Eritek, Inc.
Altair Engineering, Inc.

VisSim/Comm User’s Guide - Version 9.0

Copyright © 2014 Eritek, Inc. All rights reserved. Eritek, Inc. 18450 Sydnor Hill Ct. Leesburg, VA 20175 Altair Engineering, Inc. 487 Groton Road Westford, MA 01886

Trademarks VisSim is a trademark of Altair Engineering. VisSim/Comm is a joint trademark of Eritek and Altair Engineering. Other products mentioned in this manual are trademarks of their respective manufacturers.

Copy and use restrictions The information in this document is subject to change without notice and does not represent a commitment by either Eritek or Altair Engineering. Eritek and Altair Engineering do not assume responsibility for errors that may appear in this document.

No part of this manual may be reprinted or reproduced or utilized in any form or by any electronic, mechanical, or other means without permission in writing from Eritek. The Software may not be copied or reproduced in any form, except as stated in the terms of the Software license agreement.

Use, duplication, or disclosure by the US Government is subject to restrictions as set forth in FAR 52.227-19, subparagraph (c)(i)(ii) of DOD FAR SUPP 252.227-7013, or equivalent government clause for other agencies.

Portions of the Turbo Code library Copyright 2000 by Matt Valenti.
Contents

Preface ............................................................................................................................... xi
   Online VisSim/Comm documentation ............................................................................ xi
   Conventions used in this manual ................................................................................. xi
   For VisSim/Comm Personal Edition (PE) users ......................................................... xii
   Technical support service ............................................................................................. xiii

Introduction ...................................................................................................................... 1
   A typical communication system ................................................................................. 1
   Lowpass equivalent systems ....................................................................................... 3
   Communication blocks .............................................................................................. 3

VisSim/Comm Overview .................................................................................................. 7
   Starting VisSim/Comm ................................................................................................. 7
   The VisSim/Comm environment .................................................................................. 7
      Inserting blocks ....................................................................................................... 8
      Connecting blocks ................................................................................................. 8
      Block connectors ................................................................................................... 8
      Diagram timing ...................................................................................................... 9
      Random numbers .................................................................................................. 9
      Setting up block parameters ............................................................................... 10
      Global variables .................................................................................................. 10
      Choosing an integration method .......................................................................... 11
   Multirate diagrams .................................................................................................... 11
      Configuring compound blocks for local rate mode ............................................ 12
   Output plots ............................................................................................................... 12
      BER curves ........................................................................................................ 12
      Eye plots ............................................................................................................ 13
      Frequency domain plots .................................................................................... 13
      Phase scatter plots ............................................................................................. 14
   Filter Viewer .............................................................................................................. 15
   Generating BER curves ............................................................................................ 16
      Enabling multiple consecutive runs ..................................................................... 17
      Varying the SNR for each run .............................................................................. 17
      Calculating BER measurements ...................................................................... 17
   Sample communication simulation ......................................................................... 19

Comm Block Set ............................................................................................................. 21
   Channels category .................................................................................................... 21
      AWGN (Complex or Real) .................................................................................. 21
      Binary Symmetric Channel ............................................................................... 22
      Jakes Mobile ..................................................................................................... 23
Mobile Fading ................................................................. 23
Multipath ........................................................................... 24
Propagation Loss .............................................................. 25
Rice/Rayleigh Fading ....................................................... 26
Rummler Multipath ......................................................... 26
Saleh-Valenzuela ............................................................. 27
TWTA (Analytical) .......................................................... 29
TWTA (Table Lookup) ..................................................... 31
Vector AWGN ............................................................... 31
Complex Math category ................................................... 32
  Complex Addition .......................................................... 32
  Complex Conjugate ....................................................... 32
  Complex Division .......................................................... 32
  Complex Inverse ............................................................ 32
  Complex Multiplication .................................................. 33
  Complex Power .............................................................. 33
  Complex Square Root ..................................................... 33
  Complex to Mag/Phase .................................................... 33
  Complex to Real/Img ....................................................... 33
  Mag/Phase to Complex ................................................... 34
  Real/Img to Complex ....................................................... 34
Demodulators category .................................................... 34
  Differential PSK Detector ............................................. 34
  FM Demodulator .......................................................... 35
  IQ Detector ................................................................. 35
  PPM Demodulator ....................................................... 36
  PSK Detector ............................................................... 37
  QAM/PAM Detector ..................................................... 37
Digital category .............................................................. 38
  Accumulate & Dump ..................................................... 38
  Binary Counter ........................................................... 39
  Bits to Symbol ............................................................ 39
  Buffer ................................................................. 40
  D Flip Flop ............................................................... 41
  Divide by N ............................................................... 41
  JK Flip Flop ............................................................... 42
  Mux/Demux .............................................................. 43
  Packet Timing ............................................................ 43
  Parallel to Serial .......................................................... 45
  Pulse Extend ............................................................. 45
  Queue ................................................................. 46
  Serial to Parallel ......................................................... 46
  State Machine ........................................................... 47
  Symbol to Bits ........................................................... 48
Unbuffer ............................................................................................................. 49
Encode / Decode category ............................................................................................ 49
Block Interleaver ........................................................................................................... 49
Convolutional Encoder............................................................................................... 50
Convolutional Interleaver ............................................................................................ 51
Depuncture .................................................................................................................. 52
Gray Map ................................................................................................................... 53
Gray Reverse Map ...................................................................................................... 53
Hamming Decoder ..................................................................................................... 53
Hamming Encoder ....................................................................................................... 54
Manchester Encoder ................................................................................................. 54
Puncture ..................................................................................................................... 55
Reed-Solomon Decoder ............................................................................................... 57
Reed-Solomon Encoder ............................................................................................... 58
Trellis Decoder .......................................................................................................... 60
Trellis Encoder .......................................................................................................... 60
Viterbi Decoder (Hard) ............................................................................................... 62
Viterbi Decoder (Soft) ................................................................................................. 62
Estimators category ...................................................................................................... 64
Average Power (Complex or Real) ............................................................................ 64
BER Control (# Errors) ............................................................................................. 65
BER Curve Control .................................................................................................... 66
Bit/Symbol Error Rate ............................................................................................... 67
Correlation ................................................................................................................ 68
Delay Estimator ......................................................................................................... 69
Event Time ................................................................................................................ 69
File Correlation ......................................................................................................... 69
Frequency Counter .................................................................................................... 71
Mean ......................................................................................................................... 71
Median ....................................................................................................................... 72
MinMax ...................................................................................................................... 72
Variance ..................................................................................................................... 73
Vector Correlation .................................................................................................... 73
Weighted Mean ......................................................................................................... 74
Filters category ......................................................................................................... 74
Adaptive Equalizer (Complex or Real) ................................................................... 74
Discrete Equalizer (Complex or Real) ....................................................................... 77
File FIR Filter .......................................................................................................... 79
FIR Filter .................................................................................................................. 80
IIR Filter ................................................................................................................... 81
MagPhase Filter ....................................................................................................... 82
Pulse Shaping Filter ................................................................................................. 84
Sampling File FIR Filter ............................................................................................. 85
Sampling FIR Filter ................................................................................................. 86
<table>
<thead>
<tr>
<th>Variable Spaced Equalizer (Complex or Real)</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Point category</td>
<td>91</td>
</tr>
<tr>
<td>Fixed Point FIR Filter</td>
<td>91</td>
</tr>
<tr>
<td>Fixed Point IIR Filter</td>
<td>94</td>
</tr>
<tr>
<td>Fixed Point VCO (Complex or Real)</td>
<td>96</td>
</tr>
<tr>
<td>Instruments category</td>
<td>97</td>
</tr>
<tr>
<td>BER Curve Display</td>
<td>97</td>
</tr>
<tr>
<td>Oscilloscope Display</td>
<td>97</td>
</tr>
<tr>
<td>Spectrum Analyzer Display (Complex)</td>
<td>97</td>
</tr>
<tr>
<td>Spectrum Analyzer Display (Real)</td>
<td>97</td>
</tr>
<tr>
<td>Modulators categories - Complex and Real</td>
<td>97</td>
</tr>
<tr>
<td>AM Modulator</td>
<td>98</td>
</tr>
<tr>
<td>ASK Modulator</td>
<td>98</td>
</tr>
<tr>
<td>Differential PSK Modulator</td>
<td>99</td>
</tr>
<tr>
<td>FM Modulator</td>
<td>102</td>
</tr>
<tr>
<td>FSK Modulator</td>
<td>102</td>
</tr>
<tr>
<td>GFSK Modulator</td>
<td>103</td>
</tr>
<tr>
<td>GMSK Modulator</td>
<td>103</td>
</tr>
<tr>
<td>IQ Modulator</td>
<td>104</td>
</tr>
<tr>
<td>MSK Modulator</td>
<td>104</td>
</tr>
<tr>
<td>PM Modulator</td>
<td>105</td>
</tr>
<tr>
<td>PPM Modulator</td>
<td>106</td>
</tr>
<tr>
<td>PSK Modulator</td>
<td>107</td>
</tr>
<tr>
<td>BPSK</td>
<td>109</td>
</tr>
<tr>
<td>QPSK</td>
<td>109</td>
</tr>
<tr>
<td>8-PSK</td>
<td>110</td>
</tr>
<tr>
<td>16-PSK</td>
<td>110</td>
</tr>
<tr>
<td>32-PSK</td>
<td>110</td>
</tr>
<tr>
<td>QAM/PAM Modulator</td>
<td>111</td>
</tr>
<tr>
<td>16-QAM</td>
<td>113</td>
</tr>
<tr>
<td>32-QAM</td>
<td>113</td>
</tr>
<tr>
<td>64-QAM</td>
<td>113</td>
</tr>
<tr>
<td>128-QAM</td>
<td>114</td>
</tr>
<tr>
<td>256-QAM</td>
<td>114</td>
</tr>
<tr>
<td>4-PAM</td>
<td>114</td>
</tr>
<tr>
<td>8-PAM</td>
<td>115</td>
</tr>
<tr>
<td>16-PAM</td>
<td>115</td>
</tr>
<tr>
<td>SQPSK Modulator</td>
<td>115</td>
</tr>
<tr>
<td>Multirate Support category</td>
<td>117</td>
</tr>
<tr>
<td>Clock Edge</td>
<td>117</td>
</tr>
<tr>
<td>Clock Extend</td>
<td>117</td>
</tr>
<tr>
<td>Interpolator</td>
<td>117</td>
</tr>
<tr>
<td>Operator category</td>
<td>118</td>
</tr>
</tbody>
</table>
### Contents

- A/D Converter ................................................................. 118
- Comander ............................................................................. 119
- Complex Exponential .......................................................... 120
- Complex FFT/IFFT ............................................................. 120
- Conversions ........................................................................... 122
- D/A Converter ................................................................. 124
- Delay (Complex) ............................................................... 124
- Delay (Real) .......................................................................... 125
- Gain (dB) ................................................................................ 125
- Integrate & Dump (Complex or Real) ................................. 126
- IQ Mapper .............................................................................. 127
- Max Index .............................................................................. 128
- Modulo ................................................................................... 128
- Oscilloscope ........................................................................... 129
- Phase Rotate .......................................................................... 130
- Phase Unwrap ........................................................................ 130
- Polynomial ............................................................................. 130
- Spectrum (Complex or Real) ............................................... 131
- Subsample ........................................................................... 133
- Vector FFT ............................................................................ 133

#### PLL category

- Charge Pump ......................................................................... 134
- Loop Filter (2nd Order PLL) .................................................. 135
- Loop Filter (3rd Order PLL) .................................................. 137
- Type-2 Phase Detector ......................................................... 137
- Type-3 Phase Detector ......................................................... 138
- Type-4 Phase Detector ......................................................... 138

#### RF category

- Amplifier .............................................................................. 139
- Antenna ................................................................................ 140
- Attenuator ........................................................................... 141
- Cable ..................................................................................... 141
- Coupler ................................................................................ 142
- Double Balanced Mixer ....................................................... 143
- RF Conversions .................................................................... 144
- RF Gain ............................................................................... 145
- Splitter/Combiner ............................................................... 146
- Switch .................................................................................. 147
- Variable Attenuator ............................................................ 148

#### Signal Sinks category

- File Write ............................................................................. 148
- Final Value ........................................................................... 150
- Wave Write ........................................................................... 150

#### Signal Sources category

- ......................................................................................... 151
Contents

Complex Tone .................................................................................................. 151
File Data .......................................................................................................... 152
Frequency Sweep ............................................................................................. 153
Impulse ............................................................................................................. 154
Impulse Train ................................................................................................... 154
Noise ................................................................................................................ 154
PN Sequence .................................................................................................... 155
Poisson Arrivals ............................................................................................... 157
Random Distribution ........................................................................................ 157
Random Seed (Obsolete) .................................................................................. 158
Random Symbols ............................................................................................. 159
Rectangular Pulses ........................................................................................... 159
Sinusoid ............................................................................................................ 160
Spectral Mask ................................................................................................... 161
Vector Constant ................................................................................................ 162
VCO (Complex or Real) ................................................................................... 162
Walsh Sequence ............................................................................................... 163
Wave Data ........................................................................................................ 164
Waveform Generator ........................................................................................ 165

Turbo Codes category ............................................................................................. 166
LTE Turbo Decoder ......................................................................................... 166
LTE Turbo Encoder ......................................................................................... 167
TC Interleaver Generator .................................................................................. 168
Turbo Code Decoder ........................................................................................ 169
Turbo Code Encoder ....................................................................................... 172
UMTS Turbo Decoder...................................................................................... 174
UMTS Turbo Encoder ..................................................................................... 175

Vector Operators category .................................................................................. 176
Matrix to Vector ............................................................................................... 176
SubVector ......................................................................................................... 176
Vector Bits to Symbol ...................................................................................... 176
Vector Demux .................................................................................................. 177
Vector Merge ................................................................................................... 178
Vector Mux ...................................................................................................... 178
Vector Symbol to Bits ...................................................................................... 178
Vector to Matrix ............................................................................................... 179

VisSim/Comm Library ....................................................................................... 181
Compound Blocks .............................................................................................. 181
COSTAS_C.VSM ............................................................................................ 181
COSTAS_R.VSM ............................................................................................ 181
GFSK.VSM ...................................................................................................... 182
GMSK.VSM ..................................................................................................... 182
PLL1CPLX.VSM ............................................................................................. 182
Contents

PLL1REAL.VSM ................................................................. 182
PLL2CPLX.VSM ................................................................. 182
PLL2REAL.VSM ................................................................. 183
TWTA_TBL.VSM ................................................................. 183
V32DIFDE.VSM ................................................................. 183
V32DIFEN.VSM ................................................................. 183
VCPG.VSM ............................................................................. 184

Data Files .................................................................................. 184
AMAM.DAT ................................................................................. 184
AMPM.DAT ................................................................................ 184
DATA_IN.DAT ............................................................................. 184
PSK_GRAY.DAT ......................................................................... 184
QAM_GRAY.DAT ......................................................................... 184
TABLFILT.DAT ........................................................................... 184
V32QAM.DAT ............................................................................... 185
V32TRELS.DAT ........................................................................... 185
VTB3SOFT.DAT ........................................................................... 185

Turbo Codes Overview ............................................................ 187
Origins of turbo codes ............................................................... 187
Convolutional codes .................................................................. 187
Turbo encoding .......................................................................... 189
Turbo Code performance factors .................................................. 189
The UMTS turbo code .................................................................. 193
Turbo decoding ............................................................................. 193
Channel model ........................................................................... 193
Decoder architecture ................................................................. 194
The max* function ..................................................................... 194
Log-MAP algorithm .................................................................... 195
Max-log-MAP algorithm .......................................................... 195
Constant-log-MAP algorithm .................................................... 196
Linear-log-MAP algorithm ........................................................ 196
MAP algorithm in log-domain .................................................... 196
Trellis structure and branch metrics ............................................. 196
Backward recursion .................................................................... 197
Forward recursion and LLR calculation ...................................... 198
Comparison of algorithms ......................................................... 198
Dynamic Halting Condition ....................................................... 200

References ................................................................................. 201
Sample turbo code simulation .................................................... 202

Acronyms and Abbreviations ....................................................... 205
Installing VisSim/Comm ............................................................ 207
Welcome to version 9.0 of VisSim/Comm. VisSim/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 VisSim/Comm to decrease design cycle time, minimize hardware prototyping, and build better products.

