with its own length specified as a multiple of an underlying clock interval. For example, a packet may have three sections comprised of the packet header, payload data, and a CRC, each with different lengths. Clock timing can be either generated internally or provided externally.

As the block cycles through the specified number of sections (N), it provides control signals (ON/OFF) to individual outputs corresponding to each section. Only one of these outputs will be ON at any given time. In addition, a "state" output is provided to indicate the active section of the packet. Note: The State output is zero indexed and ranges over [0, N-1]. Once the last section has been completed, the block restarts at the beginning.

In a typical configuration, the block's control lines can be used to enable the clocking (ON/OFF) of individual signal sources corresponding to each section of the packet (e.g. header, payload, CRC), and the state output can be used to control which signal is passed to the next stage in the simulation diagram using a `case` block (Blocks/Nonlinear).

x_1 = External clock (impulse train) [optional]

x_2 = Section #0 length [optional]

x_3 = Section #1 length [optional]

...

x_{N+1} = Section #N-1 length [optional]

y_1 = Current state [0, N-1]

y_2 = Output clock (impulse train)

y_3 = Section #0 enable {0, 1}

...

y_{N+2} = Section #N-1 enable {0, 1}

Number of Sections

Specifies the size of the correlation buffer in simulation steps. This number may NOT be a global variable (numeric input is required).

Start Position

Specifies the initial state (active section number) for the block. This is the state that will be active, and remain active for the specified section duration, at the first clock pulse. In case the initial clock pulse is not at t= 0 (i.e. Start Time > 0), then this also specifies the dormant state for the block.

Clock Rate

Specifies the block's output clock rate in *Hertz* when in Internal Timing mode. This value typically corresponds to the bit rate or symbol rate of the packet.

Start Time

Specifies the start time of the block's internal clock in *seconds.* This parameter is only required when in Internal Timing mode.

Clock Timing

External

Specifies that block timing is provided externally. Note: The first clock pulse is used to mark the beginning of the specified initial section and does NOT cause a change in the active state.

Internal

Specifies that block timing is generated internally. The first clock pulse occurs at the specified Start Time, and signals the beginning of the specified initial section. Note: The active state does NOT change at the first clock pulse.

Length Mode

Internal

Specifies that the length of each packet section is specified internally and thus remains fixed. Note: It is acceptable for a section length to be defined as zero; in this case the particular section will be skipped.

External

Specifies that the length of each packet section is specified externally and thus may change during the simulation. This mode can be used to generate variable length packets. Section lengths are measured in clock cycles. Note: It is acceptable for a section length to be defined as zero; in this case the particular section will be skipped.

Section Length [#0, #1, …, #7]

Specifies the size of each packet section as measured in clock cycles, which typically corresponds to a bit or symbol period. This parameter is only applicable when in Internal Length Mode.

Parallel to Serial

This block accepts parallel data represented by symbol numbers and outputs a serial binary stream. The output bits are obtained by decomposing the binary representation of the input symbol number. The bits can be output either LSB first or MSB first. You must specify the number of bits in the parallel input data word and the output bit rate. Input symbols are read in when the x_2 clock input goes high. Output clock pulses are provided for each consecutive output bit. If the bit rate is insufficient to output all the bits prior to the next symbol input pulse, any remaining bits are discarded. On the other hand, once all output bits have been shifted out, the last bit value is held and clock pulses cease until the next input symbol is processed. Proper timing is the user's responsibility. It is recommended that the duration of each output bit be represented by an integer number of simulation steps.

x_1 = Input symbol number (0, 1, .., 2^n-1) n = number of bits

x_2 = Input symbol clock (impulse train)

y_1 = Output serial bit stream

y_2 = Output bit clock (impulse train)

Bit Order

LSB First

Indicates that y_1 is considered the LSB.

MSB First

Indicates that y_1 is considered the MSB.

Bits per Symbol

Indicates the number of bits *n* per parallel symbol. Valid range is 2 to 16.

Output Bit Rate

Indicates the bit rate to be used for outputting the serial data.

Pulse Extend

This block is used to extend the duration of a clock impulse for a specified time interval, represented as either a number of simulation steps or time in seconds. This block preserves the amplitude of the original input pulse.

x = Input clock signal (impulse train)

y = Output extended pulse

Pulse Duration

Specifies the total duration of the output pulse in either simulation steps or *seconds*, depending on the Units selection.

Units

Sim Steps

Indicates the pulse duration is specified in simulation steps.

Seconds

Indicates the pulse duration is specified in seconds.

Queue

This block implements a digital queue. The queue service can be specified as either *first in first out* (FIFO) or *last in first out* (LIFO). This block can be used to simulate buffers used in communication systems. Input values are stored in the queue according to an input clock and are read out according to an output clock. If the input and output clocks occur simultaneously and the queue is empty, then the current input is immediately provided at the output. Besides the data output, the `Queue` block also provides the current number of values stored in the queue and an overflow indicator flag.

x_1 = Data in

x_2 = Input clock (impulse train)

x_3 = Output clock (impulse train)

y_1 = Data out

y_2 = Queue count

y_3 = Overflow flag (see below)

Queue Size

Specifies the size of the queue buffer. Valid range is 1 to 32,766.

Empty Output

Specifies the value to output when the queue is empty.

Queue Type

FIFO

Specifies that the queue operates as a FIFO queue.

LIFO

Specifies that the queue operates as a LIFO queue.

The overflow flag output behaves as follows:

Flag = 0: Normal condition

Flag = 1: An overflow has occurred during the simulation

Flag = 2: Overflow in progress

Serial to Parallel

This block accepts a serial binary stream and outputs parallel data as symbol numbers. The bits can be provided either LSB first or MSB first. The symbol value is obtained by combining sets of *n* input bits at a time, where *n* is specified by you. Input bits are read in each time the x_2 clock input goes high. Once *n* serial bits have been read in, the output symbol value is output upon occurrence of the *next* input clock pulse. Output clock pulses are provided for each output symbol. Proper timing is your responsibility. It is recommended that input bits be represented by an integer number of simulation steps.

x_1 = Input serial bit n = number of bits

x_2 = Input bit clock (impulse train)

y_1 = Output symbol number (0, 1, .., 2^n-1)

y_2 = Output data clock (impulse train)

Bit Order

LSB First

Indicates that the first out of *n* input bits is considered the LSB.

MSB First

Indicates that the first out of *n* input bits is considered the MSB.

Bits per Symbol

Indicates the number of bits *n* per parallel symbol. Valid range is 2 to 16.

State Machine

This block implements a digital State Machine. Each time the clock input for the block is “high”, the block will transition to a new “state” depending on its current state and the value of the block’s input. The block can control up to 10 outputs.

An external State Transition Map File is used to control the block’s behavior. For each possible combination of “current state/input value”, this file specifies the next state and unique values for each of the block’s *N* outputs.

x_1 = Input integer value (range [0, *k*-1])

x_2 = Input clock (high ≥ 1) (impulse train)

y_1 = Current state [0, L-1]

y_2 = Output clock (impulse train)

$y_{3...N+2}$ = State machine outputs

Number of Inputs

Specifies the number k of discrete input values for the block. The input is assumed to be integer values in the range of [0, k-1]. This value should match the number of inputs specified in the State Map file.

Number of Outputs

Specifies the number N of desired state machine outputs. The maximum value for N is 10. This value must be specified numerically (global variable not allowed), and should match the number of outputs specified in the State Map file.

Number of States

Specifies the number of states L for the state machine. This value should match the number of states specified in the State Map file. The state values used in the map file are assumed to be in the range of [0, L-1].

Select File

Opens the Select File dialog box for selecting a state machine map file.

Browse File

Opens the selected state machine map file using Notepad.

State Map File Path

Specifies the DOS path to the desired state machine map file. A description of the file format is provided below:

File header *(anything)*
k N L *(#inputs, #outputs, #states)*
Column header line *(used to improve file readability)*
current state, input value, new state, out #1, out #2 ... out #N
...

The *current state* values must be entered in increasing order. The *input value* entries may be entered in random order. Table entries may be separated by blank spaces, tabs, or commas. The table should contain ($k \cdot$ *#states*) total entries.

An example state machine file corresponding to a V.32 trellis encoder is shown below:

V.32 Trellis Map File (*1st header line)*
16 2 8 *(16 inputs, 2 outputs, 8 states)*

state	input	new state	out #1	out #2	(*2nd header line)*
0	0	0	-4	1	
0	1	0	0	-3	
...	...	...	...	...	
0	15	2	-2	1	

1	0	2	-4	1
...	...	...	...	...
7	15	7	-1	4

In this example, there are 16 possible input values in the range of [0, 15], eight possible state values [0, 7], and two outputs.

Symbol to Bits

This block accepts a symbol number and outputs *n* parallel binary bit streams. The mapping is obtained by decomposing the binary representation of the symbol number. You can specify the number of output data streams.

x = Input symbol number (0, 1, .., 2^n-1)

y_1 = Output bit stream #1

.. ...

y_n = Output bit stream #*n*

Bit Order

LSB First

Indicates that y_1 is considered the LSB.

MSB First

Indicates that y_1 is considered the MSB.

Number of Output Bits

Indicates the number *n* of output binary data streams. The valid range is 2 to 16.

Toggle

This block accepts an impulse clock signal and produces an OFF / ON level output [0, 1]. Each time an impulse is received at the input, the output state of the block toggles. Provisions are made for allowing a "dead time" after each impulse before a new impulse is recognized. Any input larger than 0.5 is considered "high".

x = Input impulse train ("high" is > 0.5)

y_1 = OFF / ON [0, 1] toggle output

Number Dead Sim Steps

Indicates the number *n* of "dead time" steps required after occurrence of a pulse before the block will recognize a new input pulse. The valid range is 0 to 65,535.

Initial State

Indicates the initial state of the block as either 0 (OFF) or 1 (ON).

Unbuffer

This block accepts an input data vector of the specified size and outputs a serial data stream. The reading of the input vector is controlled via an external read strobe. The serial output can be controlled using either an internal or external clock. If a new read strobe (frame clock) occurs before all the previous data has been output, the remaining data in the buffer is lost and replaced by new values.

x_1 = Frame Clock (impulse)

x_2 = Input data vector (size *N*)

x_3 = External clock [optional] (impulse train)

y_1 = Output serial data

y_2 = Output clock (impulse train)

Output Mode

FIFO

Indicates that the first element of the input data vector is output first.

LIFO

Indicates that the last element of the input data vector is output first.

Output Timing

Internal

Indicates internal clock timing. An output rate needs to be specified when in this mode.

External

Indicates external timing. An external clock must be provided at the *x*3 input when in this mode.

Buffer Size

Specifies the size *N* of the output buffer. The valid range is 1 to 8192.

Cyclic Output

When selected, the block's output will repeat in cyclic fashion from the beginning once the last element has been output. If not selected, the block's output will be held at the last element once all the data has been output.

Output Rate

Specifies the output rate for the serial data stream in hertz when in internal timing mode.

Encode / Decode category

Blocks in the Encode / Decode category include `Block Interleaver`, `Convolutional Encoder`, `Convolutional Interleaver`, `Depuncture`, `Gray Decoder`, `Gray Encoder`, `Hamming Decoder`, `Hamming Encoder`, `Puncture`, `Reed-Solomon Decoder`, `Reed-Solomon Encoder`, `Trellis Decoder`, `Trellis Encoder`, `Viterbi Decoder (Hard)`, and `Viterbi Decoder (Soft)`.

Block Interleaver

This block implements block interleaving or de-interleaving. It is normally used to scramble encoded channel bits in order to spread out the effects of burst type channel errors. It is particularly useful when used in conjunction with a `Convolutional Encoder`, `Viterbi Decoder (Hard)`, or `Viterbi Decoder (Soft)` block.

When interleave mode is set, the input data is written in rows and read out by columns, starting at location [1, 1]. The opposite occurs when deinterleave mode is set. Since an entire block of data must be received before the output can start, there is an initial delay equal to the number of cells in the block. The data is read in when the input clock goes high. The output clock starts after the appropriate delay amount.

The `Block Interleaver` block accepts a real value and outputs a real value. The internal buffer size (rows x columns) has an upper limit of 32,767.

x_1 = Input data stream

x_2 = Input data clock (0, 1) (impulse train)

y_1 = Output data stream

y_2 = Output data clock (0, 1) (impulse train)

Example: Assuming a 4 x 3 block interleaver, the following input sequence (read left to right) results in the output shown:

Input: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 …

Output: 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11 …

Mode

Interleave

Indicates that data is read-in by row and output by column.

Deinterleave

Indicates that data is read-in by column and output by row.

Rows

Specifies the number of rows used in the block. The max number of rows allowed is 16,384.

Columns

Specifies the number of columns used in the block. The max number of columns allowed is 16,384.

Convolutional Encoder

This block implements a convolutional encoder. Block parameters include the numbers of input bits (k) and coded bits (n), the encoder constraint length (L), the input bit rate, and the generator coefficients.

The `Convolutional Encoder` block implements a single shift register of kL length internally. At simulation start, the shift register is initialized to all zeros. The coded output bits are produced in serial fashion, with the first output bit coming from generator coefficient #1. There is no internal block delay between an input bit and the corresponding first output coded bit when $k = 1$. For larger values of k, the first output bit is produced as soon as all k input bits have been clocked in. The default generator coefficients (133, 171) represent an often used set of coefficients for a rate ½ convolutional code.

This block attempts to generate an output clock and data stream at a rate nR/k, where R is the input bit rate. Note: If the output (coded) bit rate does not divide evenly into the simulation rate, the encoder block will still operate but the output bits may not be equally spaced.

x_1 = Input data bit stream

x_2 = Input data clock (impulse train)

y_1 = Coded output bit stream

y_2 = Output data clock (impulse train)

No. Information Bits

Specifies the number of input bits k. Valid range is from 1 to 7.

No. Coded Bits

Specifies the number of coded output bits n. Valid range is from 1 to 8.

Constraint Length

This parameter specifies the constraint length L of the encoder, and represents the size of the internal buffer in words of size k. The memory size m of the encoder is simply L-1. The maximum allowed value of kL is 15.

Input Bit Rate

Specifies the bit rate *R* of the incoming data in hertz.

Generator Coefficients

Specifies the value of each generator coefficient in octal format.

Convolutional Interleaver

This block implements convolutional interleaving or de-interleaving. It is normally used to scramble encoded channel bits in order to spread out the effects of burst type channel errors. It is particularly useful when used in conjunction with the `Convolutional Encoder`, `Viterbi Decoder (Hard)`, or `Viterbi Decoder (Soft)` blocks.

In convolutional interleaving data is written into rows of increasing buffer size, starting with a no delay path. The depth of the interleaver (number of rows) and the row increment can both be set by the user. When interleave mode is set, the input data is written in rows of increasing length. The opposite occurs when deinterleave mode is set, i.e. the size of the row decreases from row to row, with the last row being a straight through path. An external input clock controls reading of the data.

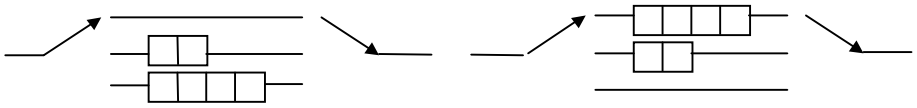
The `Convolutional Interleaver` block accepts a real value and outputs a real value.

x_1 = Input data stream

x_2 = Input data clock (0, 1) (impulse train)

y_1 = Output data stream

y_2 = Output data clock (0, 1) (impulse train)



Interleaver *Deinterleaver*

Example: Assuming a [3,2] `Convolutional Interleaver` (Depth= 3, Increment= 2), the following input sequence (read left to right) results in the output shown below:

Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 …

Output: 1, 0, 0, 4, 0, 0, 7, 2, 0, 10, 5, 0, 13, 8, 3, 16, 11, 6, 19, 14, 9 …

Mode

Interleave

Indicates that rows increase in length from one row to the next.

Deinterleave

Indicates that rows decrease in length from one row to the next.

Rows

Specifies the number of rows used in the block. The max number of rows allowed is 16,384.

Row Increment

Specifies the cell increment from row to row. The max increment allowed is 256.

Depuncture

This block performs depuncturing of an input data vector as specified by an external puncture pattern file. This operation is commonly used in conjunction with a coding block to achieve a variety of different code rates.

At each external frame clock event, the `Depuncture` block reads in a previously punctured data vector and inserts erasures at all the punctured locations as specified by the puncture pattern. As each element of the input vector is read, successive elements of the puncture mask are read and used to determine where to insert the erasures. A mask value of 1 indicates that an element was transmitted, while a 0 indicates that an element was omitted and thus an erasure should be inserted. The mask is applied in cyclical fashion (that is, once the end of the mask is reached, it restarts from the beginning).

Block parameters include the output vector size, the number of encoders used, erasure value, and the file path to the puncture specification file. The number of encoders information is only used for display purposes to compute the effective coding rate provided by the puncturing.

x_1 = Frame clock (impulse)

x_2 = Input data vector (size M)

y_1 = Frame clock (impulse)

y_2 = Output data vector (size N)

Number of Encoder Outputs

Specifies the number of encoder data streams, including any systematic output, present in the input data vector. This information is used to compute the effective code rate provided by the puncturing step.

Output Vector Size

Specifies the size N of the output data buffer. The valid range is 1 to 100000.

Compute Input Vector Size

Provides an indication of the input vector size and effective code rate by reading and processing the specified puncture pattern.

Select File

Opens the Select File dialog box for selecting the desired Puncture Pattern definition file.

Browse File

Opens the selected Puncture Pattern file using Notepad.

Puncture Pattern File Path

Specifies the path and filename for the puncture pattern external specification file. The puncture pattern file format is described below:

Header line (can be anything)
# of entries *(period of pattern)*
mask value #1
mask value #2, mask value #3
etc…

Multiple entries are allowed per line. Supported delimiters include commas, blank spaces, tabs and carriage returns. For additional information regarding the puncture pattern format, please refer to the `Puncture` block description.

Gray Map

This block performs Gray mapping (encoding) on the input signal. The input is first rounded to the closest integer and then encoded. Gray coding is used to map neighboring integer input values into encoded symbols that differ by only one bit.

For example, this block will map the input values below

0, 1, 2, 3, 4, 5, 6, 7

to the following

0, 1, 3, 2, 6, 7, 5, 4

Note: To implement a Gray encoded constellation when using a linear constellation map file at the modulator (e.g. the modulator's map file is simply {0, 1, 2, 3, 4, 5, 6, 7} in the case of 8-PSK), one must actually use a `Gray Reverse Map` block at the transmitter, and a `Gray Map` block at the receiver. This is because, in this case, the modulator block uses the input symbol value as the constellation point location. For example, the correct 8-PSK Gray mapping for an input data symbol value of 4 (100 binary) should be the last constellation point (-22.5 degrees), i.e. symbol location number 7 (range is [0, 7]). As one can see, the correct mapping in this case is provided by the Gray Reverse Map block, not the Gray Map block.

x = Input value

y = Gray encoded integer

Gray Reverse Map

This block performs Gray reverse mapping (decoding) of the input signal. The input is first rounded to the closest integer and then decoded. Gray coding is used to map neighboring integer input values into encoded symbols that differ by only one bit.

For example, this block will map the input values

0, 1, 2, 3, 4, 5, 6, 7

to the following

0, 1, 3, 2, 7, 6, 4, 5

Note: To implement a Gray encoded constellation when using a linear constellation map file at the modulator (e.g. the modulator's map file is simply {0, 1, 2, 3, 4, 5, 6, 7} in the case of 8-PSK), one must actually use a `Gray Reverse Map` block at the transmitter, and a `Gray Map` block at the receiver. This is because, in this case, the modulator block uses the input symbol value as the constellation point location. For example, the correct 8-PSK Gray mapping for an input data symbol value of 4 (100 binary) should be the last constellation point (-22.5 degrees), i.e. symbol location number 7 (range is [0, 7]). As one can see, the correct mapping in this case is provided by the `Gray Reverse Map` block, not the `Gray Map` block.

x = Gray encoded value

y = Output value

Hamming Decoder

This block implements a Hamming decoder. Hamming codes are a form of block codes, and operate on binary symbols (bits). In a Hamming (n, k) code, n - k parity bits are added to the k input bits, resulting in a total of n output bits. Hamming codes are capable of correcting a single bit error in each data frame.

This block supports Hamming codes in the range of (3, 1) to (255, 247). This block accepts an input coded vector of size n and outputs a data vector of size k. The input is assumed to be in systematic form (data bits followed by parity bits). An external clock is used to signal when each decoding operation should take place.

Note: The `Buffer` block can be used to pack serial bits into the required vector format, while the `Unbuffer` block can be used for the reverse operation.

x_1 = Input coded vector (size n)

x_2 = Input clock (pulse)

y_1 = Output data vector (size k)

y_2 = Output clock (pulse)

Hamming Code Size

Specifies the (*n*, *k*) size of the Hamming code. The following selections are available: (3, 1), (7, 4), (15, 11), (31, 26), (63, 57), (127, 120) and (255, 247).

Hamming Encoder

This block implements a Hamming encoder. Hamming codes are a form of block codes, and operate on binary symbols (bits). In an Hamming (*n*, *k*) code, *n* - *k* parity bits are added to the *k* input bits, resulting in a total of *n* output bits. Hamming codes are capable of correcting a single bit error in each data frame.

This block supports Hamming codes in the range of (3, 1) to (255, 247). This block accepts an input data vector of size *k* and outputs a coded vector of size *n*. The output is made in systematic form (data bits followed by parity bits). An external clock is used to signal when each encoding operation should take place.

Note: The `Buffer` block can be used to pack serial bits into the required vector format, while the `Unbuffer` block can be used for the reverse operation.

x_1 = Input data vector (size *k*)

x_2 = Input clock (pulse)

y_1 = Coded output vector (size *n*)

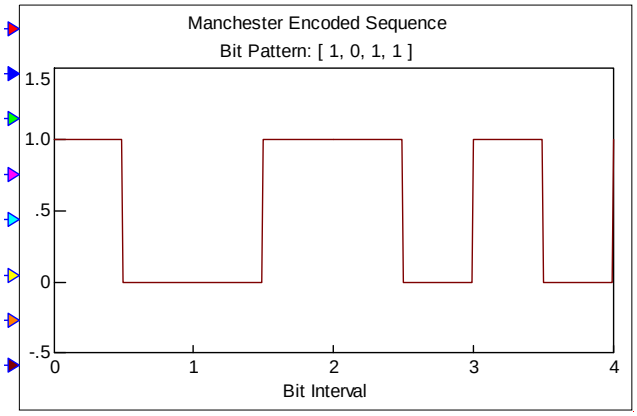
y_2 = Output clock (pulse)

Hamming Code Size

Specifies the (*n*, *k*) size of the Hamming code. The following selections are available: (3, 1), (7, 4), (15, 11), (31, 26), (63, 57), (127, 120) and (255, 247).

Manchester Encoder

This block implements Manchester encoding of the input bit stream, which always includes a data transition at the half point of the bit interval. With Manchester encoding, zeros and ones are represented as shown in the figure below.



In order for the block to operate properly, the user must supply the input bit rate. It's recommended that an even number of samples per bit be specified when using this block, although the above is not required.

x_1 = Input bit value

x_2 = Input clock (pulse)

y_1 = Manchester encoded output

y_2 = Output clock (pulse)

Input Bit Rate

Specifies the input bit rate for the block in *bits/sec*.

Output Mid-Period Clock Pulse

When selected, forces the block to produce an output clock pulse at both the bit period boundary and the bit midway transition point.

Output Mode

Bilevel

Specifies the output signal levels to be {-1, 1}.

Binary

Specifies the output signal levels to be {0, 1}.

Puncture

This block performs puncturing of an input data vector as specified by an external puncture pattern file. This operation is commonly used in conjunction with a coding block to achieve a variety of different code rates.

At each external frame clock event, the block reads in a data vector and applies the specified puncture pattern, thus removing a number of the elements of the original vector. As each element of the input vector is read, successive elements of the puncture mask are read and used to determine whether each data element is to be kept or removed. A mask value of 1 indicates to keep the element, while a 0 is used to indicate its removal. The mask is applied in cyclical fashion (that is, once the end of the mask is reached, it restarts from the beginning). The new data vector thus formed is then output.

Block parameters include the input vector size, the number of encoders used, and the file path to the puncture specification file. The number of encoders information is only used for display purposes to compute the effective coding rate provided by the puncturing.

x_1 = Frame clock (impulse)

x_2 = Input data vector (size N)

y_1 = Frame clock (impulse)

y_2 = Output data vector (size M)

Number of Encoder Outputs

Specifies the number of encoder data streams, including any systematic output, present in the input data vector. This information is used to compute the effective code rate provided by the puncturing step.

Input Vector Size

Specifies the size N of the input data buffer. The valid range is 1 to 100000. This value should be a multiple of the pattern period as specified in the puncture pattern description file, but is not required to be so.

Compute Output Vector Size

Provides an indication of the output vector size and effective code rate by reading and processing the specified puncture pattern.

Select File

Opens the Select File dialog box for selecting the desired Puncture Pattern definition file.

Browse File

Opens the selected Puncture Pattern file using Notepad.

Puncture Pattern File Path

Specifies the path and filename for the puncture pattern external specification file. The puncture pattern file format is described below:

Header line *(can be anything)*
# of entries *(period of pattern)*
mask value #1
mask value #2, mask value #3
etc…

Multiple entries are allowed per line. Supported delimiters include commas, blank spaces, tabs and carriage returns.

In the following example, a rate 1/3 encoder is assumed to provide the block's vector input, being comprised of systematic (data) bits, encoder #1 bits and encoder #2 bits (i.e. the vector is comprised as follows: [data, enc#1, enc#2, data, enc#1, enc#2, data, … enc#2]). The pattern period in this case is 24 bits. The puncture pattern specified below retains all the systematic bits (column #1) and keeps only two parity bits out of each group of 24 total bits, resulting in a rate 4/5 code (for every 8 data bits, 10 output bits are generated).

Rate 4/5 puncture pattern (keeps two parity bits for every eight data bits)

24 *(pattern period)*
1 0 0 *(keep input vector bit #1, discard bits #2 and #3)*
1 0 0 *(keep input vector bit #4, discard bits #5 and #6)*
1 0 0
1 0 0 . . .
1 0 0
1 0 1 *(keep input vector bit #16, discard bit #17 and keep #18)*
1 1 0 *(keep input vector bits #19 and #20, discard bit #21)*
1 0 0 . . .

Reed-Solomon Decoder

This block implements a Reed-Solomon (RS) decoder. RS codes are a type of block code and are a subclass of BCH codes. RS codes use non-binary symbols from a Galois Field (GF) and are systematic. In an RS(*n*, *k*) code, *n* - *k* parity symbols are added at the end of the *k* input symbols, resulting in a total of *n* output symbols. An RS(*n*, *k*) code is capable of correcting up to (*n* - *k*)/2 errors.

Standard RS block sizes are 2^*M* -1, where *M* is the symbol size in bits (e.g. *n*= 255 for *M*= 8). When *n* is less than 2^*M* -1, the corresponding RS code is referred to as a shortened code. Such codes provide a higher degree of error protection by applying the FEC properties of the code over fewer transmitted symbols. Shortened codes are implemented internally by zero padding the input prior to the encoding stage, and then omitting these zeros from the systematic coded output. An example is the RS(204, 188) used in the DVB specification. This code is implemented by using 51 padded zeros and an RS(255, 239) code. The resulting code is capable of correcting up to eight errors out of the 204 output symbols.

Block parameters should match those used by the corresponding RS encoder. These include the symbol size (*M*) in bits, which also specifies the sizes of the underlying RS block length and GF size, and the number of information and coded symbols (*k* and *n* respectively). This block accepts an input coded data vector of size *n* and outputs an information data vector of size *k*. An input

clock is used to trigger each block decoding operation. The `Buffer` and `Unbuffer` blocks can be used to pack serial symbols into the required vector format.

The error output indicates the total number of corrected symbol errors found in each block. If the decoder detects an uncorrectable number of symbol errors, then it simply outputs the received systematic symbols and the error output is set to a value of -1.

x_1 = Input coded data vector (size *n*) (Integers in range [0, 2^M-1])

x_2 = Input clock (pulse)

y_1 = Decoded output vector (size *k*)

y_2 = Output clock (pulse)

y_3 = Error count (-1 indicates RS failure)

Symbol Size

Specifies the symbol size *M* for the RS code in *bits*. The resulting RS code has an underlying block length of 2^*M* -1, and operates over GF(2^*M*). The valid range for M is from 3 to 10.

Information Symbols

Specifies the number of input information symbols *k* for each RS block. The valid range for *k* is from 1 to 2^*M* -2.

Coded Symbols

Specifies the number of coded output symbols *n*. The valid range for *n* is from *k* + 1 to 2^*M* - 1.

Advanced Settings

These settings do not need be modified for most RS encoding/decoding applications. One exception is the CCSDS standard, which specifies an RS(255, 223) code with a value of B0= 112 for the first root of the generator polynomial and a value of PRIM= 11 for the power of alpha used to generate the roots.

For given values of B0 and PRIM, the generator polynomial for the RS code will be:

@^PRIM*B0, @^PRIM*(B0+1), @^PRIM*(B0+2)...@^PRIM*(B0 + *n* - *k*)

where "@" represents a lower case alpha.

First root of generator polynomial

This setting lets you specify the first root of the generator polynomial (B0) in alpha form. The default value is 1.

Power of alpha used for roots of generator polynomial

This setting lets you specify the power of alpha (PRIM) used to generate the roots of the generator polynomial in alpha form. The default value is 1.

The primitive polynomials used by the RS Decoder block are shown below (for reference see Lin & Costello, *Error Control Coding*, Appendix A):

M=3: 1 + x + x^3

M=4: 1 + x + x^4

M=5: 1 + x^2 + x^5

M=6: 1 + x + x^6

M=7: 1 + x^3 + x^7

M=8: 1+x^2+x^3+x^4+x^8

M=9: 1+x^4+x^9

M=10: 1+x^3+x^10

Note: Portions of the Reed-Solomon library module are Copyright 1999 Phil Karn, and are provided under the terms of the Lesser General Public License (LGPL). The source code to the core Reed-Solomon processing routines, and a copy of the LGPL, are included with the Embed/Comm distribution materials.

Reed-Solomon Encoder

This block implements a Reed-Solomon (RS) encoder. RS codes are a type of block code and are a subclass of BCH codes. RS codes use non-binary symbols from a Galois Field (GF) and are systematic. In an RS(*n*, *k*) code, *n* - *k* parity symbols are added at the end of the *k* input symbols, resulting in a total of *n* output symbols. An RS(*n*, *k*) code is capable of correcting up to (*n* - *k*)/2 errors.

Standard RS block sizes are 2^*M* -1, where *M* is the symbol size in bits (e.g. *n*= 255 for *M*= 8). When *n* is less than 2^*M* -1, the corresponding RS code is referred to as a shortened code. Such codes provide a higher degree of error protection by applying the FEC properties of the code over fewer transmitted symbols. Shortened codes are implemented internally by zero padding the input prior to the encoding stage, and then omitting these zeros from the systematic coded output. An example is the RS(204, 188) used in the DVB specification. This code is implemented by using 51 padded zeros and an RS(255, 239) code. The resulting code is capable of correcting up to eight errors out of the 204 output symbols.

Block parameters include the symbol size (*M*) in bits, which specifies the sizes of the underlying RS block length and GF size, and the number of information and coded symbols (*k* and *n* respectively). This block accepts an input data vector of size *k* and outputs a data vector of size *n*. An input clock is used to trigger each block encoding operation. The `Buffer` and `Unbuffer` blocks can be used to pack serial symbols into the required vector format.

x_1 = Input data vector (size *k*)

x_2 = Input clock (pulse)

y_1 = Coded output vector (size *n*)

y_2 = Output clock (pulse)

Symbol Size

Specifies the symbol size *M* for the RS code in *bits*. The resulting RS code has an underlying block length of 2^*M* -1, and operates over GF(2^*M*). The valid range for M is from 3 to 10.

Information Symbols

Specifies the number of input information symbols *k* for each RS block. The valid range for *k* is from 1 to 2^*M* -2.

Coded Symbols

Specifies the number of coded output symbols *n*. The valid range for *n* is from *k* + 1 to 2^*M* - 1.

Advanced Settings

These settings do not need be modified for most RS encoding/decoding applications. One exception is the CCSDS standard, which specifies an RS(255, 223) code with a value of B0= 112 for the first root of the generator polynomial and a value of PRIM= 11 for the power of alpha used to generate the roots.

For given values of B0 and PRIM, the generator polynomial for the RS code will be:

@^PRIM*B0, @^PRIM*(B0+1), @^PRIM*(B0+2)...@^PRIM*(B0 + *n* - *k*)

where "@" represents a lowercase alpha.

First root of generator polynomial

This setting lets you specify the first root of the generator polynomial (B0) in alpha form. The default value is 1.

Power of alpha used for roots of generator polynomial

This setting lets you specify the power of alpha (PRIM) used to generate the roots of the generator polynomial in alpha form. The default value is 1.

The primitive polynomials used by the RS Decoder block are shown below (for reference, see Lin & Costello, *Error Control Coding*, Appendix A):

M=3: 1 + x + x^3
M=4: 1 + x + x^4
M=5: 1 + x^2 + x^5
M=6: 1 + x + x^6
M=7: 1 + x^3 + x^7
M=8: 1+x^2+x^3+x^4+x^8
M=9: 1+x^4+x^9
M=10: 1+x^3+x^10

Note: Portions of the Reed-Solomon library module are Copyright 1999 Phil Karn, and are provided under the terms of the Lesser General Public License (LGPL). The source code to the core Reed-Solomon processing routines, and a copy of the LGPL, are included with the Embed/Comm distribution materials.

Trellis Decoder

This block decodes trellis-coded modulated signals. Block parameters include the number of data bits (*k*), coded bits (*n*), the number of states in the trellis, and the trellis truncation length. There is a delay through this block equivalent to the specified truncation length. The specific trellis state transition mapping and corresponding constellation output points are provided via an external file.

The `Trellis Decoder` block accepts a baseband (I, Q) pair as its input and outputs a decoded symbol number data stream.

x_1 = I channel input

x_2 = Q channel input

x_3 = Data clock (impulse train)

y_1 = Output symbol value

y_2 = Output data clock

y_3 = Best metric value

Number of Input Bits

Specifies the number of input bits *k*. Valid range is from 1 to 7.

Number of Output Bits

Specifies the number of coded bits *n*. Valid range is from 1 to 8.

Number of States

Specifies the number of states in the trellis. This value must be a power of 2.

Trellis Truncation Length

Specifies the truncation length of the decoded trellis. The maximum value is 100.

Trellis File Path

Specifies the DOS path to the desired trellis state transition map file. The file provides both the state transition mapping and the associated (I, Q) channel outputs. For a description of the file format, refer to the `Trellis Encoder` block description.

Trellis Encoder

This block implements trellis-coded modulation. In trellis-coded modulation, the encoding process and modulation scheme are designed together. The mapping of the output constellation points is selected to maximize the minimum Euclidean distance between coded signal pairs. The CCITT V.32 modem communication standard is an example of trellis-coded modulation. Block parameters include the numbers of input data bits (*k*) and encoded output bits (*n*), and the number of states in the trellis.

Unlike the `Convolutional Encoder` block, this block accepts *k* input bits simultaneously as a symbol value. The coded output symbol value (*n* bits) is then determined internally and the corresponding constellation point in (I, Q) space is output. There is no delay through this block. The specific trellis state transition mapping and corresponding constellation output points are provided via an external file.

The `Trellis Encoder` block accepts a symbol number and outputs a baseband (I, Q) pair corresponding to the output constellation point. This block is typically followed by an `I/Q Mapper` block.

x_1 = Input symbol number

x_2 = Input data clock (impulse train)

y_1 = I channel output

y_2 = Q channel output

y_3 = Output data clock

Number of Input Bits

Specifies the number of input data bits *k*. Valid range is from 1 to 7.

Number of Output Bits

Specifies the number of coded output bits *n*. Valid range is from 1 to 8.

Number of States

Specifies the number of states in the trellis. This value must be a power of 2.

Trellis File Path

Specifies the DOS path to the desired trellis state transition map file. The file provides both the state transition mapping and the associated (I, Q) channel outputs. The file format is described below:

File header
K n #states
Column header line
current state input value new state I output Q output
...

The *current state* values must be entered in increasing order. The *input value* entries may be entered in random order. The table should contain a total of *N* entries, where

$N = 2^k \cdot \#states$

Table entries may be separated by blank spaces, tabs, or commas. An example of a trellis map file is shown below.

V.32 Trellis Map File (*1st header line)*
4 5 8 *(4 input bits, 5 output bits, 8 states)*

state	input	new state	I out	Q out	(*2nd header line)*
0	0	0	-4	1	
0	1	0	0	-3	
...	...	...	...	...	
0	15	2	-2	1	
1	0	2	-4	1	
...	...	...	...	...	
7	15	7	-1	4	

In this example, there are four input bits indicating an input symbol range of 0 to 15. The number of coded output bits is five, which specifies one of 32 different constellation points. This particular trellis has eight states (0 - 7).

Viterbi Decoder (Hard)

This block implements a hard decision Viterbi decoder to decode convolutionally encoded data. Block parameters include the numbers of information bits (k), coded bits (n), the encoder constraint length (L), the trellis truncation length M, the output bit rate, and the generator coefficients.

This block takes as input a binary stream {0, 1} and outputs a binary stream {0, 1}. Hamming distance is used as the internal metric. Assuming that no uncorrected channel errors have occurred, the best metric output (y_2) is an indication of the number of channel bit errors. The decoding delay through this block, from when the first coded bit is received to when the first information bit is produced, is equal to ($kMT - kT/n$), where k and n are the number of information bits and coded bits respectively, M is the Truncation Length, $T = 1/R$ and R is the information bit rate.

x_1 = Input coded bit stream {0, 1}

x_2 = Input coded clock (impulse train)

y_1 = Decoded bit stream {0, 1}

y_2 = Decoded data clock (impulse train)

y_3 = Best path metric (Hamming distance)

No. Information Bits

Specifies the number of information bits k. Valid range is from 1 to 7.

No. Coded Output Bits

Specifies the number of coded bits *n*. Valid range is from 1 to 8.

Constraint Length

Specifies the constraint length *L* of the associated encoder. The memory size *m* of the encoder is simply *L*-1. The maximum allowed value of *kL* is 15.

Trellis Truncation Length

Specifies the trellis truncation length *M* in words of size *k*. The maximum allowed value of *kM* is 96.

Output Bit Rate

Specifies the bit rate *R* in hertz of the output decoded data.

Generator Coefficients

Specifies the value of the associated encoder generator coefficients in octal format.

Viterbi Decoder (Soft)

This block implements a soft decision Viterbi decoder to decode convolutionally encoded data. Block parameters include the numbers of input bits (*k*) and coded bits (*n*), the encoder constraint length (*L*), the trellis truncation length (*M*), the number of quantization bits, the output bit rate, and the generator coefficients. An external metric file must also be specified.

This block accepts a real value (typically in the range of [-1, 1]) and outputs a binary stream {0, 1}. The decoding delay through this block, from when the first coded bit is received to when the first information bit is produced, is equal to ($kMT - kT/n$), where *k* and *n* are the number of information bits and coded bits respectively, *M* is the Truncation Length, $T = 1/R$ and *R* is the information bit rate.

x_1 = Input coded data stream

x_2 = Input coded data clock (impulse train)

y_1 = Decoded bit stream (0, 1)

y_2 = Decoded data clock (impulse train)

y_3 = Best path metric

No. Information Bits

Specifies the number of information bits *k*. Valid range is from 1 to 7.

No. Coded Output Bits

Specifies the number of coded bits *n*. Valid range is from 1 to 8.

Constraint Length

Specifies the constraint length *L* of the associated encoder. The memory size *m* of the encoder is simply *L*-1. The maximum allowed value of *kL* is 15.

Trellis Truncation Length

Specifies the trellis truncation length *M*, in words of size *k*. The maximum allowed value of *kM* is 96.

Quantization Bits

Specifies the number of quantization bits used in the decoding process.

Output Bit Rate

Specifies the bit rate of the output decoded data in hertz.

Generator Coefficients

Specifies the value of the associated encoder generator coefficients in octal format.

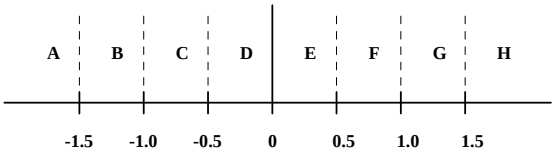
Metric File Path

Specifies the path and filename of the metric file to be used in decoding. The metric file format is described below:

Header line (can be anything)			Q = # quantization bits
Q	*m(A/0)*	*m(A/1)*	T_{AB} = threshold value between
T_{AB}	*m(B/0)*	*m(B/1)*	regions "A" and "B"
T_{BC}	*m(C/0)*	*m(C/1)*	*m(C/1)* = metric for region "C"
...	...	...	given a "1" was sent

Note: File entries can be separated by a comma, tab, or whitespace. A metric value of 0 is considered best.

Example Metric File



Viterbi Decoder Metric File, Q = 3

3,	7.0,	0.0	
-1.5,	6.0,	1.0	For example:
-1.0,	5.0,	2.0	for an input $0.5 \le x < 1.0$,
-0.5,	4.0,	3.0	the metric for a "0" bit is "2.0",
0,	3.0,	4.0	and the metric for a "1" bit is "5.0".
0.5,	2.0,	5.0	
1.0,	1.0,	6.0	
1.5,	0.0,	7.0	

Estimators category

Blocks in the Estimators category include `Average Power (Complex & Real)`, `BER Curve Control`, `BER Control (#Errors)`, `Bit/Symbol Error Rate`, `Correlation (Complex & Real)`, `Delay Estimator`, `Event Time`, `File Correlation (Complex & Real)`, `Frequency Counter`, `Mean`, `Median`, `MinMax`, `Variance`, `Vector Correlation` and `Weighted Mean`.

Average Power (Complex or Real)

These blocks estimate the average (or complex) power of the input (or complex) signal. Two versions of this block exist: one for complex signals and one for real signals. Two power estimation modes are available: running and sliding window. The two modes are described in more detail below.

The output can be reset during the simulation (running mode only) by sending a unity impulse on the reset connector. The reset becomes effective at the next simulation step, at which time the output will represent the power in the first new sample.

x_1 = Input signal ([Re, Im] for complex)

x_2 = Reset signal (resets when $x_2 \geq 1$)

y = Power estimate

$$y[k] = \frac{1}{N} \cdot \sum_{i=k-N-1}^{k} \left| x_1[i] \right|^2$$

N = window size or # of samples since reset

k,i = simulation step indices

Load

1 Ohm

Specifies a load resistance of 1 Ohm.

50 Ohms

Specifies a load resistance of 50 Ohms.

Output Units

dBm

Watts

dBW

Specifies the average power in dBm, watts, or dBW.

Mode

Sliding

Specifies that the average power estimate is computed over a sliding window.

Running

Specifies that the average power estimate is computed using all simulation samples since the last reset pulse or simulation start. The reset signal is optional.

Window Size

Specifies the size of the sliding window averaging buffer in simulation steps. This parameter is available only when sliding mode is activated.

Shift Reg. Initial Value

Specifies the initial value stored in the sliding window buffer at simulation start. This parameter is available only when sliding mode is activated.

BER Control (# Errors)

This block is used to automatically control the generation of BER curves by monitoring an externally supplied running count of observed errors. When the desired number of errors is achieved during an individual run, the block will halt the current run and advance to the next run. In order for this block to function properly, it is necessary to activate the Auto Restart parameter in the Simulation Properties dialog box.

The `BER Control (# Errors)` block allows up to ten consecutive iterations of the simulation, each with its own number of desired error events. The `BER Control (# Errors)` block accepts the current E_s/N_o value (from an `AWGN` block or other custom source), and the current error count and error rate from a `Bit/Symbol Error Rate` block. Care should be taken to match the number of runs in this block with those specified in the `AWGN` block (if used).

This block provides outputs to be used with a `plot` block configured for external trigger, XY plotting, and log Y scaling. The BER curve result is updated at the end of each run in a multi-run scenario (Auto Restart mode). At the end of the last run, an optional written BER summary message is provided, as shown in the figure below.

x_1 = Current E_s/N_o level

x_2 = Error rate estimate (from `Bit/Symbol Error Rate` block)

x_3 = Running input error count (from `Bit/Symbol Error Rate` block)

y_1 = Trigger for BER plot

y_2 = Error rate results for BER plot (y-axis signal - use log scale)

y_3 = SNR data for BER plot (x-axis signal)

Number of Runs

Specifies the number of simulation iterations. The valid range is from 1 to 10.

Mode

This entry is used for label display purposes only and does not affect the block's numeric results.

Bit Error Rate

Forces the use of the Eb/No label in the results summary. Use this setting when providing a reference Eb/No input to the block.

Symbol Error Rate

Forces the use of the Es/No label in the results summary. Use this setting when providing a reference Es/No input to the block.

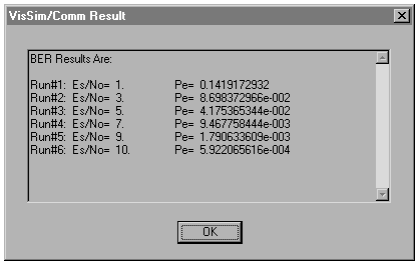
Max Errors

Specifies the desired maximum number of errors for each run.

Note: Due to the block's implementation, it's possible for the actual number of observed errors to occasionally slightly exceed this value.

Show Results

Displays the last set of results obtained using the `BER Control (# Errors)` block, as illustrated in the figure below. The same message is also displayed automatically at the end of the simulation.



Suppress Result Notification

Suppresses the automatic display of the BER results at the end of the run.

BER Curve Control

This block is used to automatically control the generation of BER curves. It allows the user to specify individual run times for each of a BER simulation's multiple runs. In order for this block to function properly, it is necessary to activate the Auto Restart parameter in the Simulation Properties dialog box.

Note: To control a BER simulation by specifying the number of desired errors, please use the `BER Control (# Errors)` block.

The `BER Curve Control` block allows up to ten consecutive iterations of the simulation, each with its own time duration expressed in seconds. The `BER Curve Control` block accepts the current E_s/N_o value (from an `AWGN` block or other custom source) and an output error rate from a `Bit/Symbol Error Rate` block. Care should be taken to match the number of runs in this block with those specified in the `AWGN` block (if used).

This block provides outputs to be used with a `plot` block configured for external trigger, XY plotting, and log Y scaling. The BER curve result is updated at the end of each run in a multi-run scenario (Auto Restart mode). At the end of the last run, an optional written BER summary message is provided, as shown in the figure below.

x_1 = Current E_s/N_o level

x_2 = Error rate estimate (from `Bit/Symbol Error Rate` block)

y_1 = Trigger for BER plot

y_2 = Error rate results for BER plot (y-axis signal - use log scale)

y_3 = SNR data for BER plot (x-axis signal)

Number of Runs

Specifies the number of simulation iterations. The valid range is from 1 to 10.

Mode

This entry is used for label display purposes only and does not affect the block's numeric results.

Bit Error Rate

Forces the use of the Eb/No label in the results summary. Use this setting when providing a reference Eb/No input to the block.

Symbol Error Rate

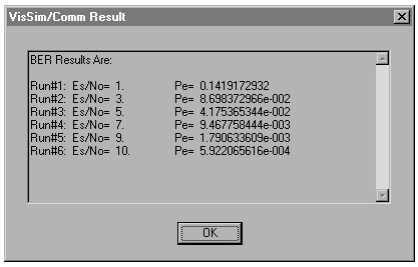
Forces the use of the Es/No label in the results summary. Use this setting when providing a reference Es/No input to the block.

Duration

Specifies each run's duration in seconds. As the SNR is made larger, a longer duration is usually necessary to obtain a reliable error rate estimate.

Show Results

Displays the last set of results obtained using the `BER Curve Control` block, as illustrated in the figure below. The same message is also displayed automatically at the end of the simulation.



Suppress Result Notification

Suppresses the automatic display of the BER results at the end of the run.

Bit/Symbol Error Rate

This block accepts either bits or symbols as input and can output either a Bit Error Rate (BER) or a Symbol Error Rate (SER) by comparing a recovered data stream to a reference data stream.

In order for this block to operate properly, the reference data stream must be delayed by the same amount as the recovered data stream. An external sampling clock must be provided to the `Bit/Symbol Error Rate` block. Sampling at approximately the half symbol point is recommended.

x_1 = Recovered data stream

x_2 = Reference data stream

x_3 = External Clock (0, 1) (impulse train)

y_1 = Error rate (symbol or bit)

y_2 = Error count ((symbols or bits) (optional))

y_3 = Total count ((symbol or bits) (optional))

Count Start Delay

Specifies the initial delay in symbol counts before starting the error counting process. A symbol count occurs each time the sampling clock goes high.

Output Mode

Bit Error Rate

Specifies the output error rate as a BER. In this mode, the total number of bits that are in error within a symbol is counted. The total count output shows the total number of symbols processed times the number of bits/symbol.

Symbol Error Rate

Specifies the output error rate as an SER. In this mode, regardless of how many bits within a symbol are in error, a single symbol error is recorded.

Bits per Symbol

Specifies the number of bits per symbol. This parameter is only available when bit error rate mode is selected.

Correlation

Two versions of this block are provided, one Real and the other Complex. These blocks perform a Real or Complex sliding cross-correlation between the input signal and a reference signal. The cross-correlation is performed over a variable window size. Two modes are available: standard and gated control. In standard correlation mode, the two signals are continuously shifted through

the correlation window. In gated control mode, the input signal is still continuously shifted through the window, but the reference signal is only shifted when the external gate signal is low. This latter mode can be used to implement a matched filter or convolution against a fixed waveform by "sliding in" the desired waveform and then locking the control gate.

