commSIM 7

Network & Communications Simulation

User Guide & Block Reference

Electronics
WORKBENCH

CS7-E-1721 Rev. 1

Preface

Welcome to Commsim 7. Commsim provides the ideal solution for designing and simulating analog and digital end-to-end communication links. The Commsim library supports digital and analog modulation, channel modes, demodulation, phase locked loops, error correcting codes, and bit error rate analysis, to mention a few.

Through its support of complex math, Commsim enables the use of complex envelope simulations. By using lowpass equivalent models, you can significantly reduce the computing load required to support most communication analysis problems.

This guide provides a detailed discussion of advanced topics for users of Commsim as well as a complete reference of the Comm and Core Blocks available in Commsim. Depending on your version of Commsim, your software may have some feature limitations. See the section entitled *Version Details* (in the *Commsim 7 Getting Started Guide*) for more information.

Conventions used in this book

The following typographical conventions are used in this manual:

Convention	Where it's used
Shortcut key combinations	Shortcut key combinations are joined with the plus sign (+). For example, the command CTRL+C means to hold down the CTRL key while you press the C key.
Hot keys	Hot keys are the underlined keys in Commsim's menus, commands, and dialog boxes. To use a hot key, press ALT and then the key for the underlined character. For instance, to execute the File menu's Save command, hold down the ALT key while you press the F key, then release both keys and press the S key.
SMALL CAPS	To indicate the names of the keys on the keyboard.
ALL CAPS	To indicate directory names, file names, and acronyms.
Initial Caps	To indicate menu names, commands names, and dialog box options.

In addition, unless specifically stated otherwise, when you read "click the mouse…" or "click on…," it means click the left mouse button.

What’s new in Commsim 7

Feature	Function
Multiple open diagrams	Switch to different diagram windows easily.
Close diagram	Close a diagram without quitting Commsim.
Save embedded files	Save the changes to embedded diagram files.
Save all diagrams	Save changes to all open diagrams.
Pop-up Edit menu	Easy access to frequently used Edit menu commands.
Multiple level Undo	Undo previous editing actions.
Auto wiring	Allows bulk connection of aligned connectors.
Align blocks	Group block alignment commands.
Navigation tags	Go to specific diagram locations quickly.
Report generation	Create statistical reports for block diagrams.
Optimize matrix usage	Speed up simulations.
Compare diagrams	Mark differences in two diagrams.
Complex number support	Support of complex numbers in `const`, `mul`, `div`, `add`, `gain`, `merge`, `case`, `unitDelay`, and `sampleHold` blocks.
Matrix generation	Use of diag (), eye (), ones (), and zeros () as input to `const` and `expression` blocks.
Matrix manipulation	`reshape` block
Matrix operations	`complexToReIm` and `magPhase` blocks
Event-driven simulation	`stateTransition` block
Vector notation for 1/Z block	Set initial condition as scalar or vector value.
Subplots	Create multiple plot windows.
Signal generation blocks	`triangleWave, squareWave,` and `sawtooth` blocks
Dialog tables	Automatic dialog creation from data table.

References to other books

For information on	Refer to
Block diagram modeling and simulation	Karayanakis, Nicholas M., *Computer-Assisted Simulation of Dynamic Systems with Block Diagram Languages.* CRC Press, 1993.
Scientific computing	Abramowitz, M.; Stegun, I. A. *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55, Washington: National Bureau of Standards; reprinted Dover Publications, New York, 1968.
	D'Azzo, John J.; Houpis, Constantine H. *Linear Control System Analysis & Design - Conventional and Modern.* McGraw-Hill Book Company, 1988.
	Fitzgerald, A. E.; Kingsley, Charles Jr.; Umans, Stephen D. *Electric Machinery.* McGraw-Hill Book Company, 1983.
	Flannery, B. P.; Press, W. H.; S. A.; Vetterling, W. T. *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, 1989.
	Franklin, Gene F.; Powell, David J. *Digital Control of Dynamic Systems*. Addison-Wesley Publishing Company, 1980.
	Gear, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, 1971.
	Stoer, J.; Bulirsh, R. *Introduction to Numerical Analysis.* New York: Springer-Verlag, 1980.
Computer Programming	Darnell, Peter A.; Margolis, Philip E. *C: A Software Engineering Approach*. Springer-Verlag, 1990.
Industrial Controls	McMillan, Gregory. *Tuning and Control Loop Performance: A Practitioner's Guide*. ISA, 1993.

Getting help

For information on getting help, refer to the *Commsim 7 Getting Started Guide*.

Technical support

For information on technical support, refer to the *Commsim 7 Getting Started Guide*.