VisSim/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.

VisSim/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 VisSim/Comm.

This manual assumes that you are already familiar with the use of the VisSim simulation environment. Please consult the VisSim User’s Guide for more details on the core VisSim environment.

If you’ve purchased the VisSim/Comm Personal Edition (PE), be sure to read the related section later in this chapter to learn about program functionality limitations associated with this version.

Online VisSim/Comm documentation

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

- **Online help.** Extensive online information is provided for the VisSim simulation environment from the Help menu. For online information on VisSim/Comm, use the VisSim/Comm Help command under the Comm menu. Online help for a particular block can also be accessed by clicking on the Help button in the Setup dialog box for the block.

- **Online release notes.** A VisSim/Comm readme file (COMMREAD.WRI) is installed in your VisSim directory. This file contains last minute information on changes that were made after this manual went to print. For your convenience, you should read this file immediately and print a copy of it to keep with this manual.

You may also find it helpful to browse through the sample block diagrams included with VisSim/Comm. These diagrams, which are listed in Appendix B, “Sample Block Diagrams,” demonstrate how VisSim/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 VisSim graphical user interface. If you need to review the interface, consult your VisSim 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
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:

<table>
<thead>
<tr>
<th>Visual convention</th>
<th>Where it's used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italics</td>
<td>To reference a book, chapter, or section. Also used to emphasize certain keywords.</td>
</tr>
<tr>
<td>SMALL CAPS</td>
<td>To indicate the names of keys on the keyboard.</td>
</tr>
<tr>
<td>Shortcut key combinations</td>
<td>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.</td>
</tr>
<tr>
<td>ALL CAPS</td>
<td>To indicate directory names, filenames, and acronyms.</td>
</tr>
<tr>
<td>Initial Caps</td>
<td>To indicate menu names, command names, and dialog box options.</td>
</tr>
</tbody>
</table>

**For VisSim/Comm Personal Edition (PE) users**

If you've purchased VisSim/Comm Personal Edition (PE), your software will have the following limitations:

- Block diagrams in excess of 100 blocks cannot be saved.
- The `userFunction` block and `embed` block are not available.
- The following communication blocks are not available in the Comm menu:

**Channels:**
- Jakes Mobile
- Mobile Fading
- Propagation Loss
- Saleh-Valenzuela

**Demodulators:**
- DPSK Detector (8,16,32 not available), IQ Detector, PSK Detector (8,16,32 n/a), QAM Detector (64,128,256 n/a)

**Digital Elements:**
- Mux/Demux
- Packet Timing
- Queue
- State Machine

**Encoders/Decoders:**
- Convolutional Interleaver, Depuncture, Puncture, Trellis Encoder and Decoder

**Estimators:**
- Correlation, File Correlation, Median, Vector Correlation, Weighted Mean

**Filters:**
- Discrete Equalizer, MagPhase, Sampled FIR, Sampled File FIR, Variable Spaced Equalizer

**Fixed Point:**
- Fixed Point FIR, Fixed Point IIR, Fixed Point VCO

**Modulators:**
- DPSK (8,16,32 not available), IQ, PSK (8,16,32 n/a), QAM (64,128,256 n/a)

**Multirate:**
Clock Edge, Clock Extend, Interpolator

Operators:
IQ Mapper, Phase Unwrapper

PLL Blocks:
Loop Filter (3rd Order PLL), Type 4 Phase/Frequency Detector

RF Components:
Antenna, Cable, Double Balanced Mixer, Variable Attenuator

Signal Sources:
Poisson Arrivals, Spectral Mask, Walsh Sequence

Technical support service

When you need assistance for this product, first look in the manual, review the readme file, and consult the online Help program. If you cannot find the answer, contact the Technical Support group via toll call between 9:00 am and 6:00 pm EST, Monday through Friday, excluding holidays. The phone number is 978-392-0100.

When you call in, please have the following information at hand:

- The version of VisSim, VisSim/Comm DLL and Windows that you're using
  (Click on “Help/About VisSim…” and “Help/About VisSim/Comm Module…” to obtain this information).
- Your VisSim/Comm serial number.
- The type of hardware that you're using.
- All screen messages.
- What you were doing when the problem happened.
- How you tried to solve the problem.

Support may also be reached online at www.vissim.com or via the following fax and e-mail addresses:

<table>
<thead>
<tr>
<th>Address/Number</th>
<th>What it's for</th>
</tr>
</thead>
<tbody>
<tr>
<td>(978) 692-3102</td>
<td>Fax number</td>
</tr>
<tr>
<td><a href="mailto:bugs@vissol.com">bugs@vissol.com</a></td>
<td>Bug report</td>
</tr>
<tr>
<td><a href="mailto:doc@vissol.com">doc@vissol.com</a></td>
<td>Documentation errors and suggestions</td>
</tr>
<tr>
<td><a href="mailto:sales@vissol.com">sales@vissol.com</a></td>
<td>Sales, pricing, and general information</td>
</tr>
<tr>
<td><a href="mailto:tech@vissol.com">tech@vissol.com</a></td>
<td>Technical support</td>
</tr>
</tbody>
</table>
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 be 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 $\omega_c$, and can be represented as

$$s(t) = a(t)\cos[\omega_c t + \phi(t)] \quad (2.1)$$

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

$$s(t) = a(t)\cos(\omega_c t) - a(t)\sin(\omega_c t)$$

$$s(t) = x(t)\cos(\omega_c t) - y(t)\sin(\omega_c t) \quad (2.2)$$

$$x(t) = a(t)\cos\phi(t)$$

$$y(t) = a(t)\sin\phi(t)$$

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

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

$$s(t) = \text{Re}[u(t)e^{j\omega_c t}] \quad (2.3)$$

$$u(t) = a(t)e^{j\phi(t)} = x(t) + jy(t)$$

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, VisSim/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 VisSim/CommPE, 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; 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; FLR; 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; I Q*; MSK; PM; PPM (Real only); PSK**; QAM/PAM**; SQPSK.
Chapter 1  Introduction

Multirate Support

Clock Edge*; Clock Extend*; Interpolator*.

Operators

A/D Converter; Compressor; 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*; 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

VisSim/Comm Overview

This chapter provides information on:

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

Starting VisSim/Comm

To start VisSim/Comm from the desktop

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

![VisSim/Comm environment window]

The VisSim/Comm environment

The VisSim/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. VisSim/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. VisSim/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.

![QPSK Modulator Phase Complex Output](image)

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 VisSim/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 VisSim 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 VisSim 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:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 VisSim constant.
All VisSim/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 VisSim/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 VisSim transferFunction block and Comm Filter blocks are not affected by the integration method you choose.

### Multirate diagrams

The VisSim/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 **Interpolation** 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 VisSim/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.

**Note:** The Multirate Support blocks are not available in the PE/LE versions.
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 CTRL 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 VisSim 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 $E_b/N_0$, 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 ($E_b/N_0$).
   - 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_0).

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.

![Eye plot example](image)

**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 VisSim/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, and Vector Ops.

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

If you are using the Personal Edition (PE) of VisSim/Comm, please refer to the Preface section for the list of communication blocks not supported in the PE version of the software.

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 $E_s/N_0$, as opposed to $E_b/N_0$.

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)
\[ y_2 = \text{Current run’s } \frac{E_s}{N_0} \text{ value} \]

**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 \( \frac{E_s}{N_0} \)**

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)

\[ \begin{array}{c}
0 \\
1 \\
\end{array} \begin{array}{c}
1-p \\
p \\
1-p \\
\end{array} \begin{array}{c}
0 \\
1 \\
\end{array} \]

**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 $1.5 \times \text{#terms} \times 1/f_m$ seconds.

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

$x = \text{Complex input signal } [\text{Re}, \text{Im}]$

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

$N = 2(N_0 + 1)$

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

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

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

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 ($\omega_m$) 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 $\approx 8x$ 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.

\[ x = \text{Complex input signal } [\text{Re}, \text{Im}] \]
\[ y = \text{Complex output signal } [\text{Re}, \text{Im}] \]

\[ y(t) = h(t) * x(t) \]
\[ H(f) = \frac{1}{2\pi f_m} \left[ \frac{1}{1 - \left( \frac{f}{f_m} \right)^2} \right] |f| \leq f_m ; H(f) = 0 \text{ otherwise} \]

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.

\[ x = \text{Complex input signal } [\text{Re}, \text{Im}] \]
\[ y = \text{Complex output signal } [\text{Re}, \text{Im}] \]

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

where \( \tau_k \) = delay of path \( k \)
\( \alpha_k \) = relative weight for path \( k \)
\( \phi_k \) = 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.

\[ y(t) = x(t) \frac{c}{4\pi f d} \quad \text{where } c = \text{speed of light} \]

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.

\[ y(t) = Ax(t) \cdot \left[ \alpha + \beta(t)e^{-j\Phi(t)} \right] \]
\( \beta(t) \) is Rayleigh distributed
\( \beta(t) = \sqrt{u_1(t)^2 + u_2(t)^2} \) and \( u_1(t), u_2(t) \) are \( N(0, \sigma^2) \)
\( \Phi(t) \) is uniform over \((-\pi, \pi)\)
\( \alpha^2 + 2\sigma^2 = 1 \)
\( r = \frac{\alpha^2}{2\sigma^2} \)
\( A = (10)^{\frac{-L_{\text{dB}}}{20}} \)

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_{\text{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.

\[ x = \text{Complex input signal } [\text{Re}, \text{Im}] \]
\[ y = \text{Complex output Signal } [\text{Re}, \text{Im}] \]

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

where
\[ \tau = \text{reflected path delay} \]
\[ \alpha = \text{overall path attenuation} \]
\[ \beta = \text{reflected path relative gain} \]
\[ f_0 = \text{fade null frequency} \]

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 \( \beta \). 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 \( \alpha \) 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 \([0, 2\pi]\).

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.

\[ x = \text{Input signal } ( [\text{Re}, \text{Im}] \text{ for Complex version of block}) \]
\[ y = \text{Output signal } ([\text{Re}, \text{Im}] \text{ for Complex version of block}) \]
\[ h(t) = \sum_{l=0}^{\infty} \sum_{k=0}^{\infty} \beta_{kl} e^{j\theta_{kl}} \delta(t - T_l - \tau_{kl}) \]

where \( T_l \) represents the arrival time of the \( l^{th} \) cluster,
\( \tau_{kl} \) represents the arrival time of the \( k^{th} \) ray within the \( l^{th} \) cluster,
\( \beta_{kl} \) represents the ray's gain, and \( \theta_{kl} \) its phase (complex version only)

\[ p(T_l | T_{l-1}) = \Lambda \exp[-\Lambda(T_l - T_{l-1})], \ l > 0 \]
\[ p(\tau_{kl} | \tau_{(k-1)l}) = \lambda \exp[-\lambda(\tau_{kl} - \tau_{(k-1)l})], \ k > 0 \]

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

where \( \beta_{00}^2 \) 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 \( \Lambda \) for the clusters. The default value is 300 nsec.

**1 / Ray Arrival Rate**
Specifies the inverse of the average Poisson arrival rate \( \lambda \) for the rays within each cluster. The default value is 5 nsec.

**Cluster Decay Time Constant**
Specifies the decay time constant \( \Gamma \) to be applied to subsequent arriving clusters. The default value is 60 nsec.

**Ray Decay Time Constant**
Specifies the decay time constant \( \gamma \) 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 VisSim/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.

\[ x_1 = \text{Complex input signal [Re, Im]} \]

\[ x_2 = \text{Reference average power level} \]

\[ y = \text{Complex output signal [Re, Im]} \]

\[ y(t) = AG(r)e^{i\Phi(r)}x(t) \]

where

\[ G(r) = \frac{\alpha_a r^2}{1 + \beta_a r^2} \]

\[ \Phi(r) = \frac{\alpha_\phi r^2}{1 + \beta_\phi r^2} \]

\[ r = \sqrt{\frac{|x_1|^2}{P_{av}}} \]

\[ A = \text{scaling factor} \]

**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.
### 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 VisSim 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.

\[
x_1 = \text{Complex input signal } [\text{Re}, \text{Im}]
\]

\[
x_2 = \text{Saturation reference point power level (not in dB)}
\]

\[
y = \text{Complex output signal } [\text{Re}, \text{Im}]
\]

\[
y = G(r)e^{i\Phi(r)}x_1
\]

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

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

### 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 \( E_s/N_0 \), as opposed to \( E_b/N_0 \). 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.

\[
x_1 = \text{Input signal vector (Real elements)}
\]

\[
x_2 = \text{Frame clock (impulse)}
\]

\[
y_1 = \text{Output signal vector}
\]

\[
y_1 = \text{Frame clock (impulse)}
\]

\[
y_3 = \text{Current run’s } E_s/N_0 \text{ value}
\]

### 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_0

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.

---

**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.

\[ y = x_1 + x_2 \]

**Complex Conjugate**

This block outputs the complex conjugate of the input.

\[ y = x^* \]

**Complex Division**

This block performs complex division.

\[ y = x_1 \div x_2 \]

**Complex Inverse**

This block outputs the inverse of the complex input.

\[ y = 1 / x \]
$y = \text{Complex output [Re, Im]}$

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

**Complex Multiplication**

This block performs complex multiplication.

$x_1 = \text{Complex signal #1 [Re, Im]}$

$x_2 = \text{Complex signal #2 [Re, Im]}$

$y = \text{Complex output [Re, Im]}$

$y = x_1 \cdot x_2$

**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.

$x = \text{Complex input [Re, Im]}$

$y = \text{Complex output [Re, Im]}$

$y = x^\alpha$

**Exponent**

Specifies the exponent $\alpha$ 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.

$x = \text{Complex input [Re, Im]}$

$y = \text{Complex output [Re, Im]}$

$y = \sqrt{x}$

**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 $[-\pi, \pi]$. If both the real and imaginary parts of the input are 0, zero phase is returned.