Note: The complex version of this block takes the complex conjugate of the reference signal prior to performing the internal complex-multiply operation.

x_1 = Input signal ([Re, Im] for complex version)

x_2 = Reference signal ([Re, Im] for complex version)

x_3 = External gate control {0, 1} (0 = shift in reference; 1 = lock reference)

y = Correlation output ([Re, Im] for complex version)

$$y[k] = \sum_{i=0}^{N-1} (x_1[k-i] \cdot *x_2[j]) \quad j = \begin{cases} k-i & \text{sliding mode} \\ j_0 - 1 - i & \text{gated mode} \end{cases} \qquad N = \text{window size}$$

i, j, k = simulation step indices $\qquad j_0$ = gate lock step#

Mode

Gated Control

Indicates gated correlation mode. In gated correlation, the shifting of the reference signal (x_2) is controlled by the external gate control. When the gate voltage is low, the reference signal is shifted into the correlation buffer, along with the input signal (x_1). When the gate voltage becomes high, the reference signal is no longer shifted and a sliding correlation with the input signal is performed.

Standard

Indicates standard correlation mode. In standard correlation, the two signals are simply shifted into the correlation buffer and continuously correlated. The gate control input has no effect.

Window Size

Specifies the size of the cross-correlation buffer in simulation steps.

Delay Estimator

This block estimates the propagation time delay from input to output in a simulation. The delay is estimated by performing a sliding correlation between the desired output signal and an undelayed version of the input signal (or, reference signal). The output signal can be a distorted version of the input signal. The size of the correlation window is specified as a parameter. An output flag is provided that indicates when the entire delay range was successfully searched. The result flag is 0 during computation and toggles to 1 upon completion. The delay estimate output is 0 at simulation start. The total simulation time should be greater than the sum of the correlation start time, the correlation window size (expressed in seconds), and the maximum delay.

x_1 = Simulation output signal

x_2 = Reference signal

y_1 = Delay estimate

y_2 = Result flag (0, 1)

Window Size

Specifies the size of the correlation window used in simulation steps.

Max. Search Delay

Specifies the upper end of the delay search range. The search range starts with 0 delay. The maximum delay parameter is specified in seconds.

Start Time

Specifies the starting time of the correlation process in seconds. This allows, for example, a tracking loop to complete its acquisition process before the delay estimate is made.

Event Time

This block provides the simulation time at which the input signal value first meets the specified condition.

x = Input signal

y = Time of occurrence

Event Mode

Specifies the logical comparison to be tested on the input signal relative to the threshold value parameter. Available choices include: =, >=, <=, >, and <.

Threshold Value

Specifies the threshold value against which the input signal is compared.

File Correlation

Two versions of this block are provided, one Real and the other Complex. These blocks perform a Real or Complex sliding cross-correlation between the input signal and a fixed reference signal specified via an external file. The cross-correlation is performed over a sliding window of size N, where N is a user-defined parameter. The external file defines the reference signal, which can also be viewed as specifying a series of tap values for the correlator. The tap values from the file can be loaded in either ascending or descending order. In the default mode, the file tap values are assumed to be in increasing time order.

For efficiency purposes, an external enable input is used to control when the correlation calculation is performed. An input clock signal is also required for reading the input signal.

Note: The complex version of this block takes the complex conjugate of the reference signal prior to performing the internal complex-multiply operation.

x_1 = Input signal ([Re, Im] for complex version)

x_2 = Input sample clock {0, 1}

x_3 = Enable input {0= disabled, 1= enabled}

y = Correlation output ([Re, Im] for complex version)

$$y_k = \sum_{i=0}^{N-1} (x_{1[k-i]} \cdot *h_{[N-1-i]}) \qquad N = \text{window size} \qquad h[\,i\,] = i^{\text{th}} \text{ internal tap value}$$

Taps File Order

Reverse Order

Specifies that the reference file tap values are to be loaded in reverse order than normally used for a cross correlation. In this mode the first file tap value corresponds to the last position ($h[N\text{-}1]$) of the internal tap delay structure.

Default

Specifies that the reference file tap values are to be loaded in the order normally used for a cross correlation. In this mode the first file tap value corresponds to the first position ($h[0]$) of the internal tap delay structure.

Window Size

Specifies the size of the cross-correlation buffer (i.e. the number of taps). This value must match the number of taps specified by the external data file.

Select File

Opens the Select File dialog box for selecting the correlator reference file.

Browse File

Opens the selected correlator reference file using Notepad.

Taps File Path

Specifies the DOS path to the desired correlation reference file. The format of the correlation reference file differs slightly between the Real and Complex versions of the block.

For the **Real** version, the format is as follows:

File header *(this can be anything)*

number of taps

tap value #1

tap value #2, tap value #3

...

Multiple tap values can be specified on a given line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 100 characters.

For the **Complex** version, the format is as follows:

File header *(this can be anything)*

number of taps

real tap value #1, imaginary tap value #1

real tap value #2, imaginary tap value #2

...

Only one complex pair may be specified on each line. Valid data delimiters are commas, blank spaces, and tabs. The maximum allowed line length is 100 characters.

Frequency Counter

This block implements a digital frequency counter. The block operates by counting the number of high-to-low and low-to-high signal transitions within a specified time interval. Block parameters include the high and low signal thresholds and the measurement interval if appropriate. The measurement interval can be either specified as a fixed time span, or controlled externally via an enable input.

At the completion of each measurement interval, a new frequency estimate is generated and the result update clock is pulsed high (unity impulse). This block can be triggered repeatedly to provide continuous frequency estimates. Note: The output frequency estimate from this block is always positive.

x_1 = Input signal

x_2 = Trigger or Enable input

y_1 = Frequency estimate

y_2 = Result update clock (impulse train)

Time Span

Specifies the measurement interval in *seconds* when in Time Duration mode.

High Threshold

Specifies the voltage level that the input signal must rise above before it is considered "high".

Low Threshold

Specifies the voltage level that the input signal must drop below before it is considered "low".

Time Span Mode

External Enable

Specifies that the measurement interval is controlled externally via the Enable input. The block begins its frequency measurement once the Enable signal is high (> 0.5), and completes it once the enable signal drops low (< 0.5).

Time Duration

Specifies that the measurement interval is defined by the Time Span parameter. When in this mode, a trigger input must be provided to begin each measurement.

Mean

This block estimates the mean of the input signal. Two modes are available: running and sliding window.

The output can be reset during the simulation (running mode only) by sending a unity impulse on the reset connector. The reset becomes effective at the next simulation step, at which time the mean will equal the input value.

x_1 = Input signal

x_2 = Reset signal (resets when $x_2 \geq 1$)

y = Mean estimate

$$y[k] = \frac{1}{N} \cdot \sum_{i=k-N-1}^{k} x[i]$$

N = window size or number of samples since reset

k,i = simulation step indices

Mode

Sliding Window

Indicates that the mean estimate is computed over a sliding window.

Running

Indicates that the mean estimate is based on all simulation samples after the last reset pulse or simulation start. The reset signal is optional.

Window Size

Specifies the size of the sliding window buffer in simulation steps. This parameter is available only when sliding mode is activated.

Shift Reg. Initial Value

Specifies the initial value stored in the sliding window buffer at simulation start. This parameter is available only when sliding mode is activated.

Median

This block computes the moving median of the input signal. The median is defined as the value where half the data points are larger and half are smaller. The block operates by sorting the input data points in ascending order and returning the value closest to the middle (Odd *N* case). When the size of the sliding window *N* is Even, the returned value is the average of the two center data points.

x = Input data

y = Median output

Window Size

Specifies the size *N* of the sliding window.

Shift Register Initial Value

Specifies the initialization value for the internal shift register contents. Default value is zero.

MinMax

This block outputs the minimum and maximum values of the input signal since the beginning of the simulation or the last reset event. An external input is provided to reset the min and max values during the course of a simulation.

x_1 = Input signal

x_2 = Reset signal (Resets when >= 1)

y_1 = Max value

y_2 = Min value

This block does not have any internal parameters.

Variance

This block estimates the variance and mean of the input signal. Two modes are available: running and sliding window. The mean is also provided as an optional connector, since it is computed in the process of obtaining the variance.

The output can be reset during the simulation (running mode only) by sending a unity impulse on the reset connector. The reset becomes effective at the next simulation step, at which time the variance will be reset to zero and the mean will equal the input value.

x_1 = Input signal

x_2 = Reset signal (resets when $x_2 \geq 1$)

y_1 = Variance estimate

y_2 = Mean estimate (optional)

$$y_{[k]} = \frac{1}{N} \cdot \sum_{i=k-N-1}^{k} x_{[i]}^2 - \left(\frac{1}{N} \cdot \sum_{i=k-N-1}^{k} x_{[i]} \right)^2$$

$$y_{2[k]} = \frac{1}{N} \cdot \sum_{i=k-N-1}^{k} x_{[i]}$$

N = window size or number of samples since reset

k,i = simulation step indices

Mode

Sliding Window

Indicates that the mean and variance estimates are computed over a sliding window.

Running

Indicates that the mean and variance estimates are based on all simulation samples since the last reset pulse or simulation start. The reset signal is optional.

Window Size

Specifies the size of the sliding window buffer in simulation steps. This parameter is available only when sliding window mode is activated.

Initial Value

Specifies the initial value stored in the sliding window buffer at simulation start. This parameter is available only when sliding window mode is activated.

Vector Correlation

This block performs a real valued sliding correlation between the input signal and an external data vector. The correlation is performed over a sliding window of size N, where N represents the size of the external vector and is a user-defined parameter.

For efficiency purposes, an external enable input is used to control when the correlation calculation is performed. The calculation is performed only when the enable input is high (>= 1). An input clock signal must be provided for the input signal.

x_1 = Input signal

x_2 = Input sample clock {0, 1} (impulse train)

x_3 = Vector data [size N]

x_4 = Enable input {0, 1}

y = Correlation output

$$y_k = \sum_{i=0}^{N-1} (x_{1[k-i]} \cdot x_3[i]) \quad N = \text{window size} \qquad x_3[\ i\ ] = i^{\text{th}} \text{ vector element}$$

Window Size

Specifies the size of the internal correlation buffer (i.e. the number of internal taps). This value must match the size of the external data vector.

Weighted Mean

This block computes the weighted mean of the input signal. Two modes are available: running and sliding window. The weighted mean can also be reset during the course of the simulation (running mode only) by sending a unity impulse on the reset connector. The reset becomes effective at the next simulation step, at which time the mean will equal the input value.

x_1 = Input signal

x_1 = Input weight (w in formula below)

x_3 = Reset signal (resets when $x_3 \geq 1$)

y = Weighted mean

$$y_{[k]} = \frac{\sum_{i=k-N-1}^{k} x_{[i]} \cdot w_{[i]}}{\sum_{i=k-N-1}^{k} w_{[i]}} \qquad \sum_{i=k-N-1}^{k} w_{[i]} \neq 0 \;;\; y_{[k]} = 0 \quad \text{otherwise}$$

N = window size or number of samples since reset

k,i = simulation step indices

Mode

Sliding Window

Indicates that the weighted mean is computed over a sliding window.

Running

Indicates that the weighted mean is based on all simulation samples after the last reset pulse or simulation start. The reset signal is optional.

Window Size

Specifies the size N of the sliding window buffer in simulation steps. This parameter is available only when sliding mode is activated.

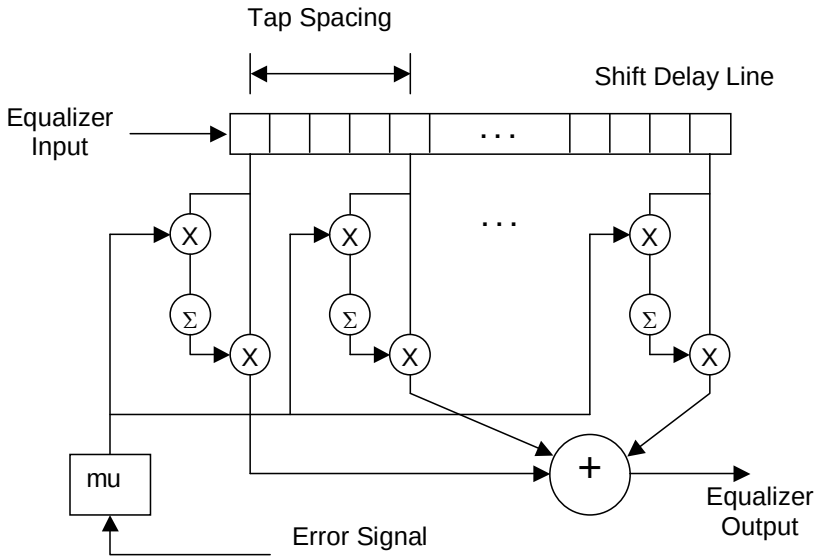
Filters category

Blocks in the Filters category include `Adaptive Equalizer (Complex)`, `Adaptive Equalizer (Real)`, `FIR`, `File FIR`, `IIR`, `Pulse Shaping Filter`, `Sampled FIR`, `Sampled File FIR`, and `MagPhase`.

Adaptive Equalizer (Complex or Real)

This block implements a conventional or fractionally-spaced adaptive equalizer suitable for use with both analog and digital waveforms. Two versions of this block exist, one complex and the other real.

Key block parameters include the total number of taps N, the number of taps per symbol, initial tap values, and the convergence coefficient “mu”. The basic equalizer structure is shown below. Each cell in the equalizer’s internal shift delay line corresponds to a single simulation step. The tap spacing is a derived internal parameter obtained by looking at the input symbol rate and the desired number of taps per symbol. This block requires that the tap spacing be an integer number of simulation time steps. This can be achieved my making the simulation rate an integer multiple of the symbol rate times the number of taps per symbol.



This block uses a Least Mean Square (LMS) convergence algorithm to adapt the tap values with the goal of minimizing the average error. The user is responsible for providing a suitable error input to the block, for example the error vector between a received point in the IQ plane and the closest constellation point. The error value is only read when an internal/external clock pulse occurs and is read with a one simulation sample delay from the clock pulse. This one sample delay within the block allows the error signal to include a direct feedback term of the block's output without the danger of forming an algebraic loop.

The block updates its tap values at each internal/external clock pulse based on the error input, the value of "mu", and the contents of the shift register. If desired, the tap values may be locked by:

- Not providing an update clock in external timing mode
- Setting "High" (1) the Lock input in internal timing mode

Taps may be reset at any time during the simulation by pulsing the Clock/Lock input with a value of -1. Taps may be initialized either internally (sets all taps to zero except for a specified tap, typically the center tap) or by using the external vector input. For an ODD number of taps, the center tap is initialized, while for an EVEN number of taps, the tap to the right of center is initialized. The user can specify an offset from the above default initialization.

Note: when implementing a fractionally spaced equalizer (e.g. more than one tap per symbol), and selecting Internal Tap Initialization, only the first tap of the "subgroup" (taps corresponding to the same symbol) is initialized. This is done to synchronize the input/output clock (at the symbol rate) to the initialized tap.

There is a delay of one simulation step across this block in addition to the output delay associated with the specific tap arrangement in use. This additional delay is necessary to allow diagram configurations where the block's output is used to produce the feedback error signal. Note: in such cases, the user should use the block's output clock to compute the error term if applicable.

x_1 = Input signal (Real or complex depending on type of block)

x_2 = Error signal (Real or complex depending on type of block)

x_3 = Clock / Lock input [high > 0.5] (impulse train); Reset [≤ -1] (pulse)

x_4 = Tap initialization vector (size = number of taps)

y_1 = Output signal

y_2 = Clock output (impulse train)

y_3 = Tap output vector (size = N for real eq.; size = $2N$ for complex eq.)
(alternating Real/Imag elements for complex eq.)

Number of Taps

Specifies the number of equalizer taps N. Valid range is 1 to 32,767.

Taps per Symbol

Specifies the number of taps per symbol period.

Mu

Specifies the feedback coefficient applied to the error signal in the adaptation process. A value in the range of 0.005 is typically specified.

Symbol Rate

Specifies the input symbol rate in *Hertz*. This value also corresponds to the inverse of the tap spacing period when the "Taps per Symbol" setting is 1.

Input Time Delay

Specifies the time delay in seconds until the start of a symbol period at the equalizer input. This value is used to synchronize the equalizer with the incoming data signal when internal timing mode is specified.

Initialized Tap Value

Specifies the value to be used in initializing the equalizer's center tap (or other tap if an offset is used). This parameter is only applicable when internal tap initialization mode is specified. All other taps are initialized to 0.

Initialized Tap Offset

Specifies an offset from center as to which tap is to be initialized using the value specified above. A value of 0 causes the center tap to be initialized. This parameter is applicable only when you activate the Internal parameter under Tap Initialization, described below. Positive values correspond to a taps past the center.

Show Taps

Displays the current values of the equalizer internal taps.

Timing

External

Specifies that an external clock is provided to the equalizer. The clock should go high at the center of the symbol period.

Internal

Specifies that the equalizer's sampling clock is to be generated internally.

Tap Initialization

External

Specifies that tap initial values are provided via the x_4 vector input.

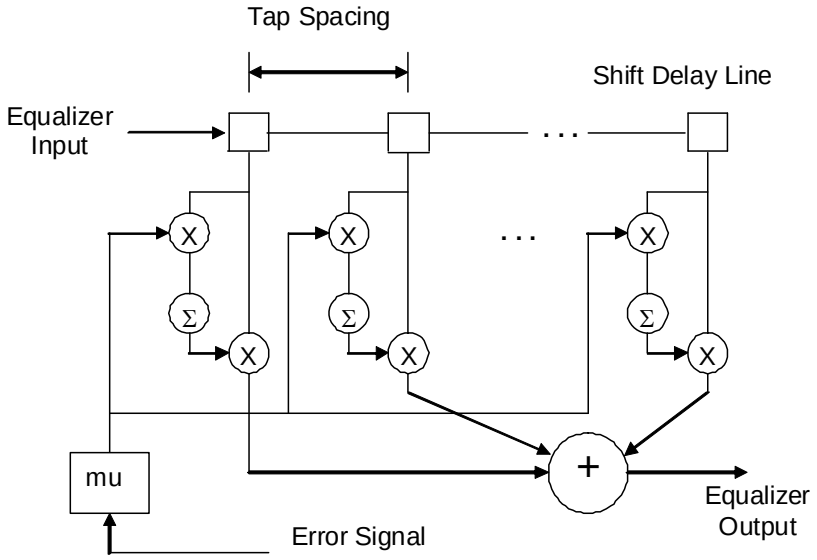
Internal

Internal taps are initialized based on the specified Initialized Tap Value and Initialized Tap Offset parameters, described above.

Discrete Equalizer (Complex or Real)

Two versions of this block exist, one complex and the other real. This block implements a discrete fractionally-spaced adaptive equalizer suitable for use with sampled (digital) waveforms. This block is similar to the Adaptive Equalizer block except that it does not use an internal shift delay line operating at the master simulation rate. Instead, it operates on discrete input samples read in at the block's internal (or external) sample rate.

Key block parameters include the total number of taps *N*, initial tap values, and the convergence coefficient "mu". The basic equalizer structure is shown in the figure below. To implement a fractionally spaced design (e.g. multiple taps per symbol), simply specify an internal (or external) sample rate that's a multiple of the underlying symbol rate of the input signal.



This block uses a Least Mean Square (LMS) convergence algorithm to adapt the tap values with the goal of minimizing the average error. The user is responsible for providing a suitable error input to the block, for example the error vector between a received point in the IQ plane and the closest constellation point. The error value is only read when an internal/external clock pulse occurs and is read with a one simulation sample delay from the clock pulse. This one sample delay within the block allows the error signal to include a direct feedback term of the block's output without the danger of forming an algebraic loop.

The block updates its tap values at each internal/external clock pulse based on the error input, the value of "mu", and the contents of the shift register. If desired, the tap values may be locked by:

- Not providing an update clock in external timing mode
- Setting "High" (1) the Lock input in internal timing mode

Taps may be reset at any time during the simulation by pulsing the Clock/Lock input with a value of -1. Taps may be initialized either internally (sets all taps to zero except for a specified tap, typically the center tap) or by using the external vector input. For an ODD number of taps, the center tap is initialized, while for an EVEN number of taps, the tap to the right of center is initialized. The user can specify an offset from the above default initialization.

There is a delay of one simulation step across this block in addition to the output delay associated with the specific tap arrangement in use. This additional delay is necessary to allow diagram configurations where the block's output is used to produce the feedback error signal. Note: in such cases, the user should use the block's output clock to compute the error term if applicable.

x_1 = Input signal (Real or complex depending on type of block)

x_2= Error signal (Real or complex depending on type of block)

x_3= Clock / Lock input [high > 0.5] (impulse train); Reset [≤ -1] (pulse)

x_4= Tap initialization vector (size = number of taps)

y_1 = Output signal

y_2 = Clock output (impulse train)

y_3 = Tap output vector (size = N for real eq.; size = $2N$ for complex eq.)
(alternating Real/Imag elements for complex eq.)

Number of Taps

Specifies the number of equalizer taps *N*. Valid range is 1 to 32,767.

Mu

Specifies the feedback coefficient applied to the error signal in the adaptation process. A value in the range of 0.005 is typically specified.

Sample Rate

Specifies the equalizer input sample rate in *Hertz* when in Internal Timing Mode. This value also corresponds to the inverse of the tap spacing period.

Input Time Delay

Specifies the time-delay in seconds until the start of the first input clock. This value can be used to synchronize the equalizer with the incoming data signal when Internal Timing Mode is specified.

Initialized Tap Value

Specifies the value to be used in initializing the equalizer's center tap (or other tap if an offset is used). This parameter is only applicable when internal tap initialization mode is specified. All other taps are initialized to 0.

Initialized Tap Offset

Specifies an offset from center as to which tap is to be initialized using the value specified above. A value of 0 causes the center tap to be initialized. This parameter is applicable only when you activate the Internal selection under Tap Initialization, described below. Positive values correspond to tap locations past the center.

Tap Spacing (1 – 10)

Specifies the spacing between successive equalizer taps in simulation steps.

Show Taps

Displays the current values of the equalizer internal taps.

Timing

External

Specifies that an external clock is provided to the equalizer.

Internal

Specifies that the equalizer's sampling clock is to be generated internally.

Tap Initialization

External

Specifies that tap initial values are provided via the x_4 vector input.

Internal

Internal taps are initialized based on the specified Initialized Tap Value and Initialized Tap Offset parameters, described above.

File FIR Filter

This block implements a Finite Impulse Response (FIR) filter based on user-supplied tap values provided in a file. Up to 32,767 taps can be specified. Because the input file represents the impulse response of the user-specified filter, this block can also be viewed as performing a convolution of the input signal with the truncated waveform specified by the input file. The effective sampling frequency of the filter can be specified to obtain a consistent filter response regardless of the simulation sampling rate. When the filter sampling frequency is a fraction of the simulation sampling rate, additional delay elements are introduced in the internal shift register between the filter's active tap locations.

The `File FIR` block accepts a real signal and outputs a real signal.

x = Input signal

y = Filtered output signal

Tap Spacing Frequency

Specifies the time spacing of the filter's internal taps as a frequency in hertz. This value must divide exactly into the simulation sampling frequency.

Normalize For Unity Gain at DC

Specifies that the filter's impulse response be normalized (scaled) so that the DC gain is unity (0 dB), i.e. the sum of all the taps equals 1. This setting is useful in scaling the filter's output to a desired level.

Reverse Tap Order

This setting specifies that the taps read from the file should be applied in reverse order. This option is useful when wanting to implement a matched filter to a known signal, or convolve a signal with a desired impulse response, in which case a time-reversed version of the response is needed.

Taps File Path

Specifies the DOS path to the desired FIR filter taps file. Taps are provided in increasing order and correspond to increasing delays. The format of the FIR filter tap file is described below:

File header (anything)

number of taps

tap value #1

tap value #2, tap value #3

...

Multiple tap values can be specified on a given line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 100 characters.

Select File

Opens the Select File dialog box for selecting the desired FIR filter tap file.

Browse File

Opens the selected FIR filter tap file using Notepad.

View Response

Invokes the Embed/Comm filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

FIR Filter

This block implements a Finite Impulse Response (FIR) filter. It employs the windowing method for filter design and allows you to implement lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian filters with your choice of window function.

Most filters designed with the `FIR Filter` block will have unity gain in the passband. The cutoff frequencies for all filter types except root raised cosine correspond to the half amplitude point; that is, they are down 6 dB (3 dB for root raised cosine). You should check the impulse response of the filter to ensure it meets your design criteria. The effective sampling frequency of the filter can be specified to obtain a consistent filter response regardless of the simulation sampling rate. When the filter sampling frequency is a fraction of the simulation sampling rate, additional delay elements are introduced in the internal shift register between the filter's active tap locations.

A warning message is issued if the time span of the filter (number of taps * sampling frequency) is less than the reciprocal of the filter cutoff frequency (1/Fc). This indicates that an insufficient number of taps may have been selected to effectively achieve the desired cutoff frequency. This warning message can be temporarily disabled by checking the appropriate box. This block accepts a real signal and outputs a real signal.

x = Input signal

y = Filtered output signal

Number of Taps

Indicates the desired number of filter taps to be used in realizing the filter. Valid range is from 1 to 32,767. Note that for highpass and bandstop types, the number of taps must be odd.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass, Raised Cosine, Root Raised Cosine, Hilbert, Gaussian or Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Cutoff Freq 2

Specifies the desired upper cutoff frequency for Bandpass and Bandstop filter types. Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Window Type

Lists the available window functions that can be used to realize the filter. When you select Kaiser, you must also enter a value in the Beta box.

Beta

Specifies the shape parameter associated with the Kaiser window.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Rolloff Factor

Specifies the rolloff factor (alpha) associated with the raised cosine or root raised cosine filter type. Valid range is from 0 to 1.

Filter Type

Indicates the desired filter type. Click on the DOWN ARROW to select from a list of filter types. Available types are lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian. When you select either the Raised Cosine or Root Raised Cosine parameter, you must supply a Rolloff Factor value.

Normalize LPF to 0 dB at dc

Specifies that the calculated filter's impulse response is to be normalized (scaled) so that the dc gain is unity (0 dB), i.e. the sum of all the taps equals 1. This setting only applies when the lowpass, raised cosine, root raised cosine, Hilbert or Gaussian filter type is selected. It is usually only necessary when the number of taps is relatively low and the cutoff frequency is a small fraction of the sampling frequency.

Allow FIR Decimation

Allows the user to specify a tap spacing frequency.

Tap Spacing Frequency

Specifies the time spacing of the filter's internal taps as a frequency in hertz. This value must divide exactly into the simulation sampling frequency. This option is only available if the "Allow FIR Decimation" checkbox is selected.

Show Taps

Displays the current FIR Filter tap values in the Embed/Comm Filter Result dialog box. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by clicking on the Save to File button. If you activate the Use FIR File Format option, the values are saved in a format compatible with the `File FIR` block.

View Response

Invokes the Embed/Comm filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

IIR Filter

This block implements an Infinite Impulse Response (IIR) filtering. You can choose from several well-known analog filter prototypes, including Butterworth and Chebyshev designs. The desired filter is implemented using bilinear transformation to map the *s*-domain analog design to the digital *z*-domain.

The `IIR` block differs from a discrete-time `transferFunction` block (listed under the Linear Systems category in the Blocks menu) in that the simulation time step is updated automatically prior to each run.

The `IIR` block accepts a real signal and outputs a real signal. You should verify that a stable impulse response is obtained, especially when specifying a very narrowband filter (<1% Fs) that is also of high filter order. The View Response button can be helpful for this purpose.

x = Input signal

y = Filtered output signal

Filter Method

Indicates the filter design method to be used. You can choose from Butterworth, Chebyshev Type I, Chebyshev II (Inverse Chebyshev), and Bessel.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass and Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Cutoff Freq 2

Specifies the upper cutoff frequency for Bandpass and Bandstop filter types.

Passband Specification

Epsilon

Indicates that the filter response at the cutoff frequency is determined from the value of Epsilon. A value of 1 for Epsilon, corresponds to an attenuation of 0.5.

Ripple

Indicates that the filter response at the cutoff frequency is determined from the value of Ripple. Ripple is a positive value, expressed in decibels, and corresponds to the desired attenuation at the cutoff frequency.

Stopband Atten.

Specifies the stopband attenuation for the desired filter in decibels. This parameter only applies to Chebyshev Type II filters. Its value must exceed the Ripple value.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Filter Order

Indicates the desired filter order. Valid range is 1 to 20. If you select either the bandpass or bandstop filter type, the filter order must be even.

Filter Type

Indicates the desired filter type. You can choose from lowpass, highpass, bandpass, and bandstop.

Show Coeff.

Displays the polynomial coefficients of the IIR filter's numerator and denominator in powers of z^{-1}.

View Response

Invokes the Embed/Comm Filter Viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

MagPhase Filter

This block implements an arbitrary complex FIR filter based on user-specified magnitude and phase responses supplied via an external file. The magnitude response is specified in decibels, while the phase response may be provided in either degrees, radians, or as a group delay. The filter is realized using the overlap-save method. The input signal is mapped to the frequency domain via FFT, multiplied by the specified frequency response, and then mapped back to the time domain via IFFT. The size of the FFT is twice that of the equivalent complex FIR filter tap length.

The input file should include points from $-f_s/2$ to $+f_s/2$ (0 to $+f_s/2$ for a real filter) in increasing order, but the frequency spacing need not be uniform. Linear interpolation is used to compute intermediate points as required. When a single-sided response is provided (real case), the routine internally creates a mirror image of the response (with opposite sign phase response) for the negative portion of the spectrum. In the event that the frequency range specified by the input file does not include the entire $-f_s/2$ to $+f_s/2$ range, values outside the specified range are linearly extrapolated based on the closest two specified frequency points.

An implementation delay equal to the equivalent FIR filter length plus one (in simulation steps) is experienced when using this block. This is in addition to any filter delay due to its response.

This block accepts a complex signal and outputs a complex signal. It also outputs the internal interpolated response used by the FFT routine. This is provided by the second output connector in vector form [3x1], and may be used to drive a `plot` block configured in XY mode.

x = Complex input signal [Re, Im]

y_1 = Filtered complex output signal [Re, Im]

y_2 = Interpolated response used by FFT routine [mag, phase, freq]

Equivalent FIR Filter Tap Length

Specifies the length of the filter's impulse response. This value must be a power of two as it determines the size of the internal FFT computations.

Phase Units

Degrees

Indicates that the file phase response is specified in units of degrees.

Group Delay

Indicates that the file phase response is specified as a group delay response in seconds.

Radians

Indicates that the file phase response is specified in units of radians.

Filter File Data

[0, fs/2]

Used when the input file only includes data over the range of [0, +fs/2] (real filter).

[-fs/2, +fs/2]

Used when the input file includes data over the range of [-fs/2, +fs/2] (complex filter).

Add Linear Phase

Automatically adds linear phase to the input phase specification. The amount of linear phase added corresponds to a delay of ½ the FIR filter's impulse response duration. This option is useful when the input file phase specification represents the filter's deviation from linear phase.

Normalize Phase

Automatically normalizes the internal phase response (derived from the group delay data) so that the phase response is zero at the zero frequency point. This setting only applies when the Group Delay specification method is chosen.

Select File

Opens the Select File dialog box for selecting a filter frequency response file.

Browse File

Opens the selected filter frequency response file using Notepad.

Filter File Path

Specifies the DOS path to the desired filter frequency response file. The format of the file is described below:

File header (anything)

number of entries (n)

frequency point #1, magnitude, phase

frequency point #2, magnitude, phase

...

frequency point #n, magnitude, phase

Entries are to be provided in increasing frequency order and should cover the range from $-f_s/2$ to $+f_s/2$ or 0 to $+f_s/2$. Entries may be separated by commas, blank space, or tabs. The maximum allowed line length is 100 characters. The expected units are hertz for frequency, decibels for magnitude, degrees or radians for phase, and seconds for group delay.

Care should be taken to ensure that the correct amount of linear phase is built into the phase specification. Alternatively the Add Linear Phase box may be checked. Since the filter's delay is typically equal to ½ the specified FIR filter impulse response length, the required amount of linear phase can be obtained as follows:

$$delay = \frac{FIRlength}{2 \cdot f_s} \text{ sec} \qquad phase(f) = -360 \cdot f \cdot delay$$

Pulse Shaping Filter

This block implements pulse shaping using a *finite impulse response* (FIR) approach. It allows the use of a variety of windowing shapes, as well as Nyquist type pulse shapes, such as the raised cosine and root raised cosine forms. This block also supports Gaussian pulse shapes.

The `Pulse Shaping Filter` block expects an impulse pulse train representing the input symbol values. If a rectangular input signal is provided instead (e.g. NRZ data), an inverse sinc function can be applied to convert, internally to the block, the rectangular pulse train into an impulse one.

The `Pulse Shaping Filter` block normally introduces a delay equal to $N / 2$ simulation steps, where N is the number of filter taps (assuming the filter sampling frequency equals the simulation rate). The number of taps is equal to the pulse span interval times the number of samples per symbol. This block accepts a real signal and outputs a real signal.

x = Input signal (impulse train or rectangular pulses)

y = Pulse shaped output

Pulse Span

Specifies the span of the pulse shaping filter in units of Symbol Periods (T). Note: The filter tap at the upper span boundary is usually omitted. For example, in the case of a $4T$ pulse span, the filter taps will range from $[-2/T, 2/T - \Delta t]$, where Δt is the simulation time step.

Samples per Symbol

Specifies the number of samples per symbol associated with the input data signal.

Rolloff Factor

Specifies the rolloff factor (alpha) associated with the raised cosine or root raised cosine filter types. The valid range is from 0 to 1, with 0 corresponding to a truncated sin(x)/x impulse response.

Beta

Specifies the shape parameter associated with the Kaiser window.

BT Product

Specifies the BT product value associated with the Gaussian filter type.

Filter Type

Indicates the desired pulse shaping filter type. Click on the DOWN ARROW to select from a list of filter types. Available types are: window only, raised cosine, root raised cosine, and Gaussian. A Rolloff Factor must be supplied when the Raised Cosine or Root Raised Cosine filters are selected, and a BT Product value for the Gaussian case.

Window Type

Specifies the window function to be used in the realization of the FIR filter. If the Kaiser window is selected, a value for the Kaiser Beta parameter must also be supplied.

Apply Inverse Sinc

When selected, an inverse sinc function is cascaded with the pulse shaping FIR response. This option should be used whenever a rectangular pulse train, rather than an impulsive one, is used at the block's input.

Show Taps

Displays the current FIR Pulse Shaping Filter tap values. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by pressing the Save To File button. If the Use FIR File Format box is checked, the values are saved in a format compatible with the `File FIR` block.

View Response

Displays the current FIR Pulse Shaping Filter response by launching the Embed/Comm Filter Viewer.

Sampling File FIR Filter

This block implements a sampling Finite Impulse Response (FIR) filter based on user-supplied tap values provided in a file. Up to 32,767 taps can be specified. The block will sample the input signal at the specified rate and propagate these values through its internal shift register. The `Sampling File FIR` output signal can be either held or interpolated when the filter sampling rate is below the simulation rate. A filter delay adjustment is provided to allow fine-tuning of the sampling instant.

The `Sampling File FIR` block accepts a real signal and outputs a real signal.

x = Input signal

y_1 = Filtered output signal

y_2 = Filter sampling clock (impulse train)

Sampling Frequency

Specifies the filter's sampling frequency in hertz. This value must divide exactly into the simulation sampling frequency.

Initial Delay

Specifies the initial sampling delay of the filter in simulation steps or seconds, depending on the selected Delay Mode.

Normalize for unity gain at dc

Specifies that the filter's impulse response is to be normalized (scaled) so that the dc gain is unity (0 dB), i.e. the sum of all the taps equals 1.

Delay Mode

Sim Steps

Indicates the initial delay is specified in simulation steps.

Seconds

Indicates the initial delay is specified in seconds.

Output Mode

This setting only applies when the filter sampling frequency is a fraction of the simulation sampling rate.

Interpolated

Specifies the filter output to be linearly interpolated between computed output points. An additional delay of one filter sampling period is introduced by this selection.

Held

Specifies the filter output to be held constant between computed output points.

Zero Pad

Specifies the filter output is zero padded between computed output points.

Select File

Opens the Select File dialog box for selecting the desired FIR filter tap file.

Browse File

Opens the selected FIR filter tap file using Notepad.

View Response

Invokes the Embed/Comm filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

Taps File Path

Specifies the DOS path to the desired FIR filter taps file. Taps are provided in increasing order and correspond to increasing delays. The format of the FIR filter tap file is described below:

File header (anything)

number of taps

tap value #1

tap value #2, tap value #3

...

Multiple tap values can be specified on a given line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 100 characters.

Sampling FIR Filter

This block implements a sampling Finite Impulse Response (FIR) filter. The block will sample the input signal at the specified rate and propagate these values through its internal shift register. The `Sampling FIR` output signal can be either held or interpolated when the filter sampling rate is below the simulation rate. A filter delay adjustment is provided to allow fine-tuning of the sampling instant. The `Sampling FIR` block employs the windowing method for filter design and allows you to implement lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian filters with your choice of window function.

Most filters designed with the `FIR Filter` block will have unity gain in the passband. The cutoff frequencies for all filter types except root raised cosine correspond to the half amplitude point; that is, they are down 6 dB (3 dB for root raised cosine). You should check the impulse response of the filter to ensure it meets your design criteria. The effective sampling frequency of the filter can be specified so as to obtain a consistent filter response regardless of the simulation sampling rate. When the filter sampling frequency is a fraction of the simulation sampling rate, additional delay elements are introduced in the internal shift register between the filter's active tap locations.

This block accepts a real signal and outputs a real signal.

x = Input signal

y_1 = Filtered output signal

y_2 = Filter sampling clock (impulse train)

Number of Taps

Indicates the desired number of filter taps to be used in realizing the filter. Valid range is from 1 to 32,767. Note that for highpass and bandstop types, the number of taps must be odd.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass, Raised Cosine, Root Raised Cosine, Hilbert, Gaussian or Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Cutoff Freq 2

Specifies the desired upper cutoff frequency for Bandpass and Bandstop filter types. Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Rolloff Factor

Specifies the rolloff factor (alpha) associated with the raised cosine or root raised cosine filter type. Valid range is from 0 to 1.

Beta

Specifies the shape parameter associated with the Kaiser window.

Cutoff Frequency Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Output Mode

This setting only applies when the filter sampling frequency is a fraction of the simulation sampling rate

Interpolated

Specifies the filter output to be linearly interpolated between computed output points. An additional delay of one filter sampling period is introduced by this selection.

Held

Specifies the filter output to be held between computed output points.

Zero Pad

Specifies the filter output is zero padded between computed output points.

Delay Units

Sim Steps

Indicates the initial delay is specified in simulation steps.

Seconds

Indicates the initial delay is specified in seconds.

Window Type

Lists the available window functions that can be used to realize the filter. When you select Kaiser, you must also enter a value in the Beta box.

Filter Type

Indicates the desired filter type. Click on the DOWN ARROW to select from a list of filter types. Available types are lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian. When you select either the Raised Cosine or Root Raised Cosine parameter, you must supply a Rolloff Factor value.

Normalize LPF to 0 dB at dc

Specifies that the computed filter's impulse response is to be normalized so that the dc gain is unity (0 dB), i.e. the sum of all the taps equals 1. This setting only applies when the lowpass, raised cosine, root raised cosine, Hilbert or Gaussian filter type is selected. It is usually only necessary when the number of taps is relatively low and the cutoff frequency is a small fraction of the sampling frequency.

Sampling Frequency

Specifies the filter's sampling frequency in hertz. This value must divide exactly into the simulation sampling frequency.

Initial Delay

Specifies the initial sampling delay of the filter in simulation steps or seconds, depending on the selected Delay Mode. This setting only applies when Internal Timing mode is selected.

Timing

External

Specifies that the block's sampling clock is provided externally. An approximate value for the Sampling Frequency parameter must still be specified; this is required for internal tap computation purposes.

Internal

Specifies that the block's sampling clock is internally generated according to the Sampling Frequency and Initial Delay parameters.

Show Taps

Displays the current FIR Filter tap values in the Embed/Comm Filter Result dialog box. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by clicking on the Save to File button. If you activate the Use FIR File Format option, the values are saved in a format compatible with the `File FIR` block.

View Response

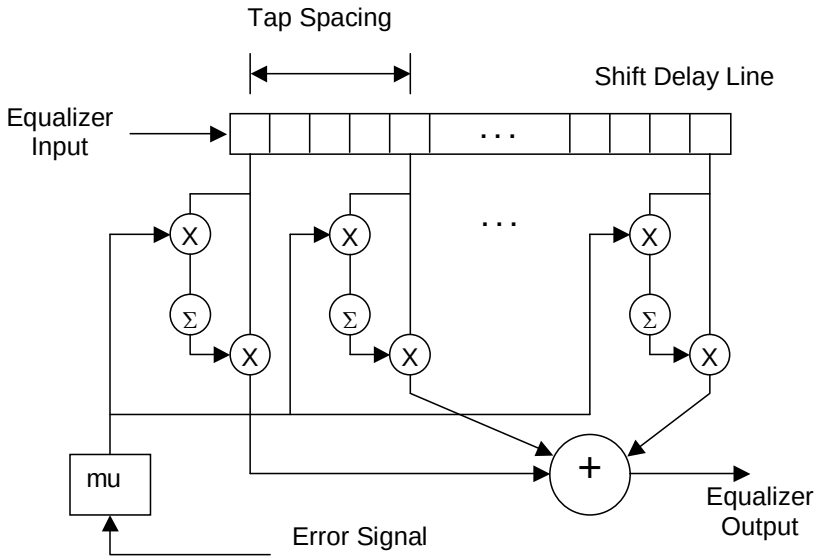
Invokes the Embed/Comm filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

Variable Spaced Equalizer (Complex or Real)

This block implements a conventional or fractionally-spaced adaptive equalizer suitable for use with both analog and digital waveforms. Unlike the regular Adaptive Equalizer block, this version allows the user to specify non-uniform sample spacing between filter taps. Two versions of this block exist, one complex and the other real.

This version of the equalizer block is most useful for channels that include an echo or multipath term that occurs at a delay larger than many symbol periods. The user can then create two groups of taps, separated by a fixed delay, to span the entire channel impulse response without requiring a large number of equalizer taps. It's recommended that external tap initialization be used for such configurations (see included example).

Key block parameters include the total number of taps N (11 max), the number of taps per symbol, initial tap values, and the convergence coefficient "mu". The basic equalizer structure is shown below. Each cell in the equalizer's internal shift delay line corresponds to a single simulation step. The spacing between successive taps is specified by the user in units of simulation steps and need not be uniform. To emulate a uniformly-spaced equalizer, the tap spacing should be set to the ratio of the simulation rate divided by the symbol rate (assuming a one tap per symbol case).



This block uses a Least Mean Square (LMS) convergence algorithm to adapt the tap values with the goal of minimizing the average error. The user is responsible for providing a suitable error input to the block, for example the error vector between a received point in the IQ plane and the closest constellation point. The error value is only read when an internal/external clock pulse occurs and is read with a one simulation sample delay from the clock pulse. This one sample delay within the block allows the error signal to include a direct feedback term of the block's output without the danger of forming an algebraic loop.

The block updates its tap values at each internal/external clock pulse based on the error input, the value of "mu", and the contents of the shift register. If desired, the tap values may be locked by:

- Not providing an update clock in external timing mode
- Setting "High" (1) the Lock input in internal timing mode

Taps may be reset at any time during the simulation by pulsing the Clock/Lock input with a value of -1. Taps may be initialized either internally (sets all taps to zero except for a specified tap, typically the center tap) or by using the external vector input. For an ODD number of taps, the center tap is initialized, while for an EVEN number of taps, the tap to the right of center is initialized. The user can specify an offset from the above default initialization.

Note: when implementing a fractionally spaced equalizer (e.g. more than one tap per symbol), and selecting Internal Tap Initialization, only the first tap the "subgroup" (taps corresponding to the same symbol) is initialized. This is done to synchronize the input/output clock (at the symbol rate) to the initialized tap.

There is a delay of one simulation step across this block in addition to the output delay associated with the specific tap arrangement in use. This additional delay is necessary to allow diagram configurations where the block's output is used to produce the feedback error signal. Note: in such cases, the user should use the block's output clock to compute the error term if applicable.

x_1 = Input signal (Real or complex depending on type of block)

x_2 = Error signal (Real or complex depending on type of block)

x_3 = Clock / Lock input [high > 0.5] (impulse train); Reset [≤ -1] (pulse)

x_4 = Tap initialization vector (size = number of taps)

y_1 = Output signal

y_2 = Clock output (impulse train)

y_3 = Tap output vector (size = N for real eq.; size = $2N$ for complex eq.)
(alternating Real/Imag elements for complex eq.)

Number of Taps

Specifies the number of equalizer taps N. Valid range is 1 to 11.

Taps per Symbol

Specifies the number of taps per symbol period.

Mu

Specifies the feedback coefficient applied to the error signal in the adaptation process. A value in the range of 0.005 is typically specified.

Symbol Rate

Specifies the input symbol rate in *Hertz*. This value also corresponds to the inverse of the tap spacing period when the "Taps per Symbol" setting is 1.

Input Time Delay

Specifies the time delay in seconds until the start of a symbol period at the equalizer input. This value is used to synchronize the equalizer with the incoming data signal when internal timing mode is specified.

Initialized Tap Value

Specifies the value to be used in initializing the equalizer's center tap (or other tap if an offset is used). This parameter is only applicable when internal tap initialization mode is specified. All other taps are initialized to 0.

Initialized Tap Offset

Specifies an offset from center as to which tap is to be initialized using the value specified above. A value of 0 causes the center tap to be initialized. This parameter is applicable only when you activate the Internal parameter under Tap Initialization, described below. Positive values correspond to a taps past the center.

Show Taps

Displays the current values of the equalizer internal taps.

Timing

External

Specifies that an external clock is provided to the equalizer. The clock should go high at the center of the symbol period.

Internal

Specifies that the equalizer's sampling clock is to be generated internally.

Tap Initialization

External

Specifies that tap initial values are provided via the x_4 vector input.

Internal

Internal taps are initialized based on the specified Initialized Tap Value and Initialized Tap Offset parameters, described above.

Fixed Point category

Blocks in the Fixed Point category include `Fixed Point FIR`, `Fixed Point IIR`, and `Fixed Point VCO (Complex and Real)`.

Fixed Point FIR Filter

This block implements a Finite Impulse Response (FIR) filter using fixed point arithmetic. It employs the windowing method for filter design and allows the user to implement lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian filters with choice of window function.

This block uses the same design technique as the regular `FIR Filter` block but allows the user to specify the fixed point precision of the internal coefficients once they have been computed. The user can also specify the precision of the internal math accumulator and the fixed point precision of the output signal. An autoscaling feature is provided to automatically scale the output radix point based on observations of output overflows during a simulation run.

Most filters designed with the `Fixed Point FIR Filter` block will have unity gain in the passband. The cutoff frequencies for all filter types except root raised cosine correspond to the half amplitude point; that is, they are down 6 dB (3 dB for root raised cosine). You should check the impulse response of the filter to ensure it meets your design criteria. The effective sampling frequency of the filter can be specified to obtain a consistent filter response regardless of the simulation sampling rate. When the filter sampling frequency is a fraction of the simulation sampling rate, additional delay elements are introduced in the internal shift register between the filter's active tap locations.

A warning message is issued if the time span of the filter (number of taps * sampling frequency) is less than the reciprocal of the filter cutoff frequency (1/Fc). This indicates that an insufficient number of taps may have been selected to effectively achieve the desired cutoff frequency. This warning message can be temporarily disabled by checking the appropriate box. This block accepts a scaled integer signal and outputs a scaled integer signal.

x = Input signal

y = Filtered output signal

Number of Taps

Indicates the desired number of filter taps to be used in realizing the filter. Valid range is from 1 to 32,767. Note that for highpass and bandstop types, the number of taps must be odd.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass, Raised Cosine, Root Raised Cosine, Hilbert, Gaussian or Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Cutoff Freq 2

Specifies the desired upper cutoff frequency for Bandpass and Bandstop filter types. Due to the filter design method employed, this cutoff frequency corresponds to the half amplitude point (6 dB attenuation point) except for the Root Raised Cosine and Gaussian filter types (3 dB attenuation point).

Window Type

Lists the available window functions that can be used to realize the filter. When you select Kaiser, you must also enter a value in the Beta box.

Beta

Specifies the shape parameter associated with the Kaiser window.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Rolloff Factor

Specifies the rolloff factor (alpha) associated with the raised cosine or root raised cosine filter type. Valid range is from 0 to 1.

Filter Type

Indicates the desired filter type. Click on the DOWN ARROW to select from a list of filter types. Available types are lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian. When you select either the Raised Cosine or Root Raised Cosine parameter, you must supply a Rolloff Factor value.

Normalize LPF to 0 dB at dc

Specifies that the calculated filter's impulse response is to be normalized (scaled) so that the dc gain is unity (0 dB), i.e. the sum of all the taps equals 1. This setting only applies when the lowpass, raised cosine, root raised cosine, Hilbert or Gaussian filter type is selected. It is usually only necessary when the number of taps is relatively low and the cutoff frequency is a small fraction of the sampling frequency.