License Agreement

Please read the license agreement included in the *Commsim 7 Getting Started Guide* carefully before installing and using the software contained in this package. By installing and using the software, you are agreeing to be bound by the terms of this license. If you do not agree to the terms of this license, simply return the unused software within ten days to the place where you obtained it and your money will be refunded.

Table of Contents

1. Introduction

2. Solving Implicit Equations

3. Performing Global Optimization

4. Working with Large Diagrams

5. Designing Digital Filters

6. Working with Other Applications

7. Wireless Lan

8. Comm Block Set

9. Core Block Reference

10. Customizing Commsim

11. Extending the Block Set

12. Commsim Library

13. Sample Block Diagrams

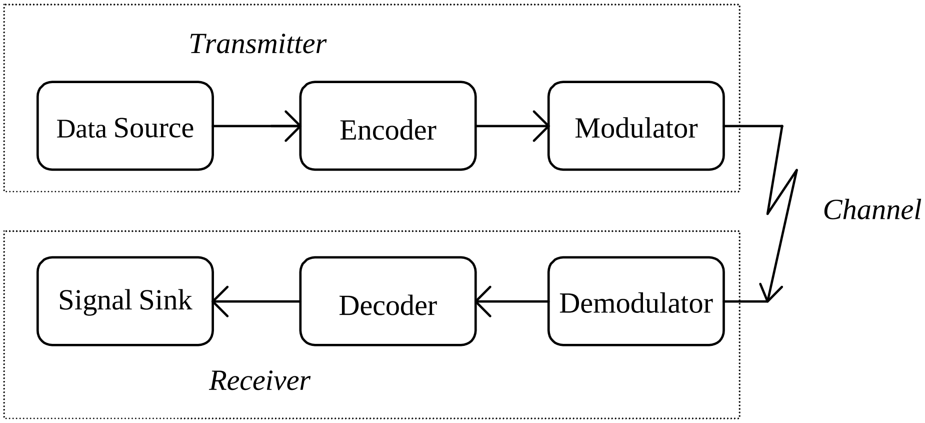
14. Acronyms and Abbreviations

Chapter 1
Introduction

The following are described in this chapter.

1.1 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. 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.

1.2 Lowpass equivalent systems

The sampling requirements of a given simulation can be reduced through the use of *complex envelope notation*. When a data signal is modulated, a sampling rate of at least eight samples per carrier cycle is usually selected in order for the simulation to accurately represent the carrier signal. This means that the simulation step size is dictated by the carrier frequency and not the data rate, which may be several orders of magnitude lower. When a simulation is transformed to a lowpass (or baseband) equivalent, its carrier frequency is mathematically translated (shifted) to 0 Hz, and the simulation sampling interval can then based on the sampling requirements of the data signal. The result is a reduction in execution time without loss of simulation accuracy. A brief description follows.

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

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

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

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

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

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

and

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

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

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

1.3 Communication blocks

To allow you to easily simulate a communication link, Commsim 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.

The growing list of Comm blocks supports most disciplines within communications. Below is a brief summary of the blocks available within each category. Blocks marked with an asterisk may not be available in all editions of Commsim.

Channels

AWGN (Complex); AWGN (Real); Binary Symmetric Channel; Jakes Mobile; Mobile Fading Channel; Multipath; Propagation Loss; Rayleigh/Rice Fading; Rummler Multipath; TWTA; and Vector AWGN.

Complex Math

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.

Decoders and Encoders

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)*.

Demodulators

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

Digital Elements

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

Estimators

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

Filters

Adaptive Equalizer (Complex); Adaptive Equalizer (Real); File FIR; FIR; IIR; MagPhase Filter; Pulse Shaping Filter*; Sampled FIR*; and Sampled File FIR*.

Instruments

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

Modulators (Complex and Real)

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

Multirate Support

Clock Edge*; Clock Extend*; and Interpolator*.

Operators

A/D Converter; Compander; Complex Exponential; Complex FFT,IFFT; Conversions; 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); and 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; and Type-4 Phase/Freq Detector.

RF Elements

Amplifier; Attenuator; Coupler*; Double Balanced Mixer*; Splitter/ Combiner; Switch; and Variable Attenuator*.

Signal Sources

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

Vector Operators

`Matrix to Vector`; `SubVector`; `Vector Bits to Symbols`; `Vector Demux`; `Vector Mux`; `Vector Merge`; `Vector Symbols to Bits`; and `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 are typically read-only but can be modified by first creating a new copy and saving under a new name.

Chapter 2
Solving Implicit Equations

The following are described in this chapter.

2.1 Setting up an implicit equation

When a system contains an implicit equation, that is, an equation defined in terms of itself, you use `unknown` and `constraint` blocks to solve it. There may be one, more than one, or no solutions for the system.