$x = \text{Complex vector input [Re, Im]}$

$y_1 = \text{Complex magnitude}$

$y_2 = \text{Complex Phase}$

$y_1 = \sqrt{\text{Re}(x)^2 + \text{Im}(x)^2}$

$y_2 = \text{atan(Im(x), Re(x))}$

**Complex to Real/Imag**

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

$x = \text{Complex vector input [Re, Im]}$

$y_1 = \text{Real part}$

$y_2 = \text{Imaginary part}$
$y_1 = \text{Re}(x) \quad y_2 = \text{Im}(x)$

**Mag/Phase to Complex**

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

$x_1 = \text{Complex magnitude}$

$x_2 = \text{Complex phase}$

$y = \text{Complex vector output } [\text{Re, Im}]$

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

**Real/Imag to Complex**

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

$x_1 = \text{Real part}$

$x_2 = \text{Imaginary part}$

$y = \text{Complex vector output } [\text{Re, Im}]$

$y = [x_1, x_2]$

**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.

$x_1 = \text{Complex input signal } [\text{Re, Im}]$

$x_2 = \text{Sample clock (impulse train)}$

$y_1 = \text{Output symbol number}$

$x_2 = \text{Output clock (impulse train)}$

**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.

\[
y = \frac{1}{2\pi\beta} \left( \frac{\text{Re}(x) \cdot \frac{d}{dt} \text{Im}(x) - \text{Im}(x) \cdot \frac{d}{dt} \text{Re}(x)}{\text{Re}(x)^2 + \text{Im}(x)^2} \right)
\]

\(x =\) Complex FM modulated signal

\(y =\) Demodulated signal

FM Deviation Index

Specifies the FM deviation index \(\beta\) 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.

\[x_1 = \text{IQ Complex input signal } [\text{Re}, \text{Im}]\]

\[x_2 = \text{Sample clock } (\text{impulse train})\]

\(y =\) 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
3 -1.05 -0.95
(blank lines are OK)
```

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)

**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.

\[ \begin{align*}
    x_1 &= \text{Complex input signal [Re, Im]} \\
    x_2 &= \text{Sample clock (impulse train)} \\
    y &= \text{Output symbol number}
\end{align*} \]

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 \( \frac{1}{2} \) 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.

\[ \begin{align*}
    x_1 &= \text{Complex input signal [Re, Im]} \\
    x_2 &= \text{Sample clock (impulse train)} \\
    x_3 &= \text{External constellation spacing reference [Optional]} \\
    y &= \text{Output symbol number}
\end{align*} \]
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 DOS 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**

**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.

$x_1 = \text{Input bit stream } \#1$

$\ldots \ldots$

$x_n = \text{Input bit stream } \#n$

$y = \text{Output symbol number } (0, 1, \ldots, 2^n-1)$

**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.

$x_1 = \text{Input serial data}$

$x_2 = \text{Input clock (impulse train)}$

$x_3 = \text{Reset (resets vector index to first [or last] position)}$

$y_1 = \text{Frame Clock (impulse)}$

$y_2 = \text{Output data vector (size } N\text{)}$

**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.

\[ x_1 = \text{D input} \]
\[ x_2 = \text{Clock (rectangular)} \]
\[ y_1 = \text{Flip flop output} \]
\[ y_2 = \text{Complemented output} \]

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.
\[ x = \text{Input clock} \]
\[ y = \text{Divided output clock [0, 1]} \]

**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.

\[ x_1 = J \text{ input} \]
\[ x_2 = K \text{ input} \]
\[ x_3 = \text{Clock (rectangular)} \]
\[ y_1 = \text{Flip flop output} \]
\[ y_2 = \text{Complemented output} \]

**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 \texttt{case} block (Blocks/Nonlinear).

\[x_1 = \text{External clock (impulse train)} \text{ [optional]}\]
\[x_2 = \text{Section \#0 length [optional]}\]
\[x_3 = \text{Section \#1 length [optional]}\]
\[\ldots\]
\[x_{N+1} = \text{Section \#N-1 length [optional]}\]
\[y_1 = \text{Current state} \ [0, N-1]\]
\[y_2 = \text{Output clock (impulse train)}\]
\[y_3 = \text{Section \#0 enable} \ \{0, 1\}\]
\[\ldots\]
\[y_{N+2} = \text{Section \#N-1 enable} \ \{0, 1\}\]

**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)  \hspace{1cm} 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**

*Rsim 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.

\[ x_1 = \text{Data in} \]
\[ x_2 = \text{Input clock (impulse train)} \]
\[ x_3 = \text{Output clock (impulse train)} \]
\[ y_1 = \text{Data out} \]
\[ y_2 = \text{Queue count} \]
\[ y_3 = \text{Overflow flag (see below)} \]

**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.

\[ x_1 = \text{Input serial bit} \quad n = \text{number of bits} \]
\[ x_2 = \text{Input bit clock (impulse train)} \]
\[ y_1 = \text{Output symbol number (0, 1, \ldots, 2^n-1)} \]
\[ y_2 = \text{Output data clock (impulse train)} \]

**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.

\[ x_1 = \text{Input integer value (range } [0, k-1]) \]
\[ x_2 = \text{Input clock (high } \geq 1\text{) (impulse train)} \]
\[ y_1 = \text{Current state } [0, L-1] \]
\[ y_2 = \text{Output clock (impulse train)} \]
\[ y_3, \ldots, N+2 = \text{State machine outputs} \]

**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.

\[
x = \text{Input symbol number (0, 1, ..., } 2^n-1) \\
y_1 = \text{Output bit stream #1} \\
\ldots \\
y_n = \text{Output bit stream #n}
\]

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.

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.

$x_1 = \text{Frame Clock (impulse)}$

$x_2 = \text{Input data vector (size } N)$

$x_3 = \text{External clock [optional] (impulse train)}$

$y_1 = \text{Output serial data}$

$y_2 = \text{Output clock (impulse train)}$

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.

\[ x_1 = \text{Input data stream} \]
\[ x_2 = \text{Input data clock (0, 1) (impulse train)} \]
\[ y_1 = \text{Output data stream} \]
\[ y_2 = \text{Output data clock (0, 1) (impulse train)} \]

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 \( \frac{1}{2} \) 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.

\[ x_1 = \text{Input data bit stream} \]
\[ x_2 = \text{Input data clock (impulse train)} \]
\[ y_1 = \text{Coded output bit stream} \]
\[ y_2 = \text{Output data clock (impulse train)} \]

**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.

\[ x_1 = \text{Input data stream} \]
\[ x_2 = \text{Input data clock (0, 1) (impulse train)} \]
\[ y_1 = \text{Output data stream} \]
\[ y_2 = \text{Output data clock (0, 1) (impulse train)} \]

---

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.

\[ x_1 = \text{Frame clock (impulse)} \]
\[ x_2 = \text{Input data vector (size } M \text{)} \]
\[ y_1 = \text{Frame clock (impulse)} \]
\[ y_2 = \text{Output data vector (size } N \text{)} \]

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.

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
e tc…

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.

\[x = \text{Input value}\]

\[y = \text{Gray encoded integer}\]

**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.

\[x = \text{Gray encoded value}\]

\[y = \text{Output value}\]

**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.

\[ x_1 = \text{Input coded vector (size } n \text{)} \]
\[ x_2 = \text{Input clock (pulse)} \]
\[ y_1 = \text{Output data vector (size } k \text{)} \]
\[ y_2 = \text{Output clock (pulse)} \]

**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.

\[ x_1 = \text{Input data vector (size } k \text{)} \]
\[ x_2 = \text{Input clock (pulse)} \]
\[ y_1 = \text{Coded output vector (size } n \text{)} \]
\[ y_2 = \text{Output clock (pulse)} \]

**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.

\[ x_1 = \text{Input bit value} \]
\[ x_2 = \text{Input clock (pulse)} \]
\[ y_1 = \text{Manchester encoded output} \]
\[ y_2 = \text{Output clock (pulse)} \]

**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.

\[ x_1 = \text{Frame clock (impulse)} \]
\[ x_2 = \text{Input data vector (size } N \text{)} \]
\( y_1 = \) 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)

<table>
<thead>
<tr>
<th>Pattern period</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>(keep input vector bit #1, discard bits #2 and #3)</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>(keep input vector bit #4, discard bits #5 and #6)</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>. . .</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
</tr>
<tr>
<td>(keep input vector bit #16, discard bit #17 and keep #18)</td>
</tr>
</tbody>
</table>
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.

\[ \begin{align*}
  x_1 &= \text{Input coded data vector (size } n) \quad \text{(Integers in range } [0, 2^M-1]) \\
  x_2 &= \text{Input clock (pulse)} \\
  y_1 &= \text{Decoded output vector (size } k) \\
  y_2 &= \text{Output clock (pulse)} \\
  y_3 &= \text{Error count (-1 indicates RS failure)} \\
\end{align*} \]

**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 to 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 BO= 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:

\[ @^\text{PRIM}B0, @^\text{PRIM}(B0+1), @^\text{PRIM}(B0+2)\ldots @^\text{PRIM}(B0 + n - k) \]
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):

\[
\begin{align*}
M=3: & \quad 1 + x + x^3 \\
M=4: & \quad 1 + x + x^4 \\
M=5: & \quad 1 + x^2 + x^5 \\
M=6: & \quad 1 + x + x^6 \\
M=7: & \quad 1 + x^3 + x^7 \\
M=8: & \quad 1+x^2+x^3+x^4+x^8 \\
M=9: & \quad 1+x^4+x^9 \\
M=10: & \quad 1+x^3+x^{10}
\end{align*}
\]

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 VisSim/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.

\[x_1 = \text{Input data vector (size } k)\]
\[x_2 = \text{Input clock (pulse)}\]
\( y_1 = \) 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 \( B_0 = 112 \) 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 \( B_0 \) and \( PRIM \), the generator polynomial for the RS code will be:

\[ \alpha^{PRIM \cdot B_0}, \alpha^{PRIM \cdot (B_0 + 1)}, \alpha^{PRIM \cdot (B_0 + 2)}...\alpha^{PRIM \cdot (B_0 + n - k)} \]

where \( \alpha \) represents a lowercase alpha.

**First root of generator polynomial**
This setting lets you specify the first root of the generator polynomial (\( B_0 \)) 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):

\( M=3: \quad 1 + x + x^3 \)
\( M=4: \quad 1 + x + x^4 \)
\( M=5: \quad 1 + x^2 + x^5 \)
\( M=6: \quad 1 + x + x^6 \)
\( M=7: \quad 1 + x^3 + x^7 \)
\( M=8: \quad 1+x^2+x^3+x^4+x^8 \)
\( M=9: \quad 1+x^4+x^9 \)
\( M=10: \quad 1+x^3+x^{10} \)
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 VisSim/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.

\[ x_1 = \text{Input symbol number} \]
\[ x_2 = \text{Input data clock (impulse train)} \]
\[ y_1 = \text{I channel output} \]
\[ y_2 = \text{Q channel output} \]
\[ y_3 = \text{Output data clock} \]

**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*

\( K \quad n \quad \#\text{states} \)

*Column header line*

current state \quad input value \quad new state \quad I output \quad 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 \( N = 2^k \times \#\text{states} \).

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)

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>new state</th>
<th>I out</th>
<th>Q out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>2</td>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>7</td>
<td>-1</td>
<td>4</td>
</tr>
</tbody>
</table>

(2nd header line)
Chapter 3  Comm Block Set

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.

\[
x_1 = \text{Input coded bit stream } \{0, 1\}
\]
\[
x_2 = \text{Input coded clock (impulse train)}
\]
\[
y_1 = \text{Decoded bit stream } \{0, 1\}
\]
\[
y_2 = \text{Decoded data clock (impulse train)}
\]
\[
y_3 = \text{Best path metric (Hamming distance)}
\]

**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.
Chapter 3  Comm Block Set

\[ x_1 = \text{Input coded data stream} \]
\[ x_2 = \text{Input coded data clock (impulse train)} \]
\[ y_1 = \text{Decoded bit stream (0, 1)} \]
\[ y_2 = \text{Decoded data clock (impulse train)} \]
\[ y_3 = \text{Best path metric} \]

**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**