Allow FIR Decimation

Specifies that the FIR filter taps will be spaced at a value greater than one simulation step according to the value specified by the Tap Spacing Frequency parameter. The tap spacing in time units is the inverse of the above parameter.

Tap Spacing Frequency

Specifies the time spacing of the filter's internal taps as a frequency in hertz. This value must divide exactly into the simulation sampling frequency. This parameter is only available when the "Allow FIR Decimation" box is checked.

Show Taps

Displays the current FIR Filter tap values in the Embed/Comm Filter Result dialog box. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by clicking on the Save to File button. If you activate the Use FIR File Format option, the values are saved in a format compatible with the `File FIR` block.

View Response

Invokes the Embed/Comm filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

FIXED POINT PARAMETERS

Coefficient Word Length

Specifies the word length of the filter coefficients.

Coefficient Radix Point

Specifies the radix point of the filter coefficients.

Accumulator Word Length

Specifies the word length of the internal math accumulator.

Accumulator Radix Point

Specifies the radix point of the internal math accumulator.

Output Word Length

Specifies the word length of the filter's output.

Output Radix Point

Specifies the radix point of the filter's output.

Autoscale Output Radix Point

Specifies to automatically scale the radix point of the output signal a based on the detection of output overflows. Each time during a simulation run that an output overflow is detected, the output radix point is immediately increased by one.

Warn on Overflow

Specifies to produce a warning message if any overflows are detected.

Math Overflow Count

Provides a readout of the number of internal math overflows detected during the most recent run.

Output Overflow Count

Provides a readout of the number of output overflows detected during the most recent run.

Fixed Point IIR Filter

This block implements an Infinite Impulse Response (IIR) filtering. You can choose from several well-known analog filter prototypes, including Butterworth and Chebyshev designs. The desired filter is implemented using bilinear transformation to map the *s*-domain analog design to the digital *z*-domain.

The `IIR` block differs from a discrete-time `transferFunction` block (listed under the Linear Systems category in the Blocks menu) in that the simulation time step is updated automatically prior to each run.

The `IIR` block accepts a real signal and outputs a real signal. You should verify that a stable impulse response is obtained, especially when specifying a very narrowband filter (<1% Fs) that is also of high filter order. The View Response button can be helpful for this purpose.

x = Input signal

y = Filtered output signal

Filter Method

Indicates the filter design method to be used. You can choose from Butterworth, Chebyshev Type I, Chebyshev II (Inverse Chebyshev), and Bessel.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass and Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Cutoff Freq 2

Specifies the upper cutoff frequency for Bandpass and Bandstop filter types.

Passband Specification

Epsilon

Indicates that the filter response at the cutoff frequency is determined from the value of Epsilon. A value of 1 for Epsilon, corresponds to an attenuation of 0.5.

Ripple

Indicates that the filter response at the cutoff frequency is determined from the value of Ripple. Ripple is a positive value, expressed in decibels, and corresponds to the desired attenuation at the cutoff frequency.

Stopband Atten.

Specifies the stopband attenuation for the desired filter in decibels. This parameter only applies to Chebyshev Type II filters. Its value must exceed the Ripple value.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Filter Order

Indicates the desired filter order. Valid range is 1 to 20. If you select either the bandpass or bandstop filter type, the filter order must be even.

Filter Type

Indicates the desired filter type. You can choose from lowpass, highpass, bandpass, and bandstop.

Show Coeff.

Displays the polynomial coefficients of the IIR filter's numerator and denominator in powers of z^{-1}.

View Response

Invokes the Embed/Comm Filter Viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response. For more details on using the filter viewer, please refer to the *Output Plots* section in Chapter 2.

FIXED POINT PARAMETERS

Coefficient Word Length

Specifies the word length of the filter coefficients.

Coefficient Radix Point

Specifies the radix point of the filter coefficients.

Autoscale Coeficient Radix Point

Specifies to automatically scale the radix point of the tap coefficients to maximize the precision of the filter.

Accumulator Word Length

Specifies the word length of the internal math accumulator.

Accumulator Radix Point

Specifies the radix point of the internal math accumulator.

Output Word Length

Specifies the word length of the filter's output.

Output Radix Point

Specifies the radix point of the filter's output.

Autoscale Output Radix Point

Specifies to automatically scale the radix point of the output signal a based on the detection of output overflows. Each time during a simulation run that an output overflow is detected, the output radix point is immediately increased by one.

Warn on Overflow

Specifies to produce a warning message if any overflows are detected.

Math Overflow Count

Provides a readout of the number of internal math overflows detected during the most recent run.

Output Overflow Count

Provides a readout of the number of output overflows detected during the most recent run.

Fixed Point VCO (Complex or Real)

This block implements a fixed point VCO. Two versions of this block are provided: one producing a complex scaled integer output and the other producing a real scaled integer output. When the input drive signal is 0, the `VCO` block outputs a tone at the specified center frequency. With a non-zero input, the output frequency deviates from the center frequency depending on the magnitude of the drive signal and the specified VCO gain.

This block always produces a unity amplitude output and uses fixed point versions of the sine and cosine functions.

x = Input drive signal

y_1 = Output signal ([Re, Im] for complex)

y_2 = Accumulated phase (rad)

$$y_1(t) = Ae^{j\theta(t)} \qquad y_2(t) = \theta(t)$$

$$\theta(t) + \int_0^t \left(2\pi f_c + x_1(\tau)K_o\right)d\tau + \phi$$

where:

f_C = translation frequency A = carrier amplitude

K_O = VCO gain ϕ = initial phase (radians)

Center Frequency

Indicates the VCO center frequency in hertz. The value may be set to 0 or even a negative frequency.

Initial Phase

Indicates the starting phase of the output complex tone. This value is specified in degrees.

VCO Gain

Indicates the gain of the VCO in Hz/volt. The value may be positive or negative.

Integration Method

Euler

Specifies the Euler integration method (forward difference).

Trapezoidal

Specifies the trapezoidal integration method.

Backward Difference

Specifies the backwards difference integration method.

Sin/Cos Precision

16 Bits

Specifies that the internal sine and cosine functions, and the block's output, use 16 bit word lenghts.

32 Bits

Specifies that the internal sine and cosine functions, and the block's output, use 32 bit word lenghts.

FIXED POINT PARAMETERS

Accumulator Word Length

Specifies the word length of the internal math accumulator.

Accumulator Radix Point

Specifies the radix point of the internal math accumulator.

Auto Adjust Accumulator Radix Point

Specifies to automatically scale the radix point of the internal accumulator based on the detection of internal overflows. Each time during a simulation run that an accumulator overflow is detected, the accumulator radix point is immediately increased by one. The default radix point is 2, which supports are range of [

Warn on Overflow

Specifies to produce a warning message if any overflows are detected.

Math Overflow Count

Provides a readout of the number of internal math overflows detected during the most recent run.

Instruments category

Blocks in the Instruments categories include `BER Curve Display`, `Oscilloscope Display`, `Spectrum Analyzer Display (Complex)`, and `Spectrum Analyzer Display (Real)`.

BER Curve Display

This pre-wired set of blocks provides quick access to a BER curve display. It includes a `BER Co`ntrol block (Estimators), and a pre-configured `plot` block.

Oscilloscope Display

This pre-wired set of blocks provides quick access to an oscilloscope display. It includes an `Oscilloscope` block (Operators), an `Impulse` block (Sources) as a trigger, and a pre-configured `plot` block.

Spectrum Analyzer Display (Complex)

This pre-wired set of blocks provides quick access to a complex spectrum analyzer display. It includes a `Spectrum (Complex)` block (Operators), an `Impulse` block (Sources) as a trigger, and a pre-configured `plot` block.

Spectrum Analyzer Display (Real)

This pre-wired set of blocks provides quick access to a real spectrum analyzer display. It includes a `Spectrum (Real)` block (Operators), an `Impulse` block (Sources) as a trigger, and a pre-configured `plot` block.

Modulators categories - Complex and Real

Blocks in the Modulators - Complex and Modulators - Real categories include `AM`, `DQPSK`, `Pi/4-DQPSK`, `FM`, `FSK`, `I/Q`, `MSK`, `PM`, `PPM`, `PSK`, `QAM`, `PAM`, and `SQPSK`. Note that the `PPM` block is available only as a real output block.

AM Modulator

This block performs *double-sideband amplitude modulation* (DSB-AM) of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `AM` block belongs to the family of analog modulators. In AM, the information is transmitted by varying the carrier signal amplitude according to the input signal level. The carrier frequency remains constant. This block accepts an analog signal as its input.

x = Input signal

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Carrier phase (rad) [optional]

$$y_1(t) = (A + mx)e^{j(2\pi f_c t + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A when the input is 0. This value is specified in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

Modulation Factor

Specifies the AM factor m. It controls the extent of carrier amplitude deviation according to the equation shown above.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the carrier phase. This forces the phase value to be in the range of [0, 2pi].

ASK Modulator

This block performs *amplitude shift keying* (ASK) modulation of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The ASK block belongs to the family of digital modulators. In ASK, the information is transmitted by varying the carrier signal amplitude in discrete levels according to the input data value. The carrier frequency remains constant. This block accepts a symbol number as its input.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

x_2 = Input data clock (impulse train)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Unmodulated carrier phase (rad)

$$y_1(t) = A_{[x_1]} e^{j(2\pi f_c t + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Number of Amplitude Levels

Specifies the number of discrete amplitude levels to be used. This value should match the number of entries in the ASK Definition File.

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Initial Phase

Specifies the initial phase of the modulator in *degrees*.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

Select File

Opens the Select File dialog box for selecting a ASK definition file.

Browse File

Opens the selected ASK definition file using Notepad.

ASK File Path

Specifies the DOS path to the desired ASK definition file.

Entry delimiters include spaces, commas and tabs. The maximum allowed line length is 100 characters. The format is shown below:

File header (anything) (one line)
symbol #1, corresponding amplitude value
...
symbol #*n*, corresponding amplitude value

A sample ASK definition file is shown below:

ASK Map File
0 1.0
1 4.0
2 3.0
3 2.0

Differential PSK Modulator

This block implements differential phase shift keying (DPSK) modulation. There are two versions of this block: one producing a complex output and the other producing a real output.

The `Differential PSK` block belongs to the family of digital modulators. In DPSK modulation the digital information is transmitted by increasing or decreasing the carrier phase depending on the input data values. The carrier amplitude remains constant. The `Differential PSK` block accepts a symbol number as its input (when the input data clock is high) and maps it to a phase transition value as specified via an external mapping file.

Supported DPSK modes include DBPSK, DQPSK, Pi/4-DQPSK, D8PSK, D16PSK and D32PSK.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

x_2= Input data clock (impulse train)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Unmodulated carrier phase (rad)

$$y_1(t) = Ae^{j(2\pi f_c + \phi + \sum_{k=0}^{n} \theta_d[k])} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi \qquad \theta_d[k](x) = k\textit{th} \text{ phase transition}$$

The phase transition value θ_d is obtained from the input symbol value according to the specified map file.

DPSK Type

Specifies the desired DPSK modulation type. Supported modes include DBPSK, DQPSK, Pi/4-DQPSK, D8PSK, D16PSK and D32PSK.

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Initial Phase

Specifies the initial phase of the modulator in *degrees*.

Gain Imbalance

Specifies the gain imbalance (Q relative to I) of the modulator in units of dBs. A positive value corresponds to greater power in the quadrature axis than in the in-phase axis.

Phase Imbalance

Specifies the phase imbalance of the modulator in degrees as a deviation from ideal. A positive value correspond to a clockwise rotation of the Q axis relative to the I axis. For example, 10 degrees imbalance implies an angle of 80 degrees between the I and Q axes, instead of the ideal 90 degrees.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

Select File

Opens the Select File dialog box for selecting a DPSK phase transition map file.

Browse File

Opens the selected DPSK phase transition map file using Notepad.

DPSK File Path

Specifies the DOS path to the desired DPSK phase transition map file.

The file format uses a "keyword" followed by a listing of input symbols and corresponding "normalized" delta-phase values, where unity (1) corresponds to the constellation spacing (e.g. 90 degrees for DQPSK, and 45 degrees for Pi4DQPSK or D8PSK). Valid modulation keywords are *dbpsk*, *dqpsk*, *pi4dqpsk*, *d8psk*, *d16psk* and *d32psk*. All keywords must be specified in lowercase.

Additional text on each line beyond the first two entries is ignored and can be used for comment purposes (see example below). Entry delimiters include spaces, commas and tabs. The maximum allowed line length is 100 characters. Each map file may contain multiple modulation mappings. The format is shown below:

File header (anything) (can be multiple lines)
...
modulation keyword
symbol #1, corresponding "normalized" phase-delta *comments*
...
symbol #*n,* corresponding "normalized" phase-delta *comments*

next modulation keyword [optional]
symbol # ... [optional]

A sample DPSK phase transition map file is shown below:

DPSK Map File for DBPSK, DQPSK, Pi/4-DQPSK and D8PSK
dbpsk

0	0	(0 deg)
1	1	(180 deg)

dqpsk

0	0	(0)
1	1	(90)
2	-1	(-90)
3	2	(180)

pi4dqpsk

0	1	(45)
1	3	(135)
2	-1	(-45)
3	-3	(-135)

d8psk

0	0	(0)
1	1	(45)
2	3	(135)
3	2	(90)
4	-1	(-45)
5	-2	(-90)
6	4	(180)
7	-3	(-135)

FM Modulator

This block performs *frequency modulation* (FM) of the input signal based on the selected block settings. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `FM` block belongs to the family of analog modulators. In FM, the information is transmitted by varying the carrier frequency according to the input signal level. The carrier amplitude remains constant.

The `FM` block takes an analog signal as its input.

x = Input signal

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Phase of complex modulated signal (optional)

$$y_1(t) = Ae^{j[2\pi(f_c+\beta x)t+\phi]} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi(f_c + \beta x)t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Carrier Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

FM Deviation

Specifies the FM deviation index β. It controls the extent of carrier frequency variation according to above equation, and is specified in hertz/volt.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

FSK Modulator

This block performs *frequency shift keying* (FSK) modulation of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The FSK block belongs to the family of digital modulators. In FSK modulation, the information is transmitted by varying the carrier frequency between N frequency settings depending on the input signal level. The carrier amplitude remains constant.

The FSK block accepts a symbol number as its input.

x = Input symbol number (integer [0 … N -1], where N is the number of tones)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Phase of complex modulated signal (optional)

$$f_{out} = f_{min} + x \cdot \Delta f$$

Number of Tones

Specifies the number N of available tones.

Lowest Frequency

Specifies the carrier frequency corresponding to the lowest desired output tone in hertz. This corresponds to input symbol # 0.

Frequency Spacing

Specifies the frequency spacing Δf between adjacent FSK tones in hertz.

Amplitude

Specifies the carrier single-sided peak amplitude in volts.

Initial Phase

Specifies the initial carrier phase in degrees. This option is only available when you select continuous phase mode.

Phase Mode

Continuous

Indicates that the signal phase is continuous across FSK tone transitions. This simulates the use of a *voltage controlled oscillator* (VCO) or NCO in generating the output FSK signal.

Discontinuous

Indicates that the signal phase is not continuous across FSK tone transitions. This simulates the use of multiple free running oscillators to generate the output signal.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase. This mode is only available when the Phase Mode is set to "Continuous".

Wrapped [0, 2pi]

Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

GFSK Modulator

This block implements a Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK modulation, the digital information is transmitted by shifting the carrier frequency between two states.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of each of these blocks may need to be specified for proper operation, in addition to setting the BT product, symbol rate, and FM deviation global parameters. The default settings reflect a Bluetooth implementation.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

GMSK Modulator

This block implements a Gaussian Minimum Shift Keying (GMSK) modulator as a compound block. In GMSK modulation, the digital information is transmitted by shifting the carrier frequency between two states with a frequency offset of +/- 0.25 the symbol rate. This represents a special case of GFSK.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of each of these blocks may need to be specified for proper operation, in addition to setting the BT product and the symbol rate global parameters.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

IQ Modulator

This block performs generic Amplitude Phase Modulation given a pair of analog I/Q signal inputs. Two versions of this block are provided: one producing a Complex output and the other producing a Real output. The block can be used to produce either digital or analog modulation. The information is transmitted by varying both the carrier amplitude and phase according to the complex input signal. Block parameters include the carrier frequency, initial phase, and amplitude scaling factor. This block takes two parallel analog signals as its inputs.

x_1 = I channel input

x_2 = Q channel input

y_1 = Modulated signal ([Re, Im] for Complex)

y_2 = Unmodulated carrier phase (rad) [optional]

$$y_1(t) = A(x_1 + jx_2)\,e^{j(2\pi f_c t + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in *hertz*. It may be set to zero when working in complex envelope representation.

Amplitude Factor

Specifies the carrier amplitude scaling factor A applied to the input (I,Q) vector.

Initial Phase

Specifies the initial phase θ of the modulating carrier in *degrees*.

MSK Modulator

This block performs *minimum shift keying* (MSK) modulation of the input signals based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `MSK` block belongs to the family of digital modulators. MSK modulation is similar to SQPSK modulation, except that sinusoidal pulse shaping is applied to the data signal prior to modulation.

MSK modulation can also be viewed as a form of FSK modulation with tones at

$$f_c \pm \frac{R}{2}$$

where R is the symbol rate.

MSK modulation results in lower sidelobe energy levels than both QPSK and SQPSK modulation. Since the Q data is delayed half a symbol (within the block), it is preferable to select a simulation step size that yields an even number of simulation steps per symbol period.

The `MSK` block accepts two binary signals as its input: I and Q data, respectively.

x_1 = I channel data (binary)

x_2 = Q channel data (binary)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Unmodulated carrier phase (rad) (optional)

$$y_1(t) = Ad_I \cos(\pi f_s(t-\tau))\cos(2\pi f_c(t-\tau)) + jAd_Q \sin(\pi f_s(t-\tau))\sin(2\pi f_c(t-\tau))$$

$$y_2(t) = 2\pi f_c(t-\tau) + \phi \qquad \phi = \frac{\pi\theta}{180} \qquad \tau = \text{data start time}$$

$$d_I(x_1) \in \{\pm 1\} \qquad d_Q(x_2) \in \{\pm 1\}$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Constellation Rotation

Specifies the constellation rotation θ in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at π/4 radians.

Data Rate

Specifies the data rate f_s in symbols/second.

Data Start Time

Specifies the start time τ of the input data in seconds.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

PM Modulator

This block performs *phase modulation* (PM) of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `PM` block belongs to the family of analog modulators. In PM, the information is transmitted by varying the carrier phase according to the input signal level. The carrier amplitude remains constant.

The `PM` block accepts an analog signal as its input.

x = Input signal

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Phase of complex modulated signal (optional)

$$y_1(t) = Ad^{j(2\pi f_c t + \beta x + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

Modulation Index

Specifies the PM index β. It controls the extent of carrier phase variation according to the above equation, and is specified in radians/volt.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

PPM Modulator

This block performs PPM of the input signal based on the selected modulation parameters. In PPM, the information is transmitted by varying the occurrence of a rectangular pulse within a pre-defined symbol frame. The location of the pulse is proportional to the input signal level. Pulse spacing is automatically calculated, and no portion of the pulse ever occurs beyond the symbol frame boundaries.

The `PPM` block belongs to the family of digital modulators.

The `PPM` block accepts a symbol number as its input, and outputs a baseband real signal. The input is rounded to the closest allowed symbol number.

x = Input symbol number (integer [0 … N -1], where N is the number of levels)

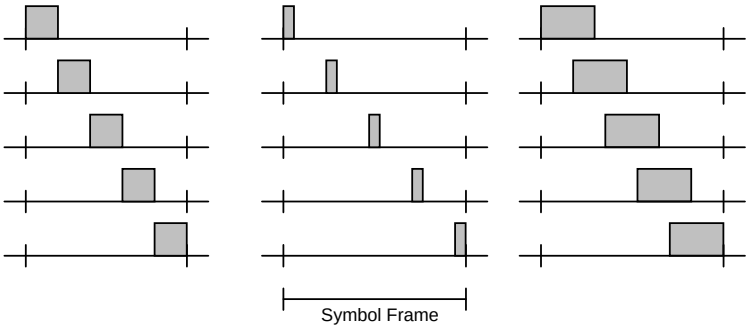
y = Baseband modulated signal

Number of Levels

Indicates the number N of possible input symbol values.

Pulse Width

Specifies the width of the rectangular pulse in seconds. The pulse width must be less than the symbol period, but may greater than, equal to, or less than the symbol period divided by the number of pulse positions. All the cases shown below are valid examples of the various output pulse positions for a five-level scheme.



Symbol Rate

Specifies the symbol rate (pulse rate) in symbols/second.

Pulse Amplitude

Specifies the amplitude of the rectangular pulse. This value is specified in volts.

Data Frame Start

Specifies the start time of the first data frame in seconds.

PSK Modulator

This block performs *phase shift keying* (PSK) modulation of the input signal based on the selected modulation parameters. In PSK modulation, the digital information is transmitted by varying the carrier phase between known phase states. The carrier amplitude remains constant. Two versions of this block are provided: one producing a complex output and the other producing a real output. The following constellations are available: BPSK, QPSK, 8-PSK, 16-PSK, and 32-PSK.

This block belongs to the family of digital modulators. It accepts as its input a binary signal (BPSK only) or a symbol number and maps it to the constellation point specified in the PSK map file.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

x_2 = Input clock (impulse train)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Unmodulated carrier phase (rad)

$$y_1(t) = Ae^{f(2\pi f_c t + \theta_d + \phi)} \qquad \phi = \frac{\pi\theta_r}{180}$$

$$y_2(t) = 2\pi f_c t + \theta \qquad \theta_d(x) = \text{data phase}$$

PSK Type

Specifies the PSK modulation type. Choices include BPSK, QPSK, 8-PSK, 16-PSK or 32-PSK. Each modulation scheme is described below in more detail.

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Constellation Rotation

Specifies the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at 0 rad for BPSK, and at π/n rad for all others, where n is the constellation size.

Gain Imbalance

Specifies the gain imbalance (Q relative to I) of the modulator in units of dBs. A positive value corresponds to greater power in the quadrature axis than in the in-phase axis.

Phase Imbalance

Specifies the phase imbalance of the modulator in degrees as a deviation from ideal. A positive value correspond to a clockwise rotation of the Q axis relative to the I axis. For example, 10 degrees imbalance implies an angle of 80 degrees between the I and Q axes, instead of the ideal 90 degrees.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

Select File

Opens the Select File dialog box for selecting a PSK constellation map file.

Browse File

Opens the selected PSK constellation map file using Notepad.

PSK File Path

Specifies the DOS path to the desired PSK constellation map file. The format of the map file is described below:

File header (anything)
modulation keyword
symbol # for 1st constellation point, symbol # for 2nd constellation point
...
symbol # for last constellation point
next modulation keyword [optional]
symbol # ... [optional]

Valid modulation keywords are *bpsk*, *qpsk*, *8psk*, *16psk* and *32psk*. They must be specified in lowercase letters. After the modulation keyword, data can be arranged as desired starting on the next line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 100 characters.

Constellation point numbering starts from the positive *I* axis and proceeds counterclockwise. Each map file may contain multiple modulation mappings: one for each PSK type. Below is an example of a Gray encoded 8-PSK map file, illustrating the use of various data delimiters. Gray coding makes neighboring constellation points differ by only 1 bit.

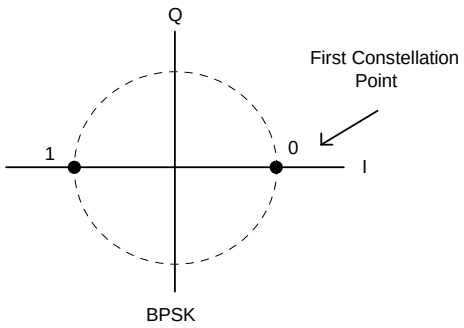
PSK Map File: Gray Coded Mapping
8psk
0 1 3, 2
6
7, 5 4

BPSK

This option performs BPSK modulation of the input signal based on the specified block parameters. In BPSK modulation, the digital information is transmitted by varying the carrier phase between two states spaced by π rad.

The BPSK block accepts a binary signal {0, 1} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).

$$\theta_d + \begin{cases} 0 & \text{if } x \le 0.5 \\ \pi & \text{if } x > 0.5 \end{cases}$$

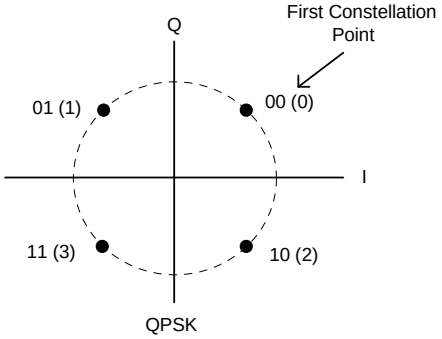


QPSK

This option performs *quadrature phase shift keying* (QPSK) modulation of the input signal based on the specified block parameters. In QPSK modulation, the digital information is transmitted by varying the carrier phase among four states equally spaced at π/2 rad increments.

The `QPSK` block accepts a symbol number {0, 1, 2, 3} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file PSK_GRAY.DAT).

$$\theta_d = \begin{cases} \pi/4 & x \le 0.5 \\ 3\pi/4 & 0.5 < x \le 1.5 \\ -\pi/4 & 1.5 < x \le 2.5 \\ -3\pi/4 & x > 2.5 \end{cases}$$

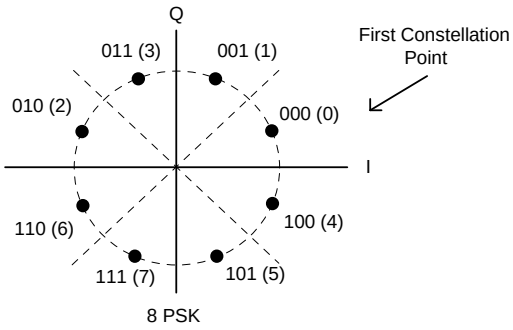


The default first constellation point is at π/4 radians.

8-PSK

This option performs *eight phase shift keying* (8-PSK) modulation of the input signal based on the specified block parameters. In 8-PSK modulation, the digital information is transmitted by varying the carrier phase among eight states equally spaced at π/4 rad increments.

The `8-PSK` block accepts a symbol number {0, 1, 2, ..., 7} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).

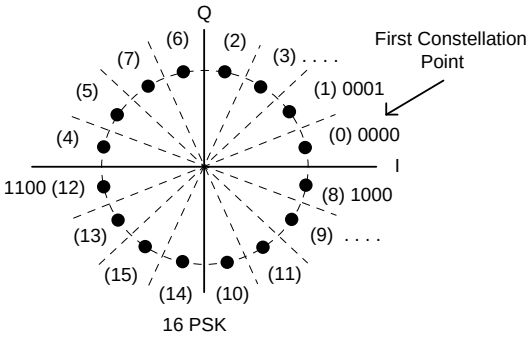


The default first constellation point is at π/8 radians.

16-PSK

This option performs *16 phase shift keying* (16-PSK) modulation of the input signal based on the specified block parameters. In 16-PSK modulation the digital information is transmitted by varying the carrier phase among 16 states equally spaced at π/8 rad increments.

The `16-PSK` block accepts a symbol number {0, 1, 2, ..., 15} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).



The default first constellation point is at π/16 rad.

32-PSK

This option performs *32 phase shift keying* (32-PSK) modulation of the input signal based on the specified block parameters. In 32-PSK modulation the digital information is transmitted by varying the carrier phase among 32 states equally spaced at π/16 rad increments.

The `32-PSK` block accepts a symbol number {0, 1, 2, ..., 31} as its input and maps it to the constellation point specified in the PSK map file.

The default first constellation point is at π/32 rad.

QAM/PAM Modulator

This block performs *quadrature amplitude modulation* (QAM) or *pulse amplitude modulation* (PAM), depending on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output. The following constellations are available: 16-QAM, 32-QAM, 64-QAM, 256-QAM, 4-PAM, and 8-PAM.

The `QAM/PAM` block belongs to the family of digital modulators. This block accepts a symbol number as its input and maps it to the constellation point specified in the QAM/PAM map file.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

y_1 = Modulated signal ([Re, Im] for complex)

y_2 = Unmodulated carrier phase (rad) (optional)

$$y_1 = \frac{\alpha}{2} A_d e^{j(2\pi f_c t + \theta_d + \phi)} \qquad \phi = \frac{\pi \theta_r}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

$\theta_d(x)$ = data phase

$A_d(x)$ = data amplitude

QAM/PAM Type

Indicates the selected modulation scheme. The available choices are 16-QAM, 32-QAM, 64-QAM, 128-QAM, 256-QAM, 4-PAM, 8-PAM, and 16-PAM. Each scheme is described below in more detail (or in Help file, click on the above links).

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Constellation Spacing

Indicates the spacing between adjacent constellation points α in volts.

Constellation Rotation

Indicates the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at ($\alpha/2$, $\alpha/2$) in the (I, Q) plane for QAM signals and ($\alpha/2$, 0) for PAM signals.

Gain Imbalance

Specifies the gain imbalance (Q relative to I) of the modulator in units of dBs. A positive value corresponds to greater power in the quadrature axis than in the in-phase axis.

Phase Imbalance

Specifies the phase imbalance of the modulator in degrees as a deviation from ideal. A positive value correspond to a clockwise rotation of the Q axis relative to the I axis. For example, 10 degrees imbalance implies an angle of 80 degrees between the I and Q axes, instead of the ideal 90 degrees.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

Select File

Opens the Select File dialog box for selecting a QAM constellation map file.

Browse File

Opens the selected QAM constellation map file using Notepad.

QAM File Path

Specifies the DOS path to the desired QAM constellation map file. The format of the map file is described below:

File header (up to 100 characters)
modulation keyword
symbol # for 1st constellation point, symbol # for 2nd constellation point
...
symbol # for last constellation point
next modulation keyword [optional]
symbol # ... [optional]

Valid modulation keywords are *16qam, 32qam, 64qam, 128qam, 256qam, 4pam, 8pam,* and *16pam*. They must be specified in lowercase letters. After the modulation keyword, data can be arranged as desired starting on the next line. Valid data delimiters are commas, blank space, tabs and carriage returns. The maximum allowed line length is 100 characters.

Constellation point numbering starts at the upper left corner and increments by row going from left to right. Each map file can contain multiple modulation mappings—one for each modulation scheme. Below is an example of a map file specifying both gray-coded 8-PAM and 16-QAM constellations. Gray coding makes neighboring constellation points differ by only 1 bit.

QAM Map File: Gray Coded Mapping

8pam
0 1 3 2 6 7 5 4

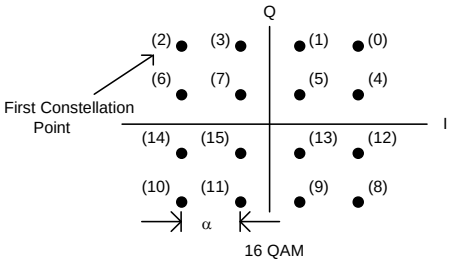
16qam
2 3 1 0
6 7 5 4
14 15 13 12
10 11 9 8

Note: The default QAM map file (QAM_GRAY.DAT) contains mappings for all QAM/PAM constellations except 32-QAM and 128-QAM.

16-QAM

This block performs *16-level quadrature amplitude modulation* (16-QAM) of the input signal based on the selected modulation parameters. In 16-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 16 equally spaced states in the (I, Q) plane.

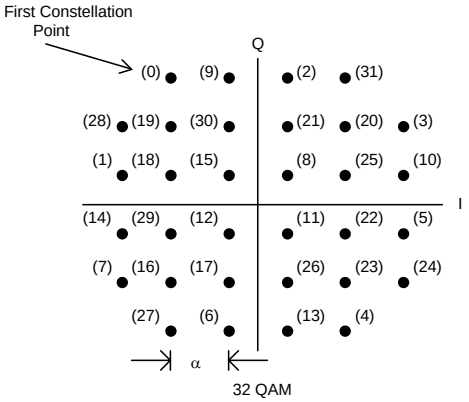
The `16-QAM` block accepts a symbol number {0, 1, 2, ..., 15} as its input and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM map file (QAM_GRAY.DAT), which implements a Gray-coded 16-QAM constellation.



32-QAM

This block performs *32 cross quadrature amplitude modulation* (32-QAM) of the input signal based on the selected modulation parameters. In 32-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 32 equally spaced states in the (I, Q) plane.

The `32-QAM` block accepts a symbol number {0, 1, 2, ..., 31} as its input and maps it to the constellation point specified in the QAM map file. The example that follows assumes the V.32 QAM map file (V32QAM.DAT).

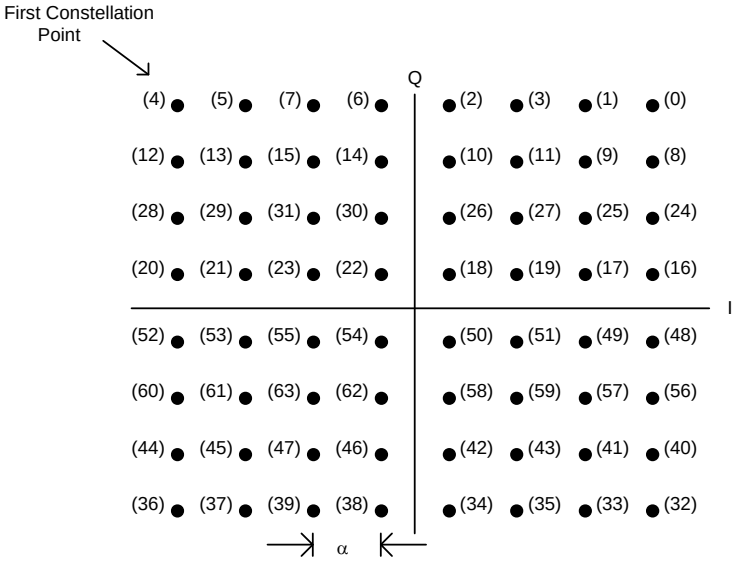


Note: For true V.32, the above constellation must be rotated +45°.

64-QAM

This block performs *64-level quadrature amplitude modulation* (64-QAM) of the input signal based on the selected modulation parameters. In 64-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 64 equally spaced states in the (I, Q) plane.

The `64-QAM` block takes a symbol number {0, 1, 2, ..., 63} as its input and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM map file (QAM_GRAY.DAT), which implements a Gray-coded 64-QAM constellation.



128-QAM

This block performs *128-level quadrature amplitude modulation* (128-QAM) of the input signal based on the selected modulation parameters. In 128-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 128 distinct states in the (I, Q) plane.

The `128-QAM` block accepts a symbol number (0, 1, 2, ..., 127) as its input and maps it to the constellation point specified in the QAM map file. The default QAM map file does not contain a 128-QAM constellation. A sample 128-QAM map file (QAM128.DAT), which is not Gray encoded, is included for reference (not shown).

256-QAM

This block performs *256-level quadrature amplitude modulation* (256-QAM) of the input signal based on the selected modulation parameters. In 256-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 256 equally spaced states in the (I, Q) plane.

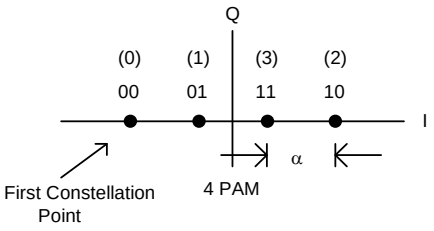
The `256-QAM` block accepts a symbol number {0, 1, 2, ..., 255} as its input and maps it to the constellation point specified in the QAM map file. The default QAM map file (QAM_GRAY.DAT) implements a Gray-coded 256-QAM constellation (not shown).

4-PAM

This block performs *four-level pulse amplitude modulation* (4-PAM) of the input signal based on the selected modulation parameters. The modulation scheme is also known as *amplitude shift keying* (ASK). In 4-PAM, the digital information is transmitted by varying the carrier amplitude among four equally spaced amplitude states. The carrier phase remains constant.

The `4-PAM` block accepts as input a symbol number {0, 1, 2, 3} and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM/PAM map file (QAM_GRAY.DAT).

$$A_d = \begin{cases} -3 & x \le 0.5 \\ -1 & 0.5 < x \le 1.5 \\ +3 & 1.5 < x \le 2.5 \\ +1 & x > 2.5 \end{cases} \qquad \theta_d$$

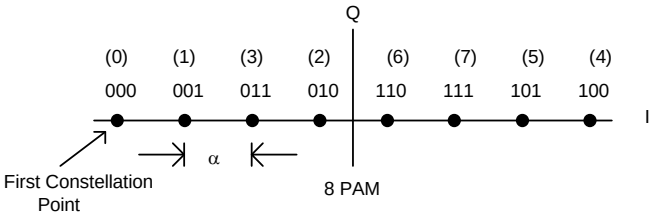


8-PAM

This block performs *eight-level pulse amplitude modulation* (8-PAM) of the input signal based on the selected modulation parameters. This modulation scheme is also known as ASK. In 8-PAM, the digital information is transmitted by varying the carrier amplitude among eight equally spaced amplitude states. The carrier phase remains constant.

The `8-PAM` block accepts a symbol number {0, 1, ..., 7} as its input and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM/PAM map file (QAM_GRAY.DAT).

$$A_d = \begin{cases} -7 & x \le 0.5 \\ -5 & 0.5 < x \le 1.5 \\ -1 & 1.5 < x \le 2.5 \\ -3 & x > 2.5 \end{cases} \qquad A_d = \begin{cases} +7 & 3.5 < x \le 4.5 \\ +5 & 4.5 < x \le 5.5 \\ +1 & 5.5 < x \le 6.5 \\ +3 & x > 6.5 \end{cases} \qquad \theta_d$$



16-PAM

This block performs *16-level pulse amplitude modulation* (16-PAM) of the input signal based on the selected modulation parameters. This modulation scheme is also known as Amplitude Shift Keying (ASK). In 16-PAM, the digital information is transmitted by varying the carrier amplitude between eight equally spaced amplitude states. The carrier phase remains constant.

The `16-PAM` block accepts a symbol number (0, 1, ..., 15) as its input and maps it to the constellation point specified in the QAM map file. The default QAM/PAM map file (QAM_GRAY.DAT) implements a Gray-coded 16-PAM constellation.

SQPSK Modulator

This block performs *staggered quadrature phase shift keying* (SQPSK) modulation of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `SQPSK` block belongs to the family of digital modulators, and is also known as *Offset QPSK* (OQPSK). In SQPSK modulation, the digital information is transmitted by varying the carrier phase among four states equally spaced at π/2 rad increments. The carrier amplitude remains constant. The data on the Q channel input is delayed ½ symbol duration relative to the I channel data. This ensures that at any given time, only one of the two data channels (I or Q) may undergo a transition. As a result, the modulated signal phase never changes by more than π/2 rad, and the modulated spectrum exhibits lower sidelobes than ordinary QPSK. Since the Q data is delayed ½a symbol (within the block), it is preferable to select a simulation step size that yields an even number of simulation steps per symbol period.

The `SQPSK` block accepts two binary data streams as its input (I and Q data, respectively) and uses Gray encoding in its mapping. A real-to-integer conversion (rounding) is performed on the inputs.

Note: Due to the staggered timing of the I and Q data bits, demodulation of a SQPSK signal requires two separate detectors. One approach is to use two BPSK detectors configured with 90 and 0 degrees of constellation rotation (for I and Q respectively), as shown in the "SQPSK_Modulator.vsm" diagram located in the Modulators examples folder.

x_1 = I channel data (binary)

x_2 = Q channel data (binary)

y_1 = Modulated signal ([Re, Im] for complex)

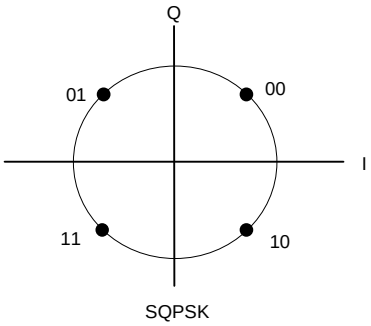
y_2 = Unmodulated carrier phase (rad) (optional)

$$y_1(t) = Ae^{j(2\pi f_c t + \theta_d + \phi)} \qquad \phi = \frac{\pi\theta_r}{180}$$

$$y_2(t) = 2\pi f_c t + \phi \qquad \theta_d(x_1, x_2) = \text{data phase}$$

T = symbol duration

$$\theta_d = \begin{cases} \pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 \le 0.5 \\ 3\pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 > 0.5 \\ -\pi/4 & x_1 > 0.5 \text{ and } \tilde{x}_2 \le 0.5 \\ -3\pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 > 0.5 \end{cases} \qquad \text{where } \tilde{x}_2 = x_2(t - T/2)$$



Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Constellation Rotation

Specifies the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at $\pi/4$ rad.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of [0, 2pi].

Multirate Support category

Blocks in the Multirate Support category include `Clock Edge`, `Clock Extend`, and `Interpolator`.

Clock Edge

This block is used to "latch" and propagate a clock pulse generated from a slower rate "locally stepped compound block". Without the use of this block, it's possible for a clock's "high" state to span multiple simulation time steps across such a boundary, resulting in the erroneous perception of multiple clock pulses having occurred.

This block operates by forcing a clock's "high" state to last for only one simulation sample irrespective of the actual duration of the input pulse. Once a high state is detected, the occurrence of a low state is required to reset the latch. This block should be placed just outside of the slower locally stepped compound block (output side).

x = Input clock signal

y = Output clock signal

Clock Extend

This block is used to propagate a clock pulse into a slower rate "locally stepped compound block". Without the use of this block, it's possible for clock pulses to be "dropped" across such a boundary.

This block operates by extending a clock's "high" state across multiple simulation steps so that the slower running block can detect it. For this block to operate properly, it's important to accurately specify the local clock rate of the compound block that follows. This block should be placed just outside of the slower locally stepped compound block (input side).

x = Input clock signal

y = Output clock signal

Local Sim Rate of Next Block

Specifies the simulation rate in hertz of the locally stepped compound block that follows.

Interpolator

This block provides linear waveform interpolation for a signal passing between differing sample rate regions. This block is best suited for analog waveforms and is especially useful when the sample rates in question are not integer multiples of each other (i.e. the sample times of the two blocks do not generally coincide).

Use of this block avoids the "staircase" effect that can otherwise occur on analog signals when crossing from one sample rate region to another. Such an effect occurs when two or more sample times at the faster simulation rate occur between sample times at the lower rate. This block also provides added accuracy when going

This block introduces a single time step delay to the output at the sample rate of the input signal , e.g. the main sim rate for a High-to-Low case and the local rate for a Low-to-High case.

x = Input signal

y = Output interpolated signal

Local Sim Rate of Next Block

Specifies the simulation rate in hertz of the locally stepped compound block that follows (high to low case) or precedes (low to high case) the `Interpolator` block.

Mode

High to Low

Use this setting when going from a higher sample rate environment to a lower sample rate environment.

Low to High

Use this setting when going from a lower sample rate environment to a higher sample rate environment.

Operator category

Blocks in the Operators category include `A/D Converter`, `Compander`, `Complex FFT/IFFT`, `Conversions`, `D/A Converter`, `Delay (Complex)`, `Delay (Real)`, `Gain (dB), I/Q Mapper`, `Max Index`, `Modulo`, `Integrate and Dump (Complex)`, `Integrate and Dump (Real)`, `Oscilloscope`, `Phase Rotate`, `Phase Unwrap`, `Polynomial`, `Spectrum Analyzer, Variable Delay, Vector FFT,` and `Vector Merge.`

A/D Converter

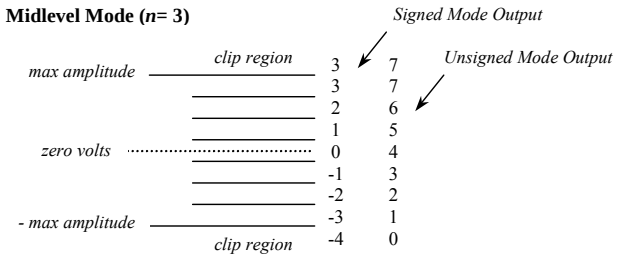
This block implements an *analog-to-digital converter* (ADC). The input signal is quantized according to the number of specified resolution bits *n* and output as an integer over the range [0, 2^*n* -1] or [-2^(*n*-1), 2^(*n*-1) -1] depending on the output mode selection (can be signed or unsigned). For the case of an 8-bit ADC, the output range would correspond to either [0, 255] or [-128, 127] respectively.

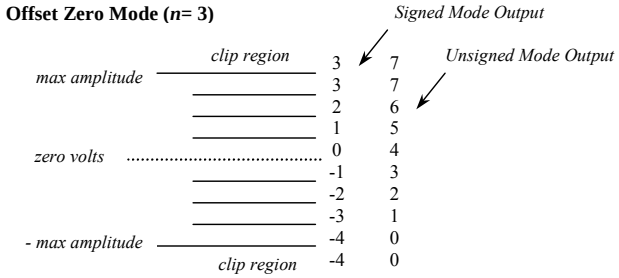
The input signal is assumed to vary over a range of +/- *Max Amplitude* volts centered about zero volts. To sample a non-zero centered waveform, please indicate the desired center of the ADC range by specifying an appropriate Input Range Offset.

Two options exist for handling conversion of the center-range input level: midlevel and offset. Assuming a zero input range offset, *midlevel* mode causes an input value in the range [-0.5Δ*q*, 0.5Δ*q*] to be converted to level 0 (for *Signed* output mode), where Δ*q* is a quantization level. In *offset* mode, a positive value in the range [0, Δ*q*] is converted to level 0, while a negative value in the range [-Δ*q*, 0-] is converted to level -1 (for *Signed* output mode). The distinction between the two modes is further illustrated in the diagrams below (a zero input range offset is assumed).

x = Input signal

y = Quantized integer output





Number of Bits

Specifies the number of bits *n* of the quantizer. This parameter determines the number of ADC quantization levels, which is equal to 2^*n*.

Max Amplitude

Indicates the signal amplitude in volts (zero-centered) that corresponds to the ADC clip level (see figures above). The ADC range, assuming the range offset is zero, is then equivalent to [-maxAmp, +maxAmp].

Input Range Offset

Allows for a voltage offset to be applied to the ADC input range. Using this parameter, the ADC input range can be easily shifted up or down by a fixed offset and becomes [-maxAmp+offset, +maxAmp+offset].

Center Range Mode

Midlevel

In this mode, the ADC mid-range input voltage (nominally zero) falls at the center of the 0 quantization level (signed mode) or at the center of the (2^*n*)/2 quantization level (unsigned mode). Note: In midlevel mode the ADC range of [-maxAmp, maxAmp] is divided into

(2^*n* - 1) levels (e.g. 255 levels for an 8-bit ADC). Please refer to the figure above for additional clarification.

Offset

In this mode, the ADC mid-range input voltage (nominally zero) falls at the boundary between levels 0 and 1 (signed mode) or at the boundary between levels (2^*n*)/2 -1 and (2^*n*)/2 (unsigned mode). Note: In offset mode the ADC range of [-maxAmp, maxAmp] is divided into 2^*n* levels, e.g. 256 levels for an 8-bit ADC. Please refer to the figure above for additional clarification.

Output Mode

Unsigned

Indicates that the ADC output will be in the range [0, 2^*n* -1].

Signed

Indicates that the ADC output will be in the range [-2^(*n*-1), 2^(*n*-1) -1].

Compander

This block implements signal companding and its inverse. Companding refers to a process of nonlinear amplitude compression usually performed prior to A/D quantization of a signal. This process is employed when it is desirable to obtain a nonlinear quantization of the original signal. You can specify either μ-law or *A*-law companding, as described below. A standard value used in μ-law companding is μ=255, while for *A*-law companding *A*=87.56 is often used. Since the compander expects values in the range [-1, 1], the input signal is normalized by the Max Value parameter.

x = Input signal

y = Companded output

In the equations below $x = \frac{x_1}{x_{\max}}$

$$y = \operatorname{sgn}(x) \cdot \frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)} \qquad |x| \le 1 \qquad \mu - \text{law companding}$$

$$y = \begin{cases} \operatorname{sgn}(x) \cdot \dfrac{1 + \ln A|x|}{1 + \ln A} & \dfrac{1}{A} < |x \le 1| \\ \operatorname{sgn}(x) \cdot \dfrac{A|x|}{1 + \ln A} & 0 \le |x| \le \dfrac{1}{A} \end{cases} \qquad A - \text{law companding}$$

Compander Mode

Compress

Indicates that the block is operating in compression mode.

Expand

Indicates that the block is operating in expansion mode. This operation is the inverse of compression.

Compander Type

μ-Law

Indicates μ-law companding.

A-*Law*

Indicates *A*-law companding.

Max Value

Indicates the maximum allowed magnitude (x_{max}) of the input signal. This value is used to normalize the input to the range [-1, 1]. When in expand mode, Max Value scales the output.

A Value

Indicates the value of *A* for *A*-law companding. The value of *A* greater than or equal to 1. A value of *A* equal to 1 indicates no compression.

μ Value

Indicates the value of μ for μ-law companding. The value of μ must be greater than 0.

Complex Exponential

This block outputs the complex exponential of the input signal.

$$y(t) = e^{j\,x(t)}$$

x = Phase angle (rad)

y = Complex output [Re, Im]

Complex FFT/IFFT

This block computes either the forward *fast Fourier transform* (FFT) or inverse IFFT of the input complex signal. You may select from a variety of window functions in performing the FFT. The *N* point FFT/IFFT operation is single shot and is started by an external trigger. Results are viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line and *x*-axis output (either frequency or time) are provided for driving the `plot` block. FFT results are presented over the range [-*fs*/2, *fs*/2], where *fs* is the simulation sampling rate in hertz.