The key steps to setting and solving implicit equations follow:

1. Define the variable that needs to be determined as an unknown using the `unknown` block. The order is very important — an unknown must be defined first and then given a variable name.
2. Isolate zero on the right-hand side of the equation by moving all terms to the left-hand side.
3. Construct the left-hand side of the equation, and equate the right-hand side by using the `constraint` block to denote zero.

Algebraic loops

In the case of connections backward to earlier blocks already evaluated (often called feedback), Commsim checks to see that such feedback loops contain at least one `integrator`, `transferFunction`, `unitDelay`, or `timeDelay` block. If there is no such block in the feedback, the result is numerically ill-defined and is referred to as an algebraic loop. Commsim detects such algebraic loops and produces a warning message.

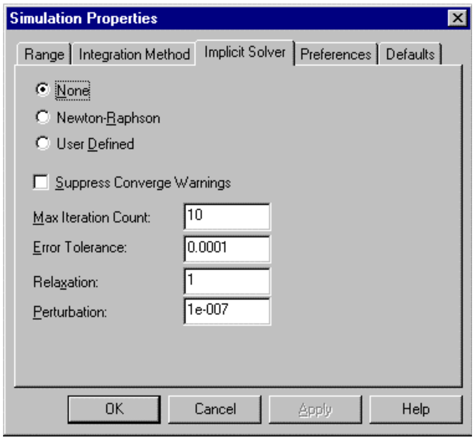
2.2 Solving an implicit equation

When solving an implicit equation, you can use the Newton Raphson solver or a custom solver. You can also set the error tolerance, maximum iteration count, perturbation, and relaxation parameters.

➢ To solve an implicit equation:

1. Choose Simulate > Simulation Properties.
2. Click on the Implicit Solver tab.

   The Implicit Solver sheet in the Simulation Properties dialog box appears.



● None
○ Newton-Raphson
○ User Defined

□ Suppress Converge Warnings

Max Iteration Count: 10
Error Tolerance: 0.0001
Relaxation: 1
Perturbation: 1e-007

3. Choose the options you want, then click the OK button, or press ENTER. (For more information on the options, see the descriptions below.)
4. Start the simulation.

2.2.1 Using the Implicit Solver property sheet

The Implicit Solver property sheet provides options for selecting the solver, suppressing convergence warnings, specifying the maximum iteration count, and more. The property sheet options are:

None: When you're not solving an implicit equation, activate None.

Newton-Raphson: Newton-Raphson is a singular value decomposition (SVD) based solver that performs static optimization at each time step. Commsim derives an *n*-dimensional slope by numerically perturbing the `unknown` outputs and observing the effects on the `constraints`. Commsim uses the slope matrix to compute values for the `unknown` blocks that drive the `constraints` to a minimum. Newton-Raphson is particularly useful for solving static equations in the presence of concurrent dynamics.

User Defined: When User Defined is activated, Commsim uses the DLL file named VSOLVER.DLL in your current directory to solve the equation.

Suppress Convergence Warnings: If you're solving for a set of roots that are equidistant from zero, you must initialize the `unknown` block to a value other than zero to force the equations to converge. To suppress the convergence warnings when solving an implicit system with nonlinear dynamics, activate Suppress Convergence Warnings.

Max Iteration Count: This option lets you specify the number of iterations the solver performs while attempting to meet the error tolerance criterion. The default value is 10.

Error Tolerance: This option lets you specify the maximum allowable difference in total error between two successive iterations. A total error is computed as the sum of individual errors squared, where the error signal is the input signal to a `constraint` block. Newton-Raphson ceases iterating when the difference in total error between two successive iterations becomes less than the tolerance.

Use Error Tolerance in conjunction with Max Iteration Count to control the time spent converging. The larger the tolerance, the quicker and less accurate the solution. The default value is 0.0001.

Relaxation: This option attenuates the iteration update value to attain convergence for equations that prove difficult to converge. As a side effect, it slows the convergence process because it forces the iteration to take smaller steps. The typical range is from slightly greater than 0 to 2. Select values less than 1 for systems that appear to be unstable. The default value is 1.

Perturbation: This option indicates the value by which the `unknown` blocks are numerically perturbed to evaluate the Jacobian (matrix of first partials). Each element of the Jacobian is a ratio of constraint change with respect to block perturbation value applied to the `unknown` blocks. The perturbation value should be at least one order of magnitude less than the `unknown` initial value, but greater than 1e-12 of the initial value for the `unknowns`. The default value is 1e-007.