```
A | B | C | D | E | F | G | H
---|---|---|---|---|---|---|---
-1.5 | -1.0 | -0.5 | 0 | 0.5 | 1.0 | 1.5 |
```
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 ≤ 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.

\[ x_1 = \text{Input signal ([Re, Im] for complex)} \]
\[ x_2 = \text{Reset signal (resets when } x_2 > 1) \]
\[ y = \text{Power estimate} \]
\[ y[k] = \frac{1}{N} \sum_{i=k-N+1}^{i=k} |x[i]|^2 \]

\[ N = \text{window size or # of samples since reset} \]
\[ k,i = \text{simulation step indices} \]

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 $E_b/N_0$ 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.

\[ x_1 = \text{Current } E_b/N_0 \text{ level} \]

\[ x_2 = \text{Error rate estimate (from Bit/Symbol Error Rate block)} \]

\[ x_3 = \text{Running input error count (from Bit/Symbol Error Rate block)} \]

\[ y_1 = \text{Trigger for BER plot} \]

\[ y_2 = \text{Error rate results for BER plot (y-axis signal - use log scale)} \]

\[ y_3 = \text{SNR data for BER plot (x-axis signal)} \]

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.

Supress 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 $E_s/N_0$ 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.
$x_1 = \text{Current } E_s/N_0 \text{ level}$
$x_2 = \text{Error rate estimate (from Bit/Symbol Error Rate block)}$
$y_1 = \text{Trigger for BER plot}$
$y_2 = \text{Error rate results for BER plot (y-axis signal - use log scale)}$
$y_3 = \text{SNR data for BER plot (x-axis signal)}$
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.

\[ x_1 = \text{Recovered data stream} \]
\[ x_2 = \text{Reference data stream} \]
\[ x_3 = \text{External Clock (0, 1) (impulse train)} \]
\[ y_1 = \text{Error rate (symbol or bit)} \]
\[ y_2 = \text{Error count ((symbols or bits) (optional))} \]
\[ y_3 = \text{Total count ((symbol or bits) (optional))} \]
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.

\[
x_1 = \text{Input signal ([Re, Im] for complex version)}
\]

\[
x_2 = \text{Reference signal ([Re, Im] for complex version)}
\]

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

\[
y = \text{Correlation output ([Re, Im] for complex version)}
\]

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

\[
N = \text{window size}
\]

\[
i, j, k = \text{simulation step indices} \quad j_0 = \text{gate lock step#}
\]

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.

\[ x_1 = \text{Simulation output signal} \]
\[ x_2 = \text{Reference signal} \]
\[ y_1 = \text{Delay estimate} \]
\[ y_2 = \text{Result flag (0, 1)} \]

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.

\[ x = \text{Input signal} \]
\[ y = \text{Time of occurrence} \]

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.

\[ x_1 = \text{Input signal ([Re, Im] for complex version)} \]

\[ x_2 = \text{Input sample clock } \{0, 1\} \]

\[ x_3 = \text{Enable input } \{0= \text{disabled, 1= enabled}\} \]

\[ y = \text{Correlation output ([Re, Im] for complex version)} \]

\[
y_k = \sum_{i=0}^{N-1} (x_1[k-i] \cdot *h[i_{N-1-i}])
\]

\[ N = \text{window size} \quad h[i] = \text{i}^{th} \text{internal tap value} \]

**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 (\( h[N-1] \)) 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:

\[ \text{File header} \quad (\text{this can be anything}) \]

number of taps

tap value #1

tap value #2, tap value #3

...\n
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:

\[ \text{File header} \quad (\text{this can be anything}) \]
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.

\[ x_1 = \text{Input signal} \]
\[ x_2 = \text{Trigger or Enable input} \]
\[ y_1 = \text{Frequency estimate} \]
\[ y_2 = \text{Result update clock (impulse train)} \]

**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.
Chapter 3  Comm Block Set

\[ x_1 = \text{Input signal} \]
\[ x_2 = \text{Reset signal (resets when } x_2 \geq 1) \]
\[ y = \text{Mean estimate} \]
\[
y[k] = \frac{1}{N} \sum_{i=k-N+1}^{k} x[i] \quad N = \text{window size or number of samples since reset}
\]
\[ k, i = \text{simulation step indices} \]

**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.

\[ x = \text{Input data} \]
\[ y = \text{Median output} \]

**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.

\[ x_1 = \text{Input signal} \]
\[ x_2 = \text{Reset signal (Resets when } \geq 1) \]
\[ y_1 = \text{Max value} \]
\[ y_2 = \text{Min value} \]

*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.

\[ x_1 = \text{Input signal} \]

\[ x_2 = \text{Reset signal (resets when } x_2 \geq 1) \]

\[ y_1 = \text{Variance estimate} \]

\[ y_2 = \text{Mean estimate (optional)} \]

\[ y[k] = \frac{1}{N} \sum_{i=k-N}^{k} x[i]^2 - \left( \frac{1}{N} \sum_{i=k-N}^{k} x[i] \right)^2 \]

\[ y_2[k] = \frac{1}{N} \sum_{i=k-N}^{k} x[i] \]

\[ N = \text{window size or number of samples since reset} \]

\[ k,i = \text{simulation step indices} \]

**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 (\( \geq 1 \)). An input clock signal must be provided for the input signal.

\[ x_1 = \text{Input signal} \]

\[ x_2 = \text{Input sample clock } \{0, 1\} \text{ (impulse train)} \]

\[ x_3 = \text{Vector data [size } N\text{]} \]

\[ x_4 = \text{Enable input } \{0, 1\} \]

\[ y = \text{Correlation output} \]
\[ y_k = \sum_{i=0}^{N-1} (x_i[k-i] \cdot x_3[i]) \]

This equation represents a weighted sum of the input signal over a sliding window of size \( N \). Each term \( x_i[k-i] \cdot x_3[i] \) is the product of the input signal sampled at time \( k-i \) and the weight \( x_3[i] \) at the current simulation step \( i \).

**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

The weighted mean is calculated as:

\[
y[k] = \frac{\sum_{i=k-N+1}^{k} x_i[k] \cdot w[i]}{\sum_{i=k-N+1}^{k} w[i]} \quad \text{if} \quad \sum_{i=k-N+1}^{k} w[i] \neq 0 \quad ; \quad y[k] = 0 \quad \text{otherwise}
\]

- \( N \): 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 by 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.

\[ x_1 = \text{Input signal (Real or complex depending on type of block)} \]

\[ x_2 = \text{Error signal (Real or complex depending on type of block)} \]
$x_3 =$ Clock / Lock input \([\text{high} > 0.5]\) (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.

\[ x_1 = \text{Input signal (Real or complex depending on type of block)} \]
\[ x_2 = \text{Error signal (Real or complex depending on type of block)} \]
\[ x_3 = \text{Clock / Lock input \[ high > 0.5 \] (impulse train); Reset \[ \leq -1 \] (pulse)} \]
\[ x_4 = \text{Tap initialization vector (size = number of taps)} \]
\[ y_1 = \text{Output signal} \]
\[ y_2 = \text{Clock output (impulse train)} \]
\[ y_3 = \text{Tap output vector (size = } N \text{ for real eq.; size = } 2N \text{ for complex eq.)} \]
\[ \text{(alternating Real/Imag elements for complex eq.)} \]

**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.

\[ x = \text{Input signal} \]

\[ y = \text{Filtered output signal} \]

**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.

**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.

**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.
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.

\[ x = \text{Input signal} \]
\[ y = \text{Filtered output signal} \]

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.

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.

Show Taps

Displays the current FIR Filter tap values in the VisSim/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 VisSim/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.

\[ x = \text{Input signal} \]
\[ y = \text{Filtered output signal} \]

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 VisSim/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 \([3 \times 1]\), and may be used to drive a plot block configured in XY mode.

\[
\begin{align*}
x &= \text{Complex input signal} \ [\text{Re, Im}] \\
y_1 &= \text{Filtered complex output signal} \ [\text{Re, Im}] \\
y_2 &= \text{Interpolated response used by FFT routine} \ [\text{mag, phase, freq}]
\end{align*}
\]

**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, \(f_s/2\)]**
  
  Used when the input file only includes data over the range of \([0, +f_s/2]\) (real filter).

- **[-\(f_s/2\), \(+f_s/2\)]**
  
  Used when the input file includes data over the range of \([-f_s/2, +f_s/2]\) (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 \(\frac{1}{2}\) 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 $\frac{N}{2}$ the specified FIR filter impulse response length, the required amount of linear phase can be obtained as follows:

$$\text{delay} = \frac{\text{FIRlength}}{2 \cdot f_s} \text{ sec} \quad \text{phase}(f) = -360 \cdot f \cdot \text{delay}$$

**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 $\frac{N}{2}$ 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.

\[ x = \text{Input signal (impulse train or rectangular pulses)} \]
\[ y = \text{Pulse shaped output} \]

**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 $\Delta t$ 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 VisSim/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.

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 VisSim/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.

\[ x = \text{Input signal} \]
\[ y_1 = \text{Filtered output signal} \]
\[ y_2 = \text{Filter sampling clock (impulse train)} \]

**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 VisSim/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 VisSim/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.

\[ x_1 = \text{Input signal (Real or complex depending on type of block)} \]
\[ x_2 = \text{Error signal (Real or complex depending on type of block)} \]
\[ x_3 = \text{Clock / Lock input [high > 0.5] (impulse train); Reset [\leq -1] (pulse)} \]
\[ x_4 = \text{Tap initialization vector (size = number of taps)} \]
\[ y_1 = \text{Output signal} \]
\[ y_2 = \text{Clock output (impulse train)} \]
\[ y_3 = \text{Tap output vector (size = } N \text{ for real eq.; size = } 2N \text{ for complex eq.)} \]
\[ \text{alternating Real/Imag elements for complex eq.)} \]

**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.
Chapter 3  Comm Block Set

\[ x = \text{Input signal} \]
\[ y = \text{Filtered output signal} \]

**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 VisSim/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 VisSim/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.

\[ x = \text{Input signal} \]
\[ y = \text{Filtered output signal} \]

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 VisSim/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 Coefficient 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.

\[ x = \text{Input drive signal} \]

\[ y_1 = \text{Output signal ([Re, Im] for complex)} \]

\[ y_2 = \text{Accumulated phase (rad)} \]

\[ y_1(t) = Ae^{j\theta(t)} \quad y_2(t) = \theta(t) \]

\[ \theta(t) = \int_0^t (2\pi f_c + x_1(\tau)K_0) d\tau + \phi \]

where:

\[ f_c = \text{translation frequency} \quad A = \text{carrier amplitude} \]

\[ K_0 = \text{VCO gain} \quad \phi = \text{initial phase (radians)} \]

**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 lengths.

- **32 Bits**
  
  Specifies that the internal sine and cosine functions, and the block’s output, use 32 bit word lengths.
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 a 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 Control 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.

\[
x = \text{Input signal}
\]
\[
y_1 = \text{Modulated signal ([Re, Im] for complex)}
\]
\[
y_2 = \text{Carrier phase (rad) [optional]}
\]

\[
y_1(t) = (A + mx)e^{j(2\pi f_c t + \phi)}
\]
\[
\phi = \frac{\pi \theta}{180}
\]
\[
y_2(t) = 2\pi f_c t + \phi
\]

**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 \( \theta \) 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 \([ 0, 2\pi ]\)**

Specifies a wrapped output mode for the carrier phase. This forces the phase value to be in the range of \([ 0, 2\pi ]\).

**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 = \text{Input symbol number (integer \([ 0 \ldots N -1]\), where \( N \) is the constellation size)}
\]
\[
x_2 = \text{Input data clock (impulse train)}
\]
\[
y_1 = \text{Modulated signal ([Re, Im] for complex)}
\]
\[
y_2 = \text{Unmodulated carrier phase (rad)}
\]
\[ y_1(t) = A_{\delta} e^{j(2\pi f_s t + \phi)} \quad \phi = \frac{\pi \theta}{180} \]

\[ y_2(t) = 2\pi f_c t + \phi \]

**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 \[ 0, 2\pi \]**
  - Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of \[ 0, 2\pi \].

**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.

\[ x_1 = \text{Input symbol number (integer } [0 \ldots N-1], \text{ where } N \text{ is the constellation size}) \]

\[ x_2 = \text{Input data clock (impulse train)} \]

\[ y_1 = \text{Modulated signal } ([\text{Re}, \text{Im}] \text{ for complex}) \]

\[ y_2 = \text{Unmodulated carrier phase (rad)} \]

\[
y_1(t) = Ae^{j(2\pi f_c t + \sum_{k=0}^{N-1} \theta_d[k])} \]

\[
\phi = \frac{\pi \theta}{180} \]

\[
y_2(t) = 2\pi f_c t + \phi \]

\[
\theta_d[k](x) = k\text{th phase transition} \]

The phase transition value \( \theta_d \) 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 \([ 0, 2\pi ]\).

**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, pi4dapsk, 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, Pi4-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)

pi4dapsk
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.

\[ x = \text{Input signal} \]

\[ y_1 = \text{Modulated signal ([Re, Im] for complex)} \]

\[ y_2 = \text{Phase of complex modulated signal (optional)} \]

\[ y_1(t) = Ae^{j[2\pi(f_c + \beta x) + \phi]} \quad \phi = \frac{\pi \theta}{180} \]

\[ y_2(t) = 2\pi(f_c + \beta x)t + \phi \]

**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 \( \theta \) in degrees.

**FM Deviation**

Specifies the FM deviation index \( \beta \). 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, 2\pi ]*

Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of \([ 0, 2\pi ]\).

**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.

\[ x = \text{Input symbol number (integer [ 0 \ldots N-1], where } N \text{ is the number of tones)} \]

\[ y_1 = \text{Modulated signal ([Re, Im] for complex)} \]
\[ y_2 = \text{Phase of complex modulated signal (optional)} \]

\[ f_{out} = f_{\text{min}} + x \cdot \Delta f \]

**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 \( \Delta f \) 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 \([0, 2\pi]\)**
  - Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of \([0, 2\pi]\).

**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.

\[ x = \text{Input data signal (binary)} \]

\[ y = \text{Complex output signal [Re, Im]} \]

**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.

\[ x = \text{Input data signal (binary)} \]
\[ y = \text{Complex output signal [Re, Im]} \]

**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.

\[ x_1 = \text{I channel input} \]
\[ x_2 = \text{Q channel input} \]
\[ y_1 = \text{Modulated signal ([Re, Im] for Complex)} \]
\[ y_2 = \text{Unmodulated carrier phase (rad) [optional]} \]

\[
y_1(t) = A(x_1 + jx_2) e^{j(2\pi f_c t + \phi)} \quad \phi = \frac{\pi \theta}{180}
\]
\[
y_2(t) = 2\pi f_c t + \phi
\]

**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 \( \theta \) 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)

\[
y_1(t) = Ad_I \cos(\pi f_c (t - \tau)) \cos(2\pi f_c (t - \tau)) + j Ad_Q \sin(\pi f_c (t - \tau)) \sin(2\pi f_c (t - \tau))
\]

\[
y_2(t) = 2\pi f_c (t - \tau) + \phi \\
\phi = \frac{\pi \theta}{180} \\
\tau = \) data start time
\]

\( d_I(x_1) \in \{ \pm 1 \} \\
d_Q(x_2) \in \{ \pm 1 \} \)

**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 \( \theta \) in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at \( \pi/4 \) radians.

**Data Rate**

Specifies the data rate \( f_s \) in symbols/second.