In order to view the entire result, the simulation time range should extend at least 2x*N* simulation steps from where the external trigger is applied. When the output is represented in dBm or dBm/Hz, the output is limited to -300 dBm.

x_1 = Input trigger (high > 0.5)

x_2 = Real part (or magnitude) of complex input

x_3 = Imaginary part (or phase) of complex input

y_1 = Output trigger (0 or 1) for plot block

y_2 = Real part (or magnitude) of complex output

y_3 = Imaginary part (or phase) of complex output

y_4 = *X*-axis drive signal: Frequency (*x*Hz) in FFT mode or time (seconds) in IFFT mode

FFT Mode

Forward FFT

Inverse FFT

Specifies either an FFT or IFFT operation.

FFT Window Type

Selects the desired window function to be used in computing the FFT. This parameter is not applicable to the IFFT operation.

FFT Representation

Mag / Phase

Indicates that the FFT is represented by magnitude and phase components.

Real / Imaginary

Indicates that the FFT is represented by real and imaginary components.

FFT Size

Specifies the size of the FFT or IFFT operation. Valid range is 8 to 8 Meg.

Unwrap Phase

Specifies that the computed phase response should be unwrapped. When not selected, the output phase is restricted to $[-\pi, +\pi]$.

Remove Linear Phase

Specifies that linear phase, based on the provided delay input, should be removed from the FFT phase response result.

Delay

Specifies the delay, in seconds, corresponding to the amount of linear phase to be removed from the phase result. This setting only applies when the Remove Linear Phase option is selected.

Number of FFT Averages

Specifies the number of consecutive FFTs (each of N points), which are averaged before producing the output spectrum.

Output Freq. Units (FFT only)

Specifies the frequency units for the x-axis drive signal (y_4). Choices include hertz, kilohertz, megahertz, and gigahertz.

Output Mode (FFT only)

Standard

Indicates unscaled FFT output. This mode is most practical when viewing a filter response, as a level of 1 corresponds to unity gain.

Power Spectrum

Indicates that the FFT result represents the signal's power spectrum in dBm. The output value represents total energy per bin. A load of 50 Ohms is assumed.

Spectral Density

Indicates that the FFT result represents the signal's power spectral density in dBm/hertz. A load of 50 Ohms is assumed.

Conversions

This block implements many common conversions. The desired conversion is selected by choosing the appropriate radio button in the block's setup dialog box. Two versions of this block are available, one for scalar signals and another for vector inputs.

Decibels to Power

The block converts a decibel input to a power value.

x = Input value in decibels

y = Power output value

$$y = 10^{(x/10)}$$

Decibels to Real

The block converts a decibel input to a real number.

x = Input value in decibels

y = Output real value

$$y = 10^{(x/20)}$$

Degrees to Radians

The block converts from degrees to radians.

x = Input value in degrees

y = Output value in radians

$$y = \frac{x\pi}{180}$$

Hertz to Rad/sec

The block converts from hertz to radians/second.

x = Input value in hertz

y = Output in radians/second

$$y = 2\pi x$$

Mag/Phase to Real/Im

The block converts a complex input represented in magnitude/phase format to real/imaginary format. This option is not available in the vector version of the block.

x_1 = Input complex magnitude

x_2 = Input complex phase

y_1 = Real part of complex output

y_2 = Imaginary part of complex output

$$y_1 = x_1 \cos(x_2) \qquad y_2 = x_1 \sin(x_2)$$

Power to Decibels

The block converts a power input value to decibels. The input must be greater than zero.

x = Input power value

y = Output in decibels

$$y = 10\log(x) \qquad x > 0$$

Radians to Degrees

The block converts from radians to degrees.

x = Input value in radians

y = Output value in degrees

$$y = \frac{180x}{\pi}$$

Rad/sec to Hertz

The block converts from radians/second to hertz.

x = Input value in radians/second

y = Output in hertz

$$y = \frac{x}{2\pi}$$

Real to dB

The block converts a real input to decibels. The input must be greater than 0.

x = Input real value

y = Output in decibels

$$y = 20\log(x) \qquad x > 0$$

Real/Im to Mag/Phase

The block converts a complex input represented in real/imaginary format to magnitude/phase format. A four-quadrant arctangent function is used, which returns values in the range of -π to π. If both x_1 and x_2 are 0, zero phase is returned. This option is not available in the vector version of the block.

x_1 = Real part of complex input

x_2 = Imaginary part of complex input

y_1 = Complex magnitude

y_2 = Complex phase

$$y_1 = \sqrt{(x_1)^2 + (x_2)^2} \qquad y_2 = \text{atan}(x_2, x_1)$$

D/A Converter

This block implements a Digital to Analog Converter (DAC). Model parameters include the number of quantization bits (N) and the minimum and maximum output amplitude levels. For internal efficiency, this block does not perform any range checking of the input value to ensure it is within the expected range. To implement range limiting, simply add a `limit` block prior to the DAC.

x = Input digital signal (integer)

y = Output analog signal

$$y = A_{\min} + x \cdot \Delta q \quad \text{(unsigned mode)}$$

$$y = A_{\min} + (x + \frac{k}{2}) \cdot \Delta q \quad \text{(signed mode)}$$

$$\text{where} \quad \Delta q = \frac{(A_{\max} - A_{\min})}{k - 1} \; ; \; k = 2^N \; ; \; N = \text{num. bits}$$

Number of Bits

Specifies the number of bits N for the DAC. The number of levels is 2^N.

Input Mode

Unsigned Int

Specifies that the range of the input digital signal is unsigned and should be limited to [0, (2^*N*) -1].

Signed Int

Specifies that the range of the input digital signal is signed and should be limited to [-2^(*N* - 1), 2^(*N* - 1) - 1].

Max Output Level

Specifies the maximum output level for the DAC in volts. This corresponds to the analog output level associated with the largest expected integer input value (e.g. level 255 for an 8 bit DAC in unsigned mode).

Min Output Level

Specifies the minimum output level for the DAC in volts. This corresponds to the analog output level associated with the smallest expected integer input value (e.g. level 0 for unsigned mode or level -128 for an 8 bit DAC in signed mode).

Delay (Complex)

This block implements a multiple unit-delay block. This block only operates on main simulation steps and will disregard the intermediate steps associated with some integration methods (e.g., Runge Kutta). For these steps, the previous value is held. The internal real and imaginary shift registers are initialized to the Initial Condition parameter values.

x = Complex input signal [Re, Im]

y = Delayed version of complex input signal [Re, Im]

$y_{[n]} = x_{[n-k]}$ k = delay size n = simulation step index

Delay

Specifies the delay in simulation steps or seconds, depending on the selected Delay Mode. Valid range is from 1 to 262,143 steps.

Initial Condition (Real)

Indicates the block's real output value for the first Delay simulation steps.

Initial Condition (Imaginary)

Indicates the block's imaginary output value for the first Delay simulation steps.

Delay Mode

Sim Steps

Indicates the delay size is specified in simulation steps.

Seconds

Indicates the delay size is specified in seconds.

Delay (Real)

This block implements a multiple unit-delay block. This block only operates on main simulation steps and will disregard the intermediate steps associated with some integration methods (e.g., Runge Kutta). For these steps, the previous value is held. The internal shift register is initialized to the Initial Condition parameter value.

x = Input signal

y = Delayed version of input signal

$y_{[n]} = x_{[n-k]}$ k = delay size n = simulation step index

Delay

Specifies the delay in simulation steps or seconds, depending on the selected Delay Mode. Valid range is from 1 to 262,143 steps.

Initial Condition

Indicates the block's output value for the first Delay simulation steps.

Delay Mode

Sim Steps

Indicates the delay size is specified in simulation steps.

Seconds

Indicates the delay size is specified in seconds.

Gain (dB)

This block lets you specify a gain value in decibels.

x = Input signal

y = Scaled output value

$y = x \cdot 10^{(gain/20)}$

Gain

Indicates the gain value in decibels. This value may be positive or negative.

Integrate & Dump (Complex or Real)

These blocks implement integrate and dump operations on the input signal. The input signal is continuously integrated and the integral output is periodically dumped and reset to a specified value. The dump rate can be specified internally or through an external clock.

When one of these blocks is used to demodulate a baseband phase modulated signal, it should be followed by the appropriate detector block for the modulation used. These blocks accept and output either a real signal or a complex signal depending on the selected version of the block.

x_1 = Input signal ([Re, Im] for complex)

x_2 = Optional external clock (0, 1) (impulse train)

y_1 = Integrator output ([Re, Im] for complex)

y_2 = Output clock (impulse train)

Reset Value

Indicates the value to which the integral resets when dumped. This parameter is also used as the integrator initial condition at simulation start.

Scale Factor

Specifies the output scaling factor. This parameter is usually set to the inverse of the data rate so as to cancel out the integrator Δt scaling.

Dump Timing

Internal

Indicates that the Dump Rate and Initial Delay (prior to the first dump) are specified internally.

External

Indicates that an outside dump clock is supplied at the clock input port (x_2).

Suppress First Output Clock

When selected, the first output clock pulse is suppressed when in external clock mode. The first external clock pulse still dumps and resets the internal I&D buffer. This option makes it easier to achieve frame synchronization in some diagrams by making the first output pulse from the I&D block correspond with the first valid dumped value.

Dump Rate

Specifies the block's dump rate in hertz. This option is only available when internal dump timing mode is selected.

Initial Delay

Specifies the initial delay in seconds to the beginning of the first integration period. The first dump event will occur at t = Initial Delay + Dump Period. This parameter is only available when internal dump timing mode is selected.

Output Mode

Continuous

Indicates that the output is the instantaneous integrator output.

Held

Indicates that the output is held constant between dump clock pulses.

Integration Method

Euler

Specifies the Euler integration method (forward difference).

Trapezoidal

Specifies the trapezoidal integration method.

Backward Diff.

Specifies the backwards difference integration method.

IQ Mapper

This block lets you specify an arbitrary IQ constellation via an external file. The block accepts a symbol value in the range of [0, N-1] and outputs a pair of I and Q values as specified by the mapping file. This block can be followed by the `IQ Modulator` block to achieve modulation of arbitrary user defined constellations.

x = Input symbol number (0,1..N-1)

y_1 = I output value

y_2 = Q output value

Constellation Size

Specifies the size N of the constellation used by the `IQ Mapper` block.

Select File

Opens the Select File dialog box for selecting the IQ Mapper constellation file.

Browse File

Opens the selected IQ Mapper constellation file using Notepad.

File Path

Specifies the DOS path to the desired IQ Mapper constellation file. The format of the mapping file is described below:

File header (user defined)

Symbol number I output Q output

...

The *symbol number* values need not be entered in increasing order, although it is highly recommended to do so for clarity. The table must contain a total of *N* entries, where *N* is the constellation size.

Table entries may be separated by blank spaces, tabs, or commas. Blank lines are also acceptable. An example of an IQ map file is shown below:

```
Arbitrary IQ Constellation Map File     (Header line)
0   1.2       1
1   -0.9      1.1
2   -1        -1.15

3   -1.05     -0.95
```

The above example illustrates a QPSK constellation having slight imbalance characteristics.

Max Index

This block returns the index of the largest input signal. The block can be configured to accept up to 16 inputs. A typical application of this block is in the creation of M-ary decision circuits (e.g. detection of MFSK). Should two or more inputs share the largest value, the output index will be that of the lowest input connection in the set.

x_1 = Input #1

x_2 = Input #2

...

x_n = Input #*N*

y = Output index value (either {0,1,.., *N*-1} or {1, 2, .. *N*})

Number of Inputs

Specifies the number of input connections *N*. Valid range is 2 to 16.

Index Mode

Start at Zero

Specifies the output range to be {0,1,.., *N*-1}.

Start at One

Specifies the output range to be {1, 2, .. *N*}.

Modulo

This block performs either real-valued or integer modulo operations. The output represents the remainder after integer division of the input value by the specified modulo value. The result will fall in the range [0, mod val) when specifying a positive modulo value, and (mod val, 0] when specifying a negative modulo value.

When in integer modulo mode, the input value undergoes a tolerance adjustment and is then truncated to an integer prior to the modulo operation. Also, the input value and the modulo value must fall within the accepted range for signed long variables.

x = Input signal

y = Remainder from modulo operation

Modulo

Indicates the modulo value, which can be a real number or an integer. In integer mode, this entry is truncated.

Modulo Mode

Integer

Indicates integer modulo operation.

Real

Indicates real modulo operation.

Tolerance

Indicates the value added to the input signal before performing the modulo operation. This step is necessary due to possible floating-point rounding errors. For example, after an arbitrary operation, an expected integer value of 5 might be represented in floating point notation as 4.99999...9.

This value is only applicable to integer mode.

The default tolerance value is 1e-6.

Oscilloscope

This block emulates the use of a two input channel oscilloscope. You can specify either input as the trigger source, specify falling or rising edge triggering, and set the trigger threshold level. The overall time span is also user defined. The output is viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line, the *A* and *B* channel traces, and time axis outputs are provided for driving the `plot` block.

Once enabled by an external start pulse, the `Oscilloscope` block monitors the input signal for a threshold crossing, i.e. a trigger event. Once triggered (and the specified trigger delay has elapsed) it reads in *N* data points on both channels, where *N* is based on the desired time span, and then displays both traces all at once as a vector. Upon completing each trace, the `Oscilloscope` block resets itself and awaits the next trigger event before repeating the above cycle.

x_1 = Start pulse (high > 0.5)

x_2 = *A* Channel input

x_3 = *B* Channel input

y_1 = Output trigger (0 or 1) for plot block

y_2 = *A* Channel vector trace output

y_3 = *B* Channel vector trace output

y_4 = Time axis for plot's *X*-axis

Trigger Channel

A Channel

Specifies Channel *A* is to be used for triggering the display.

B Channel

Specifies Channel *B* is to be used for triggering the display.

Triggering Mode

Rising Edge

Forces the displayed trace to start following a rising crossing of the specified threshold level.

Falling Edge

Forces the displayed trace to start following a falling crossing of the specified threshold level.

Time Span

Specifies the desired time span in seconds for the display.

Threshold

Specifies the voltage level (in volts) used for triggering the display.

Trigger Delay

Specifies the time delay (in seconds) to wait after a trigger event before starting the trace.

Number of Sweeps per Update

Specifies the number of traces to overlay on the display. Note: to avoid the appearance of retrace lines in the display, the Plot's "Line Type" should be configured for "points" mode when this setting is > 1.

Phase Rotate

This block multiplies the complex input by the complex exponential $e^{j\phi}$, which results in a phase rotation of ϕ radians.

x_1 = Complex input signal [Re, Im]

x_2 = Rotation angle (radians)

y = Complex output signal [Re, Im]

$$y = x_1(\cos x_2 + j \sin x_2)$$

Phase Unwrap

This block performs a phase unwrapping operation on the input signal, which is assumed to be represented in the range of [-180, +180] degrees or [-π, +π] radians. The block operates by comparing the input phase value to a weighted average of the previous several points. When a phase jump in excess of 180 degrees is detected, a phase wrap is declared on the input, and a value of 360 degrees is added to (or subtracted from) the output waveform. For this block to work properly, the phase variations from point to point should not be excessive (<45 degrees).

x = Input phase signal

y = Unwrapped phase signal

Units

Degrees

Indicates the input phase signal is specified in *degrees*.

Radians

Indicates the input phase signal is specified in radians.

Polynomial

This block computes *g(x)* for *g( )* up to a fifth order polynomial.

x = Input signal

y = Output signal *g(x)*

$$g(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

x^0 Coefficient

Indicates the coefficient of the x^0 term (*f* in the above formula).

x^1 Coefficient

Indicates the coefficient of the x^1 term (*e* in the above formula).

x^2 Coefficient

Indicates the coefficient of the x^2 term (*d* in the above formula).

x^3 Coefficient

Indicates the coefficient of the x^3 term (*c* in the above formula).

x^4 Coefficient

Indicates the coefficient of the x^4 term (*b* in the above formula).

x^5 Coefficient

Indicates the coefficient of the x^5 term (*a* in the above formula).

Show Formula

When selected, the polynomial formula is displayed as part of the block's name.

Decimal Digits

Specifies the number of decimal digits to use for the polynomial coefficients in the displayed block name (if the Show Formula checkbox is selected).

Spectrum (Complex or Real)

This block outputs the complex power spectrum of the input signal. The spectrum can be continuously updated (once started by the external trigger) or produced at user-defined intervals (again, using the external trigger). Results are viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line and x-axis output are provided for driving the `plot` block.

Two versions of this block exist: one complex and the other real. The real version of the block only outputs the positive side of the spectrum (since it's mirror imaged), but includes all signal energy, including that from the negative side of the spectrum. This is equivalent to doubling the FFT result at all points except dc and *fs*/2. So, for example, the complex spectrum of a -10 dBm real sinusoid of frequency f_o results in two peaks of -13 dBm at $-f_o$ and $+f_o$, while the real spectrum results in a single peak of -10 dBm at f_o.

Once triggered, the `Spectrum` block reads in *N* data points, performs an *N* point FFT, and then outputs the FFT result all at once as a vector. If you have selected to average multiple FFTs, then an averaging step is also performed prior to outputting the result. You may also select from a variety of window functions when performing the underlying FFT. Once completed, the `Spectrum` block either immediately reads in the next *N* data points (continuous mode) or waits until the next input trigger before repeating the above cycle.

Spectral results are presented over the range [-*fs*/2, *fs*/2] for the complex version, and over the range of [0, *fs*/2] for the real version, where *fs* is the simulation sampling rate in *hertz*. The output spectrum can be output in watts, dBm, or dBm/Hz. When in dBm or dBm/Hz mode, the output is limited to -300 dBm.

Note: When using this block to measure the power level of individual tones it is recommended to use the "dBm" units setting. For general spectral displays, or when wanting to measure noise levels, it is recommended to use the "dBm/Hz" units setting.

x_1 = Input trigger (high > 0.5)

x_2 = Complex input (real for real version of block)

y_1 = Output trigger (0 or 1) for `plot` block

y_2 = Magnitude (or Real) output

y_3 = Phase (or Imaginary) output

y_4 = Frequency (*x*Hz) for plot's x-axis

Trigger Mode

Continuous

Once a result has been output, the block immediately reads in more data.

Triggered

Once a result has been output, the block will wait for a new trigger before reading in new data.

FFT Window Type

Selects the desired window function to be applied to the input data prior to computing the FFT. Please note that a normalization step is also applied internally for windows other than Rectangular. This process ensures that absolute power levels in the output spectrum are not distorted by the choice of window function.

Spectral Output

Mag / Phase

Indicates that the result is represented by magnitude and phase components.

Real / Imaginary

Indicates that the result is represented by real and imaginary components.

FFT Size

Specifies the size *N* of the underlying FFT computation. Valid range is from 8 to 8 Meg.

Unwrap Phase

Specifies that the computed phase output should be unwrapped. When not selected, the output phase is restricted to [-π, +π].

Remove Linear Phase

Specifies that linear phase, based on the provided delay input, should be removed from the phase output.

Delay

Specifies the delay, in seconds, corresponding to the amount of linear phase to be removed from the phase result. This setting only applies when the Remove Linear Phase option is selected.

Number of FFT Averages

Specifies the number of consecutive power spectra (each of *N* points), which are averaged before producing the final result.

Output Freq. Units

Specifies the frequency units for the *x*-axis drive signal (y_4). Choices include hertz, kilohertz, megahertz, and gigahertz.

Power Spectrum Units

Watts

The output spectrum is represented in Watts.

dBm

The output spectrum is represented in dBm. The output values represent total energy per bin.

dBm/Hz

The output spectrum is represented as a power spectral density in dBm/Hz.

dBm/MHz

The output spectrum is represented as a power spectral density in dBm/MHz.

Load

1 Ohm

The output spectrum is referenced to a 1 ohm load.

50 Ohms

The output spectrum is referenced to a 50 ohms load.

Subsample

This block samples the input signal at a constant interval specified as an integer number of simulation time steps. An initial offset may be specified. The output can either be held constant between samples or specified as an impulse train (zero padded).

x = Input signal

y = Sampled output

Decimate by

Indicates the decimation factor. If, for example, it is set to 5, then every fifth simulation step is sampled and posted to the output port.

Offset

Indicates the offset from the first simulation step for applying the sampling. When set to 0, the sampling starts with the first simulation step.

Output Mode

Held Output

Indicates that the output signal is held constant (last decimated value) between sampled points.

Pulsed Output

Indicates that the output signal is 0 between sampled points.

Indicates the delay size is specified in seconds.

Variable Delay

This block implements a variable delay block, where the delay value (in simulation steps) is specified via an external block input. This block only operates on main simulation steps and will disregard the intermediate steps associated with some integration methods (e.g., Runge Kutta). For these steps, the previous value is held. The internal shift register is initialized to the Initial Condition parameter value. Optional data interpolation can be specified.

x_1 = Input signal

x_2 = Delay input (integer)

y = Delayed version of input signal

$y[n] = x[n-k]$ k = delay size n = simulation step index

Max Delay

Specifies the maximum delay in simulation steps implemented by the block. Valid range is from 1 to 262,143 steps. This setting is used to manage the internal memory requirements of the block. If the delay input exceeds the max value, the max value is used.

Initial Condition

Indicates the block's output value until the first delayed input sample is output.

Interpolate Data

When checked, the block applies linear interpolation to the delayed data when the delay input value is changed. The interpolation only applies to the sample being output immediately following the external change in delay setting.

Vector FFT

This block provides vector-based FFT and IFFT capabilities. The input and output signals are vectors of size N, where N is a power of two.

The `Vector FFT` block performs an FFT or IFFT operation in a single simulation step. A frame input clock controls when the calculation is to be performed. Each time the clock goes high, the input vectors are read, the FFT or IFFT operation is performed, and the result is posted to the output vectors. This block supports either real/imaginary format or magnitude/phase for the FFT data.

The `Vector FFT` block expects two input vectors at all times. If only real data is to be used, a 0 input vector of size N must be supplied as the imaginary input (this can be done using the `Vector Constant` block). Not all outputs have to be connected.

x_1 = Frame clock (high > 0.5) (impulse)

x_2 = Input signal vector (real or magnitude component; size N)

x_3 = Input signal vector (imaginary or phase component; size N)

y_1 = Frame clock (impulse)

y_2 = Output signal vector (real or magnitude component; size *N*)

y_3 = Output signal vector (imaginary or phase component; size *N*)

y_4 = Output signal vector (frequency or time output vector; size *N*)

FFT Mode

Forward FFT

Inverse FFT

Specifies either an FFT or IFFT operation.

FFT Window Type

Selects the desired window function to be used in computing the FFT. This parameter is not applicable to the IFFT operation.

FFT Representation

Mag / Phase

Indicates that the FFT data is represented by magnitude and phase components.

Real / Imaginary

Indicates that the FFT data is represented by real and imaginary components.

FFT Size

Specifies the size *N* of the FFT or IFFT operation. The valid range is from 8 to 8 Meg.

Output Freq. Units (FFT only)

Specifies the frequency units for the x-axis drive signal (y4). The choices include hertz, kilohertz, megahertz, and gigahertz.

Use Simulation Time Step

When checked, the time step (or frequency resolution) of the input data is assumed to be the current simulation settings.

Sampling Frequency

Specifies an alternate sampling frequency to associate with the vector data, when use of the current simulation time step value is not desired. This parameter only affects the scaling of the Frequency or Time output vector (y4), which is provided primarily for plotting purposes (x-axis drive signal).

PLL category

Blocks in the PLL category include `Charge Pump`, `Loop Filter (2nd Order PLL)`, `Loop Filter (3rd Order PLL)`, `Type-2 Phase Detector`, `Type-3 Phase Detector`, `Type-4 Phase Detector`, `VCO (Complex)`, and `VCO (Real)`.

Charge Pump

This block implements a first order active lag-lead loop filter for use in a second order digital PLL. The UP and DOWN inputs are assumed to be binary and represent the output error signals from a type-3 or type-4 digital phase/frequency detector. The output is used to drive a `VCO` block.

It is important to ensure that the Phase Detector Gain and VCO Gain parameters actually match the values used in the connected blocks. The approximate PLL noise loop bandwidth, as well as the computed transfer function coefficients (tau1, tau2), are displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency.

x_1 = UP error signal

x_2 = DOWN error signal

y = Output signal (VCO drive)

Charge Pump transfer function (Laplace format):

$$F(s) = \frac{s\tau_2 + 1}{s\tau_1}$$

Natural Frequency

Indicates the natural frequency (ω_n) of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase or frequency step.

Loop Bandwidth

Indicates the approximate noise loop bandwidth of the PLL in hertz. This represents the double-sided bandwidth (3 dB) over which the PLL is able to follow frequency variations in the input signal. The loop bandwidth parameter provides an alternative method to specifying the PLL natural frequency for describing the response time of the PLL.

Damping Factor

Specifies the damping factor of the PLL. The default value is 0.7071068, which yields a critically damped second order loop. As this value is decreased, the PLL response becomes underdamped. Increasing the value creates an overdamped response.

VCO Gain

Indicates the gain (K_o) of the connected VCO in (radians/sec)/volt.

Detector Gain

Indicates the gain (K_d) of the connected Phase Detector in volts/radian. A phase detector gain value of 1 (default) indicates that an input error signal of 0.1 V represents a 0.1 rad phase error. The detector gain for a Type-3 phase detector is approximately $1/\pi$ and approximately $1/2\pi$ for a Type-4 detector.

Specification Method

Loop Bandwidth

Indicates that the loop bandwidth is used to specify the time constant properties of the charge pump block.

Natural Frequency

Indicates that the natural frequency is used to specify the time constant properties of the charge pump block.

Update

This button updates the loop bandwidth (or natural frequency), Tau1, and Tau2 displays based on any changes made to the natural frequency (or loop bandwidth), damping factor, and gain parameters. The values are not permanently stored until you click on the OK button.

Loop Filter (2nd Order PLL)

This block implements a first order lag-lead loop filter for use in a second order PLL. A choice of active or passive loop design is provided. The input is assumed to be an error signal from a phase

detector. The output is typically used to drive a `VCO` block. Both input and output are expressed in volts.

The `Loop Filter (2nd Order PLL)` block is used in several of Embed/Comm's compound blocks, including the PLLs and Costas Loop. For more information about these compound blocks, see Appendix A, "Embed/Comm Library."

A `Loop Filter (2nd Order PLL)` block is able to track a phase or frequency step, but not Doppler rate. For this, you need to use the `Loop Filter (3rd Order PLL)` block, as described in the next section. Because `Loop Filter (2nd Order PLL)` blocks are more stable than `Loop Filter (3rd Order PLL),` you should use a second order PLL unless Doppler rate is a factor.

It is important to ensure that the Detector Gain and VCO Gain parameters actually match the values used in the connected blocks. The PLL noise loop bandwidth, as well as the computed transfer function coefficients (tau1, tau2), are displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency. Also, for good PLL operation when using the passive loop type, the loop gain (K_oK_d) should be much larger than the natural frequency (ω_n).

x = Input drive signal

y = Output signal (VCO drive)

Loop filter transfer function (Laplace format):

$$F(s) = \frac{s\tau_2 + 1}{s\tau_1} \quad \text{(active)} \qquad F(s) = \frac{s\tau_2 + 1}{s\tau_1 + 1} \quad \text{(passive)}$$

Natural Frequency

Indicates the natural frequency (ω_n) of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase or frequency step.

Loop Bandwidth

Indicates the approximate noise loop bandwidth of the PLL in hertz. This represents the double-sided bandwidth (3 dB) over which the PLL is able to follow frequency variations in the input signal. The loop bandwidth parameter provides an alternative method to specifying the PLL natural frequency for describing the response time of the PLL.

Damping Factor

Specifies the damping factor of the PLL. The default value is 0.7071068, which yields a critically damped second order loop. As this value is decreased, the PLL response becomes underdamped. Increasing the value creates an overdamped response.

VCO Gain

Indicates the gain K_o of the connected VCO in (radians/second)/volt.

Detector Gain

Indicates the gain K_d of the connected phase detector in volts/radian. A phase detector gain value of 1 (default) indicates that an input error signal of 0.1 V represents a 0.1 rad phase error. Commonly used phase detectors actually output $\sin(\theta_\varepsilon) \simeq \theta_\varepsilon$ for small error angles only. The gain value is based on assuming a small error angle.

Loop Filter Type

Active

Specifies an active loop filter implementation, also referred to as Type 3.

Passive

Specifies a passive loop filter implementation, also referred to as Type 2.

Specification Method

Loop Bandwidth

Indicates that the Loop Bandwidth is used to specify the time constant properties of the loop filter.

Natural Frequency

Indicates that the Natural Frequency is used to specify the time constant properties of the loop filter.

Update

This button updates the Noise Bandwidth, Tau1, and Tau2 values based on changes made to the Natural Frequency, Damping Factor, VCO Gain, and Detector Gain parameters. The values are not permanently stored until you click on the OK button.

Loop Filter (3rd Order PLL)

This block implements a Wiener Optimal second order loop filter for use in a third order PLL. A third order PLL is able to track a frequency ramp (Doppler rate). The input is assumed to be an error signal from a phase detector. The output is typically used to drive a `VCO` block. Both input and output are expressed in volts.

It is important to ensure that the Phase Detector Gain and VCO Gain parameters match the values used in the connected blocks. The PLL Noise Loop Bandwidth is displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency.

x = Input drive signal

y = Output signal (VCO drive)

Loop filter transfer function (Laplace format):

$$F(s) = \frac{\omega_3^3 + 2\omega_3^2 s + 2\omega_3 s^2}{K_o K_d s^2}$$

Natural Frequency

Indicates the natural frequency ω_3 of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase, frequency, or frequency ramp step.

VCO Gain

Indicates the gain K_o of the connected VCO in (radians/second)/volt.

Detector Gain

Indicates the gain K_d of the connected phase detector in volts/radian. A phase detector gain value of 1 (default) indicates that an input error signal of 0.1 V represents a 0.1 rad phase error. Commonly used phase detectors actually output $\sin(\theta_\varepsilon) \simeq \theta_\varepsilon$ for small error angles only. The gain value is based on assuming a small error angle.

Update

This button updates the Noise Bandwidth value based on any change made to the Natural Frequency parameter. The values are not permanently stored until you click on the OK button.

Type-2 Phase Detector

This block implements an XOR based digital phase detector. Block parameters include the input threshold voltage level, which is used to internally convert the input signals to digital waveforms [0, 1]. This block is usually followed by a `Loop Filter` block if used within a PLL.

Note: A type-2 PLL will lock at a 90 degree offset from the reference signal. The approximate phase detector gain for this type of detector is pi/2 V/rad (~1.5708).

x_1 = Reference input

x_2 = VCO input

y = Error signal [0, 1]

Threshold

Specifies the voltage level above which the input is considered high.

Type-3 Phase Detector

This block implements an edge triggered digital phase/frequency detector based on the JK flip-flop. Unlike a type-2 detector, the type-3 implementation is sensitive to frequency and is independent of the duty cycle ratio of the input signals. Block parameters include the initial output state, clock edge mode, and the low and high threshold voltage levels. This block is usually followed by a `Charge Pump` block.

Note: A type-3 PLL will lock at a 180 degree offset from the reference signal. The approximate phase detector gain for this type of detector is 1/pi V/rad (~0.31831).

x_1 = Reference input

x_2 = VCO input

y_1 = UP error signal [0, 1]

y_2 = DOWN error signal [0, 1]

Initial State

Specifies the initial state of the phase detector internal flip-flop.

High Threshold

Specifies the voltage level above which a rising input becomes high.

Low Threshold

Specifies the voltage level below which a falling input becomes low.

Edge Mode

Rising Edge

Specifies that the phase detector operates using rising clock edges.

Falling Edge

Specifies that the phase detector operates using falling clock edges.

Type-4 Phase Detector

This block implements an edge triggered digital phase/frequency detector. Compared to a type-3 detector, the type-4 implementation exhibits improved sensitivity to frequency offsets and has, at least theoretically, an infinite pull-in range. Block parameters include the low and high threshold voltage levels. This block is usually followed by a `Charge Pump` block.

Note: A type-4 PLL will lock with zero offset to the reference signal. The approximate phase detector gain for this type of detector is 1/2pi V/rad (~0.15915).

x_1 = Reference input

x_2 = VCO input

y_1 = UP error signal [0, 1]

y_2 = DOWN error signal [0, 1]

High Threshold

Specifies the voltage level above which a rising input becomes high.

Low Threshold

Specifies the voltage level below which a falling input becomes low.

RF category

Blocks in the RF category include `Amplifier`, `Antenna`, `Cable`, `Coupler`, `Double Balanced Mixer`, `Feko Far Field`, `RF Conversions`, `RF Gain`, `Switch`, `Splitter/Combiner`, and `Variable Attenuator`.

Amplifier

This block implements a nonlinear RF amplifier. Block parameters include the amplifier small signal gain, the 1 dB compression point, second, third, and fourth order intermodulation (IM) intercept points, and the amplifier noise figure. The block can also be modeled as a noiseless device. A 50 Ohm impedance is assumed.

The amplifier is modeled according to a fifth order Taylor polynomial, as shown below, until the saturation point is achieved. The saturation point corresponds to the peak of the polynomial output, and is actually slightly different for negative vs. positive input voltages due to the contributions from the even order terms. Once either the positive or negative saturation voltage is reached, the amplifier output remains constant at this level as long as the input voltage magnitude stays above the corresponding threshold.

The polynomial coefficients are computed based on the specified gain, IP2, IP3, IP4, and the 1 dB compression point. Depending on the specified parameters values, saturation is typically achieved a few dB beyond the 1 dB compression point. Since the contributions from the IP2 and IP4 terms introduce a dc offset in the output, the user is given the option of specifying this offset to be either positive or negative. Note: The 1dB, IP2, IP3 and IP4 points are all specified relative to the amplifier's output power level.

x = Input signal

y = Amplifier output signal

$$y = ax + bx^2 + cx^3 + dx^4 + ex^5 + noise \qquad (until\ saturation)$$

Gain

Indicates the small signal gain of the amplifier in decibels (dB).

1 dB Compression Point

Specifies the output power level in dBm where the amplifier output is 1 dB below the ideal gain.

IP2

Specifies the theoretical output power level in dBm where the power in the second order terms would equal the power in the fundamental (linear) term. The value of IP2 is typically 20 ~ 25 dB above the 1 dB compression point.

Output Offset

Positive

Specifies that the output dc offset due to the IP2 and IP4 terms is positive.

Negative

Specifies that the output dc offset due to the IP2 and IP4 terms is negative.

IP3

Specifies the theoretical output power level in dBm where the power in the third order terms would equal the power in the fundamental (linear) term. The value of IP3 is typically 10 ~ 15 dB above the 1 dB compression point.

IP4

Specifies the theoretical output power level in dBm at which the power in the intermods due to fourth order terms equals the power in the fundamental term.

Noise Figure

Specifies the equivalent input noise figure of the amplifier in dB when the Add Noise option is selected.

Add Noise

When this option is selected, white noise is added to the output according to the specified Noise Figure and Gain.

Antenna

This block models an RF antenna with gain and noise temperature specifications. Noise is added to the output based on the specified noise temperature and is not affected by the antenna gain setting. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

The antenna gain can be specified as either fixed, or arrival-angle dependent according to a specified antenna gain pattern. In the latter case, the x2 input is used to specify the arrival angle in degrees. The block uses linear interpolation (in dB) between the specified gain pattern points.

x_1 = Input signal

x_2 = Arrival angle (deg)

y = Antenna output signal

$$y = x \cdot 10^{(gain/20)} + noise \qquad \text{for fixed mode}$$

$$y = x \cdot 10^{(gain(x_2)/20)} + noise \qquad \text{for pattern mode}$$

Antenna Gain

Specifies the power gain of the antenna in dB.

Noise Temperature

Specifies the noise temperature seen by the antenna in degrees Kelvin when the Add Noise option is selected. The default value is 290 K.

Add Noise

Adds white noise to the output according to the specified Noise Temperature value.

Gain Mode

Pattern (dBc File)

Specifies an angle dependent antenna gain according to the specified dBc gain pattern and external off-boresight arrival angle input. The fixed antenna gain value is added to the pattern data, which is assumed to be in dBc units (dB relative to center).

Fixed

Specifies a fixed antenna gain. The angle input has no effect.

View Response

Invokes the Embed/Comm Response Viewer, which displays the file based antenna gain pattern. Linear interpolation (in dB) is used between the specified gain pattern points.

Select File

Opens the Select File dialog box for selecting the desired antenna gain pattern file.

Browse File

Opens the selected antenna gain pattern file using Notepad.

Antenna Gain (dBc) File Path

Specifies the DOS path to the desired antenna gain profile. The format of the antenna gain pattern file is described below. Angles are in degrees and gain values in dBc.

File header (anything)
number of entry pairs
angle #1, gain #1
angle #2 gain #2
...

Only a single pair of entries is allowed on a given line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 128 characters. The file angle data is restricted to angles in the [-180, +180] range.

Attenuator

This block implements a passive RF attenuator. Block parameters include the attenuator loss in decibels and the physical temperature of the device. The `Attenuator` block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x = Input signal

y = Attenuated output signal

$$y = x \cdot 10^{(-loss/20)} + noise$$

Loss

Indicates the loss of the device in decibels. This parameter is specified as a positive value.

Physical Temperature

When the Add Noise option is selected, specifies the physical temperature of the attenuator in degrees Kelvin. The default value is 290 K.

Add Noise

When this option is selected, white noise is added to the output according to the specified Loss and Physical Temperature parameters.

Cable

This block models a passive RF cable with a specified loss per unit distance. Noise is added to the output, when enabled, based on the specified length of the cable and its physical temperature. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x = Input signal

y = Cable output signal

$y = x \cdot 10^{(-loss/20)} + noise$ where *loss= cable length * loss/meter*

Loss per Meter

Specifies the power loss (positive value) of the cable per unit meter in *dB*.

Cable Length

Specifies the length of the cable in meters.

Phys. Temperature

Specifies the physical temperature of the cable in degrees Kelvin when the Add Noise option is selected. The default value is 290 K.

Add Noise

Adds white noise to the output according to the overall cable loss and Physical Temperature.

Coupler

This block models an RF coupler. Block parameters include the coupling sense, direct path loss, coupled loss, and noise figure of the device. The `Coupler` block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

Input Coupling	**Output Coupling**
x_1 = Primary input signal	x = Input signal
x_2 = Coupled signal	y_1 = Primary output
y = Output signal	y_2 = Coupled output

Input Coupling

$y = x_1 \cdot 10^{(-directLoss/20)} + x_2 \cdot 10^{(-coupledLoss/20)} + noise$

Output Coupling

$y_1 = x \cdot 10^{(-directLoss/20)} + noise$

$y_2 = x \cdot 10^{(-coupledLoss/20)}$

Direct Path Loss

Specifies the direct path loss of the coupler in decibels. This parameter is specified as a positive value.

Coupled Loss

Specifies the coupled path loss of the coupler in decibels. This parameter is specified as a positive value.

Noise Figure

When the Add Noise option is selected, specifies the equivalent input noise figure of the coupler in decibels. This value is typically set to the same value as the Direct Path Loss.

Coupling Mode

Input Coupling

Configures the device as an input coupler.

Output Coupling

Configures the device as an output coupler.

Add Noise

When this option is selected, white noise is added to the primary output according to the specified Noise Figure.

Double Balanced Mixer

This block implements a nonlinear double balanced mixer. Block parameters include the input 1 dB compression point, third order intermodulation (IM) intercept point (IP3), conversion loss, LO power and harmonic levels, isolation, dc offset, and the mixer noise figure. The block can also be modeled as a noiseless device. A 50 Ohm impedance is assumed.

The mixer is modeled as a nonlinear amplifier (RF input) followed by a multiplier. The amplifier coefficients (3rd and 5th order) are calculated from the 1 dB compression point and IP3 setting. The amplifier output is then multiplied by the LO signal and its harmonics, which include 3rd and 5th order terms, to generate the IF output. Once the amplifier stage reaches saturation (either negative or positive), its input to the multiplier is held constant until the RF input signal drops back down below the saturating drive level.

This block can also be used to implement an unbalanced or a single balanced mixer by adjusting the RF and LO isolation settings, so as to obtain the desired level of RF or LO feed through at the IF output.

Note: The 1dB and IP3 points are specified relative to the mixer's RF input power level.

x_1 = RF input

x_2 = LO input

y = IF output

$$y = k \cdot \left(x_1 + ax_1^3 + bx_1^5\right) \cdot \left(x_2 + cx_2^3 + dx_2^5\right) + noise \qquad (linear\ region)$$

$$y = k \cdot \left(\pm\, satAmpOutput\right) \cdot \left(x_2 + cx_2^3 + dx_2^5\right) + noise \qquad (saturated\ region)$$

Conversion Loss

Specified the conversion loss of the mixer in decibels. This is the ratio of output IF power to input RF power.

1 dB Compression Point

Specifies the input power level in dBm where the mixer conversion loss increases by 1 dB.

IP3

Specifies the theoretical input power level in dBm where the power in the third order intermods would equal the power in the fundamental. The value of IP3 is typically 10 ~ 15 dB above the 1 dB compression point.

LO Power

Specifies the mixer LO input power in dBm. This parameter is used internally to scale the output IF signal so as to achieve the specified conversion loss. Incorrect scaling will occur if the true LO input is not as specified.

RF Power

Specifies the mixer RF input power in dBm. This parameter is used internally to scale the output IF signal so as to achieve the specified conversion loss. Incorrect scaling will occur if the true RF input is not as specified.

LO 3rd Harmonic

Specifies the level for the IM products due to the LO's third harmonic. The level is specified as the number of dBs below the fundamental terms in dBc.

LO 5th Harmonic

Specifies the level for the IM products due to the LO's fifth harmonic. The level is specified as the number of dBs below the fundamental terms in dBc. This value must be at least 14 dB more than the LO 3rd harmonic setting.

DC Offset

Specifies the mixer dc offset in volts. This parameter is typically specified when the mixer is being used as a phase detector.

RF Isolation

Specifies the RF to IF port isolation in decibels. This parameter controls the extent to which the input RF signal appears at the IF output port.

LO Isolation

Specifies the LO to IF port isolation in decibels. This parameter controls the extent to which the input LO signal appears at the IF output port.

Noise Figure

When the Add Noise option is selected, specifies the equivalent input noise figure of the mixer in decibels.

Add Noise

When this option is selected, white noise is added to the output according to the specified Noise Figure.

Feko Far Field

This block models a far-field RF antenna pattern, whose Gain (or Directivity) is specified via an external Feko ".ffe" file. In addition, the block allows for specifying the antenna efficiency (in directivity mode) and optionally its noise temperature. Noise is added to the output based on the noise temperature parameter and is not affected by the antenna gain pattern value. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

The internal gain applied by the block is computed by reading the Total Gain (or Total Directivity) value specified by the Feko .ffe antenna pattern file corresponding to the Theta and Phi input angles. The x_2 input is used to specify the Theta angle in degrees, and the x_3 input to specify the

Phi angle in degrees. The block uses linear interpolation (in dB) between the specified gain pattern points in the file.

This block will only accept a Gain or Directivity .ffe file conforming to the Feko file standard. Currently, this block ignores the "Frequency" value in the .ffe file. If data for multiple frequencies is provided in the file, the block will only read in the far field pattern values corresponding to the first data set.

This block assumes that the Phi angle (azimuth) has a range of [0, 360] degrees. The entire Phi angle range need not be specified in the .ffe file, but the user is responsible for insuring that the x_3 input values do not fall outside of the provided range. Values outside of the [*minPhi*, *maxPhi*] range provided by the .ffe file, will be truncated to remain within this range.

This block assumes that the Theta angle (off-nadir) has a range of [-180, 180] degrees, where zero is nadir. The .ffe file data may include data either in the range of [-180, 180] or [0, 180] degrees. The entire Theta angle range need not be provided in the .ffe file, but the user is responsible for insuring that the x_2 input values do not fall outside of the provided range, except as explained below* for negative Theta values under very specific conditions. Values outside of the [*minTheta*, *maxTheta*] range provided by the .ffe file, are otherwise truncated to remain within this range.

*When the .ffe file includes a Theta range of [0, 180] and a Phi range of [0, 360], the block will determine the internal gain value for negative input Theta values (i.e. range of [-180, 0]) by adding (or subtracting) 180 deg from the Phi value and negating the sign of the Theta input value. For example, a [*Theta*, *Phi*] input vector of [-40, 70] deg will result in the look-up of the gain corresponding to the [+40, 250] deg coordinates.

x_1 = Input signal

x_2 = Theta angle (deg)

x_3 = Phi angle (deg)

y = Antenna output signal

$$y = x \cdot 10^{(gain(x_2, x_3)/20)} + noise$$

Antenna Efficiency

Specifies the efficiency of the antenna as a percentage. Valid range is [0, 100] %. This setting is only applicable for "Directivity" far-field antenna files.

Noise Temperature

Specifies the noise temperature seen by the antenna in degrees Kelvin when the Add Noise option is selected. The default value is 290 K.

Add Noise

Adds white noise to the output according to the specified Noise Temperature value.

Feko File Mode

Directivity (dBi File)

Specifies angle-dependent antenna directivity according to the specified .ffe pattern data and external arrival angle inputs. The specified antenna efficiency is applied to the pattern data, which is assumed to be in dBi units (dB relative to isotropic).

Gain (dBi File)

Specifies angle-dependent antenna gain according to the specified .ffe dBi gain pattern and external arrival angle inputs. The antenna efficiency setting has no effect in this mode.

Phi

Specifies the Phi value to be used when plotting the gain/directivity response from the .ffe file using the "View Response" button. Valid range is [0, 360] degrees.

View Response

Invokes the Embed/Comm Response Viewer and displays the specified Phi angle slice of the antenna gain (or directivity) pattern. Linear interpolation (in dB) is used between the specified gain pattern points.

Select File

Opens the Select File dialog box for selecting the desired Feko antenna far field pattern file.

Browse File

Opens the selected Feko antenna far field pattern file using Notepad.

Antenna Gain (dBi) File Path

Specifies the DOS path to the desired Feko antenna far field pattern file.

RF Conversions

This block implements a variety of conversions relevant to RF diagrams. The desired conversion is selected by choosing the appropriate radio button in the block's setup dialog box.

dBm to dBW

The block converts decibels relative to a milliWatt to decibels relative to a Watt.

x = Input value in dBm

y = Output value in dBW

$$y = x - 30$$

dBW to dBm

The block converts decibels relative to a Watt to decibels relative to a milliWatt.

x = Input value in dBW

y = Output value in dBm

$$y = x + 30$$

dBm to dBm/Hz

The block scales the input signal according to the current simulation frequency.

x = Input value in dBm

y = Output value in dBm/Hz

$$y = x - 10\log(F_s)$$

dBm/Hz to dBm

The block scales the input signal according to the current simulation frequency.

x = Input value in dBm/Hz

y = Output value in dBm

$$y = x + 10\log(F_s)$$

dBm/Hz to deg K

The block converts a noise density in dBm/Hz into its equivalent noise temperature.

x = Input value in dBm/Hz

y = Output value in deg K

$$y = \frac{10^{(x-30)/10}}{1.3806 \cdot 10^{-23}}$$

deg K to dBm/Hz

The block converts a noise temperature in degrees Kelvin into its equivalent noise density in dBm/Hz.

x = Input value in deg K

y = Output value in dBm/Hz

$$y = 30 + 10\log(x \cdot 1.3806 \cdot 10^{-23})$$

dBm 1 Ohm to 50 Ohm

The block scales the input signal .

x = Input value in dBm relative to a 1 Ohm load

y = Output value in dBm relative to a 50 Ohm load

$$y = x - 10\log(50)$$

dBm 50 Ohm to 1 Ohm

The block converts from radians/second to hertz.

x = Input value in dBm relative to a 50 Ohm load

y = Output value in dBm relative to a 1 Ohm load

$$y = x + 10\log(50)$$

RF Gain

This block models a perfect RF gain element (no intermodulation products). Noise is added to the output based on the specified noise figure or noise temperature of the device and its current gain setting. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x = Input signal

y = Output signal

$$y = x \cdot 10^{(gain/20)} + noise$$

Noise Figure

Specifies the noise figure of the gain element in dB when the Add Noise option is selected and the block is in the Noise Figure units mode.

Noise Temperature

Specifies the noise temperature of the gain element in degrees Kelvin when the Add Noise option is selected and the block is in the Degrees Kelvin units mode.

Device Gain

Specifies the gain of the device in dB.

Add Noise

Adds white noise to the output according to the specified Loss and Physical Temperature.

Splitter/Combiner

This block models an RF splitter or combiner. Block parameters include the splitter mode, number of connections, additional path loss, and noise figure of the device. The block can also be modeled as a noiseless device. A 50 Ohm impedance is assumed.

Splitter Mode	Combiner Mode
x = Input signal	x_1 = Input signal #1
y_1 = Output #1	**...**
...	x_n = Input signal #n
y_n = Output #n	y = Combined output

Split 0

$$y_1 = y_2 = ... = y_n = x\sqrt{10} \cdot^{(-loss/20)} + noise$$

Split 180

$$y_1 = x\sqrt{2} \cdot 10^{(-loss/20)} + noise$$

$$y_2 = -y_1$$

Combiner

$$y = \sum_{i=1}^{n} x_i \sqrt{n} \cdot 10^{(-loss/20)} + noise$$

Number of Outputs *or*

Number of Inputs

Specifies the number of outputs (in splitter mode) or inputs (in combiner mode) for the device. Valid range is 2 to 16. This value is forced to 2 in split 180 mode.

Additional Loss

Specifies an additional path loss in decibels above the loss associated with the number of inputs/outputs. This parameter is specified as a positive value.