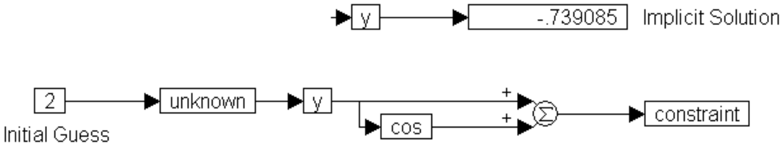
2.3 Implicit equation examples

2.3.1 Simple nonlinear implicit equation

Consider the equation:

$$y + \cos(y) = 0$$

This equation can be realized as:



In this configuration, the output of the `summingJunction`, namely *y* + cos(*y*) must equal zero. Further, from left to right, the entire diagram reads

> Starting with an initial guess of 2, find a value for the `unknown variable` called *y* such that *y* + cos(*y*) = 0.

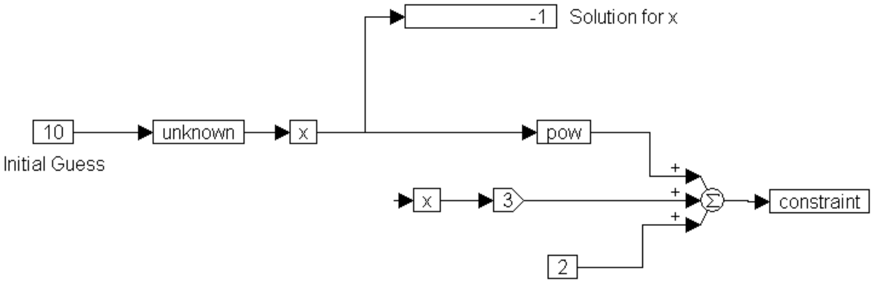
The result indicates that *y* = -0.739085 is one possible solution. Other solutions may exist and can be identified by using different initial guess values for the `unknown` block.

2.3.2 Advanced nonlinear implicit equation

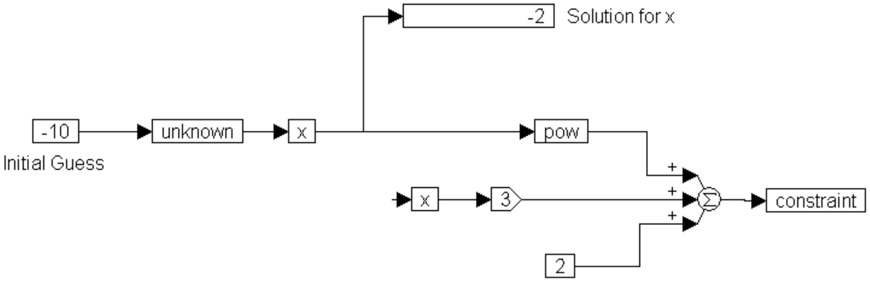
Consider the equation:

$$x^2 + 3x + 2 = 0$$

The roots of this equation can be obtained analytically as $x = -2$ and $x = -1$ for comparison. This equation can be solved implicitly in Commsim as follows.



In this configuration, the output of the `summingJunction`, namely $x^2 + 3x + 2$, must equal zero. The simulation yields one of the roots to be $x = -1$, as shown, starting from a guess of +10. When a guess value of -10 is used, the diagram becomes:



In this case, the second root, $x = -2$ is obtained.

This model shows that the solution obtained depends on the initial guess supplied to the `unknown` block. When the initial guess value is larger than -1, the solution converges to -1. When the initial guess value is smaller than -2, the solution converges to -2.

Chapter 3
Performing Global Optimization

The following are described in this chapter.

3.1 Global optimization basics

Global optimization involves the automatic adjustment of system parameters to maximize or minimize a specified quantity, while satisfying one or more global constraints.

During global optimization, Commsim iteratively updates the parameter vector such that the cost function generally decreases until it finds a minimum. The resulting parameter values become the optimum values because they minimize the cost function.

3.1.1 Cost functions with many local minimum values

Most cost functions will have many local minimum values and although Commsim tries to avoid local minima, the one Commsim finds may not be the overall minimum. To be sure that it is the global minimum, you may want to perform several runs of the optimizer, using different initial `parameterUnknown` values.

3.1.2 Cost functions with no minimum values

It is possible that a cost function has no minimum, or has a flat surface away from the minima. In this case, the global optimizer will get confused and wander aimlessly (*how can you run downhill when there is no hill?*). If the optimizer appears to run for a long time with little convergence, you should suspect flat spots in your cost function. In such cases, you may have to reformulate the cost function such that it has at least one minimum. A common mistake is to put a `limit` block just before the cost input. In this case, the optimizer will experiment with larger and larger unknown values to no avail. If a `limit` block is used in the cost function, it must be placed before an integration of total error.

3.2 Performing global optimization

Global optimization is almost always a nonlinear problem and rarely is there a single best method for minimizing the cost function. Commsim provides three optimization methods: Powell, Polak-Ribiere, and Fletcher Reeves.