**Data Start Time**

Specifies the start time \( \tau \) 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)
Chapter 3 Comm Block Set

\[ y_2 = \text{Phase of complex modulated signal (optional)} \]
\[ y_1(t) = Ad e^{j(2\omega t + \beta k + \phi)} \]
\[ \phi = \frac{\pi \theta}{180} \]
\[ y_2(t) = 2\pi f_c t + \phi \]

**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 \( \theta \) in degrees.

**Modulation Index**
- Specifies the PM index \( \beta \). 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 \([ 0, 2\pi ]\)**
  - Specifies a wrapped output mode for the modulated carrier phase. This forces the phase value to be in the range of \([ 0, 2\pi ]\).

**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 = \text{Input symbol number (integer \([ 0 \ldots N-1]\), where } N \text{ is the number of levels)} \]
\[ y = \text{Baseband modulated signal} \]

**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
Specifications the symbol rate (pulse rate) in symbols/second.

Pulse Amplitude
Specifications the amplitude of the rectangular pulse. This value is specified in volts.

Data Frame Start
Specifications 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.

\[ x_1 = \text{Input symbol number (integer } [0 \ldots N-1], \text{ where } N \text{ is the constellation size} \]
\[ x_2 = \text{Input clock (impulse train)} \]
\[ y_1 = \text{Modulated signal ([Re, Im] for complex)} \]
\[ y_2 = \text{Unmodulated carrier phase (rad)} \]
\[ y_1(t) = Ae^{i(2\pi f_c t + \theta_d + \phi)} \]
\[ \phi = \frac{\pi \theta_d}{180} \]
\[ y_2(t) = 2\pi f_c t + \theta \]
\[ \theta_d(x) = \text{data phase} \]

PSK Type
Specifications 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
Specifications the carrier single-sided peak amplitude \( A \) in volts.
**Constellation Rotation**

Specifies the constellation rotation \( \theta \), 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 \( \pi/n \) 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 corresponds 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, 2\pi ].

**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 $\pi$ 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).

$$\theta_d = \begin{cases} 0 & \text{if } x \leq 0.5 \\ \pi & \text{if } x > 0.5 \end{cases}$$

![BPSK constellation diagram]

**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 $\pi/2$ 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).

$$\theta_d = \begin{cases} \pi/4 & x \leq 0.5 \\ 3\pi/4 & 0.5 < x \leq 1.5 \\ -\pi/4 & 1.5 < x \leq 2.5 \\ -3\pi/4 & x > 2.5 \end{cases}$$

![QPSK constellation diagram]
The default first constellation point is at $\pi/4$ 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 $\pi/4$ rad increments.

The $\text{8-PSK}$ block accepts a symbol number $\{0, 1, 2, \ldots, 7\}$ 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 ($\text{PSK_GRAY.DAT}$).

![8-PSK constellation diagram]

The default first constellation point is at $\pi/8$ 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 $\pi/8$ rad increments.

The $\text{16-PSK}$ block accepts a symbol number $\{0, 1, 2, \ldots, 15\}$ 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 ($\text{PSK_GRAY.DAT}$).

![16-PSK constellation diagram]

The default first constellation point is at $\pi/16$ 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 $\pi/16$ rad increments.

The $\text{32-PSK}$ block accepts a symbol number $\{0, 1, 2, \ldots, 31\}$ as its input and maps it to the constellation point specified in the PSK map file. The default first constellation point is at $\pi/32$ 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.

\[ x_1 = \text{Input symbol number (integer } [0 \ldots N-1], \text{ where } N \text{ is the constellation size}) \]

\[ y_1 = \text{Modulated signal ([Re, Im] for complex)} \]

\[ y_2 = \text{Unmodulated carrier phase (rad) (optional)} \]

\[ y_1 = \frac{\alpha}{2} A_d e^{j(2\pi f_c t + \theta_d + \phi)} \quad \phi = \frac{\pi \theta_r}{180} \]

\[ y_2(t) = 2\pi f_c t + \phi \]

\[ \theta_d(x) = \text{data phase} \]

\[ A_d(x) = \text{data amplitude} \]

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 \( \alpha \) in volts.

Constellation Rotation

Indicates the constellation rotation \( \theta_r \) 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 \(\{0, 1, 2, \ldots, 127\}\) 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 \(\{0, 1, 2, \ldots, 255\}\) 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).

\[
A_f = \begin{cases} 
  -3 & x \leq 0.5 \\
  -1 & 0.5 < x \leq 1.5 \\
   +3 & 1.5 < x \leq 2.5 \\
   +1 & x > 2.5 
\end{cases}
\]
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).

\[
\begin{align*}
A_d &= \begin{cases} 
-7 & x \leq 0.5 \\
-5 & 0.5 < x \leq 1.5 \\
-1 & 1.5 < x \leq 2.5 \\
-3 & x > 2.5 
\end{cases} \\
\theta_d &= \begin{cases} 
+7 & 3.5 < x \leq 4.5 \\
+5 & 4.5 < x \leq 5.5 \\
+1 & 5.5 < x \leq 6.5 \\
+3 & x > 6.5 
\end{cases}
\]

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 $\pi/2$ rad increments. The carrier amplitude remains constant. The data on the Q channel input is delayed $\frac{1}{2}$ 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 $\pi/2$ rad, and the modulated spectrum exhibits lower sidelobes than ordinary QPSK. Since the Q data is delayed $\frac{1}{2}$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 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.

\[
x_1 = \text{I channel data (binary)}
\]

\[
x_2 = \text{Q channel data (binary)}
\]

\[
y_1 = \text{Modulated signal ([Re, Im] for complex)}
\]

\[
y_2 = \text{Unmodulated carrier phase (rad) (optional)}
\]

\[
y_1(t) = Ae^{j(2\pi f_c t + \theta_d + \phi)}
\]

\[
y_2(t) = 2\pi f_c t + \phi
\]

\[
\phi = \frac{\pi \theta_r}{180}
\]

\[
\theta_d(x_1, x_2) = \text{data phase}
\]

\[
T = \text{symbol duration}
\]

\[
\theta_d = \begin{cases} 
\pi / 4 & x_1 \leq 0.5 \text{ and } x_2 \leq 0.5 \\
3\pi / 4 & x_1 \leq 0.5 \text{ and } x_2 > 0.5 \\
-\pi / 4 & x_1 > 0.5 \text{ and } x_2 \leq 0.5 \\
-3\pi / 4 & x_1 \leq 0.5 \text{ and } x_2 > 0.5 
\end{cases}
\]

where \( x_2 = x_2(t - T/2) \)

\[
\text{Carrier Frequency}
\]

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 \( \theta_r \) 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 \([0, 2\pi]\)**  
Specifies a wrapped output mode for the unmodulated carrier phase. This forces the phase value to be in the range of \([0, 2\pi]\).

---

**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).

\[x = \text{Input clock signal}\]

\[y = \text{Output clock signal}\]

**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).

\[x = \text{Input clock signal}\]

\[y = \text{Output clock signal}\]

**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.

\[x = \text{Input signal}\]

\[y = \text{Output interpolated signal}\]
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, Compressor, 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, 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 \([-0.5\Delta q, 0.5\Delta q]\) to be converted to level 0 (for Signed output mode), where \( \Delta q \) is a quantization level. In offset mode, a positive value in the range \([0, \Delta q]\) is converted to level 0, while a negative value in the range \([-\Delta q, 0]\) 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

<table>
<thead>
<tr>
<th>Midlevel Mode (( n = 3 ))</th>
<th>Signed Mode Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip region 3</td>
<td>7</td>
</tr>
<tr>
<td>max amplitude 3</td>
<td>3</td>
</tr>
<tr>
<td>zero volts 1</td>
<td>4</td>
</tr>
<tr>
<td>- max amplitude -3</td>
<td>1</td>
</tr>
</tbody>
</table>

| Clip region 4 | 0 |

<table>
<thead>
<tr>
<th>Unsigned Mode Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip region 7</td>
</tr>
<tr>
<td>max amplitude 6</td>
</tr>
<tr>
<td>zero volts 5</td>
</tr>
<tr>
<td>- max amplitude -4</td>
</tr>
</tbody>
</table>
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 $[-\text{maxAmp}, +\text{maxAmp}]$.

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 $[-\text{maxAmp}+\text{offset}, +\text{maxAmp}+\text{offset}]$.

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 $[-\text{maxAmp}, \text{maxAmp}]$ 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 $[-\text{maxAmp}, \text{maxAmp}]$ 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 $\mu$-law or $A$-law companding, as described below. A standard value used in $\mu$-law companding is $\mu=255$, 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.
\( x = \text{Input signal} \)

\( y = \text{Companded output} \)

In the equations below, \( x = \frac{x_1}{x_{max}} \)

\[
y = \text{sgn}(x) \frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)} \quad |x| \leq 1 \quad \mu - \text{law companding}
\]

\[
y = \begin{cases} 
\text{sgn}(x) \frac{1 + \ln A |x|}{1 + \ln A} & \frac{1}{A} < |x| \leq 1 \\
\text{sgn}(x) \frac{A |x|}{1 + \ln A} & 0 \leq |x| \leq \frac{1}{A}
\end{cases} \quad A - \text{law companding}
\]

**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**

\( \mu \)-Law

Indicates \( \mu \)-law companding.

\( A \)-Law

Indicates \( A \)-law companding.

**Max Value**

Indicates the maximum allowed magnitude \( (x_{\text{max}}) \) 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.

**\( \mu \) Value**

Indicates the value of \( \mu \) for \( \mu \)-law companding. The value of \( \mu \) must be greater than 0.

**Complex Exponential**

This block outputs the complex exponential of the input signal.

\( y(t) = e^{ix(t)} \)

\( x = \text{Phase angle (rad)} \)

\( y = \text{Complex output [Re, Im]} \)

**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 2xN 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.

\[ x_1 = \text{Input trigger (high > 0.5)} \]
\[ x_2 = \text{Real part (or magnitude) of complex input} \]
\[ x_3 = \text{Imaginary part (or phase) of complex input} \]
\[ y_1 = \text{Output trigger (0 or 1) for plot block} \]
\[ y_2 = \text{Real part (or magnitude) of complex output} \]
\[ y_3 = \text{Imaginary part (or phase) of complex output} \]
\[ y_4 = \text{X-axis drive signal: Frequency (xHz) in FFT mode or time (seconds) in IFFT mode} \]

**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 (y4). 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/EXAMPLE. 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.

\[ x = \text{Input value in decibels} \]
\[ y = \text{Power output value} \]
\[ y = 10^{(x/10)} \]

**Decibels to Real**
The block converts a decibel input to a real number.

\[ x = \text{Input value in decibels} \]
\[ y = \text{Output real value} \]
\[ y = 10^{(x/20)} \]

**Degrees to Radians**
The block converts from degrees to radians.

\[ x = \text{Input value in degrees} \]
\[ y = \text{Output value in radians} \]
\[ y = \frac{x\pi}{180} \]

**Hertz to Rad/sec**
The block converts from hertz to radians/second.

\[ x = \text{Input value in hertz} \]
\[ y = \text{Output in radians/second} \]
\[ y = 2\pi x \]
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.

\[ x_1 = \text{Input complex magnitude} \]
\[ x_2 = \text{Input complex phase} \]
\[ y_1 = \text{Real part of complex output} \]
\[ y_2 = \text{Imaginary part of complex output} \]
\[ y_1 = x_1 \cos(x_2) \quad y_2 = x_1 \sin(x_2) \]

Power to Decibels
The block converts a power input value to decibels. The input must be greater than zero.

\[ x = \text{Input power value} \]
\[ y = \text{Output in decibels} \]
\[ y = 10 \log(x) \quad x > 0 \]

Radians to Degrees
The block converts from radians to degrees.

\[ x = \text{Input value in radians} \]
\[ y = \text{Output value in degrees} \]
\[ y = \frac{180x}{\pi} \]

Rad/sec to Hertz
The block converts from radians/second to hertz.

\[ x = \text{Input value in radians/second} \]
\[ y = \text{Output in hertz} \]
\[ y = \frac{x}{2\pi} \]

Real to dB
The block converts a real input to decibels. The input must be greater than 0.

\[ x = \text{Input real value} \]
\[ y = \text{Output in decibels} \]
\[ y = 20 \log(x) \quad x > 0 \]

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 \(-\pi\) to \(\pi\). 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.

\[ x_1 = \text{Real part of complex input} \]
\[ x_2 = \text{Imaginary part of complex input} \]
\[ y_1 = \text{Complex magnitude} \]
\[ y_2 = \text{Complex phase} \]
\[ y_1 = \sqrt{(x_1)^2 + (x_2)^2} \quad y_2 = \tan(x_2, x_1) \]

**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.

\[ x = \text{Input digital signal (integer)} \]
\[ y = \text{Output analog signal} \]

\[ y = A_{\min} + x \cdot \Delta q \quad \text{(unsigned mode)} \]
\[ y = A_{\min} + (x + \frac{k}{2}) \cdot \Delta q \quad \text{(signed mode)} \]

where \( \Delta q = \frac{A_{\max} - A_{\min}}{k - 1} \); \( k = 2^N \); \( N = \text{num. bits} \)

**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 (for example, 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.

\[ x = \text{Complex input signal [Re, Im]} \]
\[ y = \text{Delayed version of complex input signal [Re, Im]} \]

\[ y[n] = x[n-k] \quad k = \text{delay size} \quad n = \text{simulation step index} \]
Delay

Specifies the delay in simulation steps or seconds, depending on the selected Delay Mode. Valid range is from 1 to 32,767 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 (for example, Runge Kutta). For these steps, the previous value is held. The internal shift register is initialized to the Initial Condition parameter value.

\[ y[n] = x[n-k] \quad k = \text{delay size} \quad n = \text{simulation step index} \]

Delay

Specifies the delay in simulation steps or seconds, depending on the selected Delay Mode. Valid range is from 1 to 32,767 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.

\[ y = x \cdot 10^{\left(\frac{\text{gain}}{20}\right)} \]

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.

\[ x_1 = \text{Input signal } (\text{Re, Im} \text{ for complex}) \]
\[ x_2 = \text{Optional external clock } (0, 1) \text{ (impulse train)} \]
\[ y_1 = \text{Integrator output } (\text{Re, Im} \text{ for complex}) \]
\[ y_2 = \text{Output clock (impulse train)} \]

**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 \(\Delta t\) 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 \(t = \text{Initial Delay} + \text{Dump Period}\). 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.

\[ x_1 = \text{Input } #1 \]
\[ x_2 = \text{Input } #2 \]
\[ \ldots \]
\[ x_n = \text{Input } #N \]
\[ y = \text{Output index value (either } \{0, 1, \ldots, N-1\} \text{ or } \{1, 2, \ldots, N\} \) \]

Number of Inputs
Specifies the number of input connections \( N \). Valid range is 2 to 16.

Index Mode
\textbf{Start at Zero}
Specifies the output range to be \( \{0, 1, \ldots, N-1\} \).

\textbf{Start at One}
Specifies the output range to be \( \{1, 2, \ldots, N\} \).

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 \([0, \text{mod val})\) when specifying a positive modulo value, and \((\text{mod val}, 0]\) 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.

\[ x = \text{Input signal} \]
\[ y = \text{Remainder from modulo operation} \]

Indicates the modulo value, which can be a real number or an integer. In integer mode, this entry is truncated.

Modulo Mode
\textbf{Integer}
Indicates integer modulo operation.

\textbf{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.

\[ x_1 = \text{Start pulse (high > 0.5)} \]
\[ x_2 = A \text{ Channel input} \]
\[ x_3 = B \text{ Channel input} \]
\[ y_1 = \text{Output trigger (0 or 1) for plot block} \]
\[ y_2 = A \text{ Channel vector trace output} \]
\[ y_3 = B \text{ Channel vector trace output} \]
\[ y_4 = \text{Time axis for plot’s X-axis} \]

**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 $> 1$. 

Chapter 3  Comm Block Set
Chapter 3  Comm Block Set

Phase Rotate
This block multiplies the complex input by the complex exponential $e^{j\phi}$, which results in a phase rotation of $\phi$ 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 [-$\pi$, +$\pi$] 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 $g(x)$ 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).