Noise Figure

When the Add Noise option is selected, specifies the equivalent input noise figure of the splitter in decibels.

Splitter Mode

Split 0

Configures the device as a 0 degrees phase power splitter.

Split 180

Configures the device as a 180 degrees phase power splitter.

Combiner

Configures the device as a power combiner.

Add Noise

When this option is selected, white noise is added to the output(s) according to the specified Noise Figure.

Switch

This block models an RF switch. Block parameters include the switch sense, switch loss, isolation, and noise figure of the device. The block can also be modeled as a noiseless device and/or as having perfect isolation. A 50 Ohm impedance is assumed. The path selector input determines which input (or output) is active.

Input Switch	**Output Switch**
x_1 = Input path selector {1,2,..,*n*}	x_1 = Output path selector {1,2,..,*n*}
x_2 = Input signal #1	x_2 = Input signal
...	y_1 = Output signal #1
x_n = Input signal #*n*	**...**
y = Output signal	y_n = Output signal #*n*

Input Switch

$$y = \left(x_{sel} + \sum_{i \neq selt} x_i \cdot 10^{(-isolation/20)} \right) \cdot 10^{(-loss/20)} + noise$$

Output Switch

$$y_{sel} = x \cdot 10^{(-loss/20)} + noise$$

$$y_i = x \cdot 10^{(-isolation/20)} \qquad i \neq sel$$

Number of Connections

Specifies the number of inputs (input switch mode) or outputs (output switch mode) for the device. Valid range is 2 to 8.

Switch Loss

Specifies the loss through the switch in decibels. This parameter is specified as a positive value.

Isolation

Specifies the isolation between inputs or outputs for the switch in decibels. This parameter is specified as a positive value. This parameter is enabled only when Perfect Isolation is not selected.

Noise Figure

Specifies the equivalent input noise figure of the switch in decibels. This value is typically set to the same value as the Switch Loss.

Switch Mode

Input Switch

Indicates that there are N inputs and 1 output.

Output Switch

Indicates that there is 1 input and N outputs.

Perfect Isolation

Assumes perfect isolation between the switch inputs or outputs.

Add Noise

Adds white noise to the primary output according to the specified Noise Figure.

Variable Attenuator

This block implements a passive variable attenuator. The attenuation is controlled via external input, and can therefore be varied during the simulation. Noise is added to the output based on the specified physical temperature and the current loss value. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x_1 = Input signal

x_2 = Loss in dB (≥ 0)

y = Attenuated output signal

$$y = x \cdot 10^{(-loss/20)} + noise$$

Phys. Temperature

Specifies the physical temperature of the attenuator in degrees Kelvin when the Add Noise option is selected. The default value is 290 K.

Add Noise

Adds white noise to the output according to the specified Loss and Physical Temperature.

Signal Sinks category

Blocks in the Signal Sources category include `File Write`, `Wave Write`, and `Final Value`.

File Write

This block writes data to an external file in either ASCII or binary format. Writing of the data is controlled via an external clock. A clock value greater than 0.5 is considered high. The block can be configured to have from one to eight data inputs.

$x_1 .. x_n$ = Input value(s)

x_{n+1} = Input clock (impulse train)

File Type

ASCII

Specifies that the output file is in ASCII format. Data is written in rows of N values (one row per input clock pulse) and is comma delimited. Carriage returns are used at the end of each row. No additional file header is used besides the user specified header.

Binary

Specifies that the output file is in binary format. No additional file header is used besides the user specified header.

Reverse Byte Order

Specifies to reverse the byte order of the data written to the file. This option is only available for Binary files and allows the user to create files consistent with both little-endian and big-endian formats. The default format (not reversed) is little-endian.

Number of Inputs

Specifies the number of data inputs *N* for the block. Valid range is from 1 to 8.

Max Number of Data Entries

Specifies an upper limit to the output file size. Once the limit is reached, no additional data is written to the file. An entry is defined as an individual data element.

Data Type

Specifies the output file data type for Binary files. When the file type is ASCII, the block's input values are first cast to the specified Data Type and then written to the output file. Supported data types include the following (the number of bytes used in each case is shown in parentheses): Byte (1) (signed and unsigned), Short (2) (signed and unsigned), Long (4) (signed and unsigned), Float (4) and Double (8). The following data ranges apply:

Unsigned Byte	[0, 255]
Signed Byte	[-128, 127]
Unsigned Short	[0, 65535]
Signed Short	[-32768, 32767]
Unsigned Long	[0, 4294967295]
Signed Long	[-2147483648, 2147483647]
Float	6 decimal digits, max exponent= 38
Double	15 decimal digits, max exponent= 308

File Header

Allows the user to enter a file header prior to the data. For ASCII output files, the header is always specified in ASCII format. For Binary output files, the header is assumed to be in ASCII format unless preceded by a "0x" identifier, which indicates a binary header using hex format.

File Path

Specifies the DOS path to the desired output data file.

Select File

Opens the Select File dialog box for selecting the desired output data file.

Browse File

Opens the selected data file using Notepad.

Final Value

This block writes the final value of an input (value on last executed simulation step) to an external file in ASCII format. The block can be configured to have from one to eight data inputs.

Data is written in rows (one row per Altair Embed SE iteration) with *N* values per row, where *N* is the number of inputs. The data is comma delimited and carriage returns are used at the end of each row. An optional file header can also be specified.

This block can be used to record BER curve results or other data generated across multiple runs. To activate multiple iterations, select "Auto Restart" in the Simulation Properties dialog box.

$x_1 .. x_n$ = Input value(s)

Number of Inputs

Specifies the number of data inputs *N* for the block. Valid range is from 1 to 8.

Number of Runs

Specifies the number of Altair Embed SE iterations (runs) for which to record the final value(s) to the specified output file.

Data Type

Specifies the data type to be used by the output file. The block's input values are first cast to the specified Data Type and then written to the output file. Supported data types include: Byte (1) (signed and unsigned), Short (2) (signed and unsigned), Long (4) (signed and unsigned), Float (4) and Double (8). The number in parentheses indicates the number of bytes for each type. The following data ranges apply:

Unsigned Byte	[0, 255]
Signed Byte	[-128, 127]
Unsigned Short	[0, 65535]
Signed Short	[-32768, 32767]
Unsigned Long	[0, 4294967295]
Signed Long	[-2147483648, 2147483647]
Float	6 decimal digits, max exponent= 38
Double	15 decimal digits, max exponent= 308

File Path

Specifies the DOS path to the desired output data file.

Select File

Opens the Select File dialog box for selecting the desired output data file.

Browse File

Opens the selected output data file using Notepad.

Wave Write

This block writes data to an external file in wave (.wav) format. Writing of the data is controlled via an external clock. A clock value greater than 0.5 is considered high. The block can be configured to have from one to eight data inputs. When more than one input is specified, the data is multiplexed together into the file.

$x_1 \ldots x_n$ = Input value(s)

x_{n+1} = Input clock (impulse train)

Number of Inputs

Specifies the number of data inputs *N* for the block. Valid range is from 1 to 8.

Max Number of Data Entries

Specifies an upper limit to the output file size. Once the limit is reached, no additional data is written to the file. An entry is defined as an individual data element.

File Sample Rate

Specifies the sample rate to be stored in the wave file header.

Data Type

Specifies the data type for the wave file entries. The block's input values are first cast to the specified format and then written to the output file. Supported data types include the

following: 8 bits (unsigned), 16 bits (signed), 24 bits (signed), and 32 bits (signed). The following data ranges apply:

8 bits (unsigned) [0, 255]

16 bits (signed) [-32768, 32767]

24 bits (signed) [-8388608, 8388607]

32 bits (signed) [-2147483648, 2147483647]

File Path

Specifies the DOS path to the desired output wave file.

Select File

Opens the Select File dialog box for selecting the desired output wave file.

Browse File

Opens the selected wave file using Notepad.

Signal Sources category

Blocks in the Signal Sources category include `Complex Tone`, `File Data`, `Frequency Sweep`, `Impulse`, `Impulse Train`, `Noise`, `PN Sequence`, `Poisson Arrivals`, `Random Distribution`, `Rectangular Pulses`, `Random Seed`, `Random Symbols`, `Sinusoid`, `Spectral Mask`, `Vector Constant`, `Walsh Sequence`, `Wave Data`, and `Waveform Generator`.

Complex Tone

This block generates a rotating complex phasor according to the selected block parameters. The internal generator phase (in radians) is also available as an output.

y_1 = Complex output signal [Re, Im]

y_2 = Generator phase (rad) [Optional]

$$y_1(t) = Ae^{j(2\pi f_c t + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Frequency

Indicates the frequency f_c, in hertz, of the complex tone.

Amplitude / Power

Indicates the amplitude A, in volts, of the complex tone, or depending on the Units setting, the complex power of the tone in milli-decibels (50 Ohms).

Initial Phase

Indicates the starting phase θ of the complex tone. This value is specified in degrees.

Units

Volts

Indicates that the signal amplitude is specified in volts.

dBm

Indicates that the signal complex power is specified in milli-decibels (50 Ohms load).

File Data

This block reads data sequentially from an external ASCII or binary file. The data can be read at a fixed rate or controlled via an external clock. The block can be configured to have one to five outputs. This block does not expect the data to be in any particular format, as for example, in columns. Upon reaching the end of file, the data sequence can be optionally repeated. The output values are held constant between updates. A clock value greater than 0.5 is considered high.

x = Optional external clock (impulse train)

y_1 .. y_n = Output value(s)

y_{n+1} = Output clock (impulse train)

File Type

ASCII Text

Specifies that the input file is in ASCII format. The number of header lines must also be specified.

Binary

Specifies that the input file is in binary format. The size of the file's header and the data type need to be specified.

Timing

External

Indicates external timing. An external clock must be provided at the x_1 input.

Internal

Indicates internal clock timing. The symbol rate and delay need to be specified.

Header Size / Number of Header Lines

Specifies the size of the file's header in bytes for Binary data files or lines for ASCII files. The maximum header size is 64 KB.

Binary Data Type

Specifies the data type for binary files. Supported data types include: Byte (1) (signed and unsigned), Short (2) (signed and unsigned), Long (4) (signed and unsigned), Float (4) and Double (8). The number in parentheses indicates the number of bytes for each type.

Repeat at EOF

Upon reaching the end of file, repeats the file data sequence. Note that if the number of data points in the file is not an integer multiple of Num of Outputs, the output sequence will be shifted upon restarting.

Reverse Byte Order

Specifies to reverse the byte order of the data read from the file. This option is only available for Binary files and allows the user to read files consistent with both little-endian and big-endian formats. The default format (not reversed) is little-endian.

Num of Outputs

Specifies the number of data outputs for the block. Valid range is 1 to 5.

File Path

Specifies the DOS path to the desired data file. The expected file format is described below.

After a single line header, data can be arranged as desired. Valid data delimiters are commas, blank space, tabs and carriage returns. The maximum allowed line length is 100 characters. The following is an example:

Header line (up to 100 characters)
1,2.1, 4 5 8.2
3.1 6 10
7.5
9.9, 5

Data Rate

Specifies the data output rate in hertz. This setting is only available when internal timing mode is selected.

Start Time

Specifies the starting time, in seconds, for the output sequence. This setting is only available when internal timing mode is selected.

Empty Value

Specifies the output value to be used when no data is available (delayed start case or EOF condition).

Select File

Opens the Select File dialog box for selecting the desired data file.

Browse File

Opens the selected data file using Notepad.

Frequency Sweep

This block generates a frequency sweep according to the selected block parameters. The sweep start and stop frequencies, its duration, and the initial phase are specified. After the sweep is completed, a new sweep is started. This block outputs a real signal.

y = Sweep signal output

Start Frequency

Indicates the starting frequency of the sweep signal. This value is specified in hertz.

Stop Frequency

Indicates the stop frequency of the sweep signal. This value is specified in hertz.

Sweep Duration

Indicates the duration of the repeating sweep signal. This value is specified in seconds.

Amplitude

Indicates the amplitude of the sweep signal. This value is specified in volts.

Initial Phase

Indicates the starting phase of the sweep signal (referenced to the sine function). This value is specified in degrees.

Impulse

This block generates a single impulse of the specified magnitude at the specified time. If the specified impulse time is between simulation steps, the impulse will occur at the next simulation step. No pulse will occur if the time specified is prior to the simulation start time.

y = Output signal

Impulse Time

Specifies the occurrence time, in seconds, of the impulse.

Amplitude

Specifies the amplitude of the impulse in volts.

Impulse Train

This block generates an impulse train given the specified parameters, which include the sequence start time, frequency and amplitude.

y = Output pulse train

Pulse Frequency

Specifies the frequency, in hertz, of the impulse train. The inverse of this value is the pulse repetition period.

Amplitude

Specifies, in volts, the peak amplitude of the impulse.

Start Time

Used to specify, in seconds, the start time of the first output pulse.

Noise

This block generates Gaussian random noise according to the specified noise density or noise temperature parameter. The simulation sampling frequency is automatically taken into account when the noise is generated. The block supports both 1 Ohm and 50 Ohms load impedances.

y = Output noise signal

Load

1 Ohm

Specifies a load resistance of 1 Ohm.

50 Ohms

Specifies a load resistance of 50 Ohms.

Noise Units

dBm/Hz

Noise density is specified in milli-decibels/hertz.

Watts/Hz

Noise density is specified in watts/hertz.

Degrees Kelvin

Noise density is specified by setting the equivalent noise temperature in degrees Kelvin.

Noise Density

Noise Temperature

Specifies the source's noise density (or noise temperature) in watts/hertz, milli-decibels/hertz, or degrees Kelvin, depending on the Noise Units setting.

PN Sequence

This block generates a maximal length pseudo noise (PN) sequence, also known as a pseudo random binary sequence (PRBS). The generator's shift register size, coefficients, initial state, and bit rate can be specified. The block can also accept an external clock. A clock level greater than 0.5 is considered high.

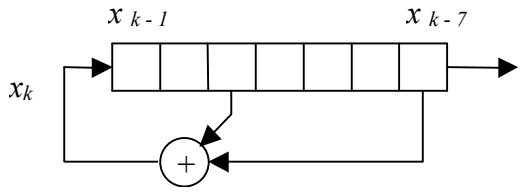
The output sequence can be optionally augmented with an extra zero. The default output sequence is generated by using the minimal weight primitive polynomial for each order.

Two different forms are used to describe PN sequences in the literature, and there are also two separate forms for describing the generator polynomial. It's important to note that both forms will produce the same output sequence, but for a given initial state of the shift register they will differ in their output (different locations within the same pattern).

The first form, implemented by Embed/Comm, defines the PN sequence as a recursion formula involving previous sequence outputs. This form uses the modulo 2 sum of the contents of one or more delay stages to create a single feedback value that is inserted in the leftmost location of the shift register. An example is shown below.

Form I Example:

$n = 7$ PN length = 127 $x_k = x_{k-7} \oplus x_{k-3}$ where $\oplus$ represents modulo 2 addition



Generator coefficient = 211 octal (10001001)

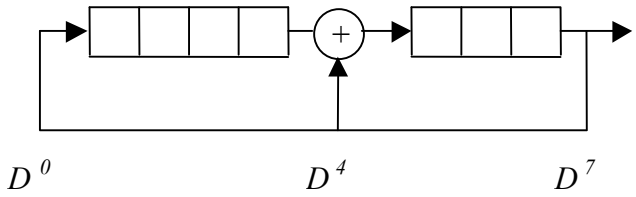
The generator coefficient value is obtained by specifying a "1" for each term where a feedback connection occurs, and writing the result as an octal number. In this form the LSB of the generator coefficient is always set to "1" by convention, and corresponds to the *x* (*k - 0)* term. Since in this case the *x* (*k - 3)* term is used, the fourth bit is set to a 1, as is the eighth bit corresponding to the *x* (*k - 7)* term. The shift register contents represent the "state" of the generator, with the LSB corresponding to the rightmost shift register position.

In the second form, the same PN sequence shown above is defined by a generator polynomial, and a different shift register implementation is used. In this case the right-most bit of the shift register is feedback to multiple locations within the shift register and modulo 2 additions are performed with their contents.

Form II Example:

$n = 7$ PN length = 127

$p(D) = D^7 + D^4 + 1$ or $p(x) = x^7 + x^4 + 1$



Generator polynomial = 221 octal (10010001)

Note that the generator coefficient value is different from before and is obtained by specifying a "1" for each x^k term that appears in the polynomial (with the x^0 term corresponding to the LSB of the generator coefficient value), and writing the result as an octal number. In this case the x^0, x^4, and x^7 terms are used, and so the first, fifth and eighth bits are set in the generator coefficient. The shift register contents represent the "state" of the generator, with the LSB corresponding to the rightmost shift register position.

A *Form II* expression or order *n* can be converted to a *Form I* expression by executing the following steps :

1) Discard the x^n term

2) Convert each remaining x^k term (including the "1", which is really x^0) to a corresponding x (*n - k)* entry in the *Form I* expression.

x = Optional external clock (impulse train)

y_1 = Pseudo-random sequence output (binary or bilevel)

y_2 = Output clock (impulse train)

Shift Register Size

Specifies the order *n* of the PN sequence and determines its length. The sequence length is $2^n - 1$ (2^n for augmented sequences), where *n* is the shift register size. Valid range is from 2 to 28.

Sequence Offset

Determines the starting position of the PN sequence. The offset value is used to advance the shift register from its known starting point state. Valid range is from 0 to $2^n - 1$, or 32,767.

Initial State

Specifies the initial state of the internal shift register in octal format. The LSB represents the initial output, while the remaining bits represent the next *n*-1 outputs. For example, 56 octal is 101110.

Generator Coeff.

Specifies the generator code for the PN sequence in octal format. The default code for a given order *n* can be obtained by selecting the Use Default Generator Coefficient option. Note that this value is always odd.

Zero Augmented Sequence

Instructs the PN generator to augment the output sequence with an extra 0. The extra 0 is inserted after *n*-1 consecutive 0s are encountered in the output sequence.

Timing

External

Indicates external timing. An external clock must be provided at the x_1 input.

Internal

Indicates internal clock timing. The Bit Rate and Start Time need to be specified.

Output Mode

Bilevel

Indicates that the signal amplitudes associated with output the sequence are {-1, 1}.

Binary

Indicates that the signal amplitudes associated with output the sequence are {0, 1}.

Use Default Generator Coefficient

Loads the default generator coefficient for each order *n*. The default code represents the minimal weight primitive polynomial.

Bit Rate

Specifies the PN sequence bit rate in bits per second. This parameter is only available when internal timing mode is selected.

Start Time

Specifies a start time, in seconds, for the PN sequence. This parameter is only available when internal timing mode is selected.

Poisson Arrivals

This block generates a Poisson arrivals process, where the pulse inter-arrival times are exponentially distributed according to the specified mean arrival rate. The output consists of a series of unit amplitude impulses beginning at the specified Start Time.

y = Output signal (Poisson distributed impulse train)

Mean Arrival Rate

Specifies the mean arrival rate for the Poisson process in Hertz.

Start Time

Specifies the simulation time of the first pulse.

Random Distribution

This block generates Random Variable (RV) values according to a variety of common distribution functions, including uniform, Gaussian, exponential and Rayleigh. The block's output is of type double regardless of the selected distribution type.

y = Random variable output per selected distribution

Distribution Type

Specifies the Random Variable distribution type. Choices include:

Long (32)

Produces uniformly distributed integer (long) random numbers over the range [0, 0xFFFFFFFF].

Long (31)

Produces uniformly distributed integer (long) random numbers over the range [0, 0x7FFFFFFF].

Int (16)

Produces uniformly distributed integer (short) random numbers over the range [0, 0xFFFF].

Int (15)

Produces uniformly distributed integer (short) random numbers over the range [0, 0x7FFF].

Uniform [0,1]

Produces uniformly distributed floating-point random numbers over the range [0, 1].

Uniform [0,1)

Produces uniformly distributed floating point random numbers over the range [0, 1) (exclusive of 1).

Uniform (0,1)

Produces uniformly distributed floating-point random numbers over the range (0, 1) (exclusive of both 0 and 1).

Gaussian (μ,σ²)

Produces Gaussian distributed floating-point random numbers with the specified mean and variance.

Exponential (μ)

Produces exponentially distributed floating-point random numbers with the specified mean.

Rayleigh (μ,σ²)

Produces Rayleigh distributed floating-point random numbers with the specified mean and variance.

Rayleigh (E(x²))

Produces Rayleigh distributed floating-point random numbers with the specified mean square value.

Mean

Specifies the mean for the distribution function, if applicable. Applies to Gaussian, Exponential and Rayleigh (μ,o²) distributions.

Variance

Specifies the variance for the distribution function, if applicable. Applies to Gaussian and Rayleigh (μ,o²) distributions.

Mean Square Value

Specifies the mean square value for the distribution when the Rayleigh (E(x²)) selection is made.

Random Seed (Obsolete)

This obsolete block was used in earlier versions of the software to set the random number generator seed for all Comm blocks. This function is now controlled via the Comm menu; please see “Random Numbers” in Chapter 2.

Seed

Specifies the random number seed to be used by the all Comm blocks.

Reset Seed on Auto Restart

Forces the Embed/Comm random number generator to reset itself when a new run begins in Auto Restart mode. This will cause the outputs of all Comm DLL random sources to repeat exactly for all iterations.

Random Symbols

This block generates uniformly distributed random symbols between 0 and *N*-1, where *N* is the number of total symbols. The value of *N*, the symbol rate, and an initial delay can be specified. This block can also accept an external clock. A clock value greater than 0.5 is considered high.

x = Optional external clock (impulse train)

y_1 = Random symbol sequence (0, ..., N-1)

y_2 = Output symbol clock (impulse train)

Number of Symbols

Specifies the number of available output symbols *N*. The output values range between 0 and *N*-1.

Timing

External

Indicates external timing. An external clock must be provided at the x_1 input.

Internal

Indicates internal clock timing. The symbol rate and delay need to be specified.

Symbol Rate

Specifies the data sequence symbol rate in symbols/second. This parameter is only available when internal timing mode is selected.

Start Time

Specifies the starting time, in seconds, for the output sequence. This parameter is only available when internal timing mode is selected.

Rectangular Pulses

This block generates a rectangular pulse train given the specified parameters. The pulse width can be entered as a pulse time duration or by specifying a duty cycle. The `Rectangular Pulses` block can be used to generate a square wave signal by specifying a 50% duty cycle.

The block's clock output produces a positive unity pulse during the waveform's rising edge and a negative unity pulse during the waveform's falling edge.

y_1 = Output signal

y_2 = Clock (impulse train) [-1, +1]

Pulse Frequency

Specifies the frequency of the pulse train. The inverse of this value is the pulse repetition period. This value is specified in hertz.

High Level

Specifies the output level associated with the pulse (ON). This value is specified in volts.

Low Level

Specifies the output level between pulses (OFF). This value is specified in volts.

Pulse Duration

Duty Cycle

Depending on the Pulse Mode setting, specifies either the pulse duration (high level) in seconds, or the pulse duty cycle in percent. When entered as a duration, this value should be less than the pulse repetition period.

Start Time

Used to specify, in seconds, the starting time of the first output pulse.

Pulse Mode

Duration

Specifies the pulse ON time.

Duty Cycle

Specifies the pulse ON time. The duty cycle is the ratio of the ON time to the OFF time.

Sinusoid

This block generates a sine or cosine wave specified in hertz according to the selected block parameters. The signal phase is also available in radians.

y_1 = Sine or cosine output (real)

y_2 = Sine generator phase (rad) (optional)

$$y_1(t) = A\sin(2\pi f_c t + \phi) \quad \text{or} \quad A\cos(2\pi f_c t + \phi) \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Frequency

Indicates the sinusoidal frequency f_c in hertz.

Amplitude

Power

Indicates the amplitude A, in volts, of the sinusoidal waveform, or depending on the Units setting, the power of the signal in milli-decibels (50 Ohms).

Initial Phase

Indicates the starting phase θ of the sinusoidal output. This value is specified in degrees.

Units

Volts

Indicates that the signal amplitude is specified in volts.

dBm

Indicates that the signal power is specified in milli-decibels (50 Ohms load).

Output Mode

Cosine

Indicates that the output waveform is a cosine.

Sine

Indicates that the output waveform is a sine.

Spectral Mask

This block outputs a user-defined spectral mask. Its purpose is to let you overlay an FCC mask onto a power spectrum being generated by the simulation environment. The mask is viewed using a `plot` block configured in XY mode with an external trigger. Both the mask information (amplitude) and frequency axis data are output as data vectors of size *N*, where *N* is a user-defined power of two. This block is meant to be used in conjunction with the `Spectrum Analyzer` block.

Once triggered, the `Spectral Mask` block reads in an external frequency vector (of size *N*+1), or an internally generated data set, and uses the specified look-up table to generate a piecewise linear graphical representation of the spectral mask. The look-up table data may be specified over the range of [0, *fs*/2] or [-*fs*/2, *fs*/2], and should include all corners of the mask. The `Spectral Mask` block automatically interpolates (and extrapolates if necessary) the table values to generate the output graph. Data points in the mask file need not be provided in uniform increments, but are required to be in ascending order. Data points should be specified in decibels.

x_1 = Input trigger (high > 0.5) (pulse)

x_2 = Input frequency vector [optional] (size *N*+1)

y_1 = Spectral mask vector for plot block (size *N*+1)

y_2 = Frequency vector for plot block (if needed)

Mask Data Range

[0, fs/2]

Used when the input file only includes data points over the range of [0, +fs/2]. The `Spectral Mask` block automatically mirror images the table's data points for the negative portion of the frequency axis.

[-fs/2, +fs/2]

Used when the input file includes data points over the range of [-fs/2, +fs/2].

Frequency Source

External

Specifies that the frequency axis data points are to be provided via the external vector input.

Internal

Specifies that the frequency axis data points are to be computed internally based on the simulation time step value and the value of *N*.

Spectrum Size

Specifies the reference FFT size (*N*) used by the corresponding `Spectrum Analyzer` block. The valid range is 8 to 16,384. The actual vector output will be *N*+1 to account for both endpoints at -*fs*/2 and *fs*/2.

Select File

Opens the Select File dialog box for selecting the spectral mask definition file.

Browse File

Opens the selected mask file using Notepad.

Mask File Path

Specifies the DOS path to the desired spectral mask definition file. Data points are to be provided in increasing order, and are to be arranged in two columns. The second line in the file is used to specify the number of total entries in the file. Thereafter, the first column

specifies each data point's frequency in hertz, and the second column the corresponding mask level in *dB*. The format of the Spectral Mask definition file is further described below:

File header (anything)
number of entries
frequency point #1, mask value #1
frequency point #2, mask value #2
...

Allowed data delimiters are commas, blank space, and tabs. The maximum allowed line length is 100 characters.

Vector Constant

This block produces a user specified column vector that outputs the same constant value for all its elements.

y = Output signal vector [size N]

Vector Size

Specifies the size N of the output column vector (range is 1 to 1,048,576).

Constant Value

Specifies the output value for all the vector elements.

VCO (Complex or Real)

This block implements a VCO. Two versions of this block are provided: one producing a complex output and the other producing a real output. When the input drive signal is 0, the `VCO` block outputs a tone at the specified center frequency. With a non-zero input, the output frequency deviates from the center frequency depending on the magnitude of the drive signal and the specified VCO gain.

x = Input drive signal

y_1 = Output signal ([Re, Im] for complex)

y_2 = Accumulated phase (rad) (optional)

$$y_1(t) = Ae^{j\theta(t)} \qquad y_2(t) = \theta(t)$$

$$\theta(t) + \int_0^t \left(2\pi f_c + x_1(\tau)K_o\right)d\tau + \phi$$

where:

f_C = translation frequency A = carrier amplitude

K_O = VCO gain ϕ = initial phase (radians)

Center Frequency

Indicates the VCO center frequency in hertz. The value may be set to 0 or even a negative frequency.

Amplitude

Indicates the amplitude of the output tone (single-sided peak amplitude). This value is specified in volts.

Initial Phase

Indicates the starting phase of the output complex tone. This value is specified in degrees.

VCO Gain

Indicates the gain of the VCO in Hz/volt. The value may be positive or negative.

Integration Method

Euler

Specifies the Euler integration method (forward difference).

Trapezoidal

Specifies the trapezoidal integration method.

Backward Difference

Specifies the backwards difference integration method.

Walsh Sequence

This block generates a repeating Walsh binary sequence, typically used in CDMA systems. Walsh sequences represent a family of orthogonal sequences, and are constructed from Hadamard matrices. The user can specify the sequence length (*N*) and row (*K*) of the sequence (*K ,N*) to be output. The output bit rate, and an initial delay can also be specified. This block can accept an external clock. A clock value greater than 0.5 is considered high.

The desired Walsh sequence is selected by specifying the "row #" of the associated Hadamard matrix. The row value can be specified as either a fixed value or via an external input.

x_1 = Optional external clock (impulse train)

x_2 = Optional external row selector

y_1 = Walsh sequence output

y_2 = Output bit clock (impulse train)

y_3 = Frame clock (indicates beginning of new row) (impulse)

$$H_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad H_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & \overline{H_N} \end{bmatrix}$$

Matrix Output Row

Specifies which row (*K*) of the (*N* x *N*) Hadamard matrix H_N is to be used as the Walsh output sequence. Valid range is 0 to *N*-1, where *N* is the sequence length. This entry is disabled when in external row selection mode.

Sequence Offset

Specifies a starting offset for the Walsh sequence. Valid range is 0 to *N*-1.

Sequence Length

Specifies the Walsh sequence length *N*. This value is a power of two, and has a valid range of 2 to 256.

Row Selection

Fixed (Internal)

The Walsh sequence row is fixed and corresponds to the Matrix Output Row parameter setting.

External Input

The Walsh sequence row is selected via the external control input. Valid input range is 0 to *N*-1.

Output Mode

Bilevel

The signal amplitudes associated with output the sequence are {-1, 1}.

Binary

The signal amplitudes associated with output the sequence are {0, 1}.

Timing

Internal

Indicates internal clock timing. The bit rate and start time need to be specified.

External

Indicates external timing. An external clock must be provided at the $x1$ input.

Bit Rate

Specifies the Walsh sequence bit rate in bits per second. This parameter is only available when in Internal Timing mode.

Start Time

Specifies a start time, in seconds, for the Walsh sequence. This parameter is only available when in Internal Timing mode.

Wave Data

This block reads data sequentially from an external wave (.wav) file. The data can be read at a fixed rate or controlled via an external clock. The block can be configured to have one to five outputs. Upon reaching the end of file, the data sequence can be optionally repeated. The output values are held constant between updates. A clock value greater than 0.5 is considered high.

x = Optional external clock (impulse train)

y_1 .. y_n = Output value(s)

y_{n+1} = Output clock (impulse train)

Timing

External

Indicates external timing. An external clock must be provided at the x_1 input.

Internal

Indicates internal clock timing. The symbol rate and delay need to be specified.

Size of Header

Displays the size of the file's header in bytes. This value is a Read-Only parameter.

Data Size

Displays the data format for the wave file in bits. This value is a Read-Only parameter.

Repeat at EOF

Upon reaching the end of file, repeats the file data sequence. Note that if the number of data points in the file is not an integer multiple of Num of Outputs, the output sequence will be shifted upon restarting.

Num of Outputs

Specifies the number of data outputs for the block. Valid range is 1 to 8.

File Path

Specifies the DOS path to the desired wave file.

Data Rate

Specifies the data output rate in hertz. This setting is only available when internal timing mode is selected.

Start Time

Specifies the starting time, in seconds, for the output sequence. This setting is only available when internal timing mode is selected.

Empty Value

Specifies the output value to be used when no data is available (delayed start case or EOF condition).

Select File

Opens the Select File dialog box for selecting the desired data file.

Browse File

Opens the selected data file using Notepad.

File Properties

Displays the wave file's properties obtained by reading its header information.

Waveform Generator

This block implements a generic waveform generator capable of producing the following waveform types: square wave, triangle wave or sawtooth wave.

y_1 = Output waveform

y_2 = Output clock (impulse train)

Waveform Type

Square Wave

Outputs a square wave at the specified frequency.

Triangle Wave

Outputs a triangle wave at the specified frequency.

Sawtooth Wave

Outputs a sawtooth wave at the specified frequency.

Waveform Frequency

Specifies the waveform output rate in hertz.

Peak-to-Peak Amplitude

Specifies the peak-to-peak amplitude in *volts* of the waveform. A 1V p-p value with an offset of zero will produce a waveform in the range of [-0.5, +0.5] V.

Offset

Specifies an offset value in volts for the output waveform.

Start Time

Specifies the starting time, in seconds, for the output waveform.

Turbo Codes category

Blocks in the Turbo Codes category include `LTE Turbo Decoder`, `LTE Turbo Encoder`, `TC Interleaver Generator`, `Turbo Code Decoder`, `Turbo Code Encoder`, `UMTS Turbo Decode`, and `UMTS Turbo Encoder.`

Portions of the Turbo Code library Copyright 2000 by Matt Valenti.

LTE Turbo Decoder

This block implements the LTE specification of a Turbo Decoder. The LTE TC encoding specification uses a Parallel Concatenated Convolutional Code (PCCC) turbo code with a code rate of 1/3 and includes termination bits for both encoders. The decoder input block size equals $3k + 12$, where k is the information block size. The interleaver is generated internally according to the LTE specification and its size is k (the input block size for the encoder). The generator polynomials for the LTE case are 13 octal (feedback) and 15 octal (feedforward).

The TC block size parameter should match the block size used at the encoder. The LTE Decoder block also allows the user to specify the decoding type to be used, set the maximum number of decoder iterations, and specify an LLR halt value for the decoding process. This block accepts a coded input vector of size n and outputs a decoded data vector of size k.

An input frame clock is used to control the decoding of each input encoded block. When the frame clock goes high, the block will read the input vector, decode the data, and output the decoded data block as a vector.

For each decode operation, the Number of Iterations output indicates the number of iterations taken by the decoder before either meeting the specified Log Likelihood Ratio (LLR) threshold, or reaching the maximum number of allowed iterations. Similarly, the LLR output corresponds to the minimum LLR value computed by the decoder.

An Unbuffer block can be used at the decoder's output to produce a serial bit stream from the block's vector format.

x_1 = Input frame clock

x_2 = Input coded data vector (size n)

x_3 = Input reference data vector (size k) [Only used in Perfect Halting mode]

y_1 = Output frame clock

y_2 = Decoded output vector (size k)

y_3 = Number of executed decoding iterations

y_4 = Minimum Log Likelihood Ratio (LLR) value

y_5 = Number of detected bit errors [Only used in Perfect Halting mode]

Output Block Size

Specifies the size k of the output data bits vector. Valid range is from 40 to 6144, inclusive. Note: not all values in this range are valid. A warning is issued for invalid block sizes.

Decoder Type

Linear Approx. log-MAP

Specifies the use of a linear approximation to the log-MAP method for computing the internal decoder metrics. This algorithm is slower than both max-log-MAP and constant-log-MAP, but performs better than either. Using the linear-log-MAP algorithm only introduces a loss in coding gain of about 0.002 dB relative to the log-MAP algorithm.

Max-log-MAP

Specifies the use the Max-log-MAP method for computing the internal decoder metrics. This is the fastest of all decoding algorithms, but introduces a 0.4 dB loss in coding gain relative to the log-MAP algorithm.

Constant-log-MAP

Specifies the use the Constant-log-MAP method for computing the internal decoder metrics. This algorithm is the next fastest algorithm, second only to max-log-MAP. However, it only introduces a loss in coding gain of only about 0.03 dB relative to log-MAP,

Log-MAP (piecewise linear)

Specifies the use of a piecewise-linear approximation for computing the internal log-MAP decoder metrics. This algorithm is faster than the log-MAP algorithm that uses C calls, yet gives approximately the same performance. It is slower than max-log-MAP, constant-log-MAP, and linear-log-MAP.

Log-MAP (C calls)

Specifies the use of standard C function calls for computing the internal log-MAP decoder metrics. This is by far the slowest of all the decoding algorithms, although it provides the best performance.

Halting Mode

LLR Halting

Specifies that the decoding process is to halt early (i.e. before executing all "Max Iterations" decoding iterations) when the minimum Log Likelyhood Ratio (LLR) of all decoded bits exceeds the "Halt LLR" threshold value.

Perfect Halting

Specifies that the decoding process is to halt as soon as zero decoded bit errors are detected. This option requires that the transmitted information bits also be supplied to the decoder for reference. This mode is intended for academic and code performance determination purposes.

Halt LLR Threshold

Specifies the Log Likelihood Ratio (LLR) threshold value to be used for stopping the decoding process when in "LLR Halting" mode. The decoding iterations will cease once the minimum LLR of all the decoded bits meets or exceeds the specified threshold. A small value of Halt LLR results in a faster decoder at the cost of reduced BER performance. A typical value of this parameter is 10.

Max Iterations

Specifies the maximum allowed number of decoder iterations. Typical values range from 4 to 15, with larger frames requiring more iterations.

LTE Turbo Encoder

This block implements the LTE specification of a Turbo Encoder. The LTE TC encoder uses a Parallel Concatenated Convolutional Code (PCCC) turbo code with a code rate of 1/3 and includes termination bits for both encoders. The output block size equals $3k + 12$, where k is the input

block size. The interleaver is generated internally according to the LTE specification and its size matches the input block size. The generator polynomials for the LTE case are 13 octal (feedback) and 15 octal (feedforward).

Block parameters include the TC block size. This block accepts an input data vector of size *k* and outputs a coded data vector of size *n*. An input frame clock is used to control the encoding of each input data vector. When the frame clock goes high, the block will read the input vector, encode the data, and produce a coded vector output. A Buffer block can be used to pack a serial bit stream into the block's input vector format.

x_1 = Input frame clock

x_2 = Input data vector (size *k*)

y_1 = Output frame clock

y_2 = Encoded output vector (size *n*)

Input Block Size

Specifies the size *k* of the input data block. Valid range is from 40 to 6144 per the LTE specification. Note: not all values in this range are valid. A warning is issued for invalid block sizes.

TC Interleaver Generator

This block generates Turbo Code interleaver files, which can then be used by the generic Turbo Code Encoder and Turbo Code Decoder blocks. This block generates interleaver files using either the UMTS specification or the S-Random algorithm, which is basically a trial and error approach.

Block parameters include the TC interleaver size and type. For the S-Random mode, the maximum number of attempts to be used in trying to produce the interleaver file is also specified, as is the separation value "*S*". In general, the larger the value of *S*, the longer it will take to generate the interleaver, but the better the performance of the resulting design

This block is intended to be run by itself within a simulation. Its two outputs provide a success flag and a "number of trials" counter. For the UMTS case, only one trial is required, since the design process follows a prescribed nondeterministic algorithn. The simulation will stop automatically when a success event is achieved. Note: for the S-Random mode, the number of simulation steps must be equal or greater than the maximum number of trials.

y_1 = Success flag (1= success)

y_2 = Number of Trials counter

Interleaver Size

Specifies the size k of the interleaver to be generated. Valid range is from 1 to 100,000 for the S-Random interleaver and between 40 and 5114 for the UMTS interleaver.

Interleaver Type

UMTS

Specifies generation of the interleaver file based on the UMTS specification.

S-Random

Specifies generation of the interleaver file by using the S-Random algorithm. The user must also supply a value of "*S*" and max number of trials.

Minimum Spacing

Specifies the minimum separation *S* for the values in the interleaver file. This parameter only applies to the S-Random case. Higher values of *S* result in a turbo code with a lower BER floor, but will require more time for the design algorithm to run. A typical value for length *K* interleavers is about sqrt(*K*)/2.

Max Trials

Specifies the maximum number of attempts to take when trying to create the interleaver using the S-Random algorithm.

Save As

Opens the Save As… dialog box for specifying the file name to be used for storing the Interleaver.

Browse File

Opens the selected Interleaver file using Notepad.

Interleaver File Path

Specifies the path and filename for the Interleaver file. The format of the generated file is shown below, as is compatible with the TC Encoder and TC Decoder blocks.

Interleaver File Format

Header line (indicates interleaver type and settings)
#of entries (size of TC interleaver)
value #1
value #2
etc…

Turbo Code Decoder

This block implements a Turbo Code (TC) Decoder suitable for decoding the general class of Parallel Concatenated Convolutional Codes (PCCCs). PCCCs are implemented by concatenating two Recursive Systematic Convolutional (RSC) codes in parallel and using a data interleaver to shuffle the order of the data bits at the input of one of the encoders. In general, PCCCs could contain more than two RSC encoders, but this implementation of the decoder can only handle two. Furthermore, the current release of the toolbox requires that each of the component RSC encoders be rate ½ and identical. The overall rate of the turbo code is ⅓ unless puncturing is used to delete every-other parity bit, in which case the rate is ½.

Block parameters should match those used by the corresponding TC encoder, and are described below. In addition to those parameters shared by the encoder, the decoder block also allows the user to specify the decoding type to be used, set the maximum number of decoder iterations, and specify an LLR halt value for the decoding process. This block accepts a coded input vector of size n and outputs a decoded data vector of size *k*.

An input frame clock is used to control the decoding of each input encoded block. When the frame clock goes high, the block will read the input vector, decode the data, and output the decoded data block as a vector.

For each decode operation, the Number of Iterations output indicates the number of iterations taken by the decoder before either meeting the specified Log Likelihood Ratio (LLR) threshold, or reaching the maximum number of allowed iterations. Similarly, the LLR output corresponds to the minimum LLR value computed by the decoder.

An Unbuffer block can be used at the decoder's output to produce a serial bit stream from the block's vector format.

x_1 = Input frame clock

x_2 = Input coded data vector (size n)

x_3 = Input reference data vector (size k) [Only used in Perfect Halting mode]

y_1 = Output frame clock

y_2 = Decoded output vector (size k)

y_3 = Number of executed decoding iterations

y_4 = Minimum Log Likelihood Ratio (LLR) value

y_5 = Number of detected bit errors [Only used in Perfect Halting mode]

Output Block Size

Specifies the size k of the input data bits vector. Valid range is from 1 to 100,000.

Trellis Termination

Not Terminated

Indicates that trellis termination bits are NOT used. The interleaver block size equals the input block size (k), and the effective code rate is not affected (i.e. code rate is exactly 1/2 or 1/3).

Terminate Upper

Indicates that the first encoder is to be terminated by the use of extra tail bits. In this case the size of the interleaver increases from k to k+L-1, where L is the constraint length of the code. The larger interleaver size is required because the L-1 tail bits are interleaved along with the k data bits and fed into the input of the second encoder. Note that although the first encoder is terminated, the second might not be. This mode also has a slight impact on the effective code rate of the block, which decreases slightly from a true rate 1/2 or rate 1/3 to rate k/(2(k+L-1)) or rate k/(3(k+L-1)) respectively.

Terminate Both

Indicates that both encoders are to be terminated using extra tail bits per the UMTS specification. In particular, each of the two encoders is decoded with its tail, and both the tail bits and the corresponding parity bits become part of the code word. In this case the interleaver block size equals the input block size, but the effective code rate is decreased in exactly the same manner as with the "Terminate Upper" case.

Generator #1

Specifies the feedback generator code polynomial for both RSC encoders. This value is specified in *octal* format.

Generator #2

Specifies the feedforward generator code polynomial for both RSC encoders. This value is specified in *octal* format.

Constraint Length

Specifies the constraint length (L) of the associated RSC encoders. The memory size (m) of the encoder is simply L-1. The choice of L is currently limited to 3, 4 or 5.

Decoder Type

Linear Approx. log-MAP

Specifies the use of a linear approximation to the log-MAP method for computing the internal decoder metrics. This algorithm is slower than both max-log-MAP and constant-log-MAP, but performs better than either. Using the linear-log-MAP algorithm only introduces a loss in coding gain of about 0.002 dB relative to the log-MAP algorithm.

Max-log-MAP

Specifies the use the Max-log-MAP method for computing the internal decoder metrics. This is the fastest of all decoding algorithms, but introduces a 0.4 dB loss in coding gain relative to the log-MAP algorithm.

Constant-log-MAP

Specifies the use the Constant-log-MAP method for computing the internal decoder metrics. This algorithm is the next fastest algorithm, second only to max-log-MAP. However, it only introduces a loss in coding gain of only about 0.03 dB relative to log-MAP,

Log-MAP (piecewise linear)

Specifies the use of a piecewise-linear approximation for computing the internal log-MAP decoder metrics. This algorithm is faster than the log-MAP algorithm that uses C calls, yet gives approximately the same performance. It is slower than max-log-MAP, constant-log-MAP, and linear-log-MAP.

Log-MAP (C calls)

Specifies the use of standard C function calls for computing the internal log-MAP decoder metrics. This is by far the slowest of all the decoding algorithms, although it provides the best performance.

Code Rate

Rate 1/2

Specifies a rate 1/2 Turbo Code, where the systematic (i.e. data) bit is always sent, and the encoder #1 and encoder #2 output parity bits are toggled. For example: Sys, Enc#1, Sys, Enc#2, Sys, Enc#1, Sys, Enc#2, …

Rate 1/3

Specifies a rate 1/3 Turbo Code, where the systematic bit is always sent, and so are both the encoder #1 and encoder #2 output parity bits. For example: Sys, Enc#1, Enc#2, Sys, Enc#1, Enc#2, …

High rate turbo codes (r >1/2) can be implemented by using this option and passing the received encoded bits though a depuncture operation using the Depuncture block.

Halting Mode

LLR Halting

Specifies that the decoding process is to halt early (i.e. before executing all “Max Iterations” decoding iterations) when the minimum Log Likelyhood Ratio (LLR) of all decoded bits exceeds the “Halt LLR” threshold value.

Perfect Halting

Specifies that the decoding process is to halt as soon as zero decoded bit errors are detected. This option requires that the transmitted information bits also be supplied to the decoder for reference. This mode is intended for academic and code performance determination purposes.

Halt LLR Threshold

Specifies the Log Likelihood Ratio (LLR) threshold value to be used for stopping the decoding process when in “LLR Halting” mode. The decoding iterations will cease once the minimum LLR of all the decoded bits meets or exceeds the specified threshold. A small value of Halt LLR results in a faster decoder at the cost of reduced BER performance. A typical value of this parameter is 10.

Max Iterations

Specifies the maximum allowed number of decoder iterations. Typical values range from 4 to 30, with larger frames requiring more iterations.

Select File

Opens the Select File dialog box for selecting the desired Interleaver definition file.

Browse File

Opens the selected Interleaver definition file using Notepad.

Interleaver File Path

Specifies the path and filename for the Interleaver external specification file.

Interleaver File Format

Header line (can be anything)
#of entries (size of TC interleaver)
value #1
value #2, value#3
etc…

Multiple entries are allowed per line. Supported delimiters are: comma, blank spaces, tabs and carriage returns.

Turbo Code Encoder

This block implements a Turbo Code (TC) Decoder suitable for decoding the general class of Parallel Concatenated Convolutional Codes (PCCCs). PCCCs are implemented by concatenating two Recursive Systematic Convolutional (RSC) codes in parallel and using a data interleaver to shuffle the order of the data bits at the input of one of the encoders. In general, PCCCs could contain more than two RSC encoders, but this implementation of the decoder can only handle two. Furthermore, the current release of the toolbox requires that each of the component RSC encoders be rate ½ and identical. The overall rate of the turbo code is ⅓ unless puncturing is used to delete every-other parity bit, in which case the rate is ½.

Block parameters include the TC block size, code polynomials, code rate, and trellis termination options. This block accepts an input data vector of size k and outputs a coded data vector of size *n*. An input frame clock is used to control the encoding of each input data vector. When the frame clock goes high, the block will read the input vector, encode the data, and produce a coded vector output. A Buffer block can be used to pack a serial bit stream into the block's input vector format.

x_1 = Input frame clock

x_2 = Input data vector (size k)

y_1 = Output frame clock

y_2 = Encoded output vector (size n)

Input Block Size

Specifies the size k of the input data bits vector. Valid range is from 1 to 100,000.

Trellis Termination

Not Terminated

Indicates that trellis termination bits are NOT used. The interleaver block size equals the input block size (k), and the effective code rate is not affected (i.e. code rate is exactly 1/2 or 1/3).

Terminate Upper

Indicates that the first encoder is to be terminated by the use of extra tail bits. In this case the size of the interleaver increases from k to $k+L-1$, where L is the constraint length of the code. The larger interleaver size is required because the $L-1$ tail bits are interleaved along with the k data bits and fed into the input of the second encoder. Note that although the first encoder

is terminated, the second might not be. This mode also has a slight impact on the effective code rate of the block, which decreases slightly from a true rate 1/2 or rate 1/3 to rate $k/(2(k+L-1))$ or rate $k/(3(k+L-1))$ respectively.

Terminate Both

Indicates that both encoders are to be terminated using extra tail bits per the UMTS specification. In particular, each of the two encoders is decoded with its tail, and both the tail bits and the corresponding parity bits become part of the code word. In this case the interleaver block size equals the input block size, but the effective code rate is decreased in exactly the same manner as with the "Terminate Upper" case.

Generator #1

Specifies the feedback generator code polynomial for both RSC encoders. This value is specified in *octal* format.

Generator #2

Specifies the feedforward generator code polynomial for both RSC encoders. This value is specified in *octal* format.

Constraint Length

Specifies the constraint length (L) of the associated RSC encoders. The memory size (m) of the encoder is simply L-1. The choice of L is currently limited to 3, 4 or 5.