Regardless of the method you select, Commsim produces a sequence of parameter updates on a per-run basis that decreases the value of the cost function. The basic parameter update equation is:

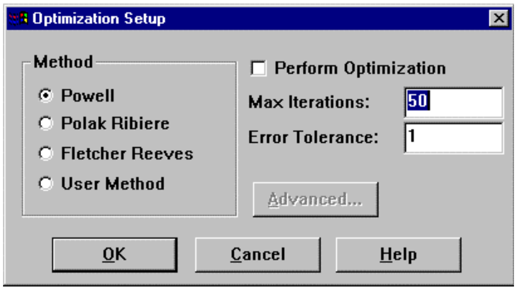
$$P_{k+1} = P_k + \Delta P_k \equiv \text{iteration index or C}\textit{ommsim}\text{ run number}$$

The difference between each method is the way ΔP_k is generated. For more information on these methods, see *Numerical Recipes*, *The Art of Scientific Computing* (Cambridge University Press).

➢ To perform global optimization:

1. Choose Simulate > Optimization Properties.

The Optimization Properties dialog box appears.



Optimization Setup

Method

● Powell
○ Polak Ribiere
○ Fletcher Reeves
○ User Method

□ Perform Optimization

Max Iterations: 50

Error Tolerance: 1

2. Activate the Perform Optimization option.
3. Choose the options you want, then click on the OK button, or press ENTER. (For more information on the options, see the descriptions below.)
4. Click on the OK button, or press ENTER.
5. Start the simulation.

3.2.1 Using the Optimization Properties dialog box

The Optimization Properties dialog box provides the following options:

Method: You can choice between three supplied optimizers or use a custom optimizer.

- **Powell:** A direction-set algorithm that typically runs faster because it does not explicitly calculate the gradient.
- **Polak Ribiere:** A conjugate gradient algorithm that is a bit more sophisticated than Fletcher Reeves for arriving at the supposed minimum of the quadratic form.
- **Fletcher Reeves:** Requires fewer iterations to convergence. This conjugant gradient algorithm is slower than Powell’s method.
- **User Method:** When User Method is activated, Commsim uses the DLL file named VOPT.DLL in your current directory to solve the equation.

Perform Optimization: This option must be activated to perform global optimization.

Max Iteration Count: Indicates the maximum number of iterations.

Error Tolerance: Indicates the maximum error between the results of two successive iterations. The default value is 10.

3.3 Global optimization examples

3.3.1 Optimized paper bag problem

Suppose you want to manufacture paper grocery bags at the lowest possible cost, where each bag has a volume of at least one cubic foot (1728 cubic inches). To minimize the cost of material, you must determine the optimal bag dimensions that minimize the amount of paper used for each bag, while ensuring that the volume of the bag ($v = whd$) is greater than or equal to 1728 cubic inches. To simplify the problem, additional material for folding and gluing the bag is ignored.

The cost function can be expressed as:

$s = 2(wh + dh) + dw$

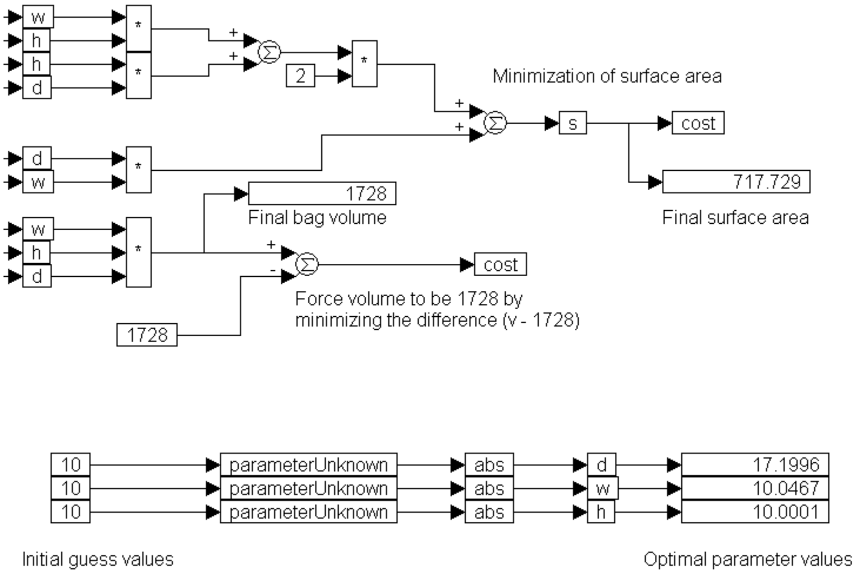
where s is the surface area of the bag, w is its width, h is its height, and d is its depth.

The cost function is also subject to a given volume constraint:

Volume $v = whd \geq 1728$ cubic inches

Though not explicitly specified, it is clear that each of the physical dimensions w, h, and d must be greater than zero.