\[ x^5 \text{ Coefficient} \]
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 \texttt{plot} block configured in XY mode with an external trigger. An output trigger line and x-axis output are provided for driving the \texttt{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 \( f_0 \) results in two peaks of -13 dBm at \(-f_0\) and \(+f_0\), while the real spectrum results in a single peak of -10 dBm at \( f_0 \).

Once triggered, the \texttt{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 \texttt{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 \texttt{plot} block
\( y_2 \) = Magnitude (or Real) output
\( y_3 \) = Phase (or Imaginary) output
\( y_4 \) = Frequency (xHz) 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 \([-\pi, +\pi]\).

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).

\[ x = \text{Input signal} \]
\[ y = \text{Sampled output} \]

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.

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.

\[ x_1 = \text{Frame clock (high > 0.5) (impulse)} \]
\[ x_2 = \text{Input signal vector (real or magnitude component; size } N \text{)} \]
\[ x_3 = \text{Input signal vector (imaginary or phase component; size } N \text{)} \]
\[ y_1 = \text{Frame clock (impulse)} \]
\[ y_2 = \text{Output signal vector (real or magnitude component; size } N \text{)} \]
\[ y_3 = \text{Output signal vector (imaginary or phase component; size } N \text{)} \]
\[ y_4 = \text{Output signal vector (frequency or time output vector; size } N \text{)} \]

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.

\[
x_1 = \text{UP error signal}
\]

\[
x_2 = \text{DOWN error signal}
\]

\[
y = \text{Output signal (VCO drive)}
\]

Charge Pump transfer function (Laplace format):
\[ F(s) = \frac{\tau_2 s + 1}{\tau_1 s} \]

**Natural Frequency**
Indicates the natural frequency (\( \omega_n \)) 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 (\( K_v \)) 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 VisSim/Comm’s compound blocks, including the PLLs and Costas Loop. For more information about these compound blocks, see Appendix A, “VisSim/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 (\( \tau_1, \tau_2 \)), 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 \( (K_p K_d) \) should be much larger than the natural frequency \( (\omega_n) \).

\[ x = \text{Input drive signal} \]
\[ y = \text{Output signal (VCO drive)} \]

Loop filter transfer function (Laplace format):
\[ F(s) = \frac{s \tau_2 + 1}{s \tau_1} \quad \text{(active)} \quad F(s) = \frac{s \tau_2 + 1}{s \tau_1 + 1} \quad \text{(passive)} \]

Natural Frequency
Indicates the natural frequency \( (\omega_n) \) 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 \( \sin(\theta_e) \sim \theta_e \) 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.

\[ x = \text{Input drive signal} \]
\[ y = \text{Output signal (VCO drive)} \]

Loop filter transfer function (Laplace format):
\[ F(s) = \frac{\omega_3^3 + 2\omega_3^2 s + 2\omega_3 s^2}{K_s K_d s^2} \]

Natural Frequency
Indicates the natural frequency \( \omega_3 \) 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 \( \sin(\theta_\varepsilon) \approx \theta_\varepsilon \) 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 \( \pi/2 \text{ V/rad} \approx 1.5708 \).
Chapter 3  Comm Block Set

\[ x_1 = \text{Reference input} \]
\[ x_2 = \text{VCO input} \]
\[ y = \text{Error signal [0, 1]} \]

**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 \( \frac{1}{\pi} \text{V/rad} \) (~0.31831).

\[ x_1 = \text{Reference input} \]
\[ x_2 = \text{VCO input} \]
\[ y_1 = \text{UP error signal [0, 1]} \]
\[ y_2 = \text{DOWN error signal [0, 1]} \]

**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 \( \frac{1}{2\pi} \text{V/rad} \) (~0.15915).

\[ x_1 = \text{Reference input} \]
\[ x_2 = \text{VCO input} \]
\[ y_1 = \text{UP error signal [0, 1]} \]
\[ y_2 = \text{DOWN error signal [0, 1]} \]
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, 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.

\[ x = \text{Input signal} \]
\[ y = \text{Amplifier output signal} \]
\[ y = ax + bx^2 + cx^3 + dx^4 + ex^5 + \text{noise} \]  
(\text{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.
Chapter 3  Comm Block Set

**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.

\[ y = \begin{cases} x \cdot 10^{(gain/20)} + noise & \text{for fixed mode} \\ x \cdot 10^{(gain(x_2)/20)} + noise & \text{for pattern mode} \end{cases} \]

**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 VisSim/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.

\[
x = \text{Input signal} \\
y = \text{Attenuated output signal} \\
y = x \cdot 10^{(-\text{loss}/20)} + \text{noise}
\]

**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.

\[
x = \text{Input signal} \\
y = \text{Cable output signal}
\]
\[ y = x \cdot 10^{(-loss/20)} + \text{noise} \quad \text{where} \quad loss = \text{cable length} \times \text{loss/meter} \]

**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.

<table>
<thead>
<tr>
<th>Input Coupling</th>
<th>Output Coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 ) = Primary input signal</td>
<td>( x ) = Input signal</td>
</tr>
<tr>
<td>( x_2 ) = Coupled signal</td>
<td>( y_1 ) = Primary output</td>
</tr>
<tr>
<td>( y ) = Output signal</td>
<td>( y_2 ) = Coupled output</td>
</tr>
</tbody>
</table>

**Input Coupling**

\[ y = x_1 \cdot 10^{(-\text{directLoss}/20)} + x_2 \cdot 10^{(-\text{coupledLoss}/20)} + \text{noise} \]

**Output Coupling**

\[ y_1 = x \cdot 10^{(-\text{directLoss}/20)} + \text{noise} \]

\[ y_2 = x \cdot 10^{(-\text{coupledLoss}/20)} \]

**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.

\[ x_1 = \text{RF input} \]
\[ x_2 = \text{LO input} \]
\[ y = \text{IF output} \]

\[ y = k \cdot (x_1 + ax_1^3 + bx_1^5) \cdot (x_2 + cx_2^3 + dx_2^5) + \text{noise} \quad \text{(linear region)} \]

\[ y = k \cdot (\pm \text{satAmpOutput}) \cdot (x_2 + cx_2^3 + dx_2^5) + \text{noise} \quad \text{(saturated region)} \]

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.

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.

\[
x = \text{Input value in dBm} \\
y = \text{Output value in dBW} \\
y = x - 30
\]

**dBW to dBm**
The block converts decibels relative to a Watt to decibels relative to a milliWatt.

\[
x = \text{Input value in dBW} \\
y = \text{Output value in dBm} \\
y = x + 30
\]

**dBm to dBm/Hz**
The block scales the input signal according to the current simulation frequency.

\[
x = \text{Input value in dBm} \\
y = \text{Output value in dBm/Hz}
\]
$y = x - 10 \log(F_y)$

**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

$y = x + 10 \log(F_y)$

**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.

<table>
<thead>
<tr>
<th>Splitter Mode</th>
<th>Combiner Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ = Input signal</td>
<td>$x_1$ = Input signal #1</td>
</tr>
<tr>
<td>$y_1$ = Output #1</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>$x_n$ = Input signal #n</td>
</tr>
<tr>
<td>$y_n$ = Output #n</td>
<td>$y$ = Combined output</td>
</tr>
</tbody>
</table>

**Split 0**

$$y_1 = y_2 = ... = y_n = x \sqrt{10} \left(\frac{-\text{loss}}{20}\right) + \text{noise}$$

**Split 180**

$$y_1 = x \sqrt{2} \cdot 10^{\left(-\text{loss}/20\right)} + \text{noise}$$

$$y_2 = y_1$$

**Combiner**

$$y = \sum_{i=1}^{n} x_i \sqrt{n} \cdot 10^{\left(-\text{loss}/20\right)} + \text{noise}$$

**Number of Outputs**

**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.

<table>
<thead>
<tr>
<th>Input Switch</th>
<th>Output Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$ = Input path selector ${1,2,...,n}$</td>
<td>$x_1$ = Output path selector ${1,2,...,n}$</td>
</tr>
<tr>
<td>$x_2$ = Input signal #1</td>
<td>$x_2$ = Input signal</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_n$ = Input signal $#n$</td>
<td>$x_n$ = Output signal $#n$</td>
</tr>
<tr>
<td>$y$ = Output signal</td>
<td>$y_n$ = Output signal $#n$</td>
</tr>
</tbody>
</table>

**Input Switch**

$$y = \left( x_{sel} + \sum_{i \neq sel} x_i \cdot 10^{(-\text{isolation} / 20)} \right) \cdot 10^{(-\text{loss} / 20)} + \text{noise}$$

**Output Switch**

$$y_{sel} = x \cdot 10^{(-\text{loss} / 20)} + \text{noise}$$

$$y_{i} = x \cdot 10^{(-\text{isolation} / 20)} \quad i \neq sel$$

**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.

\[
x = \text{Input signal}
\]
\[
x_2 = \text{Loss in dB (\geq 0)}
\]
\[
y = \text{Attenuated output signal}
\]
\[
y = x \cdot 10^{-\left(\frac{-\text{loss}}{20}\right)} + \text{noise}
\]

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.

\[
x_1 \ldots x_n \text{ = Input value(s)}
\]
\[
x_{n+1} \text{ = Input clock (impulse train)}
\]
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.
Chapter 3  Comm Block Set

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 VisSim 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.
$x_1, \ldots, x_n = \text{Input value(s)}$

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 VisSim 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)
- 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.
$x_1, \ldots, x_n = \text{Input value(s)}$
$x_{n+1} = \text{Input clock (impulse train)}$
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.

\[ y_1 = \text{Complex output signal} [\text{Re}, \text{Im}] \]
\[ y_2 = \text{Generator phase (rad)} \text{[Optional]} \]
\[ y_1(t) = Ae^{j(2\pi f_c t + \phi)} \quad \phi = \frac{\pi \theta}{180} \]
\[ y_2(t) = 2\pi f_c t + \phi \]

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 $\theta$ 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.

\[ y = \text{Sweep signal output} \]
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.

\[ y = \text{Output signal} \]

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.

\[ y = \text{Output pulse train} \]

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.

\[ y = \text{Output noise signal} \]
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 VisSim/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:

\[ n = 7 \quad \text{PN length} = 127 \quad x_k = x_{k-7} \oplus x_{k-3} \quad \text{where } \oplus \text{ represents modulo 2 addition} \]

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:**

\[ n = 7 \quad \text{PN length} = 127 \]

\[ p(D) = D^7 + D^4 + 1 \quad \text{or} \quad p(x) = x^7 + x^4 + 1 \]

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.

\[ x = \text{Optional external clock (impulse train)} \]

\[ y_1 = \text{Pseudo-random sequence output (binary or bilevel)} \]

\[ y_2 = \text{Output clock (impulse train)} \]

**Shift Register Size**

Specifies the order \( n \) of the PN sequence and determines its length. The sequence length is \( 2^n - 1 \) \((2^n \text{ for augmented sequences})\), 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.

\[ y = \text{Output signal (Poisson distributed impulse train)} \]

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.

\[ y = \text{Random variable output per selected distribution} \]
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,σ²) distributions.

**Variance**
- Specifies the variance for the distribution function, if applicable. Applies to Gaussian and Rayleigh (µ,σ²) 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 VisSim/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.

\[ y_1 = \sin(2\pi f_c t + \phi) \] or \[ y_2 = \cos(2\pi f_c t + \phi) \]

\[ \phi = \frac{\pi \theta}{180} \]

\[ y_2(t) = 2\pi f_c t + \phi \]

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 \( \theta \) 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 \([0, \text{fs}/2]\) or \([-\text{fs}/2, \text{fs}/2]\), 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.

\[
x_1 = \text{Input trigger (high > 0.5) (pulse)}
\]

\[
x_2 = \text{Input frequency vector [optional] (size N+1)}
\]

\[
y_1 = \text{Spectral mask vector for plot block (size N+1)}
\]

\[
y_2 = \text{Frequency vector for plot block (if needed)}
\]

**Mask Data Range**

\[ [0, \text{fs}/2] \]

Used when the input file only includes data points over the range of \([0, +\text{fs}/2]\). The Spectral Mask block automatically mirror images the table’s data points for the negative portion of the frequency axis.

\[ [-\text{fs}/2, +\text{fs}/2] \]

Used when the input file includes data points over the range of \([-\text{fs}/2, +\text{fs}/2]\).

**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 \(-\text{fs}/2\) and \(+\text{fs}/2\).

**Select File**

Opens the Select File dialog box for selecting the spectral mask definition file.

**Browse File**

Opens the selected mask file using Notepad.
Chapter 3  Comm Block Set

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.

\[ y = \text{Output signal vector [size } N\text{]} \]

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.

\[ x = \text{Input drive signal} \]
\[ y_1 = \text{Output signal ([Re, Im] for complex)} \]
\[ y_2 = \text{Accumulated phase (rad) (optional)} \]

\[ y_1(t) = Ae^{j\theta(t)} \]
\[ y_2(t) = \theta(t) \]

\[ \theta(t) + \int_0^t (2\pi f_c + x_1(\tau)K_o) d\tau + \phi \]

where:
\[ f_c = \text{translation frequency} \]
\[ A = \text{carrier amplitude} \]
\[ K_o = \text{VCO gain} \]
\[ \phi = \text{initial phase (radians)} \]
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.

\[ x_1 = \text{Optional external clock (impulse train)} \]
\[ x_2 = \text{Optional external row selector} \]
\[ y_1 = \text{Walsh sequence output} \]
\[ y_2 = \text{Output bit clock (impulse train)} \]
\[ y_3 = \text{Frame clock (indicates beginning of new row) (impulse)} \]

\[
H_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad H_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \quad H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & H_N \end{bmatrix}
\]

Matrix Output Row
Specifies which row \( (K) \) of the \( (N \times N) \) 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 $x_1$ 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.

\[ y_1 = \text{Output waveform} \]
\[ y_2 = \text{Output clock (impulse train)} \]

**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 algorithm. 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 $\sqrt{K}/2$. 

---

168
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 \( \frac{1}{2} \) and identical. The overall rate of the turbo code is \( \frac{2}{3} \) unless puncturing is used to delete every-other parity bit, in which case the rate is \( \frac{1}{2} \).

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 coded 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.

\[ x_1 = \text{Input frame clock} \]
\[ x_2 = \text{Input coded data vector (size } n \text{)} \]
\[ x_3 = \text{Input reference data vector (size } k \text{)} \text{ [Only used in Perfect Halting mode]} \]
\[ y_1 = \text{Output frame clock} \]
\[ y_2 = \text{Decoded output vector (size} \ k \text{)} \]
\[ y_3 = \text{Number of executed decoding iterations} \]
\[ y_4 = \text{Minimum Log Likelihood Ratio (LLR) value} \]
\[ y_5 = \text{Number of detected bit errors [Only used in Perfect Halting mode]} \]

**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 through 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.

\[ x_1 = \text{Input frame clock} \]
\[ x_2 = \text{Input data vector (size } k \text{)} \]
\[ y_1 = \text{Output frame clock} \]
\[ y_2 = \text{Encoded output vector (size } n \text{)} \]

**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 puncturing 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 $M \times N$ matrix into $N$ or less independent column vectors. This block produces an updated output each time a “high” input clock is present.

$$x_1 = \text{Input clock (high } \geq 1 \text{) (pulse)}$$

$$x_2 = \text{Input Matrix [size } M \times N \text{]}$$

$$y_1 = \text{Output clock (pulse)}$$

$$y_{2...N+1} = \text{Output column vectors [size } M \text{]}$$