Code Rate

Rate 1/2

Specifies a rate 1/2 Turbo Code, where the systematic (i.e. data) bit is always sent, and the encoder #1 and encoder #2 output parity bits are toggled. For example: Sys, Enc#1, Sys, Enc#2, Sys, Enc#1, Sys, Enc#2, …

Rate 1/3

Specifies a rate 1/3 Turbo Code, where the systematic bit is always sent, and so are both the encoder #1 and encoder #2 output parity bits. For example: Sys, Enc#1, Enc#2, Sys, Enc#1, Enc#2, …

This option should be selected when further pucturing of the encoded bits is performed by using the Puncture block.

Select File

Opens the Select File dialog box for selecting the desired Interleaver definition file.

Browse File

Opens the selected Interleaver definition file using Notepad.

Interleaver File Path

Specifies the path and filename for the Interleaver external specification file.

Interleaver File Format

Header line (can be anything)
#of entries *(size of TC interleaver)*
value #1
value #2, value#3
etc…

Multiple entries are allowed per line. Supported delimiters are: comma, blank spaces, tabs and carriage returns.

UMTS Turbo Decoder

This block implements the UMTS specification of a Turbo Decoder. The UMTS TC encoding specification uses a Parallel Concatenated Convolutional Code (PCCC) turbo code with a code rate of 1/3 and includes termination bits for both encoders. The decoder input block size equals $3k + 12$, where k is the information block size. The interleaver is generated internally according to the UMTS specification and its size is k (the input block size for the encoder). The generator polynomials for the UMTS case are 13 octal (feedback) and 15 octal (feedforward).

The TC block size parameter should match the block size used at the encoder. The UMTS Decoder block also allows the user to specify the decoding type to be used, set the maximum number of decoder iterations, and specify an LLR halt value for the decoding process. This block accepts a coded input vector of size n and outputs a decoded data vector of size k.

An input frame clock is used to control the decoding of each input encoded block. When the frame clock goes high, the block will read the input vector, decode the data, and output the decoded data block as a vector.

For each decode operation, the Number of Iterations output indicates the number of iterations taken by the decoder before either meeting the specified Log Likelihood Ratio (LLR) threshold, or reaching the maximum number of allowed iterations. Similarly, the LLR output corresponds to the minimum LLR value computed by the decoder.

An Unbuffer block can be used at the decoder's output to produce a serial bit stream from the block's vector format.

x_1 = Input frame clock

x_2 = Input coded data vector (size n)

x_3 = Input reference data vector (size k) [Only used in Perfect Halting mode]

y_1 = Output frame clock

y_2 = Decoded output vector (size k)

y_3 = Number of executed decoding iterations

y_4 = Minimum Log Likelihood Ratio (LLR) value

y_5 = Number of detected bit errors [Only used in Perfect Halting mode]

Output Block Size

Specifies the size k of the output data bits vector. Valid range is from 40 to 5114, inclusive.

Decoder Type

Linear Approx. log-MAP

Specifies the use of a linear approximation to the log-MAP method for computing the internal decoder metrics. This algorithm is slower than both max-log-MAP and constant-log-MAP, but performs better than either. Using the linear-log-MAP algorithm only introduces a loss in coding gain of about 0.002 dB relative to the log-MAP algorithm.

Max-log-MAP

Specifies the use the Max-log-MAP method for computing the internal decoder metrics. This is the fastest of all decoding algorithms, but introduces a 0.4 dB loss in coding gain relative to the log-MAP algorithm.

Constant-log-MAP

Specifies the use the Constant-log-MAP method for computing the internal decoder metrics. This algorithm is the next fastest algorithm, second only to max-log-MAP. However, it only introduces a loss in coding gain of only about 0.03 dB relative to log-MAP,

Log-MAP (piecewise linear)

Specifies the use of a piecewise-linear approximation for computing the internal log-MAP decoder metrics. This algorithm is faster than the log-MAP algorithm that uses C calls, yet gives approximately the same performance. It is slower than max-log-MAP, constant-log-MAP, and linear-log-MAP.

Log-MAP (C calls)

Specifies the use of standard C function calls for computing the internal log-MAP decoder metrics. This is by far the slowest of all the decoding algorithms, although it provides the best performance.

Halting Mode

LLR Halting

Specifies that the decoding process is to halt early (i.e. before executing all "Max Iterations" decoding iterations) when the minimum Log Likelyhood Ratio (LLR) of all decoded bits exceeds the "Halt LLR" threshold value.

Perfect Halting

Specifies that the decoding process is to halt as soon as zero decoded bit errors are detected. This option requires that the transmitted information bits also be supplied to the decoder for reference. This mode is intended for academic and code performance determination purposes.

Halt LLR Threshold

Specifies the Log Likelihood Ratio (LLR) threshold value to be used for stopping the decoding process when in "LLR Halting" mode. The decoding iterations will cease once the minimum LLR of all the decoded bits meets or exceeds the specified threshold. A small value of Halt LLR results in a faster decoder at the cost of reduced BER performance. A typical value of this parameter is 10.

Max Iterations

Specifies the maximum allowed number of decoder iterations. Typical values range from 4 to 15, with larger frames requiring more iterations.

UMTS Turbo Encoder

This block implements the UMTS specification of a Turbo Encoder. The UMTS TC encoder uses a Parallel Concatenated Convolutional Code (PCCC) turbo code with a code rate of 1/3 and includes termination bits for both encoders. The output block size equals $3k + 12$, where k is the input block size. The interleaver is generated internally according to the UMTS specification and its size matches the input block size. The generator polynomials for the UMTS case are 13 octal (feedback) and 15 octal (feedforward).

Block parameters include the TC block size. This block accepts an input data vector of size k and outputs a coded data vector of size n. An input frame clock is used to control the encoding of each input data vector. When the frame clock goes high, the block will read the input vector, encode the data, and produce a coded vector output. A Buffer block can be used to pack a serial bit stream into the block's input vector format.

x_1 = Input frame clock

x_2 = Input data vector (size k)

y_1 = Output frame clock

y_2 = Encoded output vector (size n)

Input Block Size

Specifies the size *k* of the input data block. Valid range is from 40 to 5114 per the UMTS specification.

Vector Operators category

Blocks in the Vector Operators category include `Matrix to Vector`, `SubVector`, `Vector Bits to Symbol`, `Vector Demux`, `Vector Merge`, `Vector to Matrix`, `Vector Mux`, and `Vector Symbol to Bits`.

Matrix to Vector

This block slices a *M* x *N* matrix into *N* or less independent column vectors. This block produces an updated output each time a "high" input clock is present.

x_1 = Input clock (high ≥ 1) (pulse)

x_2 = Input Matrix [size *M* x *N*]

y_1 = Output clock (pulse)

$y_{2\ldots N+1}$ = Output column vectors [size *M*]

Number of Output Vectors

Specifies the number *N* of desired output column vectors, and need not be the same as the number of columns in the input matrix (may be smaller). The maximum value for *N* is 16 or the number of columns in the matrix, whichever is smaller. This value must be specified numerically (global variable not allowed).

SubVector

This block extracts a *column* vector of size *N* from a larger or equal sized *column* vector of size *L* This block produces an updated output each time a "high" input clock is present. When desiring to extract a subvector from a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

x_1 = Input clock (high > 0.5) (pulse)

x_2 = Input column vector [size *L*]

y = Output column vector [size *N*]

Output Vector Size

Specifies the length *N* of the desired output column subvector. This value must be less or equal to the size of the input column vector.

Offset

Specifies the starting location within the input vector of the output subvector. The valid range is 0 to *N*-1.

Vector Bits to Symbol

This block combines adjacent elements of a column vector (containing binary bits) into a smaller column vector composed of symbols. This block produces an updated output each time a "high" input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

The input vector length must be evenly divisible by the number of bits per symbol *K*. This block operates by combining successive groups of *K* bits into a corresponding sequence of output vector symbols.

Example: Input Vector = [1, 0, 1 , 1, 1, 0 , 0, 0, 1]^T

Output Vector = [5, 6, 1]^T 3 bits/symbol, MSB mode

x_1 = Input clock (high $\geq$ 1) (pulse)

x_2 = Input column vector [size *M*]

y_1 = Output clock (pulse)

y_2 = Output column vector [size *M* / *K*]

Number of Bits per Symbol

Specifies the number *K* of desired bits per symbol.

Bit Order

LSB First

Indicates that the first element in each group of binary bits is to be used as the least significant bit of the output symbol.

MSB First

Indicates that the first element in each group of binary bits is to be used as the most significant bit of the output symbol.

Vector Demux

This block demultiplexes elements from a column vector into 2 or more smaller column vectors. This block produces an updated output each time a "high" input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

The output vectors can be forced to all be the same size through the use of padding when the number of output vectors *N* does not divide evenly into the input size vector *M*. Otherwise, one or more of the output vectors may be larger than others by one element. This block operates by stuffing elements from the input vector into each output vector in round robin fashion.

Example: Vector A = [a1, a2, a3, a4, a5]^T

w/o padding - Output #1 = [a1, a3, a5]^T Output #2 = [a2, a4]^T

w/ padding - Output #1 = [a1, a3, a5]^T Output #2 = [a2, a4, pad]^T

x_1 = Input clock (high $\geq$ 1) (pulse)

x_2 = Input column vector [size *M*]

y_1 = Output clock (pulse)

$y_{2...N+1}$ = Output column vectors [size *L* or *L*+1]

Number of Output Vectors

Specifies the number *N* of desired output vectors. The maximum value for *N* is 16. This value must be specified numerically (global variable not allowed).

Padding Mode

Off

Output vectors are not forced to all be the same size.

On

Output vectors are forced to be the same size by using padding when necessary.

Pad Value

Specifies the Pad Value to be used when Padding Mode in ON.

Vector Merge

This block appends a column vector of size *M* to another column vector of size *L*. The output vector will be of size $N = L + M$. The `Vector Merge` block automatically reads the size of the input vectors and computes the corresponding output vector size. Among other uses, the `Vector Merge` block can be used to zero pad a non power-or-two sized vector so that it can be used by the `Vector FFT` block.

This block produces an updated output each time a "high" input clock is present. This block has no internal parameters.

x_1 = Input clock (high > 0.5) (pulse)

x_2 = Input column vector #1 [size *L*]

x_3 = Input column vector #2 [size *M*]

y = Output signal vector [size *N*]

Vector Mux

This block multiplexes elements from two or more column vectors into a new vector. This block produces an updated output each time a "high" input clock is present. When desiring to operate on *row* vectors, first convert all row vectors to column vectors by using a matrix transpose block.

All input vectors must be of the same size *L*. The output size will then be a vector of length *N* x *L*, where *N* is the number of input vectors. The block operates by taking elements from each input vector in round robin fashion.

Example: Vector A = [a1, a2, a3]^T Vector B = [b1, b2, b3]^T

Output = [a1, b1, a2, b2, a3, b3]^T

x_1 = Input clock (high ≥ 1) (pulse)

$x_{2\ldots N+1}$ = Input column vectors [size *L*]

y_1 = Output clock (pulse)

y_2 = Output column vector [size M=N x L]

Number of Input Vectors

Specifies the number *N* of input column vectors. The maximum value for *N* is 16. This value must be specified numerically (global variable not allowed).

Vector Symbol to Bits

This block decomposes a column vector of symbol values into a larger column vector composed of binary bits. This block produces an updated output each time a "high" input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

This block operates by breaking up each symbol into its underlying binary bits and then outputting the *K* least significant bits. The output order of the *K* output bits is controlled by the MSB / LSB selection.

Example: Input Vector = [5, 2, 11]^T 3 bits/symbol, MSB mode

Output Vector = $[\ 1, 0, 1\ , 0, 1, 0\ , 0, 1, 1\ ]^T$

Note: Only the lower 3 bits of "11" (1011) were output

x_1 = Input clock (high ≥ 1) (pulse)

x_2 = Input column vector [size M]

y_1 = Output clock (pulse)

y_2 = Output column vector [size M / K]

Number of Bits per Symbol

Specifies the number of output bits per symbol K.

Bit Order

LSB First

Indicates that the first element in each output group of binary bits corresponds to the least significant bit of the input symbol value.

LSB Last

Indicates that the last element in each output group of binary bits corresponds to the least significant bit of the input symbol value.

Vector to Matrix

This block assembles two or more column vectors into a matrix. This block produces an updated output each time a "high" input clock is present. When desiring to operate on *row* vectors, first convert all row vectors to column vectors by using a matrix transpose block.

All input vectors must be of the same size M. The output size will then be a matrix of size M x N, where N is the number of input vectors.

x_1 = Input clock (high ≥ 1) (pulse)

$x_{2\ldots N+1}$ = Input column vectors [size M]

y_1 = Output clock (pulse)

y_2 = Output matrix [size M x N]

Number of Input Vectors

Specifies the number N of input column vectors. The maximum value for N is 16. This value must be specified numerically (global variable not allowed).

Wireless category

The blocks included in the Wireless category are organized in six sub-groups according to the specific Standard they support:

- Bluetooth
- 802.11 (1 Mbps and 2 Mbps modes)
- 802.11a /g (OFDM modulation at 6 ~ 54 Mbps)
- 802.11b (CCK modulation at 5.5 and 11 Mbps)
- Generic Wireless
- Ultrawideband (UWB)

Bluetooth

GFSK, Hop Generator, Scrambler, Short Hamming Encoder, Short Hamming Decoder

802.11

Barker Sequence, CRC-16, Descrambler, GFSK-2, GFSK-4, Hop Generator, Scrambler

802.11a/802.11g

Convolutional Encoder, Depuncture, Interleaver, Puncture, Scrambler, Viterbi Decoder, OFDM Demodulator, OFDM Modulator, OFDM Pilot Extract, OFDM Pilot Map, OFDM Vector Demodulator, OFDM Vector Modulator

802.11b

High Rate Hop Generator, CCK Baseband Modulator, CCK Demodulator

Generic Wireless

Frequency Hop

Ultrawideband

Gated Integrate & Dump, UWB PPM Modulator, UWB Pulse

BLUETOOTH BLOCKS

Bluetooth Hop Generator

This block generates a Bluetooth frequency-hopping pattern and a Master Clock output. Different hop patterns are generated for the Master and Follower, and a toggle output is provided to indicate which time slot (master or follower) is active. The Bluetooth Master Clock has a cycle length of $2^{\wedge 28}$ (28 bit shift register), which corresponds to $2^{\wedge 27}$ time slots as hops occur only on every other clock pulse. Master and Follower take turns transmitting, with the Master using even hop slot numbers and the Follower odd ones. The default time slot duration is 625 us.

This block supports both the 79 and 23 hop specifications. The hop sequence is derived from the current Bluetooth Master Clock value and the Device Address, as described in Section 11 of the Bluetooth Specification. The default Bluetooth hop rate is 1.6 kHz, which corresponds to a Master clock pulse rate of 3.2 kHz (hops occur on every other clock pulse).

This block support both internal and external timing. When in external mode, the user must provide both the Master Clock counter value and pulse train. This block outputs a hop pattern consisting of channel numbers in the range of [0, 78] or [0, 22] depending on the Hop Mode selection. This block should be followed by a Frequency Hop block to implement the actual signal hopping.

x_1 = Optional Master Clock counter value

x_2 = Optional Master Clock pulse train

y_1 = Master Hop Channel # [0, 78] or [0, 22]

y_2 = Follower Hop Channel # [0, 78] or [0, 22]

y_3 = Master/Follower Slot Flag (0= Master; 1= Follower)

y_4 = Master Clock counter value

y_5 = Master Clock pulse train

Device Address

Specifies the device address in hex. This value is used to select the hop sequence generated by the block.

Timing Mode

Internal

Indicates internal clock timing. The hop rate, start time, and Master Clock initial value must be specified.

External

Indicates external timing. An external clock and Master Clock counter value must be provided to the block.

Hop Mode

79 Hops

The block generates a hop sequence using 79 possible channels. This mode is used in the US and most of Europe.

23 Hops

The block generates a hop sequence using 23 possible channels. This mode is use in Japan, Spain and France.

Init Master Clock

Specifies the initial counter setting for Bluetooth Master Clock. This parameter is only available when in Internal Timing mode.

Hop Rate

Specifies the hop rate for the block in hops per second. Note that the output clock rate will be twice this rate. This parameter is only available when in Internal Timing mode. The default hop rate is 1.6 kHz.

Start Time

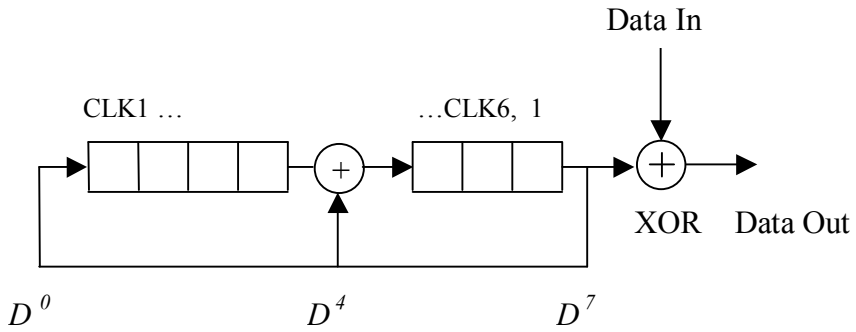
Specifies a starting time for the hop sequence and clock output. This parameter is only available when in Internal Timing mode.

Bluetooth Scrambler

This block provides a data scrambling and descrambling function as specified by the Bluetooth standard. The input data is XOR'ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized for each transmission (when a pulse is presented at the frame clock input) by using bits 1-6 of the current Bluetooth Master Clock and extending them with an MSB value of 1.

The generator polynomial for the feedback shift register is:

$$p(D) = D^7 + D^4 + 1$$



The user must provide an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired. The use must also supply the Bluetooth Master Clock value (essentially the output of a binary counter), which can be obtained from a Bluetooth Hop Generator block.

The Bluetooth Scrambler block is used for both data scrambling and unscrambling.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Frame clock pulse

x_4 = Bluetooth Master Clock value

y_1 = Output data

y_2 = Output data clock pulses

This block does not have any internal parameters.

GFSK Modulator

This block implements a Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK modulation, the digital information is transmitted by shifting the carrier frequency between two states. A "1" is represented by a positive frequency shift and a "0" is represented by a negative frequency shift.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the Bluetooth specification for the 1 Mbps mode, which specifies a BT value of 0.5 and a frequency deviation range of +/- 140 ~ 175 kHz.

x = Input data signal (binary [0, 1])

y = Complex output signal [Re, Im]

Shortened Hamming Decoder

This block implements a decoder for the shortened (15, 10) Hamming code as defined in the Bluetooth standard. The encoder matrix for this code is shown below. This code achieves a code rate of 2/3 and has the ability of correcting one bit error in a code word. The code is systematic and the parity-check matrix is shown below.

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

The decoding process includes computing the syndrome of the input vector and the parity-check matrix, locating any errors and correcting the error when possible. Since the code is systematic, the information bits reside in the first 10 bits of the codeword and the parity bits occupy the last 5 bits.

This block does not include any parameters. It accepts as input a coded vector of size 15 and outputs a decoded vector of size 10 elements.

x_1 = Input frame clock pulse

x_2 = Input coded vector (size 15)

y_1 = Output frame clock pulse

y_2 = Decoded output vector (size 10)

This block does not have any internal parameters.

Shortened Hamming Encoder

This block implements an encoder for the shortened (15, 10) Hamming code as defined in the Bluetooth standard. The encoder matrix for this code is shown below. This code achieves a code rate of 2/3 and has the ability of correcting one bit error in a code word. The code is systematic and is described by the parity-check matrix *H* shown below.

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

The encoding process involves computing the parity bits for the given input word and appending these to the input to create the coded output word. Since the code is systematic, the information bits are preserved in the output codeword and reside in its first 10 bits. The last 5 bits represent the parity bits.

This block does not include any parameters. It accepts as input a data vector of size 10 and outputs an encoded vector of size 15 elements.

x_1 = Input frame clock pulse

x_2 = Input data vector (size 10)

y_1 = Output frame clock pulse

y_2 = Encoded output vector (size 15)

This block does not have any internal parameters.

802.11 BLOCKS

Barker Sequence

This block generates a repeating Barker sequence. Barker codes are a family of pseudo-random sequences with quasi-ideal cross-correlation properties. Barker codes are often used in CDMA systems and are also part of the 802.11 2.4 GHz DSSS specification. The Barker code used in the 802.11 specification uses N= 11.

The user can select the length of the Barker code, as well as the symbol rate and an initial delay. The following illustrates the output pattern for each available length:

N = 3 1 1 0

N = 4 1 1 0 1

N = 5 1 1 1 0 1

N = 7 1 1 1 0 0 1 0

N = 11 1 0 1 1 0 1 1 1 0 0 0

N = 13 1 1 1 1 1 0 0 1 1 0 1 0 1

This block can accept an external clock or operate from an internal source. A clock value greater than 0.5 is considered high.

x_1 = Optional external clock

y_1 = Barker code sequence

y_2 = Output clock pulses

Sequence Length

Specifies the length *N* of Barker code. Available lengths include 3, 4, 5, 7, 11 and 13. The default value for the 802.11 specification is 11.

Sequence Offset

Specifies a starting offset for the Barker sequence. Valid range is 0 to *N* -1.

Output Mode

Bilevel

The signal amplitudes associated with the output sequence are {-1, 1}.

Binary

The signal amplitudes associated with the output sequence are {0, 1}.

Timing

Internal

Indicates internal clock timing. The bit rate and start time need to be specified.

External

Indicates external timing. An external clock must be provided at the x1 input.

Bit Rate

Specifies the Barker sequence bit rate in bits per second. This parameter is only available when in Internal Timing mode.

Start Time

Specifies a start time, in seconds, for the Barker sequence. This parameter is only available when in Internal Timing mode.

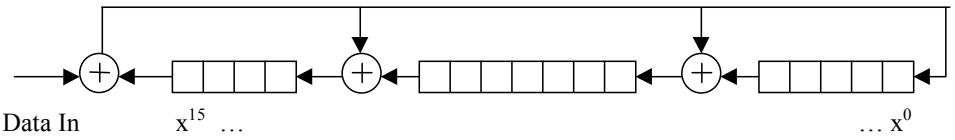
CRC-16 Generator

This block computes the CCITT-16 CRC (Cyclic Redundancy Code) based on an input binary serial data stream. At each input clock pulse, the block will update the CRC output vector, which is comprised of 16 elements (x^{15}, x^{14}, … x^1, x^0). The output can be provided as is, or flipped (one's complement mode).

A CRC is commonly used to verify that a data sequence is received error free by including it's "state" at the end of a data packet. If the CRC computed at the receiver (based on the received data) matches the CRC sent by the transmitter, then it's highly likely that the block was received correctly.

The CRC shift register can be reset to its starting value (all 1's) by providing a pulse on the Reset input. The first element of the output vector is the MSB of the CRC (x^{15} cell). The generator polynomial and internal shift register structure are shown below.

$$p(x) = x^{16} + x^{12} + x^5 + 1$$



x_1 = Binary data [0, 1]

x_2 = Data clock (pulse train)

x_3 = Reset (pulse)

y_1 = CRC output vector (size 16)

Output Mode

Regular

The block outputs the internal shift register state as is.

One's Complement

The block outputs the one's complement of internal shift register state (all 0's are flipped to 1's and all 1's are flipped to 0). This mode is used in the 802.11 specification.

802.11 Hop Generator

This block generates an 802.11 frequency-hopping pattern. This block supports both the 79 and 23 hop specifications, depending on the specified Operating Region. For specific details on the 802.11 hop sequences, please refer to Section 14.6.4 – 14.6.8 of the 802.11 specification.

This block outputs a hopping pattern represented by a hop index (for driving a Frequency Hop block) and a Channel number for information purposes. The hop index range is [0, 78] or [0, 22] depending on the Hop Mode selection. This block support both internal and external timing. When in external mode, the user must provide a pulse train indicating when the next hop slot is starting.

Note: the output Hop Index specifies the hop operating frequency using the standard 1 or 2 Mbps (Low Rate) channel assignments beginning with index value 0. For example, Channel #37 (centered at 2437 MHz) will be output as Hop Index #35, since channel numbers start with Channel #2 (2402 MHz).

This block should be followed by a Frequency Hop block to implement the actual hopping (use the Hop Index output as the drive signal).

x_1 = Pattern selection input

x_2 = Optional external hop clock (pulse train)

y_1 = Hop Index # [0, 78] or [0, 22]

y_2 = Output hop clock pulse

y_3 = Hop Channel # [2, 80] or [2, 24]

Region

Specifies the 802.11 operating geographical region. Choices include North America/Canada, Europe, Japan, Spain and France.

Timing Mode

Internal

Indicates internal clock timing.

External

Indicates external timing. An external clock signal must be provided.

Initial Hop Index

Specifies the initial internal counter value for the 802.11 hopping pattern.

Hop Rate

Specifies the hop rate for the block in hops per second. This parameter is only available when in Internal Timing mode. The default hop rate is TBD kHz.

Start Time

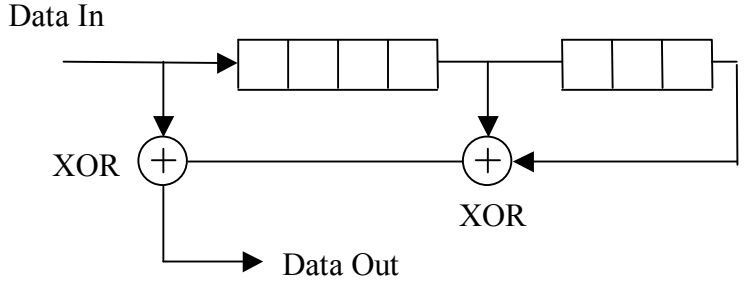
Specifies a starting time for the hop sequence. This parameter is only available when in Internal Timing mode.

802.11 Descrambler

This block provides a data descrambling function as specified in the 802.11 and 802.11b standards. The input data is XOR'ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized whenever an external pulse is presented at the frame clock input. Depending on the operating mode – either long or short PLCP preamble – the shift register is initialized with either hex 0x6C (1101100) or 0x1B (0011011) respectively, with the LSB corresponding to the rightmost shift register cell. The feed-through implementation on the scrambler and descrambler is self-synchronizing, which does not require any knowledge of the initial Tx shift register state for receive processing.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



The user must supply an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Reset clock pulse

y_1 = Output data

y_2 = Output data clock pulses

Register Init Value

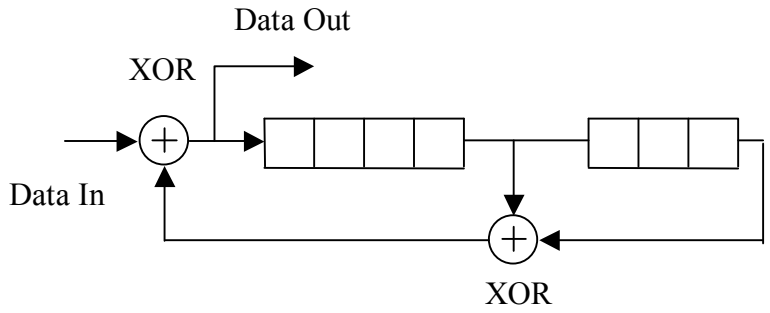
Specifies the initial value of the shift register in hex format. Since the descrambler is self-synchronizing, providing an initial value simply provides a means of achieving an immediate synchronization with the transmitter, instead of having to wait for seven clock cycles.

802.11 Scrambler

This block provides a data scrambling function as specified in the 802.11 and 802.11b standards. The input data is XOR'ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized whenever an external pulse is presented at the frame clock input. Depending on the operating mode – either long or short PLCP preamble – the shift register is initialized with either hex 0x6C (1101100) or 0x1B (0011011) respectively, with the LSB corresponding to the rightmost shift register cell.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



The user must supply an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Frame clock pulse

y_1 = Output data

y_2 = Output data clock pulses

Register Initialization

Long Preamble

The register is initialized to hex 0x6C as specified in the 802.11 specification for use with the long preamble.

Short Preamble

The register is initialized to hex 0x1B as specified in the 802.11 specification for use with the short preamble.

GFSK-2 Modulator

This block implements a two level Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK-2 modulation, the digital information is transmitted by shifting the carrier frequency between two states. The 802.11 GFSK-2 mode specifies the following frequency deviations:

Input Symbol	*Carrier Deviation*	
1	1/2 x h2 x *R* = 160 kHz	(for h2 = 0.32 and *R* = 1 Msps)
0	-1/2 x h2 x *R* = -160 kHz	

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the 802.11 specification for the 1 Mbps mode, which specifies a BT value of 0.5 and a nominal frequency deviation +/- 160 kHz.

x = Input data signal (binary [0, 1])

y = Complex output signal [Re, Im]

GFSK-4 Modulator

This block implements a four level Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK-4 modulation, the digital information is transmitted by shifting the carrier frequency between four states. The 802.11 GFSK-4 mode specifies the following frequency deviation values depending on the input data symbol (first received bit is the LSB):

Input Symbol	*Carrier Deviation*	
1 0	3/2 x h4 x *R* = 216 kHz	(for h4= 0.144 and *R* = 1 Msps)
1 1	1/2 x h4 x *R* = 72 kHz	
0 1	-1/2 x h4 x *R* = -72 kHz	
0 0	-3/2 x h4 x *R* = -216 kHz	

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the 802.11 specification for the 2 Mbps mode, which specifies a BT value of 0.5 and the frequency deviations shown above.

x = Input data signal (symbol value [0, 1, 2, 3])

y = Complex output signal [Re, Im]

802.11a/g BLOCKS

802.11a/g Convolutional Encoder

This implements an 802.11a/g rate ½ convolutional encoder. The block's settings allow specification of the generator coefficients and constraint length, but the default values correspond to those of the 802.11a/g specification (k=7, G1= 133, G2= 171 octal). All other code rates other than rate ½ are achieved by puncturing the output of this block (see Puncture block).

At each input clock pulse, the blocks internal shift register is shifted to the right and a new input data bit is read. The convolutional encoder's A and B outputs are then updated.

x_1 = Input data bits [0, 1]

x_2 = Input clock (pulse train)

y_1 = A output

y_2 = B output

y_3 = Output clock pulse

Generator #1

Specifies the value of the first generator coefficient in octal format. The default is 133, which corresponds to "1011011" in binary.

Generator #2

Specifies the value of the first generator coefficient in octal format. The default is 171, which corresponds to "1111001" in binary.

Constraint Length

Specifies the length of the internal shift register in bits.

802.11a/g Depuncture

This block converts a received serial bit stream into pairs (A B) of data suitable for use by the 802.11a/g Viterbi Decoder block. For code rates other than rate ½, this block also provides a depuncture function, which inserts dummy bits (erasures) in the locations where bits were "stolen" at the encoder side.

Knowledge of the output symbol rate (A and B pairs) is required to so that the block can properly output the depunctured bit pairs. The output data stream is always re-synched with the input serial clock at the start of each new pattern period (every 2 input clock pulses for a rate 1/2 code; every 3 input clock pulses for rate 2/3; and every 4 input clock pulses for rate 3/4). For further details on code puncturing see the description of the 802.11a/g Puncture block.

x_1 = Input bits

x_2 = Input clock (pulse train)

y_1 = A output

y_2 = B output

y_3 = Output clock (pulse train)

Code Rate

Specifies the desired code rate as Rate 1/2, Rate 2/3 or Rate 3/4.

Output Symbol Rate

Specifies the output symbol rate (A B pairs) so that the block can properly output the depunctured bit pairs. This value should match the symbol rate used at the encoder block.

Erasure Value

Specifies the received value to be used for the dummy bits inserted at the erasure locations. This value is typically set to 0 since a soft-decision Viterbi block (which usually follows this block) is typically set to operate with bi-level data [-1, +1].

802.11a/g Puncture

This block is used to convert the output of the 802.11a/g Convolutional Encoder into a serial bit stream and also implement puncturing when the selected code rate is not rate ½. Puncturing is implemented by "stealing" specific coded bits from the encoder output (i.e. not transmitting such bits). At the receiver, these stolen bit slots are filled with dummy bit values.

Knowledge of the input symbol rate (clock rate) is required to so that the block can derive the appropriate output serial timing. This block does not require that the output bit interval be comprised of an integral number of simulation samples. As a result, the duration of individual output bits can differ slightly from bit to bit within the underlying "pattern period", which is defined as the periodic time interval (in input clock cycles) required to implement each specific puncture pattern. The output stream is always re-synched with the input clock at the start of each new pattern period.

At each input clock new A and B value are read, and represented herein by increasing subscripts (i.e. A_0 B_0, A_1 B_1, A_2 B_2). The following describes the puncture pattern for each code rate (i.e. which bits are output).

Rate 1/2:

Pattern Period= 1 clock cycle Output pattern: A_0 B_0

Rate 2/3:

Pattern Period= 2 clock cycles Output pattern: A_0 B_0 A_1

Rate 3/4:

Pattern Period= 3 clock cycles Output pattern: A_0 B_0 A_1 B_2

x_1 = Input A value

x_2 = Input B value

x_3 = Input clock pulse

y_1 = Serial punctured output

y_2 = Output clock (pulse train)

Code Rate

Specifies the desired code rate as Rate 1/2, Rate 2/3 or Rate 3/4.

Input Symbol Rate

Specifies the approximate input symbol rate (A B pairs) so that the block can properly compute the output serial timing for the selected code rate.

802.11a/g Interleaver

This implements an 802.11a/g Interleaver or Deinterleaver. The block size of the interleaver is set to correspond to the number of coded bits in a single OFDM frame, depending on the intended rate of the link per Section 17.3.5.6 of the 802.11a/g specification. This block is normally inserted after the convolutional encoding process.

This block basically implements a “block type” interleaver, and introduces a delay equal to the block size (i.e. number of coded bits in the OFDM frame). The block size is set as follows depending on the selected Rate Mode:

6 Mbps, 9 Mbps:	48 bits
12 Mbps, 18 Mbps:	96 bits
24 Mbps, 36 Mbps:	192 bits
48 Mbps, 54 Mbps:	288 bits

x_1 = Input data

x_2 = Input clock (pulse train)

y_1 = Output data

y_2 = Output clock (pulse train)

802.11a Rate Mode

Specifies the data rate of the 802.11a/g link being simulated (not necessarily the simulation rate).

Mode

Interleave

The block operates as an interleaver.

Deinterleave

The block operates as a deinterleaver.

802.11a/g Scrambler

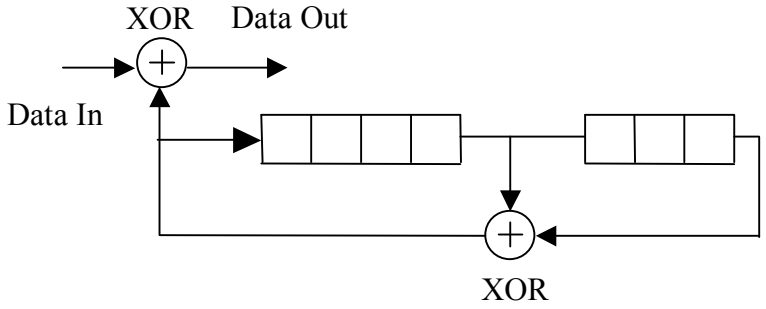
This block provides a data scrambling/descrambling function as specified in the 802.11a/g standard. The input bits are XOR'ed with the output sequence of a feedback shift register of size $N = 7$, which produces a repeating pattern of length 127.

The internal shift register is re-initialized whenever an external pulse is presented at the reset input. Depending on the operating mode – either internal or external initialization – the shift register is then initialized with either a fixed hex value or an externally provided value, where the LSB corresponds to the rightmost shift register cell in the diagram below. The initialization value is typically set to a pseudo-random non-zero value.

Note: when this block is used to scramble the 802.11a/g OFDM pilot tones, the shift register is typically initialized to all 1's.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



x_1 = Input bits

x_2 = Input clock (pulse train)

x_3 = Register initialization value (external mode only)

x_4 = Reset input (pulse)

y_1 = Output data

y_2 = Output clock (pulse train)

Register Initialization

External

The register is initialized to the current value presented at the external "Init" input.

Internal

The register is initialized to the hex value specified in the Register Init Value parameter field.

Register Init Value

Specifies the reset value for the internal shift register when in Internal register initialization mode. This value is specified in hex format.

802.11a/g Viterbi Decoder

This block implements a rate ½ soft decision Viterbi Decoder compatible with the 802.11a/g specification. This block is used to decode a convolutionally encoded bit stream. Parameters include the encoder constraint length (k), the trellis truncation length M, the number of quantization bits, and the generator coefficients. An external Metric file must also be specified.

This block accepts soft A and B ouputs from a 802.11a/g Depuncture block, which should be in bilevel format (i.e. +/- 1 without any noise).

x_1 = A Input ~[-1, +1]

x_2 = B Input ~[-1, +1]

x_3 = Input clock (pulse train)

y_1 = Decoded bit stream (0, 1)

y_2 = Ouput clock (pulse train)

y_3 = Best path metric

Generator #1

Specifies the value of the first generator coefficient in octal format. The default is 133, which corresponds to "1011011" in binary.

Generator #2

Specifies the value of the first generator coefficient in octal format. The default is 171, which corresponds to "1111001" in binary.

Constraint Length

Specifies the constraint length *k* of the associated encoder.

Trellis Truncation Length

Specifies the trellis truncation length *M*. The maximum allowed value of *M* is 96.

Quantization Bits

Specifies the number of quantization bits used in the decoding process.

Select File

Opens the Select File dialog box for selecting the desired Metric file.

Browse File

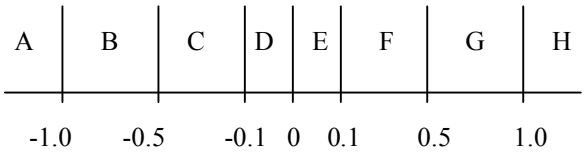
Opens the selected Metric file using Notepad.

Metric File Path

Specifies the path and filename of the metric file to be used in decoding. The metric file format is described below:

Header line (can be anything)			*Q* = # quantization bits
Q	*m(A/0)*	*m(A/1)*	T_{AB} = threshold value between
T_{AB}	*m(B/0)*	*m(B/1)*	regions "A" and "B"
T_{BC}	*m(C/0)*	*m(C/1)*	*m(C/1)* = metric for region "C"
...	...	...	given a "1" was sent

Note: File entries can be separated by a comma, tab, or whitespace. A metric value of 0 is considered best.



Example Metric File

Viterbi Decoder metric file for use with erasures (0), Quant bits = 3

3,	0.0,	7.0
-1.0,	1.0,	6.0
-0.5,	2.0,	5.0
-0.1,	3.0,	3.0
0,	3.0,	3.0
0.1,	5.0,	2.0
0.5,	6.0,	1.0
1.0,	7.0,	0.0

For example:
for an input $0.5 \leq x < 1.0$,
the metric for a "0" bit is "6.0",
and the metric for a "1" bit is "1.0".

Note: In the above example, values in the range of [-0.1, 0.1], which includes the erasure value (dummy bit) 0, are given an equal metric.

OFDM Demodulator

This block demodulates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal back to a group of data and pilot subcarriers, as specified in the IEEE 802.11a/g standard. For more details on OFDM, please refer to the OFDM Modulator block description.

The OFDM signal is demodulated by applying a forward FFT to the received I and Q complex baseband inputs. An external trigger pulse is used to signal the beginning of each received frame to the block.

Since a Guard Interval (GI) may be used in the transmission of the OFDM frame, this block allows the user to specify an FFT wraparound value to be applied to the received vector prior to performing the FFT. This feature is useful when the active samples used by the OFDM Demodulator include samples from the GI (instead of just FFT segment samples). Since the GI is generated from a cyclic shift of the FFT segment, this feature allows the OFDM Demodulator block to recover the proper IFFT samples.

The OFDM Demodulator block accepts a baseband OFDM modulated signal in I and Q component format. Other inputs include a sampling clock (pulse train) and a trigger used to signal the beginning of each OFDM frame. Block outputs include a frame clock and two data vectors representing the Real and Imaginary components of the recovered "subcarrier"data. This block can be followed by an OFDM Pilot Extract block to separate-out the data and pilot subcarriers.

x_1 = Beginning of frame trigger

x_2 = I baseband input

x_3 = Q baseband input

x_4 = Input sampling clock (pulses)

y_1 = Frame clock pulse

y_2 = Real output subcarrier vector (size $2^{\wedge N}$)

y_3 = Imaginary output subcarrier vector (size $2^{\wedge N}$)

FFT Size

Specifies the size of the forward FFT (power of two) to be performed on the received data. Valid range is from 16 to 64K. This value should match the IFFT size used by the transmitter.

FFT Wraparound Offset

Specifies a cyclic offset to be applied to the received data prior to performing the FFT operation. This parameter is useful when samples from the GI are included in the received frame.

OFDM Modulator

This block generates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a/g specification.

In OFDM modulation, the output is obtained by applying an Inverse FFT (typically a power of two) to an input vector of complex IQ points corresponding to data and pilot "subcarriers". To reduce ISI, the inverse FFT output is often padded with an optional guard interval representing a cyclic shift of the inverse FFT output itself. The resulting OFDM "symbol" is then treated as a baseband time domain signal and used for transmission.

This block accepts two input vectors representing the Real and Imaginary components of the subcarrier data. An OFDM Pilot Map block can be used to merge the data subcarriers with any pilot or null subcarriers. The input vectors are read when a frame clock pulse is detected. Additional inputs include a guard interval selector, an extended FFT period flag, and an optional external clock for the output data. Block outputs include time domain baseband I and Q outputs representing the OFDM symbol, and an output sample clock.

The external Guard Interval (GI) selection input allows the user to vary the guard interval size during the simulation. The following describes the GI corresponding to each allowed input value:

0 ➔ No GI
1 ➔ GI = FFT Size / 2
2 ➔ GI = FFT Size / 4
3 ➔ GI = FFT Size / 8
4 ➔ GI = FFT Size / 16
5 ➔ GI = FFT Size / 32

The size of the output frame is equal to GI + FFT size, unless the extended FFT input is set high. The optional extended FFT period connector is used to signal the OFDM Modulator to output the FFT segment twice (i.e. the output equals GI + FFT + FFT).

The OFDM Modulator also supports windowing between successive OFDM frames to reduce spectral sidelobes. The user can specify an overlap ranging from 0 to 3 samples. The window function follows a $\sin^2$ rolloff profile as specified in the 802.11a/g specification.

In this implementation a windowed overlap is generated by extending the GI at the beginning of the frame by a few samples, and carrying over the last few samples of the FFT section to the next output frame. For example, if an overlap of 2 samples is used, the GI length is increased by two samples (to which a rolloff is applied) and overlapped with the last two samples (also weighted by a rolloff) from the previous frame's FFT section. This implementation reduces simulation delay and internal buffering needs.

x_1 = Input Frame clock pulse

x_2 = Real input vector (size $2^{\wedge N}$)

x_3 = Imaginary input vector (size $2^{\wedge N}$)

x_4 = Guard Interval selector {0, 1, 2, 3, 4, 5}

x_5 = Extended FFT period flag (High = extend FFT period)

x_6 = Optional external sample clock

y_1 = I signal output

y_2 = Q signal output

y_3 = Output clock pulses

Timing

Internal

Indicates internal clock timing per the specified output rate.

External

Indicates external timing. An external clock must be provided at the x_6 input.

Use Simulation Sample Rate

When selected, this mode forces the OFDM Modulator block to use the current simulation sample rate as its output rate.

Output Rate

Specifies the output rate for the OFDM output samples. This parameter is only available when in Internal Timing mode. The default value is set to the current simulation rate.

FFT Size

Specifies the size of the inverse FFT (power of two) to be performed. Valid range is from 16 to 64K.

Overlap Size

Specifies an overlap between successive OFDM output frames. Specifying an overlap results in a windowing operation and provides a smoother transition from frame to frame. Valid range is from 0 to 3 samples.

OFDM Pilot Extract

This block decomposes the Re/Im output of an OFDM Demodulator into two groups of subcarrier vectors (Data and Pilot) according to a user defined mapping file.

The subcarrier mapping file should match the one used by the OFDM transmitter, and serves to specify the ordering in the received vector of the data, pilot and any null subcarriers (which are discarded).

The user must specify the number of data and pilot subcarriers, as well as the input vector size, which must be a power of two (FFT size). The user can also specify the starting position within the input vector for element #0 in the map file as either 0 or N/2, where N is the FFT size. The Data and Pilot subcarrier values are written in sequential order to their respective outputs according to their order of appearance in the Map File.

For more details on the Map File format, please refer to the OFDM Pilot Map block description.

x_1 = Input Frame clock pulse

x_2 = Real output vector from OFDM Demod (size $2^{\wedge N}$)

x_3 = Imaginary output vector from OFDM Demod (size $2^{\wedge N}$)

y_1 = Frame clock pulse

y_2 = Data subcarrier I vector

y_3 = Data subcarrier Q vector

y_4 = Pilot subcarrier I vector

y_5 = Pilot subcarrier Q vector

Input Insertion Point

Begin at N/2 Location

Forces an N/2 circular shift in the input vector, where N is the size of the FFT used by the OFDM Demodulator. Element zero in the Map File is read from the N/2 input vector location.

Begin at 0 Location

Element zero in the Map File is read from the 0 input vector location.

Number of Data Subcarriers

Specifies the size of the Data subcarrier I and Q output vectors.

Number of Data Subcarriers

Specifies the size of the Pilot subcarrier I and Q output vectors.

Data Subcarrier Weight

Specifies the weight applied to the I and Q Data subcarriers in the modulator. The received subcarrier is scaled (divided) by this value.

Pilot Subcarrier Weight

Specifies the weight applied to the I and Q Pilot subcarriers in the modulator. The received subcarrier is scaled (divided) by this value.

FFT Size

Specifies the block's input vector size *N*. This value should match the size of the FFT used by the preceding OFDM Demodulator, as well as the Map File's number of entries. Valid range is from 16 to 64K.

Select File

Launches a dialog box for selecting the desired Subcarrier Map file.

Browse File

Opens the selected Subcarrier Map file using Notepad.

Subcarrier Mapping File

Specifies the path and filename for the Subcarrier Map file. The format of the generated file is shown below. The keyword identifier specifies the subcarrier type to be used for each element. Allowed keywords are: DATA, PILOT, and NULL. Valid entry separators are commas, tabs and blank space.

Header line (for storing user defined data)
element #0 *keyword*
element #1 *keyword*
…
element #N-1 *keyword*

OFDM Pilot Map

This block generates a composite Re/Im pair of "subcarrier" vectors for use by an OFDM Modulator or Vector OFDM Modulator block.

This block accepts two pairs of input vectors (each with Real and Imaginary components) representing data subcarriers and pilot subcarriers. A user defined data mapping file is used to specify the ordering in the output vector of the data, pilot and any null subcarriers.

The user must specify the number of data and pilot subcarriers, as well as the output vector size, which must be a power of two (FFT size). The user can also specify the starting position within the output vector for the input data as either 0 or N/2, where N is the FFT size. The Data and Pilot subcarrier values are read in sequential order from their respective input ports and placed in the output vector according to the output ordering specified by the Map File.

The format of the "Map File" is as follows. The number of entries in the file should match the intended OFDM FFT size. A one line header is assumed. Thereafter, the first entry of each row represents the output vector element #, and is followed by a keyword identifying whether that location is to contain a data subcarrier (DATA), a pilot subcarrier (PILOT), or a null subcarrier (NULL). Any additional characters on the line are ignored. If the "Begin at N/2" option is selected, then the output vector is cyclically shifted by N/2 before being output (i.e. the entry for the #0 element will be output at the N/2 vector location.

x_1 = Input Frame clock pulse

x_2 = Data subcarrier I vector

x_3 = Data subcarrier Q vector

x_4 = Pilot subcarrier I vector

x_5 = Pilot subcarrier Q vector

y_1 = Frame clock pulse

y_2 = Output vector (Real component) (size $2^{\wedge N}$)

y_3 = Output vector (Imaginary component) (size $2^{\wedge N}$)

Input Insertion Point

Begin at N/2 Location

Forces an N/2 circular shift in the output vector, where N is the size of the FFT used by the OFDM Modulator. Element zero from the Map File is output in the N/2 output vector location.

Begin at 0 Location

Element zero from the Map File is output in the 0 output vector location.

Number of Data Subcarriers

Specifies the size of the Data subcarrier I and Q input vectors.

Number of Data Subcarriers

Specifies the size of the Pilot subcarrier I and Q input vectors.

Data Subcarrier Weight

Specifies a weight to be applied to the I and Q Data subcarriers.

Pilot Subcarrier Weight

Specifies a weight to be applied to the I and Q Pilot subcarriers.

FFT Size

Specifies the block's output vector size *N*. This value should match the size of the FFT used by the OFDM Modulator that will follow, as well as the Map File's number of entries. Valid range is from 16 to 64K.

Select File

Launches a dialog box for selecting the desired Subcarrier Map file.

Browse File

Opens the selected Subcarrier Map file using Notepad.

Subcarrier Mapping File

Specifies the path and filename for the Subcarrier Map file. The format of the generated file is shown below. The keyword identifier specifies the subcarrier type to be used for each element. Allowed keywords are: DATA, PILOT, and NULL. Valid entry separators are commas, tabs and blank space.

Header line (for storing user defined data)
element #0 *keyword*
element #1 *keyword*
...
element #N-1 *keyword*

OFDM Vector Demodulator

This block demodulates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a/g specification. For more details on OFDM, please refer to the OFDM Modulator block description.

This block differs from the regular OFDM Demodulator block in that it uses vector inputs for the I and Q received signals, instead of scalars. The OFDM signal is demodulated by applying a forward FFT to the received I and Q complex baseband samples. An external frame clock pulse is used to signal each new OFDM symbol.