To solve this problem in Commsim, construct the following block diagram:



Three `const` blocks, all set to 10, are used to produce the initial guess values. Their outputs are connected to three `parameterUnknown` blocks. To enforce the requirement that the physical dimensions of the bag are positive, the outputs of the `parameterUnknown` blocks are connected to three `abs` blocks, and the outputs of the `abs` blocks are connected to three `variable` blocks named *d*, *w*, and *h*. The outputs of the three `variables` are connected to `display` blocks to monitor the changes and the final values of the bag dimensions.

The outputs of *w*, *h*, and *d* are connected to a three-input * block, and the output of the * block is connected to a `summingJunction` and to a `display` block. A `const` block set to 1728 is connected to the other negated input of the `summingJunction`. A `cost` block, connected to the output of the `summingJunction` block, enforces the minimization of the difference *v* - 1728. This is equivalent to forcing the volume *v* to be very close to the desired value of 1728 cubic inches.

In order to minimize the surface area, the equation $s = 2(wh + dh) + dw$ is coded using `variable`, `*`, `summingJunction`, and `const` blocks. The output of the final `summingJunction` block is connected to a `variable` *s*, which is connected to another `cost` block and to a `display` block.

Optimization is performed using the Powell method. The number of iterations is set to 50 and the error tolerance is 0.001. The optimization results indicate that a volume of 1728 cubic inches can be attained by using an ideal surface area of 717.7 square inches, with the final bag dimensions of $w = 10.05$, $h = 10.00$, and $d = 17.20$ inches. In comparison, the intuitive solution of $w = h = d = 12$ inches has a surface area of 720 square inches.

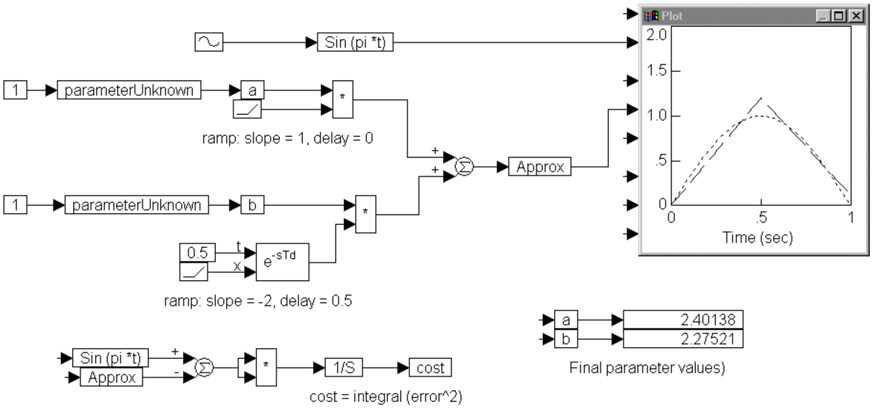
3.3.2 Two segment approximation of sin(π*t*)

Consider the problem of approximating a sinusoid sin(π*t*) in the range (0,1) by two straight line segments. In the range (0,0.5), the line segment has a positive slope, and in the range (0.5, 1.0), the second line segment has a negative slope. The approximation can be written as:

$$\sin(\pi t) \cong ar(t) - 2br(t - 0.5)$$

where *a* and *b* are unknown weight factors, $r(t)$ is a unit ramp that starts at $t = 0$, with a slope of +1, and $-2r(t - 0.5)$ is a ramp that starts at $t = 0.5$ with a slope of -2.

The optimization problem in this case involves the determination of optimal values for the unknown weight factors a and b. This equation can be realized as:



In this configuration, two `const` blocks, both set to 1, provide the initial guess values to two `parameterUnknown` blocks. The outputs of the `parameterUnknown` blocks are connected to `variable` blocks *a* and *b,* thereby defining *a* and *b* to be unknown parameters. The output of *a* is connected to a `*` block. A `ramp` block, with a slope of 1 and no delay, is connected to the other input of the `*` block.

To generate a ramp of slope -2 that is delayed by 0.5 sec, a `const` block set to 0.5 is wired to the time delay input of a `timeDelay` block and a `ramp` block with a slope of -2 is wired to the main signal input of the `timeDelay`. The output of the `timeDelay` block is connected to one input of a `*` block and the other input is connected to the output of *b*.

The outputs of the two `*` blocks are connected to a `summingJunction`, and the output of the `summingJunction` is connected to a `variable` *Approx*. The output of a `sinusoid` block, set to an amplitude of 1 and a frequency of π radians/sec, is connected to a `variable` named *Sin(pi*t)*. The output of *Sin(pi*t)* and *Approx* are connected to a `plot` block to observe the results.