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.

$$x_1 = \text{Input clock (high } > 0.5 \text{) (pulse)}$$

$$x_2 = \text{Input column vector [size } L \text{]}$$

$$y = \text{Output column vector [size } N \text{]}$$

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 = $[ 1, 0, 1, 1, 0, 0, 1 ]^T$
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 = $[ a_1, a_2, a_3, a_4, a_5 ]^T$

w/o padding - Output #1 = $[ a_1, a_3, a_5 ]^T$ Output #2 = $[ a_2, a_4 ]^T$
w/ padding - Output #1 = $[ a_1, a_3, a_5 ]^T$ Output #2 = $[ a_2, a_4, pad ]^T$

$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.

$x_1 = \text{Input clock (high > 0.5) (pulse)}$

$x_2 = \text{Input column vector #1 [size } L]$

$x_3 = \text{Input column vector #2 [size } M]$

$y = \text{Output signal vector [size } N]$

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 $N \times L$, where $N$ is the number of input vectors. The block operates by taking elements from each input vector in round robin fashion.

Example:

$\text{Vector A} = [ a1, a2, a3 ]^T$

$\text{Vector B} = [ b1, b2, b3 ]^T$

$\text{Output} = [ a1, b1, a2, b2, a3, b3 ]^T$

$x_1 = \text{Input clock (high } \geq 1) \text{ (pulse)}$

$x_2 \ldots X_{N+1} = \text{Input column vectors [size } L]$

$y_1 = \text{Output clock (pulse)}$

$y_2 = \text{Output column vector [size } M=\text{N x L]}$

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:

$\text{Input Vector} = [ 5, 2, 11 ]^T$

3 bits/symbol, MSB mode
Chapter 3  Comm Block Set

\[
\text{Output Vector} = [1, 0, 1, 0, 1, 0, 0, 1, 1]^T
\]

Note: Only the lower 3 bits of “11” (1011) were output

\(x_1 = \) Input clock (high \(\geq 1\)) (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 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 \(M \times N\), where \(N\) is the number of input vectors.

\(x_1 = \) Input clock (high \(\geq 1\)) (pulse)

\(x_{2...N+1} = \) Input column vectors [size \(M\)]

\(y_1 = \) Output clock (pulse)

\(y_2 = \) Output matrix [size \(M \times N\)]

**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).
Appendix A

VisSim/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.

\[
x = \text{Input signal (complex)} \\
y_1 = \text{Output signal (complex)} \\
y_2 = \text{VCO output signal (complex)}
\]

**COSTAS_R.VSM**
This compound block is identical to the COSTAS_C.VSM compound block, except that it accepts a modulated real signal.

\[
x = \text{Input signal (real)} \\
y_1 = \text{Output signal (complex)} \\
y_2 = \text{VCO output signal (complex)}
\]
**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.

\[ x = \text{Input data signal (binary)} \]
\[ y = \text{Complex output signal [Re, Im]} \]

**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.

\[ x = \text{Input data signal (binary)} \]
\[ y = \text{Output signal (complex)} \]

**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.

\[ x = \text{Input signal (complex)} \]
\[ y_1 = \text{VCO output signal (complex)} \]
\[ y_2 = \text{VCO phase (radians)} \]

**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.

\[ x = \text{Input signal (real)} \]
\[ y_1 = \text{VCO output signal (real)} \]
\[ y_2 = \text{VCO phase (radians)} \]

**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).

\[ x = \text{Input signal (complex)} \]
\[ y_1 = \text{VCO output signal (complex)} \]
\[ y_2 = \text{VCO phase (radians)} \]

**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.

\[ x = \text{Input signal (real)} \]
\[ y_1 = \text{VCO output signal (real)} \]
\[ y_2 = \text{VCO phase (radians)} \]

**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.

\[ x_1 = \text{Input signal (complex)} \]
\[ x_2 = \text{Reference power level} \]
\[ y = \text{Output signal (complex)} \]
\[ y = G(r)e^{j\Phi(r)}x_1 \]

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.

\[ x_1 = \text{Input symbol} \]
\[ x_2 = \text{Clock} \]
\[ y = \text{Output symbol} \]

**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.

\[ x_1 = \text{Input symbol} \]
Appendix A  VisSim/Comm Library

\[ x_2 = \text{Clock} \]
\[ y = \text{Output symbol} \]

**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.

\[ x = \text{Input signal} \]
\[ y = \text{Pulse train output (0, 1)} \]

---

**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.”
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 possess 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 \( \frac{1}{2} \) turbo codes are capable of achieving the arbitrarily small Bit Error Rate (BER) of \( 10^{-5} \) at an \( E_b/N_0 \) ratio of just 0.7 decibels, which is just \( \frac{1}{2} \) dB away from the Shannon capacity for BPSK and rate \( \frac{1}{2} \). 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 \( E_b/N_0 \), 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\textsuperscript{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\textsuperscript{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\textsuperscript{3}. The Viterbi

\textsuperscript{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.

\textsuperscript{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.

\textsuperscript{3} \url{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 \(\alpha\). 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 1/2 systematic codes is \(r = \frac{1}{3}\). This code rate can be made higher by puncturing\(^4\). The code rate of a turbo code is often increased to \(r = \frac{1}{2}\) 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 \(r = \frac{1}{2}, K = 5\) 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 \( \frac{1}{3} \) 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.
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 $[PER96]$

$$P_b \approx w_{\text{eff}} Q \left( \frac{d_{\text{free}}}{2rE_b / N_o} \right),$$

where $d_{\text{free}}$ is the free distance of the code, and $w_{\text{eff}}$ is the effective multiplicity of code words of weight $d_{\text{free}}$. 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 has $d_{\text{free}}$ of 18, while for the $K=5$ turbo code, $d_{\text{free}}$ 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, $w_{\text{eff}}$ of the convolutional code is 187 while $w_{\text{eff}}$ for the turbo code is only $6/L$. For $L = 65,536$ data bits per frame, this translates to $w_{\text{eff}} \approx 0.0001$, 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 $\frac{1}{2}$, constraint length 5, and 65,536 data bits per block) and the $K=15$ convolutional code (also rate $\frac{1}{2}$) 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, 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 $E_b/N_0$ 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 $E_b/N_0$ than it does for PCCC’s, the error floor for SCCC’s is virtually nonexistent. The current release of the VisSim 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 $\frac{1}{2}$, 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 $40 \leq L \leq 5114$. 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 $r = \frac{1}{3}$, 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 $X_{L+1}, Z_{L+1}, X_{L+2}, X_{L+3}, Z_{L+2}, Z_{L+3}, Z'_{L+1}, Z'_{L+2}, X'_{L+1}, X'_{L+2}, Z'_{L+3}$, 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 $\sigma^2 = (3L+12)/(2L(E_b/N_0))$, 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 $Z'_k$. 

---

193
It is important to note that for the VisSim/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 $\text{est}(X_k) = 1$ when $\Lambda_2(X_k) > 0$ and $\text{est}(X_k) = 0$ when $\Lambda_2(X_k) \leq 0$.

**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:

\[
\text{max}^*(x, y) = \max(x, y) + \ln\left(1 + \exp\left(-|y - x|\right)\right)
\]

\[
= \max(x, y) + f_c(|y - x|),
\]

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.

Max-log-MAP algorithm

With the max-log-MAP algorithm, the Jacobi algorithm is loosely approximated using:

\[
\text{max}^*(x, y) = \max(x, y),
\]

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 $\sigma^2$).

**Constant-log-MAP algorithm**

The constant-log-MAP algorithm, approximates the Jacobi algorithm using [GRO98]:

$$\max^* (x, y) = \max(x, y) + \begin{cases} 0 & \text{if } |y - x| > T \\ C & \text{if } |y - x| \leq T, \end{cases}$$

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]:

$$\max^* (x, y) = \max(x, y) + \begin{cases} 0 & \text{if } |y - x| > T \\ a(|y - x| - T) & \text{if } |y - x| \leq T, \end{cases}$$

where the values of $a$ and $T$ are chosen to minimize the total squared error between the exact correction function and its linear approximation:

$$\int_0^\infty \left[ a(z - T) - \ln(1 + \exp(-z)) \right]^2 dz + \int_0^\infty \left[ \ln(1 + \exp(-z)) \right]^2 dz$$

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 $\frac{1}{2}$, 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 VisSim/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 $\hat{\beta}_k(S_i)$ with $1 \leq k \leq L+3$ and $0 \leq i \leq 7$. The backward recursion is initialized with $\hat{\beta}_{L+3}(S_0) = 0$ and $\hat{\beta}_{L+3}(S_i) = -\infty \forall i > 0$.

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:

$$\hat{\beta}_k(S_i) = \max \left\{ \left( \beta_{k+1}(S_j) + \gamma_{ij} \right) \left( \beta_{k+1}(S_{j_1}) + \gamma_{j_1i} \right) \right\},$$
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 \( \beta_k(S_i) \), \( 1 \leq i \leq 7 \), 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 \( 0 \leq k \leq L-1 \) and \( 0 \leq i \leq 7 \). The forward recursion is initialized by setting \( \alpha_0(S_0) = 0 \) 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:

\[
\tilde{\alpha}_k(S_j) = \max_* \left\{ \left( \alpha_{k-1}(S_{i_1}) + \gamma_{i_1j} \right) \left( \alpha_{k-1}(S_{i_2}) + \gamma_{i_2j} \right) \right\},
\]

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 \( \alpha \)'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:

\[
\tilde{\lambda}_k(i, j) = \alpha_{k-1}(S_i) + \gamma_{ij} + \beta_k(S_j)
\]

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:

\[
\Lambda(X_k) = \max_* \left\{ \tilde{\lambda}_k(i, j) \right\} - \max_* \left\{ \tilde{\lambda}_k(i, j) \right\},
\]

where the max* operator is computed recursively over the likelihoods of all data 1 branches \( (S_i \rightarrow S_j; X_k=1) \) or data 0 branches \( (S_i \rightarrow S_j; X_k=0) \). 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 \( \alpha_k \) 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 = \( 10^{-2} \), 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 \( E_b/N_0 = 0.4 \) 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, $\Lambda_T$, i.e. the decoder halts once

$$\min_{1 \leq k \leq L} |\Lambda_2(X_k)| > \Lambda_T.$$

The performance of the stopping condition is highly dependent on the choice of $\Lambda_T$ (called Halt LLR in VisSim/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 $\Lambda_T$. 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 $\Lambda_T$ 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


[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.
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 likelihood ratio (internal decoder metric).
Appendix B  Turbo Codes Overview
### Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-PSK</td>
<td>16-phase shift keying</td>
</tr>
<tr>
<td>16-QAM</td>
<td>16-level quadrature amplitude modulation</td>
</tr>
<tr>
<td>256-QAM</td>
<td>256-level quadrature amplitude modulation</td>
</tr>
<tr>
<td>32-QAM</td>
<td>32-cross quadrature amplitude modulation</td>
</tr>
<tr>
<td>64-QAM</td>
<td>64-level quadrature amplitude modulation</td>
</tr>
<tr>
<td>8-PSK</td>
<td>eight-phase shift keying</td>
</tr>
<tr>
<td>ADC</td>
<td>analog-to-digital converter</td>
</tr>
<tr>
<td>AGC</td>
<td>automatic gain control</td>
</tr>
<tr>
<td>AM</td>
<td>amplitude modulation</td>
</tr>
<tr>
<td>ASK</td>
<td>amplitude shift keying</td>
</tr>
<tr>
<td>AWGN</td>
<td>additive white Gaussian noise</td>
</tr>
<tr>
<td>BER</td>
<td>bit error rate</td>
</tr>
<tr>
<td>BPSK</td>
<td>binary phase shift keying</td>
</tr>
<tr>
<td>BSC</td>
<td>binary symmetric channel</td>
</tr>
<tr>
<td>DOS</td>
<td>disk operating system</td>
</tr>
<tr>
<td>DPSK</td>
<td>differential phase shift keying</td>
</tr>
<tr>
<td>DQPSK</td>
<td>differential quadrature phase shift keying</td>
</tr>
<tr>
<td>DSB-AM</td>
<td>double-sideband amplitude modulation</td>
</tr>
<tr>
<td>DTTL</td>
<td>data transition tracking loop</td>
</tr>
<tr>
<td>FIFO</td>
<td>first in first out</td>
</tr>
<tr>
<td>FIR</td>
<td>finite impulse response</td>
</tr>
<tr>
<td>FM</td>
<td>frequency modulation</td>
</tr>
<tr>
<td>FSK</td>
<td>frequency shift keying</td>
</tr>
<tr>
<td>IIR</td>
<td>infinite impulse response</td>
</tr>
<tr>
<td>IM</td>
<td>intermodulation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>ISI</td>
<td>inter-symbol inference</td>
</tr>
<tr>
<td>LIFO</td>
<td>last in first out</td>
</tr>
<tr>
<td>LSB</td>
<td>least significant bit</td>
</tr>
<tr>
<td>MSB</td>
<td>most significant bit</td>
</tr>
<tr>
<td>MSK</td>
<td>minimum shift keying</td>
</tr>
<tr>
<td>OQPSK</td>
<td>offset quadrature phase shift keying</td>
</tr>
<tr>
<td>PAM</td>
<td>pulse amplitude modulation</td>
</tr>
<tr>
<td>PLL</td>
<td>phase-locked loop</td>
</tr>
<tr>
<td>PM</td>
<td>phase modulation</td>
</tr>
<tr>
<td>PN</td>
<td>pseudo noise</td>
</tr>
<tr>
<td>PPM</td>
<td>pulse position modulation</td>
</tr>
<tr>
<td>PRBS</td>
<td>pseudo random binary sequence</td>
</tr>
<tr>
<td>QAM</td>
<td>quadrature amplitude modulation</td>
</tr>
<tr>
<td>QPSK</td>
<td>quadrature phase shift keying</td>
</tr>
<tr>
<td>SER</td>
<td>symbol error rate</td>
</tr>
<tr>
<td>SNR</td>
<td>signal to noise ratio</td>
</tr>
<tr>
<td>SQPSK</td>
<td>staggered quadrature phase shift keying</td>
</tr>
<tr>
<td>TWTA</td>
<td>traveling wave tube amplifier</td>
</tr>
<tr>
<td>VCO</td>
<td>voltage controlled oscillator</td>
</tr>
<tr>
<td>VCPG</td>
<td>voltage controlled pulse generator</td>
</tr>
</tbody>
</table>
Appendix D

Installing VisSim/Comm

This appendix describes how to install VisSim/Comm.

**Computer requirements**

VisSim/Comm 9.0 runs on personal computers running the Windows operating system. To use this program, your computer must have the following components:

- WinXP/ Vista / Win7 /Win8
- 12 MB free disk space

**VisSim/Comm Installation**

To install the VisSim/Comm software, follow the instruction provided on the distribution media or accompanying installation procedures documentation.

**Readme files**

As part of the installation procedure, two readme files are copied to your distribution media: a VisSim readme file named README.TXT and a Comm-specific readme file named COMMREAD.WRI. These files contain information on last minute enhancements and corrections that were not available when the manuals went to print. For your convenience you should read these files immediately and print a copy of them to keep with your manuals.
Index

128-QAM modulation, 114
16-PAM modulation, 115
16-PSK modulation, 110
16-QAM modulation, 113

256-QAM modulation, 114

32-PSK modulation, 110
32-QAM modulation, 113

4-PAM modulation, 114

64-QAM modulation, 113

8-PAM modulation, 115
8-PSK modulation, 110

A
A/D Converter, 118
Abbreviations, 205
Accumulate & Dump, 38
Acronyms, 205
Adaptive Equalizer (Complex or Real), 74
ADC, 118
AM Modulator, 98
Amplifier, 139
Antenna, 140
ASK Modulator, 98
Attenuator, 141
Average Power (Complex or Real), 64
AWGN (Complex or Real), 21
AWGN channel model, 193

B
Backward recursion, 197
Baseband equivalent systems, 3
BER Control (# Errors), 65
BER Curve Control, 66
BER Curve Display, 97
BER curves, 12
  example simulation, 17
genrating, 16
Binary Counter, 39
Binary Symmetric Channel, 22
Bit/Symbol Error Rate, 67
Bits to Symbol, 39
Block connectors, 8
Block Interleaver, 49
Blocks
  A/D converter, 118
  accumulate & dump, 38
  adaptive equalizer (complex), 74