Since a Guard Interval (GI) may be used in the transmission of the OFDM symbol, this block allows the user to specify a limited FFT offset value to be applied to the received vector prior to performing the FFT. This feature is useful when the input vector used by the Vector OFDM Demodulator includes some GI samples (instead of just FFT segment samples). Since the GI is generated from a cyclic shift of the FFT segment, this feature allows the Vector OFDM Demodulator block to recover the proper IFFT samples.

The Vector OFDM Demodulator block accepts a baseband OFDM modulated signal in I and Q vector format and a frame (symbol) clock. Block outputs include a frame clock and two data vectors representing the Real and Imaginary components of the recovered “subcarrier”data. This block can be followed by an OFDM Pilot Extract block to separate-out the data and pilot subcarriers.

x_1 = Input frame clock

x_2 = I baseband vector (size $2^{\wedge N}$)

x_3 = Q baseband vector (size $2^{\wedge N}$)

y_1 = Frame clock pulse

y_2 = Real output data vector (size $2^{\wedge N}$)

y_3 = Imaginary output data vector (size $2^{\wedge N}$)

FFT Size

Specifies the size of the forward FFT (power of two) to be performed on the received data. Valid range is from 16 to 64K. This value should match the IFFT size used by the transmitter.

FFT Offset

Specifies a cyclic offset to be applied to the received data vector to performing the FFT operation. This parameter is useful when samples from the GI are included in the received frame. Valid range is 0 to 3.

OFDM Vector Modulator

This block generates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a/g specification.

This block differs from the regular OFDM Modulator block in that it uses vector outputs for the I and Q signals, instead of scalars. This version of the block does not support varying the Guard Interval (GI) size during a simulation, nor does it support the extended FFT mode. For more details on OFDM, please refer to the OFDM Modulator block description.

This block accepts two input vectors representing the Real and Imaginary components of the subcarrier data. An OFDM Pilot Map block can be used to merge the data subcarriers with any pilot or null subcarriers. The input vectors are read when a frame clock pulse is detected. Block outputs include I and Q vector OFDM outputs, and an output frame clock. The size of the output vector is equal to GI + FFT size. The GI size can range from none to 1/32 of the FFT size.

The Vector OFDM Modulator supports windowing between successive OFDM frames to reduce spectral sidelobes. The user can specify an overlap ranging from 0 to 3 samples. The window function follows a $\sin^2$ rolloff profile as specified in the 802.11a/g specification.

In this implementation, a windowed overlap is generated by extending the GI at the beginning of the frame by a few samples, and carrying over the last few samples of the FFT section to the next output frame. For example, if an overlap of 2 samples is used, the GI length is increased by two samples (to which a rolloff is applied) and overlapped with the last two samples (also weighted by a rolloff) from the previous frame's FFT section. This implementation reduces simulation delay and internal buffering needs.

x_1 = Input Frame clock pulse

x_2 = Real input vector (size $2^{\wedge N}$)

x_3 = Imaginary input vector (size $2^{\wedge N}$)

y_1 = Output frame clock pulse

y_1 = I signal output

y_2 = I signal output vector (size GI + FFT)

y_3 = Q signal output vector (size GI + FFT)

FFT Size

Specifies the size of the inverse FFT (power of two) to be performed. Valid range is from 16 to 64K.

Overlap Size

Specifies an overlap between successive OFDM output frames. Specifying an overlap results in a windowing operation and provides a smoother transition from frame to frame. Valid range is from 0 to 3 samples.

Guard Interval Size

Specifies the size of the desired Guard Interval as a fraction of the IFFT size. Choices include: None, ½, ¼, 1/8, 1/16 and 1/32 of the IFFT size.

802.11b BLOCKS

802.11b High Rate Hop Generator

This block generates an 802.11b frequency-hopping pattern optionally used with the High Rate modes (5.5 and 11 Mbps). This block supports both the Set 1 and Set 2 channel selections for either North America or Europe (except France and Spain).

This block supports either internal or external hop timing. When in external mode, the user must provide an input pulse train indicating when the next hop slot is starting. This block outputs a hop pattern consisting of a hop index (for driving a Frequency Hop block) and either a High Rate or Low Rate channel number as appropriate for information purposes. A High/Low flag is used to characterize the channel number as this can vary during the hop process. High Rate channels can range from 1 to 13, while Low Rate channels range from 2 to 80.

North America Set 1

Set 1 hop channels include High Rate Channels 1, 6 and 11.

For Set 1 only two hop patterns are specified:

Pattern#1: 1, 6, 11, 1, 6, 11…

Pattern#2: 1, 11, 6, 1, 11, 6…

North America Set 2

Set 2 hop channels include High Rate Channels 1, 3, 5, 7, 9 and 11.

For Set 2 the input pattern number AND the current Hop Index determine the next hop channel number according to section 18.4.6.7.3 of the 802.11b specification.

Europe Set 1 (except France and Spain)

Set 1 hop channels include High Rate Channels 1, 7 and 13.

For Set 1 only two hop patterns are specified:

Pattern#1: 1, 7, 13, 1, 7, 13…

Pattern#2: 1, 13, 7, 1, 13, 7…

Europe Set 2 (except France and Spain)

Set 2 hop channels include High Rate Channels 1, 3, 5, 7, 9, 11 and 13.

For Set 2 the input pattern number AND the current Hop Index determine the next hop channel number according to section 18.4.6.7.3 of the 802.11b specification.

This block should be followed by a Frequency Hop block to implement the actual hopping (use the Hop Index output as the drive signal).

Note: the output Hop Index specifies the hop operating frequency using the Low Rate (1, 2 Mbps) channel assignments beginning with index value 0. For example High Rate channel #6, centered at 2437 MHz and corresponding to Low Rate Channel #37, will be output as Hop Index #35. The offset value of two is due to the fact that the 802.11 channel numbers start with Channel #2 (2402 MHz).

x_1 = Pattern number selection input

x_2 = Optional external hop clock (pulse train)

y_1 = Hop Index # [0, 78]

y_2 = Output hop clock pulse

y_3 = Hop Channel # (see notes above)

y_4 = High/Low Rate flag

Region

Specifies the 802.11b operating geographical region. Choices include North America/Canada, and Europe (except Spain and France).

Timing Mode

Internal

Indicates internal hop timing.

External

Indicates external timing. An external clock signal (pulse train) must be provided to indicate each hop.

Hop Set

Specifies which hop set to use; either Set 1 or Set 2.

Initial Hop Index

Specifies the initial internal counter value for the 802.11b hopping pattern.

Hop Rate

Specifies the hop rate for the block in hops per second. This parameter is only available when in Internal Timing mode. The default hop rate is TBD kHz.

Start Time

Specifies a starting time for the hop sequence. This parameter is only available when in Internal Timing mode.

CCK Demodulator

This block provides a symbol detection function for Complementary Code Keying (CCK) modulated signals. CCK modulation, as defined in the IEEE 802.11b specification, is used to provide a High Rate mode operating at either 5.5 Mbps or 11 Mbps, which is compatible with the pre-existing lower rate data modes (1 and 2 Mbps) of the 802.11 standard. The CCK chips are modulated using QPSK. For additional details on CCK modulation, see the description of the CCK Modulator.

This block accepts a complex baseband input and a chip clock, and produces a decoded bits vector of size 4 or 8 (depending on the CCK mode). The received sequence is compared internally with the CCK waveform set, and the one with maximum correlation is selected as the best estimation of the transmitted signal.

Each time eight new chips are received, a new output vector and a frame clock pulse are presented at the block's output. An external reset capability for the internal chip counter is also provided for synchronization purposes.

x_1 = Complex baseband input (Re, Im)

x_2 = Chip clock (pulse train)

x_3 = Reference Phase {0, 1, 2, 3}

x_4 = Ref. Phase initialization strobe (pulse)

x_5 = Frame counter Reset

y_1 = Output frame clock pulse

y_2 = Output data bits vector (size 4 or 8)

y_3 = Internal "C" chip vector (size 8) {0, 1, 2, 3}

CCK Mode

(8, 4) CCK

Specifies (8, 4) CCK mode, as used in the IEEE 802.11b 5.5 Mbps rate.

(8, 8) CCK

Specifies (8, 8) CCK mode, as used in the IEEE 802.11b 11 Mbps rate.

CCK Modulator

This block generates a complex baseband modulated Complementary Code Keying (CCK) signal. CCK modulation, as defined in the IEEE 802.11b specification, is used to provide a High Rate mode operating at either 5.5 Mbps or 11 Mbps, and which is compatible with the pre-existing lower rate data modes (1 and 2 Mbps) of the 802.11 standard. CCK constructs orthogonality or semi-orthogonality from the input signal vector using an expanded Hadamard transform. The CCK chips are modulated using QPSK.

The block's function can be broken down into two main functions. The first is a mapping (encoding) of the input data vector (size 4 or 8 bits) into an eight element CCK chip vector (available as an external output). The second is the conversion of this "C vector" into a baseband time domain complex IQ signal.

For the 5.5 Mbps specification, 4 input data bits are mapped to 8 CCK chips. In this case, herein referred to as (8, 4) CCK modulation, the input data words are $\{d_0, d_1, d_2, d_3\}$, where d_0 comes first in time. CCK modulation maps the data bits into phases $\{\varphi_0, \varphi_1, \varphi_2, \varphi_3\}$ by pairs. The first pair (d_0, d_1) controls the value of φ_0, which is mapped differentially according to the previous data words, with the phase shift defined in Table 1. The remaining phase terms are defined as $\varphi_1 = d2 \times \pi + \frac{\pi}{2}$, $\varphi_2 = 0$, and $\varphi_3 = d3 \times \pi$.

Dibit pattern (d0, d1)	Even symbols Phase change (+jω)	Odd symbols Phase change (+jω)
00	0	π
01	π/2	3π/2
11	π	0
10	3π/2	π/2

Table 1. DQPSK encoding table

For the 11 Mbps specification, 8 input data bits are mapped to 8 CCK chips. In this case, herein referred to as (8, 8) CCK modulation, the input data words are $\{d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$, where d_0 comes first in time. The pair (d_0, d_1) is mapped differentially according to the previous data words as in the CCK (8, 4) case. The remaining pairs (d_i, d_{i+1}) are mapped to the remaining phase terms as defined in Table 2.

Bit pairs (d_i, d_{i+1})	Phase
00	0
01	π/2
10	π
11	3π/2

Table 2. QPSK encoding table

The resulting four phase terms are then combined as follows to obtain the actual CCK modulated code word, which is comprised of eight "chips". These 8 chips represent the output of the CCK Encoder block.

$$\vec{c} = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$$
$$= \{e^{j(\varphi_1+\varphi_2+\varphi_3+\varphi_4)}, e^{j(\varphi_1+\varphi_3+\varphi_4)}, e^{j(\varphi_1+\varphi_2+\varphi_4)}, -e^{j(\varphi_1+\varphi_4)},$$
$$e^{j(\varphi_1+\varphi_2+\varphi_3)}, e^{j(\varphi_1+\varphi_3)}, -e^{j(\varphi_1+\varphi_2)}, e^{j\varphi_1}\}$$

This block accepts an input vector of either 4 or 8 data bits and outputs a QPSK modulated baseband signal. An auxiliary output is also provided to reveal the values of the eight CCK output chips (C vector), which can assume one of four possible phase states {0, 1, 2, 3}. If desired, the modulated IQ output signal can be translated to a specific carrier frequency by using an IQ Modulator block.

x_1 = Input frame clock (pulse train)

x_2 = Input bits data vector (size 4 or 8)

x_3 = Reference Phase {0, 1, 2, 3}

x_4 = Ref. Phase initialization strobe (pulse)

x_5 = External chip clock (pulse train) [optional]

y_1 = I signal output

y_2 = Q signal output

y_3 = Chip clock (pulse train)

y_4 = Internal "C" chip vector {0, 1, 2, 3}

CCK Mode

(8, 4) CCK

Specifies (8, 4) CCK mode, as used in the IEEE 802.11b 5.5 Mbps rate.

(8, 8) CCK

Specifies (8, 8) CCK mode, as used in the IEEE 802.11b 11 Mbps rate.

Timing

Internal

Indicates internal clock timing per the specified chip rate.

External

Indicates external timing. An external clock must be provided at the x5 input.

Chip Output Rate

Specifies the output chip rate in bits per second. This parameter is only available when in Internal Timing mode. The 802.11b default is 11 MHz.

GENERIC WIRELESS BLOCKS

Frequency Hop

This block imparts frequency hopping to a complex baseband input signal according to the external channel selection input. This block does not affect the amplitude of the input signal and merely provides a frequency translation operation.

x_1 = Input signal (complex)

x_2 = Hop channel selection input

y_1 = Frequency hopped output (complex)

$$f_{out} = x_1 \cdot (f_{\min} + x_2 \cdot \Delta f)$$

Number of Hops

Specifies the number *N* of available hop channels. This value is used internally for range checking purposes only.

Lowest Hop Channel

Specifies the carrier frequency corresponding to the lowest hop channel in *hertz*. This corresponds to input channel # 0.

Hop Carrier Spacing

Specifies the frequency spacing Δf between adjacent hop channels in *hertz*.

Initial Phase

Specifies the initial carrier phase in *degrees*. This option is only available when in Continuous Phase mode.

Phase Mode

Continuous

Indicates that the signal phase is continuous across channel hops. This simulates the use of a VCO or NCO in generating the output signal.

Discontinuous

Indicates that the signal phase is not continuous across channel hops. This simulates the use of multiple free running oscillators to generate each of the hopping channel carriers.

ULTRAWIDEBAND BLOCKS

Gated Integrate & Dump

This block provides an Integrate and Dump function where the integration process is only active during a narrow user-defined time "window". The user may specify both the pulse width of the window as well as its start time relative to the last input "frame clock".

This block is useful in UWB diagrams when only a narrow time window (out of a larger frame time) is to be processed for pulse detection purposes.

x_1 = Input signal

x_2 = Frame clock pulse

y_1 = Integrated output

y_2 = Dump clock

$$y_1(t)_k = \int_a^b x_1(t)\partial t \quad \text{where } a = (t_p)_k + T_{start} \qquad b = a + T_{width}$$

$$\text{and } (t_p)_k = \text{time of kth frame pulse}$$

Window Start Time

Specifies the start time in *seconds* of the active integration window referenced to the last input frame clock pulse (input x_2).

Window Width

Specifies the active time duration in *seconds* of the integration window.

Suppress First Output Clock

When this option is selected, the very first input frame clock still internally dumps the I&D block (essentially a reset), but the corresponding output dump pulse is suppressed. This option makes it easier to achieve frame synchronization in some diagrams by making the first output pulse coincide with the first valid I&D dumped value.

Output Mode

Continuous

Indicates that the integrated output is continuously updated. Note: forward (Euler) integration is used, resulting in a one-sample delay between the receipt of an input value and its contribution appearing in the integrated output.

Held

Indicates that the integrated output is updated with each input frame clock and held constant during the entire frame's duration. As a result, a one frame period delay is present in the integrated output.

UWB PPM Modulator

This block implements a Pulse Position Modulator (PPM) suitable for use in UWB simulations. The block operates by delaying an input clock pulse by a variable amount depending on the symbol value of its data input. The user can control the number of pulse position locations and specify an optional guard interval at the end of each data frame.

x_1 = Input data symbol k in range [0, N-1]

x_2 = Frame clock pulse

y_1 = PPM modulated unit pulse

y_2 = Frame clock pulse

$$T_{slot} = \frac{1}{N} \cdot \left(\frac{1}{R_S} - T_g \right) \qquad \Delta T(k) = k \cdot T_{slot} \qquad k \in \{0,..N-1\}$$

Number of Levels

Specifies the number *N* of desired pulse positions. The input signal should be in the range of [0, .. *N*-1].

Symbol Rate

Specifies the input symbol data rate R_s in *Hertz*.

Guard Interval

This value specifies an optional time interval T_g in *seconds* between the last valid pulse position slot and the beginning of the next symbol frame.

UWB Pulse

This block generates a wide variety of commonly used ultrawideband pulses, including the Gaussian pulse shape and its time derivatives. The block operates by generating the specified pulse shape each time an impulse is presented at the block's input. The block is also designed to sense the sign of the incoming pulse and produce a same-sign output waveform. For the block to register an input pulse, its amplitude must be larger than 0.5.

User block parameters include the defined pulse duration, pulse width, pulse amplitude and pulse time offset as appropriate. The defined pulse duration refers to the actual time span for which Altair Embed SE Comm computes the pulse shape. Outside of this period, the output is zero. Pulse width is defined differently for each type of pulse shape (see below).

Note: Each time a new input pulse is sensed, the block immediately begins to output the defined pulse shape from its starting position.

This block supports the following pulse shapes:

- Monopulse
- Gaussian Pulse
- Gaussian Monocycle (1st derivative of Gaussian pulse)
- Gaussian Doublet (2nd derivative of Gaussian pulse)
- Gated Sinusoid

Please note that the list of block parameters associated with each pulse shape vary from type to type. In the section that follows, the block parameters which are common to all the pulse shapes are defined first. The remaining parameters are described under each pulse type.

x_1 = Input impulse train signal [-1, 1]

y_1 = UWB output

Pulse Amplitude Mode

Peak Amplitude

Indicates that pulse amplitude is defined by specifying the max amplitude of the pulse waveform.

Held

Indicates that pulse amplitude is defined by specifying the total power in each pulse assuming a theoretical non-time-limited pulse.

Defined Pulse Duration

Specifies the time interval in *seconds* for which Altair Embed SE Comm computes the value of the pulse waveform.

Pulse Amplitude
Pulse Power

Specifies either the pulse amplitude *A* in *Volts* or the pulse power in *dBm* depending on the selected Pulse Amplitude Mode setting.

Time Units

Specifies the time units associated with the pulse duration, time offset and pulse width parameters. Choices include *seconds*, *milliseconds*, *microseconds*, *nanoseconds* and *picoseconds*.

View Pulse Shape

Invokes the Altair Embed SE Comm Graphics Viewer to display the selected pulse waveform in both the time domain and frequency domain.

UWB Pulse Type

Specifies the desired UWB pulse shape. Available choices include: Monopulse, Gaussian Pulse, Gaussian Monocycle, Gaussian Doublet, and Gated Sinusoid. Descriptions for each pulse follow.

Monopulse

$$y(t) = A \cdot e^{\left(1 - \frac{(t - t_0)}{\tau}\right)} \qquad \text{for } t \geq t_0 \qquad y(t) = 0 \quad \text{otherwise}$$

Time Offset

Specifies the start time t_0 of the Monocycle pulse relative to the received input pulse time.

Pulse Width

Defines the width of the pulse by specifying the time interval τ between the start of the pulse and its peak amplitude time.

Gaussian Pulse

$$y(t) = A \cdot e^{-\left(\frac{2\sqrt{\ln 2}(t-t_0)}{\tau}\right)^2}$$

Pulse Peak Time

Specifies the time offset t_0, relative to the received input pulse time, when the Gaussian pulse is to reach its peak value.

Pulse Width

Defines the width of the pulse by specifying the time interval τ between the pulse's half amplitude points.

Gaussian Monocycle

$$y(t) = 2A\sqrt{e} \cdot \frac{(t-t_0)}{\tau} \cdot e^{-2\left(\frac{(t-t_0)}{\tau}\right)^2}$$

Zero Crossing Time

Specifies the time offset t_0, relative to the received input pulse time, when the Gaussian monocycle is to cross through zero.

Pulse Width

Defines the width of the pulse by specifying the time interval τ between the peaks of the monocycle waveform.

Gaussian Doublet

$$y(t) = \frac{A}{\tau} \cdot e^{-2\left(\frac{(t-t_0)}{\tau}\right)^2} \cdot \left(1 - \frac{4(t - t_0)}{\tau}\right)$$

Pulse Peak Time

Specifies the time offset t_0, relative to the received input pulse time, when the Gaussian Doublet is to reach its peak value.

Pulse Width

Defines the width of the pulse by specifying the time interval τ between the pulse's zero crossing points.

Gated Sinusoid

$$y(t) = A \cdot \sin(2\pi\ f_c\ (t-t_0) + \frac{\theta\pi}{180}) \quad \text{for } t \in [t_0, T) \qquad y(t) = 0 \quad \text{otherwise}$$

Start Time Offset

Specifies the start time t_0 of the sinusoid burst relative to the received input pulse time.

Tone Duration

Specifies the time period T during which the sinusoidal waveform is ON.

Tone Frequency

Specifies the center frequency f_c in *Hertz* of the gated sinusoid.

Starting Phase

Specifies the starting phase θ in *degrees* of the sinusoidal waveform.

Appendix A

Embed/Comm Library

This appendix describes the compound blocks and data files provided in the COMMLIB folder.

Compound Blocks

COSTAS_C.VSM

This compound block implements a complex Costas loop, which is used to track the phase of PSK modulated signals. This compound block accepts a modulated complex signal and outputs I and Q channel baseband signals and a complex VCO output. This block is based on a second order PLL design. After additional filtering, the multiplier's I and Q outputs are both used to compute the phase error term. The phase error detector provided with this compound block is tailored to BPSK. For other modulation schemes, a more suited detector should be substituted.

It is important to ensure that the phase detector gain and VCO gain are properly set in the `Loop Filter` block's parameters. The amplitude of the input signal is assumed to be 1. If this is not the case, its value must be taken into account in determining the phase detector gain. A unity phase detector gain indicates that an error signal of 0.1 V at the `Loop Filter` input represents a 0.1 rad phase error. In order for the PLL to operate properly, the simulation sampling frequency should be much larger than the PLL natural frequency (`Loop Filter` block parameter). Also, the IIR filter cutoff frequencies should be appropriately matched to the data rate.

x = Input signal (complex)

y_1 = Output signal (complex)

y_2 = VCO output signal (complex)

COSTAS_R.VSM

This compound block is identical to the COSTAS_C.VSM compound block, except that it accepts a modulated real signal.

x = Input signal (real)

y_1 = Output signal (complex)

y_2 = VCO output signal (complex)

GFSK.VSM

This compound block implements a GFSK modulator. It employs a Gaussian FIR Filter block and an FM Modulator block as its internal components. The internal parameters of each of these blocks need to be specified for proper operation. The default settings reflect a Bluetooth implementation.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

GMSK.VSM

This compound block implements a GMSK modulator. It employs a `Gaussian FIR` block and an `FM` modulator block as its internal components. The internal parameters of each of these blocks need to be specified for proper operation.

x = Input data signal (binary)

y = Output signal (complex)

PLL1CPLX.VSM

This compound block implements a complex first order PLL. The VCO output attempts to follow the input signal. The internal phase error signal is obtained by multiplying the input signal by the VCO output. This error signal is then passed through a `gain` block and used as the VCO drive signal. The loop gain is computed by multiplying the phase detector gain by the VCO gain. A phase detector gain value of 1 indicates that an error signal of 0.1 V represents a 0.1 rad phase error. A first order loop cannot remove a frequency offset and also achieve 0 phase error. If a frequency offset is present, use a second order PLL.

x = Input signal (complex)

y_1 = VCO output signal (complex)

y_2 = VCO phase (radians)

PLL1REAL.VSM

This compound block is identical to the PLL1CPLX.VSM compound block, except that it accepts a real signal input. The gain factor of -2 prior to the `multiply` block is used to compensate for the factor of two loss through the phase detector due to the formation of a double frequency term in this implementation.

x = Input signal (real)

y_1 = VCO output signal (real)

y_2 = VCO phase (radians)

PLL2CPLX.VSM

This compound block implements a second order PLL. The VCO output attempts to follow the input signal. The internal phase error signal is obtained by complex multiplying the input signal by the VCO output. This error signal is then passed through the `Loop Filter` block and its output is used as the VCO drive signal. A second order PLL can remove both a phase offset and a frequency offset (assuming a high gain loop). If Doppler rate is present, then a third order PLL is recommended. For descriptions of the `VCO` and `Loop Filter` blocks, look under their respective names earlier in this section.

It is important to ensure that the phase detector gain and VCO gain are properly set in the `Loop Filter` block's parameters. The amplitude of the input signal is assumed to be 1. If this is not the case, its value must be taken into account in determining the phase detector gain. A unity phase

detector gain indicates that an error signal of 0.1 V at the `Loop Filter` input represents a 0.1 rad phase error. In order for the PLL to operate properly, the simulation sampling frequency should be much larger than the PLL natural frequency (`Loop Filter` block parameter).

x = Input signal (complex)

y_1 = VCO output signal (complex)

y_2 = VCO phase (radians)

PLL2REAL.VSM

This compound block is identical to the PLL2CPLX.VSM compound block, except that it accepts a real signal input. The gain factor of -2 prior to the `multiply` block is used to compensate for the factor of two loss through the phase detector due to the formation of a double frequency term in this implementation.

x = Input signal (real)

y_1 = VCO output signal (real)

y_2 = VCO phase (radians)

TWTA_TBL.VSM

This compound block provides a TWTA channel based on AM/AM and AM/PM conversion look-up tables. You can modify the files AMAM.MAP and AMPM.MAP to simulate the desired tube characteristics. The AMAM.MAP file maps tube Input Power (decibels) to Output Power (decibels). The AMPM.MAP file maps tube Input Power (decibels) to Output Phase Rotation (degrees). An externally provided Reference Power Level (watts) is used to specify the saturation operating point, also referred to as the 0 dB backoff point.

x_1 = Input signal (complex)

x_2 = Reference power level

y = Output signal (complex)

$$y = G(r)e^{j\Phi(r)}x_1$$

where: $G(r) = \text{am/am function}$ $\Phi(r) = \text{am/pm function}$

$r = \text{instantaneous complex power scaled by } x_2$

V32DIFDE.VSM

This compound block implements a V.32 differential decoder for use in simulating V.32 trellis decoding. The input symbol is divided into four bit streams, and the two LSBs are differentially decoded (modulo 4). The bit streams are then recombined to form the output symbol.

x_1 = Input symbol

x_2 = Clock

y = Output symbol

V32DIFEN.VSM

This compound block implements a V.32 differential encoder for use in simulating V.32 trellis decoding. The input symbol is divided into four bit streams, and the two LSBs are differentially encoded (modulo 4). The bit streams are then recombined to form the output symbol.

x_1 = Input symbol

x_2 = Clock

y = Output symbol

VCPG.VSM

This compound block implements a *voltage controlled pulse generator* (VCPG). The output represents a pulse train where the pulse frequency is controlled by the input signal level. The block comprises a `VCO` block, a `crossDetect` block, and a `limit` block to eliminate the negative pulse at the half cycle point. When the input drive level is 0, the `VCPG` outputs a pulse train at the specified VCO center frequency. With a non-zero input, the pulse frequency will vary depending on the magnitude of the drive signal and the specified VCO gain.

x = Input signal

y = Pulse train output (0, 1)

Data Files

AMAM.DAT

This data file contains a nonlinear mapping of input power (decibels) to output power (decibels) for the TWTA_TBL.VSM compound block. It was obtained by using the coefficient values from example #1, under the description of the `TWTA` block in Chapter 3, “Comm Block Set.”

AMPM.DAT

This data file contains a nonlinear mapping of input power (decibels) to output phase (degrees) for the TWTA_TBL.VSM compound block. It was obtained by using the coefficient values example #1, under the description of the `TWTA` block in Chapter 3, “Comm Block Set.”

DATA_IN.DAT

This file is an example of an input data file for the `File Data` source block. It illustrates the use of multiple entries per line, and the use of different data delimiters.

PSK_GRAY.DAT

This data file specifies a Gray encoded constellation mapping for all the PSK modulation formats. Gray encoding ensures that neighboring constellation points only differ in one bit from one another. For a description of the file format, refer to the description of the `PSK` modulator block in Chapter 3, “Comm Block Set.”

QAM_GRAY.DAT

This data file specifies a Gray encoded constellation mapping for all the QAM and PAM modulation formats except for QAM-32. Gray encoding ensures that neighboring constellation points only differ in one bit from one another. For a description of the file format, refer to the description of the `QAM/PAM` modulator block in Chapter 3, “Comm Block Set.”

TABLFILT.DAT

This data file contains a filter response specification for the complex `MagPhase` filter block. It corresponds to a 64 tap lowpass FIR filter with a 10 Hz cutoff. The file contains magnitude and phase entries over the range of -50 to 50 Hz.

V32QAM.DAT

This data file contains a constellation mapping for QAM-32. The constellation provided is that used in the V.32 standard. For a description of the file format, refer to the description of the `QAM/PAM` modulator block in Chapter 3, “Comm Block Set.”

V32TRELS.DAT

This data file contains the trellis mapping for V.32 trellis coded modulation. For a description of the file format, refer to the description of the `Trellis Encoder` block in Chapter 3, “Comm Block Set.”

VTB3SOFT.DAT

This data file contains a metric table for three-bit soft decision Viterbi decoding. For a description of the file format, refer to the description of the `Viterbi Decoder (Soft)` block in Chapter 3, “Comm Block Set.”

Appendix B

Turbo Codes Overview

Origins of turbo codes

A well known result from information theory is that a randomly chosen code of sufficiently large block length *n* is capable of approaching channel capacity. The maximum-likelihood (ML) decoding of such a code increases exponentially with *n* up to a point where decoding becomes physically unrealizable. Over the years, the goal of coding theorists has been to develop codes that have large equivalent block lengths, yet contain enough structure that practical decoding is possible. A major milestone in this effort was reached in 1993 with the introduction of turbo codes [BER93]. Due to the use of a nonuniform interleaver, turbo codes appear random to the channel, yet posses enough structure that decoding can be physically realized.

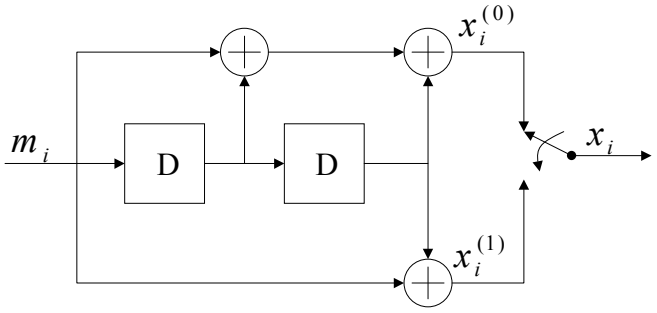
In the original paper [BER93], the authors show through simulation that binary phase shift keying (BPSK) modulated rate ½ turbo codes are capable of achieving the arbitrarily small Bit Error Rate (BER) of 10^{-5} at an E_b/N_o ratio of just 0.7 decibels, which is just ½ dB away from the Shannon capacity for BPSK and rate ½. However, there were four weaknesses with the original turbo code: (1) The original turbo code required a large block size of 65,532 data bits, which precluded their application to real-time voice or video. (2) The BER curves tended to flatten out for large values of E_b/N_o resulting in a virtual error floor for BER $\approx 10^{-6}$. (3) The original decoder was rather complex, having a per-bit complexity that was equivalent to a constraint length 15 convolutional code. (4) The decoder required accurate estimates of the symbol-by-symbol signal to noise ratio of the received signal.

Despite these initial practical shortcomings, later research alleviated many of these weaknesses. The result was code designs that, although they did not have quite the same near-capacity performance as the original turbo code, significantly improved performance over similar convolutionally coded systems. Turbo codes are now poised to be used in both emerging 3-G cellular systems (UMTS/UTRA/3GPP and cdma2000/3GPP2), for deep space communications, and various military systems. The question is no longer "*will* turbo codes be used in practice?" but rather "*when* will turbo codes start hitting the mass-marketplace?"

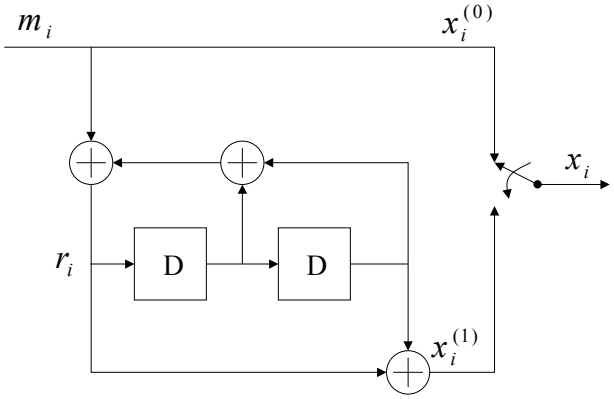
Convolutional codes

In order to fully understand turbo codes, one must first understand convolutional codes. Before the introduction of turbo codes, power efficient communications was achieved by either a strong

convolutional code or the serial concatenation of a convolutional code with a block code[1]. A convolutional code adds redundancy to a continuous stream of input data by using a linear shift register and some combinatorial logic. For a rate *r=k/n* convolutional encoder, each set of *n* output symbols is a linear combination of the current set of *k* input bits and *m* bits stored in the shift register. The total number of bits that each output depends on is called the *constraint length*, and is denoted by *K*. An example convolutional encoder with rate *r=1/2*, memory *m=2*, and constraint length *K=3* is shown below.



A convolutional code can be made systematic[2] without affecting its minimum distance properties by feeding back one of the outputs to the input. Such a code is called a *Recursive Systematic Convolutional* (RSC) code, and is the basic building block for turbo codes. An example RSC encoder derived from the above nonsystematic encode is shown below.



The decoder for a convolutional code finds the most probable sequence of data bits {*m*} given the received sequence {*y*}, where {*y*} is the set of code symbols {*x*} observed through noise and/or fading. This sequence is found using the Viterbi algorithm. Although the Viterbi algorithm goes beyond the scope of this manual, its understanding is necessary to fully understand the turbo decoding algorithm discussed later in this chapter. There are several excellent introductions to the Viterbi algorithm, including papers [FOR73], books [WIC95], and web-pages[3]. The Viterbi

---

[1] *For example, the IS-95 CDMA cellular standard uses a constraint length* K = 9 *convolutional code, while the NASA deep space communication standard uses a constraint length* K = 7 *convolutional code concatenated with a Reed Solomon code.*

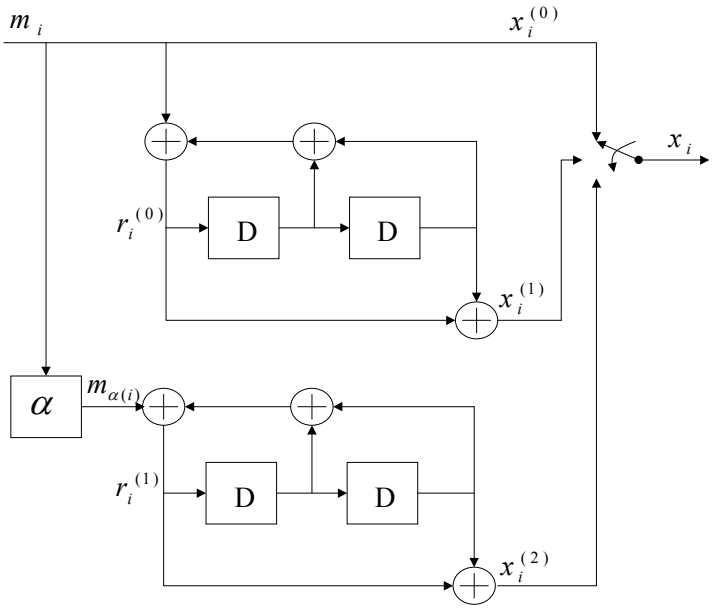
[2] *A systematic code is one for which each* n *symbol code word contains the* k *data bits. The remaining* n-k *bits are called* parity bits.

[3] *http://www.alantro.com/viterbi/viterbi.htm*

algorithm represents the convolutional encoder as a finite-state trellis and attempts to seek the best path through the trellis. The complexity of the Viterbi algorithm is related to the number of possible states, i.e. $O(2^K)$ per bit.

Turbo encoding

A turbo code is the parallel concatenation of two RSC codes separated by an interleaver. An example turbo encoder is shown below.

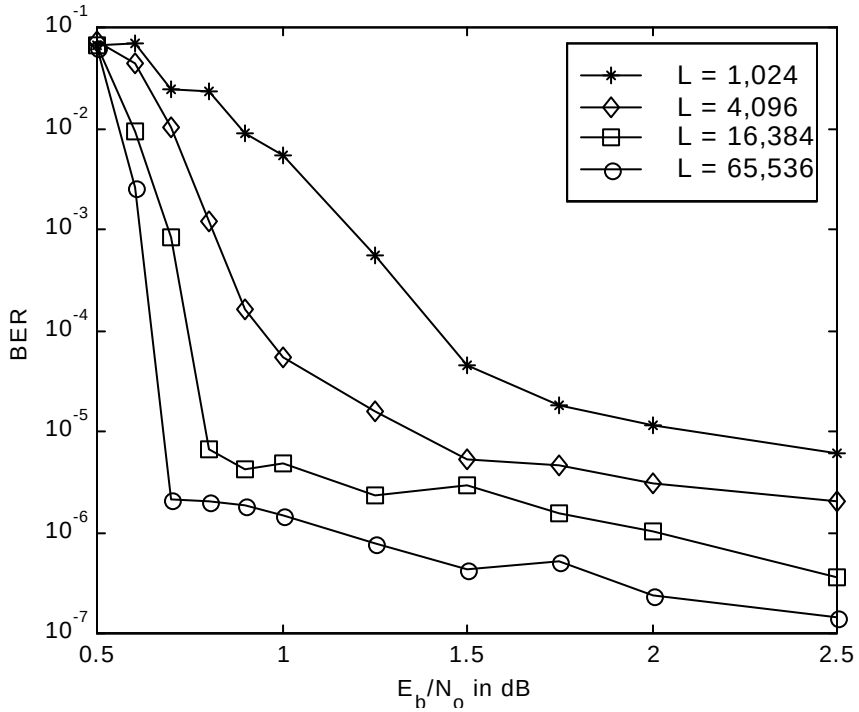


Here, the two encoders are identical rate 1/2 RSC encoders. The upper encoder receives the data directly, while the lower encoder receives the data after it has been interleaved by a permutation function α. The interleaver is in general a nonuniform interleaver --- that is it maps bits in position $\alpha(i)$ to position i according to a prescribed, but pseudo-random rule. The interleaver operates in a blockwise fashion, interleaving L bits at a time, and thus turbo codes are actually linear block codes. Since both encoders are systematic and receive the same set of data (although in permuted order), only one of the systematic outputs needs to be sent. By convention, the systematic output of the top encoder is sent while the systematic output of the lower encoder is not transmitted. However, the parity outputs of both encoders are transmitted. The overall code rate of a turbo code composed from the parallel concatenation of two rate ½ systematic codes is r = ⅓. This code rate can be made higher by *puncturing*[4]. The code rate of a turbo code is often increased to r = ½ by only transmitting the odd indexed parity bits from the upper encoder and the even indexed parity bits from the lower encoder (along with all the systematic bits from the top encoder). Other puncturing patterns can be used to attain a wider range of rates [ROW00], [ACK99].

Turbo Code performance factors

There are many factors that affect the performance of turbo codes. The most influential parameter is the interleaver size. As the frame/interleaver size increases, performance improves. Below, the performance of the r = ½, K = 5 BPSK modulated turbo code is shown in an additive white Gaussian noise (AWGN) channel for various interleaver sizes and 18 iterations of decoding.

[4] *Puncturing is the process of deleting (i.e. not transmitting) a subset of the parity bits.*



Note that as the interleaver gets smaller, performance degrades. This would imply that one would select the largest possible interleaver size. However, as the interleaver size increases so does decoder latency, since the entire code word must be received before decoding can be completed. Thus turbo codes possess an inherent tradeoff between performance and latency. In [VAL97] it is shown that this tradeoff can be exploited for wireless multimedia communication systems.

Another parameter affecting performance is the overall code rate. Just like other codes, performance improves as the code rate becomes lower. When code rates higher than ⅓ are used, then the particular puncturing pattern that is used impacts the performance. Like convolutional codes, the constraint length also influences performance. However, the impact that constraint length has on performance is not significant, and thus only the short constraint lengths of K = 3, 4, or 5 are considered for practical turbo codes.

The interleaver design plays a significant role in the performance of a turbo code, particularly for higher signal-to-noise ratios. In general, a randomly chosen interleaver design will give good performance, while highly structured interleavers such as the ``block interleaver'' should be avoided. The most widely accepted nonuniform interleaver is the S-random interleaver developed at NASA's Jet Propulsion Laboratory [DOL95]. The S-random interleaver is designed to guarantee that adjacent bits at the input of the interleaver are at least S bits apart at the output of the interleaver. Thus the parameter *S* plays a major role in determining performance and should be as large as possible. In theory the maximum value for *S* is equal to the square root of the interleaver size, although in practice it can't be more than half its theoretical maximum value. The UMTS and cdma2000 standards do not use S-random interleavers; rather each system specifies a particular deterministic algorithm for generating interleavers of various lengths.

Another factor that influences performances is the method of trellis termination. The performance of the turbo code is best when each of the constituent RSC encoders is terminated with a tail. However, due to the recursive nature of the encoder, the tails are not all-zeros as they would be for a conventional convolutional code. Thus it is not likely that the same tail will terminate both encoders. There are several options that can be taken. First, neither encoder could be terminated with a tail. Second, only one encoder could be terminated (usually the top encoder) and the tail bits for the first encoder could be fed as data into the second encoder (whose trellis is not terminated). Third, both encoders could be terminated independently with separate tails. This

option requires that both tails be transmitted, which requires a slight modification to the policy of only transmitting the first encoder's systematic bits. The turbo codes used by UMTS and cdma2000 both follow this third option, which typically offers the best performance.

While turbo codes offer extraordinary performance for bit error rates down to about 10^{-5}, the performance for very small bit error rates is not very impressive. Note the BER "floors" that appear for low bit error rates in the previous figure. For high signal-to-noise ratios, it may in fact be better to use a convolutional code. This phenomenon can be explained in terms of the Hamming distance spectrum of turbo codes. For high signal-to-noise ratios, the performance of a code is approximated by its *free distance asymptote* [PER96]

$$P_b \approx w_{eff} Q\left(\sqrt{\frac{d_{free}}{2rE_b / N_o}}\right),$$

where d_{free} is the free distance[5] of the code, and w_{eff} is the *effective multiplicity* of code words of weight d_{free}. The slope of the free distance asymptote is determined by the argument of the Q function, and thus by the free distance. Bit error performance can be improved in two ways: (1) By increasing the free distance, and thus making the asymptote steeper, or (2) Decreasing the effective multiplicity, thereby lowering the entire BER curve. The design of convolutional codes focuses on maximizing free distance, but the design of turbo codes focuses on minimizing the effective multiplicity.

The constraint length *K=15* convolutional code[6] has d_{free} of 18, while for the *K=5* turbo code, d_{free} is only 6. Thus the free distance asymptote of the convolutional code is much steeper than the free distance asymptote of the turbo code. However, w_{eff} of the convolutional code is 187 while w_{eff} for the turbo code is only 6/L. For L = 65,536 data bits per frame, this translates to $w_{eff} \approx 0.0001$, and thus the leading coefficient of the turbo code's free distance asymptote is much lower than the convolutional code's.

The number of free distance code words in a turbo code is very small due to the combination of nonuniform interleaving, recursive encoding, and parallel code concatenation. Due to their recursive nature, the impulse response of RSC encoders is infinite. Thus, most input sequences cause large weight output sequences. However there are a few input sequences that cause small weight outputs. But because of interleaving, the two RSC encoders in a turbo code do not receive their inputs in the same order. Thus the probability that both encoders produce small weight outputs is very small, accounting for the small effective multiplicity. For more information about the performance of turbo codes, please review the work of Benedetto [BEN96].

In the next figure, the simulated performance and free distance asymptotes are shown for both the Berrou turbo code (i.e. rate ½, constraint length 5, and 65,536 data bits per block) and the K=15 convolutional code (also rate ½) in AWGN with BPSK modulation.

Note that for BER of 10^{-5}, the turbo code is about 2 decibels more power efficient than the convolutional code. However, at a BER of 10^{-8} the asymptotes of the turbo code and convolutional code intersect, implying that for signal-to-noise ratios greater than about 4 dB the turbo code would actually perform worse than the convolutional code.

The BER can be lowered one of two ways, by using a primitive feedback polynomial, or by using a serial code concatenation. While the Berrou turbo code used generator polynomial (37, 21) in octal[7], the error floor can be lowered by using generator polynomial (35,23) which has a primitive

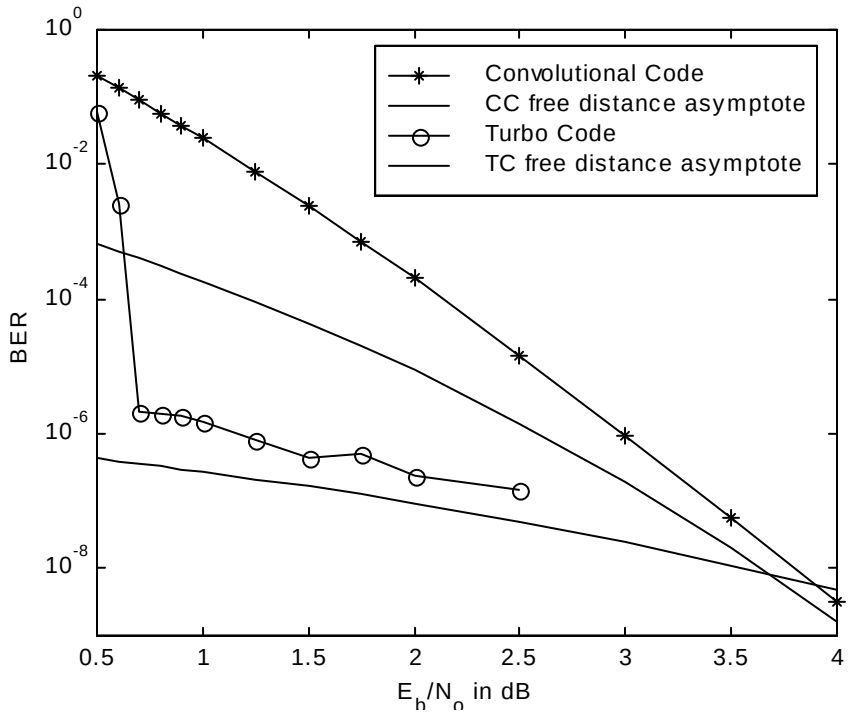
---

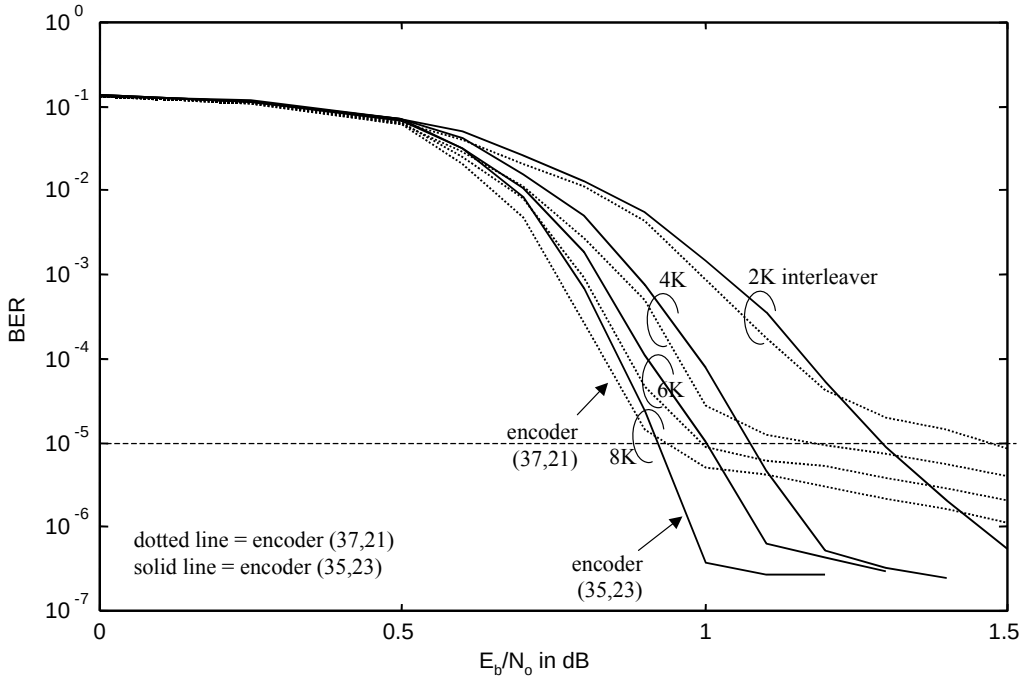
[5] *The free distance is the minimum Hamming weight of all non-zero code words. The Hamming weight is found by counting the number of ones in the code word.*

[6] *The* K=15 *was chosen for comparison purposes because its decoder has roughly the same complexity as the* K=5 *turbo code with 18 iterations of decoding.*

[7] *The convention when specifying code polynomials for RSC codes is to list the feedback polynomial first (37) and the feedforward polynomial second (21).*

feedback polynomial. A comparison of the performance of these two polynomials for various frame sizes is shown in the second figure.





Note that the "waterfall" or steep part of the BER curve occurs at a lower E_b/N_o for the Berrou encoder, but the encoder with primitive-feedback polynomial has a significantly lower error floor.

This is a common theme with turbo codes — the error floor can often be reduced at the expense of a slightly worse waterfall region.

The original turbo codes were also called Parallel Concatenated Convolutional Codes (PCCCs) due to their overall architecture. If instead of concatenating the codes in parallel, they were concatenated in serial, the result is a Serial Concatenated Convolutional Code (SCCC). Although the waterfall part of the SCCC's BER curve occurs at a higher E_b/N_o than it does for PCCC's, the error floor for SCCC's is virtually nonexistent. The current release of the Altair Embed SE TC Library does not support SCCCs, however more information about this promising technology can be found in [BEN98].