The `cost` function is constructed using a `summingJunction` to calculate the difference *Sin(pi*t) - Approx*. The output of the `summingJunction` is connected to both inputs of a `*` block to compute the squared error value. The output of the `*` block is connected to an `integrator` block to compute the integrated squared error, and the output of the `integrator` block is connected to a `cost` block, thereby defining the integrated squared error to be the cost or objective function that needs to be minimized by the optimization process. Two `display` blocks are connected to the outputs of *a* and *b* to observe the final values computed by the optimizer for the two weight factors.

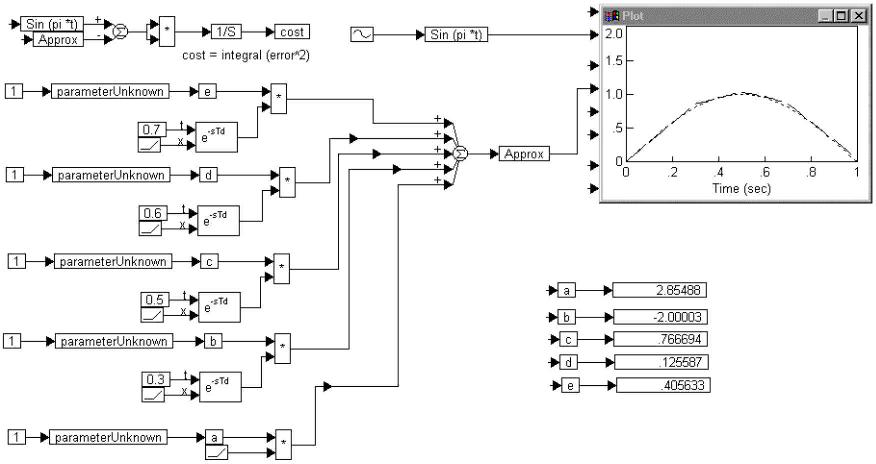
Using the Fletcher-Reeves optimization method, with maximum iterations set to 300 and error tolerance of 0.0001, the values of *a* and *b* are obtained as 2.39 and 2.28, respectively.

3.3.3 Five segment approximation of sin(πt)

By modifying the above problem, consider the usage of five line segments instead of two. The approximation can be written as:

$$\sin(\pi t) \cong ar(t) + br(t-0.3) - 2cr(t-0.5) + 3dr(t-0.6) - 4er(t-0.7)$$

This can be realized as shown in the diagram below.



The delayed versions of ramp signals are constructed using a `ramp` block with the correct slope connected to the main signal input of a `timeDelay` block, and a `const` block with the correct delay value connected to the time delayed input of the `timeDelay` block.

Five `parameterUnknown` blocks, with initial guesses set to 1, define the `variable` blocks *a*, *b*, *c*, *d*, and *e* to be unknown parameters. The outputs of the five * blocks are connected to a five-input `summingJunction` block. The output of the `summingJunction` is connected to a `variable` named *Approx*. The output of a `sin` block with an amplitude of 1 and frequency of π radians/sec is connected to a `variable` named *Sin(pi*t)*. The outputs of *Approx* and *Sin(pi*t)* are connected to a `plot` block.

The cost function is evaluated by computing the integral squared error. In this case, the optimization process yields the parameter values to be *a* = 2.85488, *b* = -2.00003, *c* = 0.766694, *d* = -0.125587, and *e* = 0.405633. From the `plot` block, it is also clear that the five-segment approximation is quite close to the original sinusoid.

3.4 Troubleshooting

How do I avoid system instability?

You should limit the cost calculation because, during optimization, some parameters may be supplied with values that drive the system into instability. The resulting large cost value can cause the optimization method to fail to converge due to the limited range of floating point numbers.

When limits are used, they must occur before the integration of the square of the error so that onset of saturation is numerically reflected in the cost function. In this way, onset of saturation is reflected in the cost value and gives the optimizer a slope to follow down.

What do I set the initial tolerance to when I know little about optimal parameter values?

Use an initial tolerance value of 10 in the Optimization Properties dialog box when you know very little about the optimal parameter values; otherwise, the algorithm will take a very long time to search a short distance in parameter space.

Once optimal values are found, the `parameterUnknowns` can be reinitialized with the new optimal values and the optimization can be rerun with a lower tolerance.

Though the algorithm tries to avoid local minima, to verify that the values found are optimal, run the optimizer with different initial values supplied to the `parameterUnknowns`.

Chapter 4
Working with Large Diagrams

The following are described in this chapter.

4.1 Creating model hierarchy

Compound blocks allow you to encapsulate one or more blocks in a single block. This gives you more flexibility in constructing and editing your block diagram models, especially if they are complex. The top-level blocks display major component connectivity, leaving the underlying levels to describe the logic of each component.

Compound blocks also encourage a modular approach to large model construction by allowing you to design and test functionally independent subcomponents concurrently. Then using the `embed` block or the File > Add command, you can incorporate each subcomponent back into the large system diagram.