  adaptive equalizer (real), 74
  addition (complex), 32
  AM modulator, 98
  amplifier, 139
  antenna, 140
  ASK modulator, 98
  attenuator, 141
  average power (complex), 64
  average power (real), 64
  AWGN (Complex), 21
  AWGN (Real), 21
  BER control (# errors), 65
  BER curve control, 66
  BER curve display, 97
  binary counter, 39
  binary symmetric channel, 22
  bit/symbol error rate, 67
  bits to symbol, 39
  block interleaver, 49
  buffer, 40
  cable, 141
  charge pump, 134
  clock edge, 117
  clock extend, 117
compander, 119
complex conjugate, 32
complex division, 32
complex exponential, 120
complex FFT/IFFT, 120
complex inverse, 32
complex multiplication, 33
complex power, 33
complex square root, 33
complex to mag/phase, 33
complex to real/imag, 33
complex tone, 151
coupler, 142
D flip flop, 41
D/A converter, 124
delay (complex), 124
delay (real), 125
delay estimator, 69
depuncture, 52
differential PSK detector, 34
differential PSK modulator, 99
discrete equalizer (complex), 77
discrete equalizer (real), 77
divide by N, 41
double balanced mixer, 143
event time, 69
file correlation, 69
file data, 152, 164
file FIR filter, 79
file write, 148
final value, 150
FIR filter, 80
Fixed Point FIR filter, 91
Fixed Point IIR filter, 94
Fixed Point VCO (complex), 96
Fixed Point VCO (real), 96
FM demodulator, 35
FM modulator, 102
frequency counter, 71
frequency sweep, 153
FSK modulator, 102
gain (dB), 125
GFSK modulator, 103
GMSK modulator, 103
Gray map, 53
Gray reverse map, 53
Hamming decoder, 53
Hamming encoder, 54
how to insert, 8
IIR filter, 81
impulse, 154
impulse train, 154
integrate & dump (complex), 126
integrate & dump (real), 126
interpolator, 117
IQ Detector, 35
IQ mapper, 127
Jakes mobile, 23
JK flip flop, 42
loop filter (2nd order PLL), 135
loop filter (3rd order PLL), 137
mag/phase to complex, 34
MagPhase filter, 82
Manchester encoder, 54
matrix to vector, 176
max index, 128
mean, 71
median, 72
minMax, 72
mobile fading, 23
modulators
   IQ modulator, 104
modulo, 128
MSK modulator, 104
multipath, 24
mux/demux, 43
noise, 154
oscilloscope, 129
oscilloscope display, 97
packet timing, 43
parallel to serial, 45
parameter settings, 10
phase rotate, 130
phase unwrap, 130
PM modulator, 105
PN sequence, 155
Poisson arrivals, 157
polynomial, 130
PPM demodulator, 36
PPM modulator, 106
propagation loss, 25
PSK detector, 37
PSK modulator, 107
pulse extend, 45
pulse shaping filter, 84
puncture, 55
QAM/PAM detector, 37
QAM/PAM modulator, 111
queue, 46
random distribution, 157
random symbols, 159
real/imag to complex, 34
rectangular pulses, 159
Reed-Solomon decoder, 57
Reed-Solomon encoder, 58
RF conversions, 144
RF gain, 145
Rice/Rayleigh fading, 26
Rummler multipath, 26
Saleh-Valenzuela, 27
sampling file FIR filter, 85
sampling FIR filter, 86
serial to parallel, 46
sinusoid, 160
spectral mask, 161
spectrum (complex), 131
spectrum (real), 131
spectrum analyzer display (complex), 97
spectrum analyzer display (real), 97
splitter/combiner, 146
SQPSK modulator, 115
state machine, 47
subsample, 133
subvector, 176
switch, 147
symbol to bits, 48
trellis decoder, 60
trellis encoder, 60
TWTA (analytical), 29
TWTA (table lookup), 31
type-2 phase detector, 137
type-3 phase detector, 138
type-4 phase detector, 138
unbuffer, 49
variable attenuator, 148
variable spaced equalizer (complex), 89
variable spaced equalizer (real), 89
variance, 73
VCO (complex), 162
VCO (real), 162
vector AWGN, 31
vector bits to symbol, 176
vector constant, 162
vector correlation, 73
vector demux, 177
vector FFT, 133
vector merge, 178
vector mux, 178
vector symbol to bits, 178
vector to matrix, 179
Viterbi decoder (hard), 62
Viterbi decoder (Soft), 62
Walsh sequence, 163
wave write, 150
waveform generator, 165
weighted mean, 74
BPSK modulation, 109
Branch metrics, 196
Buffer, 40

Cable, 141
Channel blocks
  AWGN (Complex or Real), 21
  binary symmetric channel, 22
  Jakes mobile, 23
  mobile fading, 23
  multipath, 24
  propagation loss, 25
  Rice/Rayleigh fading, 26
  Rummler multipath, 26
  Saleh-Valenzuela, 27
  TWTA (analytical), 29
  TWTA (table lookup), 31
  vector AWGN, 31
Channel, comm system element, 2
Charge Pump, 134
Chirp generator, 153
Circular buffer, 40
Clock Edge, 117
Clock Extend, 117
Combiner, see Splitter/Combiner, 146
Comm block listings, 21
Communication blocks, summary of, 3
Communication system key elements, 1
Compander, 119
Complex Addition, 32
Complex Conjugate, 32
Complex Correlation, 68
Complex Division, 32
Complex envelope notation, 3
Complex Exponential, 120
Complex FFT/IFFT, 120
Complex Inverse, 32
Complex math blocks
  addition, 32
  complex to mag/phase, 33
  complex to real/imag, 33
  conjugate, 32
  division, 32
  inverse, 32
  mag/phase to complex, 34
  multiplication, 33
  power, 33
  real/imag to complex, 34
  square root, 33
Complex Multiplication, 33
Complex Power, 33
Complex Square Root, 33
Complex to Magnitude/Phase, 33
Complex to Real/Imaginary, 33
Complex Tone, 151
Compound blocks
  Costas loop (complex), 181
Costas loop (real), 181
first order PLL (complex), 182
first order PLL (real), 182
GFSK modulator, 182
GMSK modulator, 182
second order PLL (complex), 182
second order PLL (real), 183
TWTA (table lookup), 183
V.32 differential decoder, 183
V.32 differential encoder, 183
voltage controlled pulse generator, 184
Connecting blocks, 8
connectors
labels, 8
optional, 8
unconnected, 8
Constant-log-MAP algorithm, 196
Conventions used in manual, xi
Conversions, 122
Convolutional codes, 187
Convolutional Encoder, 50
Convolutional Interleaver, 51
Correlation, 68
Cosine wave generator, 160
Costas loop
complex, 181
real, 181
Coupler, 142
Cross Correlation, 68

D
D Flip Flop, 41
D/A Converter, 124
DAC, 124
Data source, comm system element, 1
dBM 1 Ohm to 50 Ohm, 145
dBM 50 Ohm to 1 Ohm, 145
dBM to dBW, 144
dBM/Hz to dBM, 144, 145
dBM/Hz to deg K, 145
dBW to dBM, 144
Decibels to power, 122
Decibels to real, 122
Decoder, comm system element, 2
Decoding algorithm comparison, 198
deg K to dBM/Hz, 145
Degrees to radians, 122
Delay (Complex), 124
Delay (Real), 125
Delay Estimator, 69
Demodulator blocks
differential PSK, 34
FM demodulator, 35
IQ detector, 35
PPM demodulator, 36
PSK detector, 37
QAM/PAM detector, 37
Demodulator, comm system element, 2
Depuncture, 52
Diagram timing, 9
Differential phase shift keying, 99
Differential PSK Detector, 34
Differential PSK Modulator, 99
Digital blocks
accumulate & dump, 38
binary counter, 39
bits to symbol, 39
buffer, 40
D flip flop, 41
divide by N, 41
JK flip flop, 42
mux/demux, 43
packet timing, 43
parallel to serial, 45
pulse extend, 45
queue, 46
serial to parallel, 46
state machine, 47
symbol to bits, 48
unbuffer, 49
Discrete Equalizer (Complex or Real), 77
Divide by N, 41
Double Balanced Mixer, 143
DPSK, 99
DQPSK, 99
Dynamic halting, 200

E
Encode / Decode blocks
block interleaver, 49
convolutional encoder, 50
convolutional interleaver, 51
depuncture, 52
Gray map, 53
Gray reverse map, 53
Hamming decoder, 53
Hamming encoder, 54
Manchester encoder, 54
puncture, 55
Reed-Solomon decoder, 57
Reed-Solomon encoder, 58
trellis decoder, 60
trellis encoder, 60
Viterbi decoder (hard), 62
Viterbi decoder (soft), 62
Encoder, comm system element, 2
Estimator blocks
average power (complex or real), 64
BER control (# errors), 65
BER curve control, 66
bit/symbol error rate, 67
correlation, 68
delay estimator, 69
event time, 69
file correlation, 69
frequency counter, 71
mean, 71
median, 72
minMax, 72
variance, 73
vector correlation, 73
weighted mean, 74
Event Time, 69
Example Turbo Code simulation, 202
Eye plot, 13

Fading channels
Jakes mobile, 23
mobile fading, 23
multipath (fixed), 24
Rice/Rayleigh fading, 26
Rummler multipath, 26
Fast Fourier transform, 133
File Correlation, 69
File Data, 152, 164
File FIR Filter, 79
File Write, 148
Filter blocks
adaptive equalizer (complex or real), 74
discrete equalizer (complex or real), 77
file FIR, 79
FIR filter, 80
IIR filter, 81
MagPhase, 82
pulse shaping filter, 84
sampling file FIR, 85
sampling FIR, 86
variable spaced equalizer (complex or real), 89
Filter Viewer, 15
Filters
gain response, 15
group delay response, 15
impulse response, 15
phase response, 15
Final Value, 150
FIR Filter, 80
Fixed point blocks
Fixed Point FIR filter, 91
Fixed Point IIR filter, 94
Fixed Point VCO (complex or real), 96
Fixed Point FIR Filter, 91
Fixed Point IIR Filter, 94
Fixed Point VCO (Complex or Real), 96
FM Demodulator, 35
FM Modulator, 102
Forward recursion, 198
Fractional part, see Modulo, 128
Frequency Counter, 71
Frequency domain plot, 13
Frequency Sweep, 153
G
Gain (dB), 125
Gaussian filter, 80, 84
Gaussian noise, 21, 31
GFSK Modulator, 103
GFSK modulator compound block, 182
Global variables, 10
GMSK Modulator, 103
GMSK modulator compound block, 182
Gray Decoder, 53
Gray Encoder, 53
Gray Map, 53
Gray Reverse Map, 53
H
Hamming Decoder, 53
Hamming Encoder, 54
Hertz to rad/sec, 122
Hilbert filter, 80, 84
I
IIR Filter, 81
Impulse, 154
Impulse Train, 154
Index, 209
Indoor channels
Saleh-Valenzuela, 27
Inserting blocks, 8
Installation
computer requirements, 207
procedure, 207
Instruments blocks
BER curve display, 97
oscilloscope display, 97
spectrum analyzer display (complex), 97
spectrum analyzer display (real), 97
Integrate & Dump (Complex or Real), 126
Integration methods, recommended settings, 11
Interpolator, 117
Introduction, 1
IQ Detector, 35
IQ Mapper, 127
IQ Modulator, 104
IQ scatter plot, 14

J
Jakes Mobile, 23
JK Flip Flop, 42

L
Linear-log-MAP algorithm, 196
LLR calculation, 198
Local rate compound blocks, 12
Local time step, specifying, 12
Log-MAP algorithm, 195
Loop Filter (2nd Order PLL), 135
Loop Filter (3rd Order PLL), 137
Loop filters, see PLL blocks, 135
Lowpass equivalent systems, 3
LTE Turbo Decoder, 166
LTE Turbo Encoder, 167

M
Mag/phase to real/im, 123
Magnitude/Phase to Complex, 34
MagPhase Filter, 82
Manchester Encoder, 54
MAP algorithm in log domain, 196
Matrix to Vector, 176
Max Index, 128
Max* function, 194
Max-log-MAP algorithm, 195
Mean, 71
Median, 72
MinMax, 72
Mobile Fading, 23
Modulator blocks
128-QAM, 114
16-PAM, 115
16-PSK, 110
16-QAM, 113
256-QAM, 114
32-PSK, 110
32-QAM, 113
4-PAM, 114
64-QAM, 113
8-PAM, 115
8-PSK, 110
AM, 98
ASK, 98
BPSK, 109
differential PSK, 99
FM, 102
FSK, 102
GFSK, 103
GMSK, 103
IQ, 104
MSK, 104
OQPSK, 115
PM, 105
PPM, 106
PSK, 107
QAM/PAM, 111
QPSK, 109
SQPSK, 115
Modulator, comm system element, 2
Modulo, 128
MSK Modulator, 104
Multipath, 24
Multirate diagrams, 11
Multirate support blocks
clock edge, 117
clock extend, 117
interpolator, 117
Multirun simulations, 17
Mux/Demux, 43

N
Noise, 154
Non-selective fading channel, 26
Nyquist filter, 84

O
Online help, xi
Operator blocks
A/D converter, 118
comphander, 119
complex exponential, 120
complex FFT/IFFT, 120
conversions, 122
D/A converter, 124
delay (complex), 124
delay (real), 125
gain (dB), 125
integrate & dump (complex or real), 126
IQ mapper, 127
max index, 128
modulo, 128
oscilloscope, 129
phase rotate, 130
phase unwrap, 130
polynomial, 130
spectrum (complex or real), 131
subsample, 133
Vector FFT, 133
OQPSK modulator, 115
Oscilloscope, 129
Oscilloscope Display, 97
Packet Timing, 43
Parallel to Serial, 45
PBRS sequence, 155
Phase locked loops, see PLL blocks, 135
Phase Rotate, 130
Phase scatter plot, 14
Phase Unwrap, 130
Phase-locked loops
   Costas (complex), 181
   Costas (real), 181
   first order PLL (complex), 182
   first order PLL (real), 182
   second order PLL (complex), 182
   second order PLL (real), 183
   voltage controlled pulse generator, 184
Pi/4-DQPSK, 99
PLL blocks
   charge pump, 134
   loop filter (2nd order PLL), 135
   loop filter (3rd order PLL), 137
   type-2 phase detector, 137
   type-3 phase detector, 138
   type-4 phase detector, 138
Plots
   available types, 12
   BER curves, 12
   eye, 13
   filter response, 15
   frequency domain, 13
   phase scatter, 14
PM Modulator, 105
PN Sequence, 155
Poisson Arrivals, 157
Polynomial, 130
Power to decibels, 123
PPM Demodulator, 36
PPM Modulator, 106
Preface, xi
Propagation Loss, 25
Pseudo noise generation, 155
PSK Detector, 37
PSK Modulator, 107
Pulse amplitude modulation, 111
Pulse Extend, 45
Pulse position modulation, 106
Pulse Shaping Filter, 84
Puncture, 55
Queue, 46
Rad/sec to hertz, 123
Radians to degrees, 123
Raised cosine filter, 80, 84
Random Distribution, 157
Random numbers, 9
Random seed, 9
Random Seed (Obsolete), 158
Random Symbols, 159
Range checking, 10
Read me files, xi
Real to decibels, 123
Real/im to mag/phase, 123
Real/Imaginary to Complex, 34
Rectangular Pulses, 159
Reed-Solomon Decoder, 57
Reed-Solomon Encoder, 58
References, 201
RF blocks
   amplifier, 139
   antenna, 140
   attenuator, 141
   cable, 141
   coupler, 142
   double balanced mixer, 143
   RF conversions, 144
   RF gain, 145
   splitter/combiner, 146
   switch, 147
   variable attenuator, 148
RF conversions
   dBm 1 Ohm to 50 Ohm, 145
   dBm 50 Ohm to 1 Ohm, 145
   dBm to dBm/Hz, 144
   dBm to dBW, 144
   dBm/Hz to dBm, 145
   dBm/Hz to deg K, 145
   dBW to dBm, 144
   deg K to dBm/Hz, 145
RF Conversions, 144
RF Gain, 145
Rice/Rayleigh Fading, 26
Root raised cosine filter, 80, 84
Rummler Multipath, 26
Saleh-Valenzuela, 27
Sampling File FIR Filter, 85
Sampling FIR Filter, 86
Serial to Parallel, 46
Signal sink blocks
   file write, 148
Index

final value, 150
Wave write, 150
Signal sink, comm system element, 2
Signal source blocks
complex tone, 151
file data, 152, 164
frequency sweep, 153
impulse, 154
impulse train, 154
noise, 154
PN sequence, 155
Poisson arrivals, 157
random distribution, 157
random symbols, 159
rectangular pulses, 159
sinusoid, 160
spectral mask, 161
VCO (complex or real), 162
vector constant, 162
Walsh sequence, 163
waveform generator, 165
Sine wave generator, 160
Sinusoid, 160
Spectral Mask, 161
Spectrum (Complex or Real), 131
Spectrum Analyzer Display (Complex), 97
Spectrum Analyzer Display (Real), 97
Splitter/Combiner, 146
SQPSK Modulator, 115
Staggered quadrature phase shift keying, 115
Starting VisSim/Comm, 7
State Machine, 47
Subsample, 133
SubVector, 176
Switch, 147
Symbol to Bits, 48

T
TC Interleaver Generator, 168
Technical support, xiii
Traveling Wave Tube Amplifier (analytical), 29
Traveling Wave Tube Amplifier (table lookup), 31
Trellis Decoder, 60
Trellis Encoder, 60
Trellis structure, 196
Turbo Code Decoder, 169
Turbo Code Encoder, 172
Turbo Code origins, 187
Turbo Code performance factors, 189
Turbo decoder architecture, 194
Turbo decoding, 193
Turbo encoding, 189
TWTA channel (analytical), 29
TWTA channel (table lookup), 31, 183
Type-2 Phase Detector, 137
Type-3 Phase Detector, 138
Type-4 Phase Detector, 138

U
UMTS Turbo Code, 193
UMTS Turbo Decoder, 174
UMTS Turbo Encoder, 175
Unbuffer, 49
Unit conversions
decibels to power, 122
decibels to Real, 122
degrees to radians, 122
degrees to radians, 122
hertz to rad/sec, 122
mag/phase to real/im, 123
power to decibels, 123
rad/sec to hertz, 123
radians to degrees, 123
real to decibels, 123
real/im to mag/phase, 123

V
V.32 differential decoder, 183
V.32 differential encoder, 183
Variable Attenuator, 148
Variable spaced Equalizer (Complex or Real), 89
Variance, 73
VCO (Complex or Real), 162
Vector AWGN, 31
Vector Bits to Symbol, 176
Vector connectors, 8
Vector Constant, 162
Vector Correlation, 73
Vector Demux, 177
Vector FFT, 133
Vector Merge, 178
Vector Mux, 178
Vector operator blocks
matrix to vector, 176
subvector, 176
vector bits to symbol, 176
vector demux, 177
vector merge, 178
vector mux, 178
vector symbol to bits, 178
vector to matrix, 179
Vector Symbol to Bits, 178
Vector to Matrix, 179
VisSim/Comm
compound block library, 181
data files, 184
environment, 7
example simulation, 19
installing, 187, 207
multirate diagrams, 11
starting, 7
VisSim/Comm Overview, 7
VisSim/Comm Personal Edition, xii
Viterbi Decoder (Hard), 62
Viterbi Decoder (Soft), 62
Voltage controlled pulse generator, 184

W
Walsh Sequence, 163
Wave Write, 150
Waveform Generator, 165
Weighted Mean, 74