The UMTS turbo code

This section presents a brief overview of the turbo code used in the UMTS specification, as published by the Third Generation Partnership Project (3GPP). A more detailed explanation can be found in [ETSI00].

The UMTS turbo encoder is comprised of two rate ½, constraint length 4 recursive systematic convolutional (RSC) codes concatenated in parallel. The feedforward generator is (15) and the feedback generator is (13), both in octal. The number of data bits at the input of the turbo encoder is L, which takes on values in the range $40 \le L \le 5114$. Data is encoded by the first (i.e. upper) encoder in its natural order, and by the second (i.e. lower) after being interleaved (details of the interleaving are found in the specification). The data bits are transmitted together with the parity bits generated by the two encoders. Thus, the overall encoder code rate is $r = ⅓$, not including the tail bits (discussed below). The output of the encoder is in the form: $X_1, Z_1, Z'_1, X_2, Z_2, Z'_2, \ldots$ where X_k is the k^{th} systematic (i.e. data) bit, Z_k is the parity output from the upper (uninterleaved) encoder and Z'_k is the parity output from the lower (interleaved) encoder.

The trellises of both encoders are forced back to the all-zeros state by the proper selection of tail bits. Unlike conventional convolutional codes, which can be terminated with a tail of zeros, the RSC encoder is terminated with the bits that are fed back (which are determined by the state of the encoder and the feedback polynomial). The tail bits are then transmitted at the end of the encoded frame according to $X_{L+1}, Z_{L+1}, X_{L+2}, Z_{L+2}, X_{L+3}, Z_{L+3}, Z'_{L+1}, Z'_{L+1}, X'_{L+2}, Z'_{L+2}, X'_{L+3}, Z'_{L+3}$, where X represents the tail bits of the upper encoder, Z represents the parity bits corresponding to the upper encoder's tail, X' represents the tail bits of the lower encoder, and Z' represents the parity bits corresponding to the lower encoder's tail. Thus the number of coded bits is 3L+12, and the code rate is L/(3L+12) when tail bits are taken into account.

Turbo decoding

In order to understand the turbo decoding algorithm, it is first necessary to characterize the channel and the received values that are fed into the decoder.

Channel model

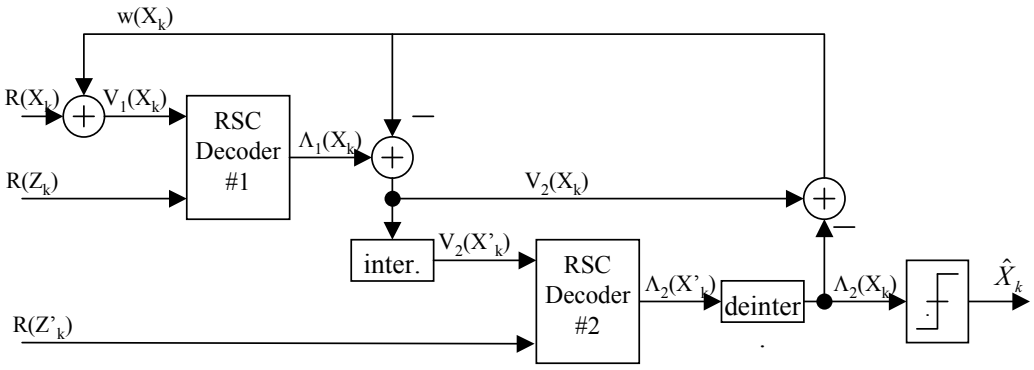
BPSK modulation is assumed along with either an AWGN or flat-fading channel. The output of the matched filter is $Y_k = a_k S_k + n_k$, where $S_k = 2X_k - 1$ for the systematic bits and $S_k = 2Z_k - 1$ for the parity bits, n_k is Gaussian noise with variance $\sigma^2 = (3L+12)/(2L(E_b/N_o))$, and a_k is the channel gain ($a_k = 1$ for AWGN and is a Rayleigh random variable for Rayleigh flat-fading).

The input to the decoder is assumed to be in log-likelihood ratio (LLR) form, which assures that the channel gain and noise variance have been properly taken into account. Specifically, the decoder input is $R_k = 2a_k S_k/\sigma^2$. For the remainder of the discussion, the notation $R(X_k)$ denotes the received LLR corresponding to systematic bit X_k, $R(Z_k)$ denotes the received LLR for the upper parity bit Z_k and $R(Z'_k)$ denotes the received LLR corresponding to the lower parity bit Z'_k.

It is important to note that for the Embed/Comm TC Library, ***the input to the decoder must be in LLR form***, i.e it must be $R_k = 2a_kS_k/\sigma^2$.

Decoder architecture

The architecture of the decoder is as shown below.



As indicated by the presence of a feedback path, the decoder operates in an iterative manner. Each full-iteration consists of two half-iterations, one for each constituent RSC code. The timing of the decoder is such that RSC decoder #1 operates during the first half-iteration, and RSC decoder #2 operates during the second half iteration. The operation of the RSC decoders is fully described later in this chapter.

The value $w(X_k)$ is the *extrinsic information* produced by decoder #2 and introduced to the input of decoder #1. Prior to the first iteration, $w(X_k)$ is initialized to all zeros. Because of the way that the branch metrics were derived, it is sufficient to simply add $w(X_k)$ to the systematic LLR input $R(X_k)$, which forms a new variable denoted $V_1(X_k)$. The input to RSC decoder #1 is both $V_1(X_k)$ and $R(Z_k)$, and the output is the LLR $\Lambda_1(X_k)$. By subtracting $w(X_k)$ from $\Lambda_1(X_k)$, a new variable $V_2(X_k)$ is formed. Similar to $V_1(X_k)$, $V_2(X_k)$ contains the sum of the systematic channel LLR and the extrinsic information produced by decoder #1 (note however that the extrinsic information for RSC decoder #1 never has to be explicitly computed). The input to decoder #2 is $V_2(X'_k)$, which is the interleaved version of $V_2(X_k)$, and $R(Z'_k)$, which is the channel LLR corresponding to the second encoder's parity bits. The output of RSC decoder #2 is the LLR $\Lambda_2(X'_k)$, which is deinterleaved to form $\Lambda_2(X_k)$. The extrinsic information $w(X_k)$ is then formed by subtracting $V_2(X_k)$ from $\Lambda_2(X_k)$ and is fed back to use during the next iteration. Once the iterations have been completed, a hard bit decision is taken using $\Lambda_2(X_k)$, where $est(X_k) = 1$ when $\Lambda_2(X_k) > 0$ and $est(X_k) = 0$ when $\Lambda_2(X_k) \le 0$.

The max* function

The two RSC decoders are each executed using a version of the classic Maximum A Posteriori (MAP) algorithm implemented in the log-domain [ROB97]. As will be discussed later, the algorithm is based on the Viterbi algorithm with two key modifications: First, the trellis must be swept through not only in the forward direction, but also in the reverse direction, and second, the add-compare-select (ACS) operation of the Viterbi algorithm is replaced with the Jacobi logarithm, also known as the max* operator [VIT98]. Because the max* operator must be executed twice for each node in the trellis during each half-iteration, it constitutes a significant, and sometimes dominant, portion of the overall decoder complexity. The manner that max* is implemented is essential to the performance and complexity of the decoder, and several methods

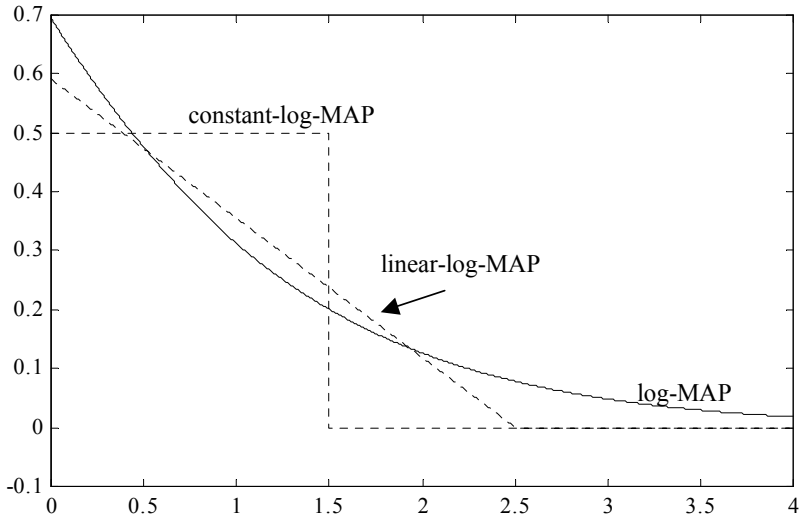
have been proposed for its computation. Below, we consider four variants of the algorithm: log-MAP, max-log-MAP, constant-log-MAP, and linear-log-MAP. The only difference between these algorithms is the manner in which the max* operation is performed.

Log-MAP algorithm

With the log-MAP algorithm, the Jacobi logarithm is computed exactly using:

$$\begin{aligned}\max^*(x,y) &= \max(x,y) + \ln\left(1+\exp\left\{-\left|y-x\right|\right\}\right) \\ &= \max(x,y) + f_c\left(\left|y-x\right|\right),\end{aligned}$$

where the correction function $f_c(|y-x|)$ can be implemented using either the log and exp functions in C, a fairly large look-up table (with at least 8 entries), or a piecewise linear approximation . The log-MAP algorithm is the most complex of the four algorithms, but offers the best BER performance. The correction function used by log-MAP is illustrated below, along with the correction functions used by constant-log-MAP and linear-log-MAP.



The Embed/Comm TC Library comes with two implementations of the log-MAP algorithm. The first implementation computes the correction function exactly by using the *log* and *exp* function calls in C. The second implementation uses an approximation to this expression composed of eight disjoint line segments. The approximation is very good and offers performance that is indistinguishable from the exact function. While both implementations have approximately the same performance, the piecewise linear approximation runs about three times faster then the exact calculation.

Max-log-MAP algorithm

With the max-log-MAP algorithm, the Jacobi algorithm is loosely approximated using:

$$\max^*(x,y) = \max(x,y)\ ,$$

i.e., the correction function of the log-MAP algorithm is not used. The max-log-MAP algorithm is the least complex of the four algorithms (it has exactly twice the complexity of the Viterbi algorithm for each half-iteration), but offers the worst BER performance (about 0.4 dB worse than log-MAP). The max-log-MAP algorithm has the additional benefit of being almost completely

tolerant of imperfect noise variance estimates when operating on an AWGN channel (i.e. performance is independent of the estimate of σ^2).

Constant-log-MAP algorithm

The constant-log-MAP algorithm, approximates the Jacobi algorithm using [GRO98]:

$$\max^*(x,y) = \max(x,y) + \begin{cases} 0 & \text{if } |y - x| > T \\ C & \text{if } |y - x| \le T, \end{cases}$$

where it is shown in [CLA00] that the optimal values for the UMTS turbo code are C = 0.5 and T = 1.5. This algorithm is equivalent to the log-MAP algorithm, with the correction function implemented with a 2-element lookup table. The performance and complexity is between that of the log-MAP and max-log-MAP algorithms, although the BER performance is only about 0.03 dB worse than log-MAP. However, a disadvantage of constant-log-MAP is that it is more susceptible to noise variance estimation errors than is log-MAP.

Linear-log-MAP algorithm

The linear-log-MAP algorithm uses the following linear approximation to the Jacobi algorithm [SUN01]:

$$\max^*(x,y) = \max(x,y) + \begin{cases} 0 & \text{if } |y - x| > T \\ a\left(|y - x| - T\right) & \text{if } |y - x| \le T, \end{cases}$$

where the values of a and T are chosen to minimize the total squared error between the exact correction function and its linear approximation:

$$\int_0^T \left[a\left(z - T\right) - \ln\left(1 + \exp\{-z\}\right)\right]^2 dz + \int_T^\infty \left[\ln\left(1 + \exp\{-z\}\right)\right]^2 dz$$

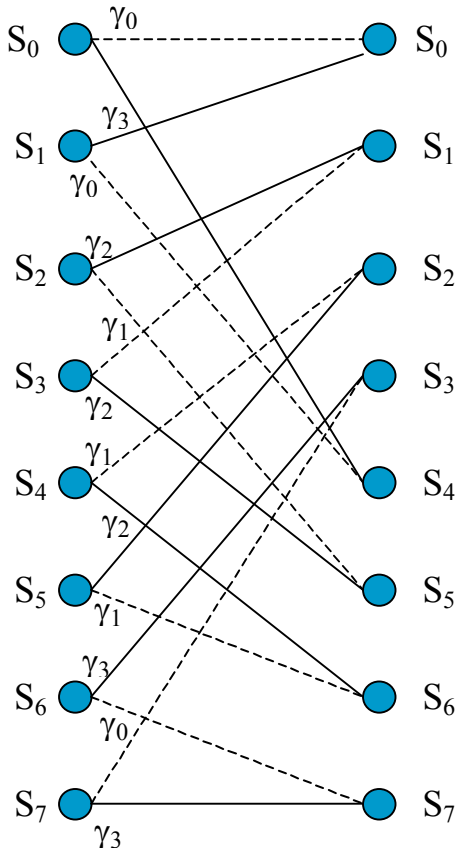
This function is minimized when a = -0.24904 and T = 2.5068. The linear-log-MAP offers performance and complexity between that of the log-MAP and constant-log-MAP algorithms. As will be shown in the simulation results, a key advantage of the linear-log-MAP algorithm is that it converges faster than constant-log-MAP.

MAP algorithm in log-domain

The RSC decoders operate by sweeping through the code trellis in both the forward and reverse directions using the modified Viterbi algorithm (the modification is that the ACS operations are replaced with max*). Then, LLR values can be computed for each stage of the trellis. Two key observations should be pointed out before going into the details of the algorithm: First, it does not matter whether the forward sweep or the reverse sweep is performed first; second, only the partial path metrics for the entire first sweep (forward or backward) must be stored in memory. The LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained for the second sweep.

Trellis structure and branch metrics

The trellis of the RSC encoder used by the UMTS turbo code is as shown below.



Solid lines indicate data $X_k = 1$ and dotted lines indicate data $X_k = 0$. The branch metric associated with the branch connecting states S_i (on the left) and S_j (on the right) is $\gamma_{ij} = V(X_k)X(i,j)+R(Z_k)Z(i,j)$, where X(i,j) is the data bit associated with the branch and Z(i,j) is the parity bit associated with the branch. Because the RSC encoder is rate ½, there are only 4 distinct branch metrics:

$$\gamma_0 = 0$$

$$\gamma_1 = V(X_k)$$

$$\gamma_2 = R(Z_k)$$

$$\gamma_3 = V(X_k) + R(Z_k)$$

The branches in the above figure are all labeled with the corresponding branch metric.

Backward recursion

The Embed/Comm turbo decoder begins with the backward recursion, saving normalized partial path metrics at all the nodes in the trellis (with an exception noted below), which will later be used to calculate the LLRs during the forward recursion. The backward partial path metric for state S_i at trellis stage k is denoted $\beta_k(S_i)$ with $1 \le k \le L+3$ and $0 \le i \le 7$. The backward recursion is initialized with $\beta_{L+3}(S_0) = 0$ and $\beta_{L+3}(S_i) = -\infty \ \forall i > 0$.

Beginning with stage k = L+2 and proceeding through the trellis in the backward direction until stage k = 1, the partial path metrics are found according to:

$$\tilde{\beta}_k(S_i) = \max^*\left\{\left(\beta_{k+1}(S_{j_1}) + \gamma_{ij_1}\right), \left(\beta_{k+1}(S_{j_2}) + \gamma_{ij_2}\right)\right\},$$

where the tilde (~) above $\beta_k(S_i)$ indicates that the metric has not yet been normalized, and S_{j_1} and S_{j_2} are the two states at stage k+1 in the trellis that are connected to state S_i at stage k. After the calculation of $\tilde{\beta}_k(S_0)$, the partial path metrics are normalized according to:

$$\beta_k(S_i) = \tilde{\beta}_k(S_i) - \tilde{\beta}_k(S_0).$$

Because after normalization $\beta_k(S_0) = 0 \ \forall k$, only the other seven normalized partial path metrics $\beta_k(S_i)$, $1 \le i \le 7$, need to be stored.

Forward recursion and LLR calculation

During the forward recursion, the trellis is swept through in the forward direction in a manner similar to the Viterbi algorithm. Unlike the backward recursion, only the partial path metrics for two stages of the trellis must be maintained: The "current" stage k and the "previous" stage k-1. The forward partial path metric for state S_i at trellis stage k is denoted $\alpha_k(S_i)$ with $0 \le k \le L-1$ and $0 \le i \le 7$. The forward recursion is initialized by setting $\alpha_0(S_o) = 0$ and $\alpha_0(S_i) = -\infty \ \forall i > 0$.

Beginning with stage k = 1 and proceeding through the trellis in the forward direction until stage[8] k = L, the unnormalized partial path metrics are found according to:

$$\tilde{\alpha}_k(S_j) = \max^*\left\{\left(\alpha_{k-1}(S_{i_1}) + \gamma_{i_1 j}\right), \left(\alpha_{k-1}(S_{i_2}) + \gamma_{i_2 j}\right)\right\},$$

where S_{i_1} and S_{i_2} are the two states at stage k-1 in the trellis that are connected to state S_j at stage k in the trellis. After the calculation of $\tilde{\alpha}_k(S_0)$, the partial path metrics are normalized using:

$$\alpha_k(S_i) = \tilde{\alpha}_k(S_i) - \tilde{\alpha}_k(S_0).$$

As the α's are computed for stage k, the algorithm can simultaneously obtain an LLR estimate for data bit X_k. This LLR is found by first noting that the likelihood of the branch connecting state S_i at time k-1 to state S_j at time k is:

$$\lambda_k(i,j) = \alpha_{k-1}(S_i) + \gamma_{ij} + \beta_k(S_j)$$

The likelihood of data 1 (or 0) is then the Jacobi logarithm of the likelihood of all branches corresponding to data 1 (or 0), and thus:

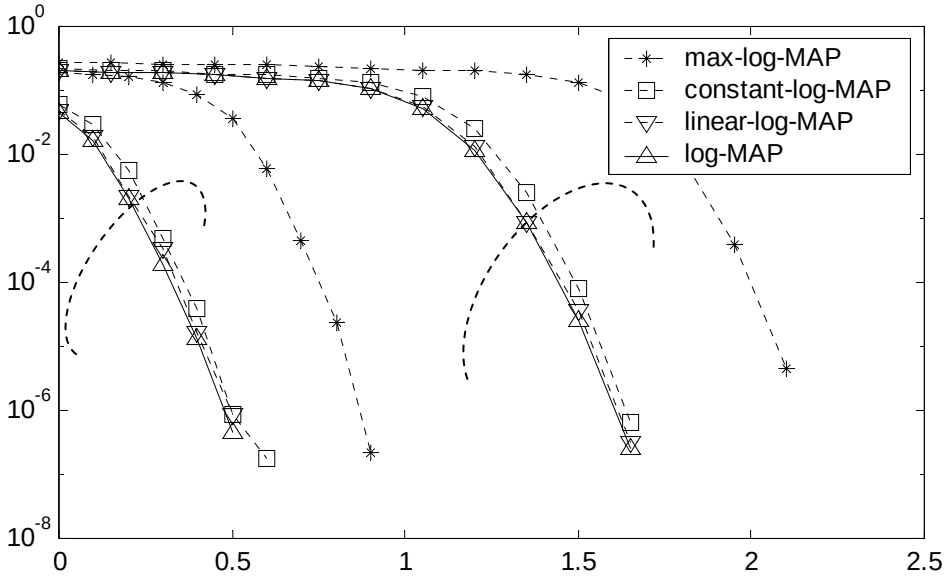
$$\Lambda(X_k) = \max^*_{S_i \to S_j : X_k = 1}\left\{\lambda_k(i,j)\right\} - \max^*_{S_i \to S_j : X_k = 0}\left\{\lambda_k(i,j)\right\},$$

where the max* operator is computed recursively over the likelihoods of all data 1 branches ($S_i \to S_j$: X_k=1) or data 0 branches ($S_i \to S_j$: X_k=0). Once $\Lambda(X_k)$ is calculated, $\alpha_{k-1}(S_i)$ is no longer needed and may be discarded.

Comparison of algorithms

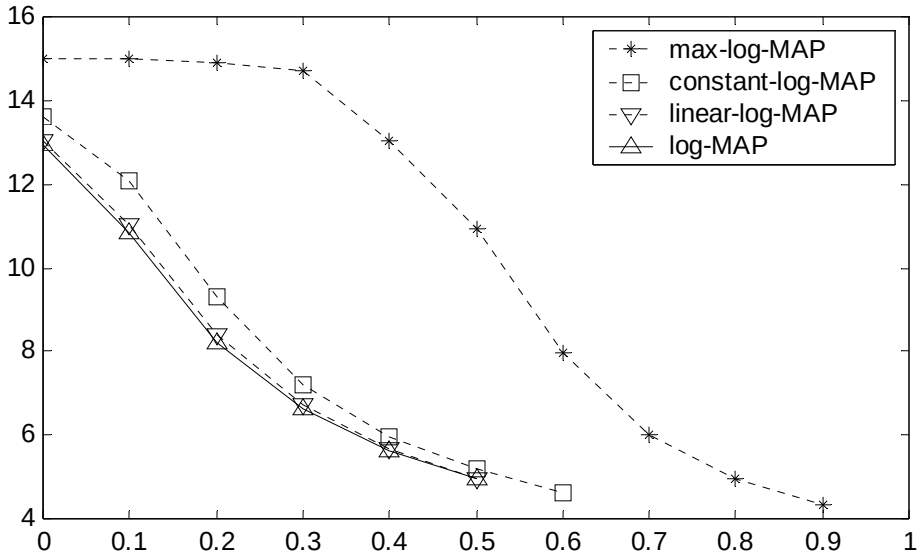
The simulated performance of the L=5114 UMTS turbo code using all four variations of the decoding algorithm is shown below for both AWGN and fading channels.

[8] *Note that α_k does not need to be computed when k = L (it is never used), although the LLR $\Lambda(X_k)$ must still be found.*



For the simulation, the frame/interleaver size of L=5114 bits was used, and up to 14 decoder iterations were performed. In all cases, the performance of max-log-MAP is noticeably worse than the other three algorithms: At BER = 10^{-5}, max-log-MAP is 0.412 dB worse than log-MAP in AWGN. The other three algorithms have roughly the same performance, although linear-log-MAP is always better than constant-log-MAP: In AWGN and at BER=10^{-5}, constant-log-MAP is 0.028 dB worse than log-MAP, while linear-log-MAP is only 0.008 dB worse than log-MAP.

The figure below shows the average number of iterations required for each of the algorithms to converge in the AWGN channel.



The figure indicates that at E_b/N_o = 0.4 dB, max-log-MAP required about 13 iterations, log-MAP about 5.63, linear-log-MAP about 5.67, and constant-log-MAP about 5.98. Thus an additional benefit of linear-log-MAP is that it requires fewer decoder iterations than constant-log-MAP.

Dynamic Halting Condition

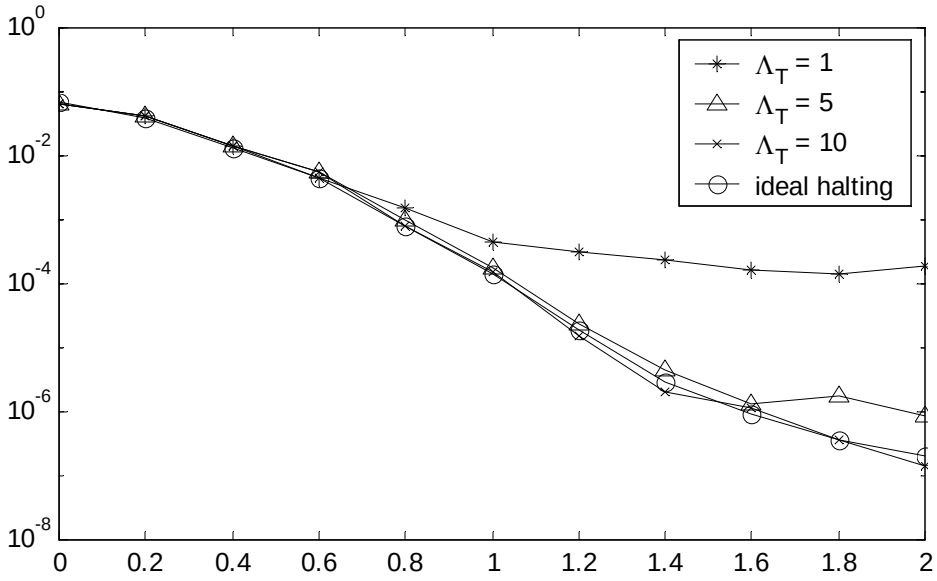
The simulation results from the previous section assumed that the decoder halted as soon as it converged, i.e. when the BER for the frame went to zero. This requires knowledge of the data, which is available when running a computer simulation. However, in practice, the decoder will not have knowledge of the data, and thus a blind method for halting the iterations must be employed. Because the decoder rarely requires the maximum number of iterations to converge, using an early stopping criterion will allow a much greater throughput in a software radio implementation.

Several other early stopping criteria have been proposed based on cross entropy between iterations or on the sign-difference ratio [WU00]. The decoder considered here uses a simpler, but effective, stopping criteria based only on the log-likelihood ratio. The decoder stops once the absolute value of all of the LLRs are above a threshold, Λ_T, i.e. the decoder halts once

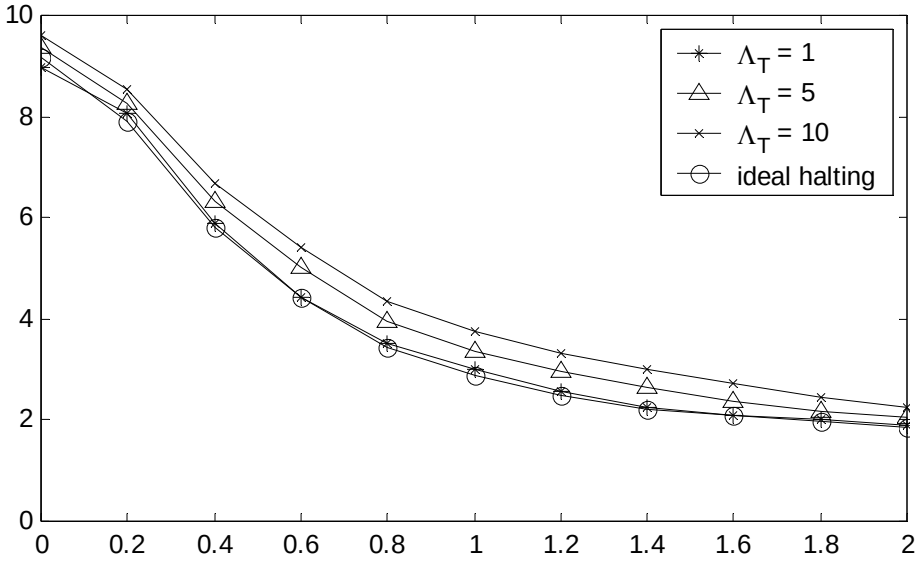
$$\min_{1 \le k \le L} \left\{ \left| \Lambda_2(X_k) \right| \right\} > \Lambda_T .$$

The performance of the stopping condition is highly dependent on the choice of Λ_T (called *Halt LLR* in Embed/Comm TC library). If it is too small, then the decoder will tend to not perform enough iterations and BER performance will suffer. If, however, it is too large, then the decoder will tend to overiterate, and the throughput will suffer. Through simulation, a value of $\Lambda_T = 10$ was found to be acceptable.

The L=640 bit UMTS turbo code was simulated in AWGN using both ideal halting (i.e. halt once the decoder converges) and halting using various values for Λ_T. The decoder used a maximum of 10 iterations of the constant-log-MAP algorithm and each curve was generated using the same received code words. BER results are shown below.



As can be seen from the figures, $\Lambda_T = 1$ and $\Lambda_T = 5$ are too small and raise the BER and FER floors. $\Lambda_T = 10$ raises the FER floor only slightly and has only a negligible effect on the BER floor. The average number of decoder iterations required to converge for various values of Λ_T is shown below.



Using the threshold $\Lambda_T = 10$ requires, on average, less than one extra iteration compared to ideal halting (for instance at Eb/No = 1.2 dB, ideal halting requires 2.5 iterations and using $\Lambda_T = 10$ requires 3.3 iterations).

It is interesting to note that the BER is sometimes lower with $\Lambda_T = 10$ than with ideal halting. The reason for this is as follows: The number of errors at the output of a turbo decoder will sometimes oscillate from one iteration to the next . If the received codeword is too corrupted to successfully decode, "ideal halting" will always run the full number of iterations; thus, the number of bit errors will be dictated by the performance at the last iteration, which due to the oscillatory nature of the decoder, could be quite high. On the other hand, the early halting decoder will stop the iterations when the LLRs are high, even if the BER is not identically zero. Thus, although early halting cannot lower the Frame Error Rate (FER), it can lower the BER by having fewer bit errors when there is a frame error.

References

[ACK99] O. Acikel and W.E. Ryan, "Punctured turbo codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, pp. 1315-1323, Sept. 1999.

[BEN96] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409-428, Mar. 1996.

[BEN98] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909-926, May 1998.

[BER93] C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc., IEEE Int. Conf. on Commun.,* (Geneva, Switzerland), May 1993, pp. 1064-1070.

[CLA00] B. Classon, K. Blankenship, and V. Desai, "Turbo decoding with the constant-log-MAP algorithm," in *Proc., Second Int. Symp. Turbo Codes and Related Application*s, (Brest, France), pp. 467-470, Sept. 2000.

[ETSI00] European Telecommunications Standards Institute, "Universal Mobile Telecommunications System (UMTS); Multiplexing and Channel Coding (FDD)" *3GPP TS 125.212 version 3.4.0,* September 23, 2000, pp. 14-20.

[FOR73] G.D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268-278, Mar. 1973.

[GRO98] W.J. Gross and P.G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders*," IEE Electronic Letters*, vol. 34, pp. 1577-1578, Aug. 6, 1998.

[PER96]L.C. Perez, J. Seghers, and D.J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1698-1708, Nov. 1996.

[ROB97] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecommun*., vol. 8, no. 2, pp. 119-125, Mar./Apr. 1997.

[ROW00] D.N. Rowitch and L.B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Trans. Commun.,* vol. 48, pp. 948-959, June 2000.

[SUN01] J. Sun and M.C. Valenti, "A linear approximation to the Jacobi logarithm and its application to turbo decoding," in Proc. *IEEE Globecom* (San Antonio, Texas), Nov. 2001.

[VAL97] M.C. Valenti and B.D. Woerner, "Variable latency turbo codes for wireless multimedia applications," in *Proc., Int. Symp. on Turbo Codes and Related Topics* (Brest, France), pp. 216-219, Sept. 1997.

[VIT98] A.J. Viterbi, "An intuitive justification and simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun*., vol. 16, no. 2, pp. 260-264, Feb. 1998.

[WIC95] S. Wicker*, Error Control Systems for Digital Communications and Storage*, Prentice Hall, 1995.

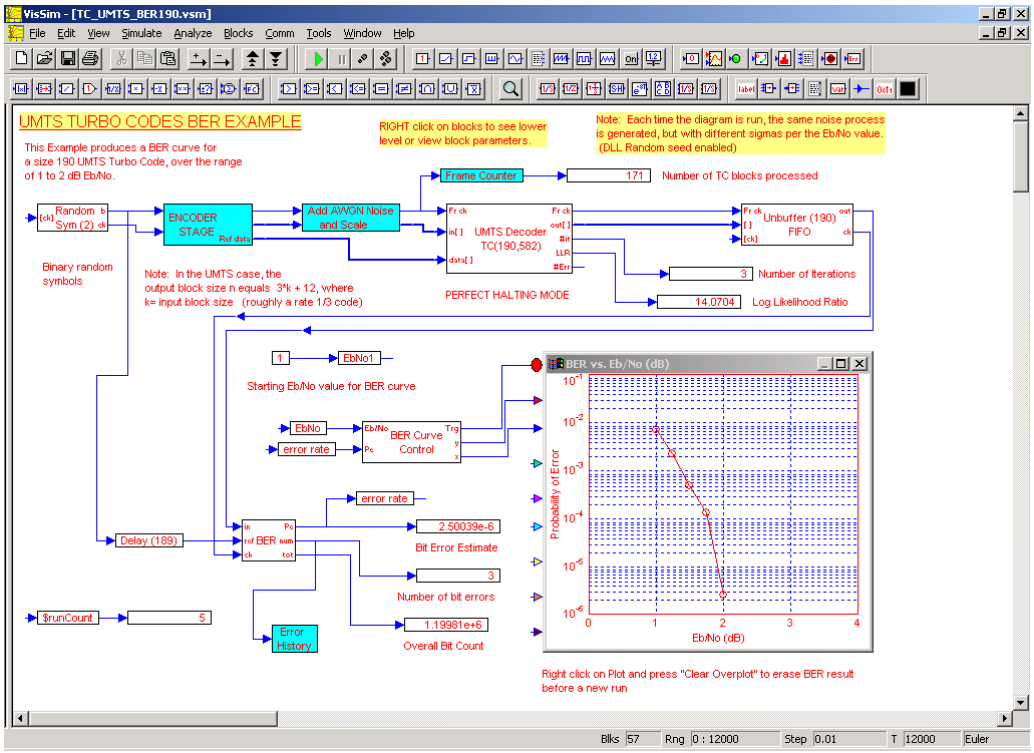
[WU00] Y. Wu, B.D. Woerner, and W.J. Ebel, "A simple stopping criterion for turbo decoding," *IEEE Commun. Letters*, vol. 4, no. 8, pp. 258-260, Aug. 2000.

Sample turbo code simulation

An example of a simulation diagram making use of turbo codes is shown below. The TC_UMTS_BER190.VSM example is located in the "TurboCodes" subdirectory of the Comm Examples folder. The actual example file may differ slightly from the diagram shown below.

In the referenced diagram, random bits are passed through a UMTS Turbo Encoder that uses a data block size of 190 bits. The encoded output after coding is a vector of size 582, including 12 termination bits. Gaussian noise is then added to each output sample using a Vector AWGN block, and its output is passed to a UMTS Turbo Decoder block. The decoder output is then compared bit by bit with the original data to obtain a Bit Error Rate (BER) curve. As demonstrated by the example, this UMTS turbo code (rate 1/3) can achieve good link performance at Eb/No levels of just a few dB.

As the simulation runs, the decoder block outputs (for each data block) the number of iterations that were required to decode the block along with the value of the lowest log likelyhood ratio (internal decoder metric).


VisSim - [TC_UMTS_BER190.vsm]
File Edit View Simulate Analyze Blocks Comm Tools Window Help
UMTS TURBO CODES BER EXAMPLE
This Example produces a BER curve for a size 190 UMTS Turbo Code, over the range of 1 to 2 dB Eb/No.
RIGHT click on blocks to see lower level or view block parameters.
Note: Each time the diagram is run, the same noise process is generated, but with different sigmas per the Eb/No value. (DLL Random seed enabled)
Frame Counter
171
Number of TC blocks processed
Random Sym (2)
Binary random symbols
ENCODER STAGE
Add AWGN Noise and Scale
UMTS Decoder TC(190,582)
PERFECT HALTING MODE
Unbuffer (190) FIFO
3
Number of Iterations
14.0704
Log Likelihood Ratio
Note: In the UMTS case, the output block size n equals 3*k + 12, where k= input block size (roughly a rate 1/3 code)
1
EbNo1
Starting Eb/No value for BER curve
EbNo
error rate
BER Curve Control
BER vs. Eb/No (dB)
Probability of Error
Eb/No (dB)
Delay (189)
BER
2.50039e-6
Bit Error Estimate
3
Number of bit errors
$runCount
5
Error History
1.19981e+6
Overall Bit Count
Right click on Plot and press "Clear Overplot" to erase BER result before a new run
Blks 57 Rng 0 : 12000 Step 0.01 T 12000 Euler

Appendix C

Wireless Overview

Bluetooth Overview

The Bluetooth standard provides wireless data and/or voice communication between multiple peripherals located in close proximity of each other (typical range is 10 meters). It can be operated in point-to-point or point-to-multipoint modes, and exhibits a channel symbol rate of 1 Mbps, with a maximum payload rate of 723 kbps. A Bluetooth network is also referred to as a "piconet".

In each piconet one Bluetooth unit acts as the master and the remaining units act as followers, with up to seven active follower units being allowed. Access to the channel is controlled by the master. A Time-Division Duplex (TDD) communication scheme is used, in which packets are exchanged between master and follower in alternating fashion.

Bluetooth operates in the 2.4 GHz ISM (Industrial Scientific Medicine) microwave band, and employs a frequency hopping scheme spanning 79 channels (1 MHz spacing) and with a hopping rate of 1.6 kHz. The pseudo-random hopping pattern is derived from a combination of the master clock time and the unique device address of each unit.

The modulation used in Bluetooth is Gaussian Frequency Shift Keying (GFSK) with a BT value of 0.5. A "1" is represented by a positive frequency shift and a "0" is represented by a negative frequency shift. The modulation index must be in the range of 0.28 – 0.35, which corresponds to a frequency deviation of 140 ~ 175 kHz.

The Bluetooth channel is organized into time slots of 625 us in duration. The start of each packet must be aligned with the beginning of a new time slot, and a packet may extend for up to five time slots in duration. The master always starts a transmission in even-numbered time slots, while the follower always starts its transmissions in odd-numbered time slots. Each packet is transmitted on a different hop frequency.

For more details on the Bluetooth specification, please visit the official Bluetooth web site at:

www.bluetooth.com

802.11 Overview

The IEEE 802.11 standard was developed to provide Local Area Network (LAN) services in a wireless communications environment. The standard supports operation in the 2.4 GHz ISM band using either a Frequency Hopped Spread Spectrum (FHSS) approach or Direct Sequence Spread Spectrum (DSSS).

The FHSS format uses binary or 4-ary GFSK modulation (BT= 0.5) and supports data rates of 1 Mbps and 2 Mbps respectively. In the US, the set of operating transmit and receive channels for

the FHSS specification includes 79 frequencies, with center frequencies ranging from 2402 MHz to 2480 MHz. Channels are spaced at 1 MHz intervals. In the US, the FHSS hopping patterns are grouped into 3 sets, each including 26 patterns.

The DSSS format also provides 1 and 2 Mbps data rates, but employs differential PSK baseband modulation formats. The 1 Mbps mode uses Differential Binary Phase Shift Keying (DBPSK), and the 2 Mbps mode uses Differential Quadrature Phase Shift Keying (DQPSK). The modulated baseband signals (both 1 Msps) are then spread using a chip rate of 11 MHz (i.e. there are 11 chips per symbol). The symbols are spread using an 11-chip Barker sequence, whose start time is aligned with the start of each symbol. In the US, 11 operating channels are available for the 802.11 DSSS specification, with center frequencies ranging from 2412 to 2462 MHz.

More details on the 802.11 standard may be obtained directly from the IEEE. The 802.11 specification is available for free download from the following URL:

http://standards.ieee.org/getieee802/

802.11a/g Overview

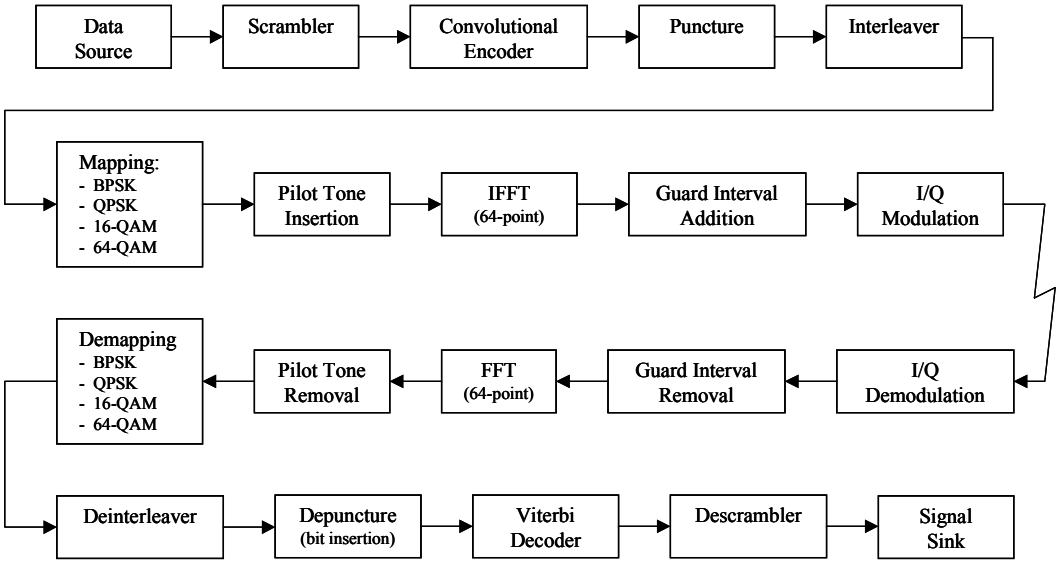
The IEEE 802.11a standard is an extension to the original 802.11 standard devised to significantly increase the system data rate. Unlike the 802.11b extension, 802.11a is not compatible with the original 802.11 frequency plan (2.4 GHz band), and operates instead in the 5 ~ 6 GHz band.

The 802.11a standard uses Orthogonal Frequency Division Multiplexing (OFDM) modulation and supports data rates ranging from 6 Mbps to 54 Mbps. The OFDM modulation uses 52 subcarriers that are modulated using BPSK, QPSK, 16-QAM, or 64-QAM. Convolutional coding is used for Forward Error Correction (FEC), at coding rates of 1/2, 2/3, or 3/4.

The newer 802.11g standard, combines the higher data rate capability of 802.11a while operating in the same 2.4 GHz band as 802.11b. It specifies the use of either PBCC modulation (not currently supported by the Wireless module) or OFDM as defined in the 802.11a specification.

In the US, the set of operating frequencies for the 802.11a OFDM specification includes 12 channels, with center frequencies ranging from 5180 to 5805 MHz.

A simplified block diagram of an 802.11a/g OFDM modulation physical layer link is shown in the following figure.



Simplified 802.11a/g Physical Layer Block Diagram

More details on the 802.11 standard may be obtained directly from the IEEE. The 802.11a and 802.11g specifications are available for free download from the following URL:

http://standards.ieee.org/getieee802/

802.11b Overview

The IEEE 802.11b standard provides a high speed physical layer extension to the original 802.11 DSSS specification. It operates in the 2.4 GHz ISM band, and remains compatible with the original 802.11 frequency plan (see above).

The 802.11b standard extends the capabilities of 802.11 by adding Complementary Code Keying (CCK) modulation at rates of 5.5 and 11 Mbps. As with 802.11, a chipping rate of 11 MHz is used. For compatibility purposes, the packet's preamble and headers are still transmitted using DBPSK and DQPSK at 1 Mbps and 2 Mbps.

An optional Packet Binary Convolutional Coding (PBCC) mode, providing a data rate of 5.5 Mbps or 11 Mbps, is also specified in the standard. The PBCC mode is not yet included in the wireless module, but is planned for future releases.

The 802.11b standard also includes an optional Channel Agility mode, in which the signal can frequency-hop across a subset of the available DSSS channels.

More details on the 802.11b specification may be obtained directly from the IEEE. The 802.11b specification is available for free download from the following URL:

http://standards.ieee.org/getieee802/

Ultrawideband Overview

Ultrawideband (UWB) signals are defined by the FCC as waveforms having a fractional bandwidth greater than 20% or occupying a bandwidth of more than 500 MHz. UWB signals typically consist of very short time duration pulses in the picosecond to nanosecond range, and are transmitted directly at RF without the use of a modulating carrier. Typical bandwidths exceed several GHz.

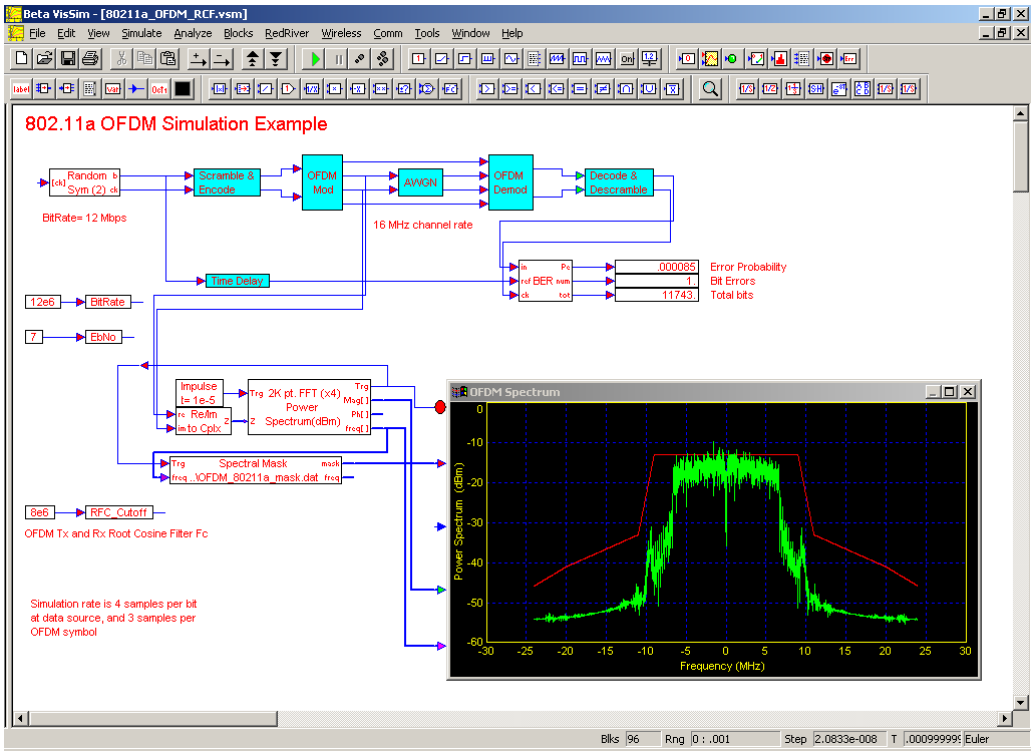
Common applications include short-range indoor communications. One of the advantages of UWB schemes is that they are capable of operating in heavy multipath environments due to the extremely short duration of the signaling pulses.

Sample Wireless simulation

An example of a communication system simulation using 802.11a components is shown below. The example diagram "80211a_OFDM_RCF.vsm" is located in the "Wireless" folder. The actual example file may differ slightly from the diagram that follows.

This example focuses on the 802.11a OFDM modulation format. The diagram displays a modulated OFDM spectrum for a 12 Mbps link along with the corresponding FCC mask. The diagram includes data scrambling, encoding, subcarrier QPSK modulation, pilot tone insertion and the OFDM modulation process via inverse FFT. The diagram also computes Bit Error Rate (BER) performance data assuming a Gaussian noise channel.

For more technical details on the OFDM modulation format, please refer to the `OFDM Modulator` block description in Chapter 3 (802.11a Section).


Beta VisSim - [80211a_OFDM_RCF.vsm]
File Edit View Simulate Analyze Blocks RedRiver Wireless Comm Tools Window Help
802.11a OFDM Simulation Example
Random Sym (2)
Scramble & Encode
OFDM Mod
AWGN
OFDM Demod
Decode & Descramble
BitRate= 12 Mbps
16 MHz channel rate
Time Delay
BER
.000085
1.
11743.
Error Probability
Bit Errors
Total bits
12e6
BitRate
7
EbNo
Impulse t= 1e-5
Re/Im to Cplx
2K pt. FFT (x4) Power Spectrum(dBm)
Spectral Mask
..\OFDM_80211a_mask.dat
8e6
RFC_Cutoff
OFDM Tx and Rx Root Cosine Filter Fc
Simulation rate is 4 samples per bit at data source, and 3 samples per OFDM symbol
OFDM Spectrum
Power Spectrum (dBm)
Frequency (MHz)
Blks 96 Rng 0 : .001 Step 2.0833e-008 T .000999999 Euler

Appendix D

Acronyms and Abbreviations

Term	Definition
16-PSK	16-phase shift keying
16-QAM	16-level quadrature amplitude modulation
256-QAM	256-level quadrature amplitude modulation
32-QAM	32-cross quadrature amplitude modulation
64-QAM	64-level quadrature amplitude modulation
8-PSK	eight-phase shift keying
ADC	analog-to-digital converter
AGC	automatic gain control
AM	amplitude modulation
ASK	amplitude shift keying
AWGN	additive white Gaussian noise
BER	bit error rate
BPSK	binary phase shift keying
BSC	binary symmetric channel
DOS	disk operating system
DPSK	differential phase shift keying
DQPSK	differential quadrature phase shift keying
DSB-AM	double-sideband amplitude modulation
DTTL	data transition tracking loop
FIFO	first in first out
FIR	finite impulse response
FM	frequency modulation
FSK	frequency shift keying
IIR	infinite impulse response
IM	intermodulation

ISI	inter-symbol inference
LIFO	last in first out
LSB	least significant bit
MSB	most significant bit
MSK	minimum shift keying
OQPSK	offset quadrature phase shift keying
PAM	pulse amplitude modulation
PLL	phase-locked loop
PM	phase modulation
PN	pseudo noise
PPM	pulse position modulation
PRBS	pseudo random binary sequence
QAM	quadrature amplitude modulation
QPSK	quadrature phase shift keying
SER	symbol error rate
SNR	signal to noise ratio
SQPSK	staggered quadrature phase shift keying
TWTA	traveling wave tube amplifier
VCO	voltage controlled oscillator
VCPG	voltage controlled pulse generator

Index

M

N

O

P

Q

R

S

W