If your compound block contains sensitive information, you can prevent other users from viewing the compound block by locking it closed. You can alternatively apply read-only attributes to the compound block. This lets other users view the contents of the compound block but denies them the ability to edit it.

If you choose to identify a compound block by name, you can change the name with the Edit > Block Properties command.

This section describes how to:

- Create compound blocks
- Drill into compound blocks
- Automatically configure compound blocks

- Trigger the execution of compound blocks
- Check the number of blocks encompassed in compound blocks
- Hide compound blocks
- Configure pictures on compound blocks
- Label connector tabs on compound blocks
- Dissolve compound blocks

4.1.1 Creating a compound block

When you create a compound block, Commsim attaches connector tabs to the compound block for each of the following situations:

- All unsatisfied connector tabs on the internal blocks (except global variables)
- All satisfied connector tabs to external blocks

You can have as many levels as you want in a compound block. (The number is limited only by your system resources.)

➢ To create a compound block:

1. Select the blocks to be encapsulated.
2. Do one of the following:
   - Choose Edit > Create Compound Block**.**
   - Click the right mouse button over a selected block and choose Create Compound from the pop-up menu.
3. Under compound name, enter a name. Avoid using the dot (.) character in the name; Commsim uses it to separate compound block names in the title bar. The default name is *Compound.*
4. Click on the OK button, or press ENTER.

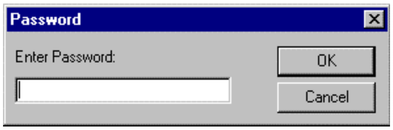
4.1.2 Drilling into a compound block

The process of moving through and displaying the levels of a compound block is referred to as *drilling.* As you drill into a compound block, Commsim adds the name of the compound block to the title bar to help you keep track of where you are.

➢ To drill down:

1. Point to a compound block.
2. Click the right mouse button.

3. If the compound block is password-protected, enter the password in the Password dialog box, then click on the OK button, or press ENTER.



The compound block remains unlocked until you close the diagram.

➢ To pop up:

1. Point to empty screen space.
2. Click the right mouse button.

4.1.3 Automatically configuring a compound block

By using a compound block's Auto Dialog dialog box, you can speed up tasks, such as setting, viewing, selecting, and changing user-defined parameter values encapsulated in the compound block.

The Auto Dialog dialog box presents all the `dialogConstant` blocks in the compound block, along with their current values. The changes you make in the Auto Dialog dialog box are automatically applied to the specific blocks. This means you no longer have to drill into a compound block and change each block parameter.

By displaying only `dialogConstant` blocks, the Auto Dialog dialog box helps ensure that only relevant information is exposed to other users. This greatly reduces the chances of accidental or unnecessary changes to the compound block.

There are three basic steps in creating Auto Dialog dialog boxes:

1. Insert and configure the `dialogConstant` blocks for the Auto Dialog dialog box.
2. Encapsulate the `dialogConstant` blocks in a compound block.
3. Activate the Auto Dialog dialog box.

4.1.3.1 Using dialogConstants to create Auto Dialog dialog boxes

A `dialogConstant` block is the only type of block that can appear in an Auto Dialog dialog box. A `dialogConstant` block has three properties — a name, a data type, and a value — that control how the `dialogConstant` block is presented in the Auto Dialog box.

You can have multiple `dialogConstant` blocks with different data types within a single Auto Dialog dialog box. Commsim automatically adjusts the size of the Auto Dialog dialog box to accommodate all the `dialogConstants`.

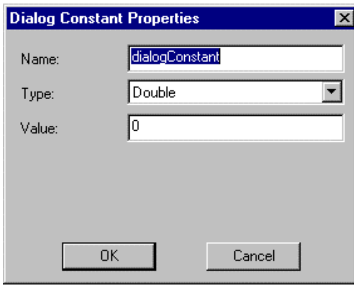
➢ To insert a dialogConstant block:

1. Choose Blocks > Signal Producer; then click on the `dialogConstant` block.
2. Click the mouse where you want to insert the `dialogConstant` block.

➢ To set up dialogConstant block properties:

1. Choose Edit > Block Properties.
2. Click on the `dialogConstant` block.

   The following dialog box appears:



Dialog Constant Properties

Name: dialogConstant

Type: Double

Value: 0

3. Make the following entries:
   - In the Name box, enter the name to be displayed on the block and in the resulting Auto Dialog dialog box. When you do not supply a name, Commsim uses *dialog constant*.
   - In the Type box, click on the DOWN ARROW and select the appropriate data type. The data type controls how the data associated with the `dialogConstant` block will be displayed in the Auto Dialog box. Your choices are shown in the table below.

Type	Definition	How the data is displayed
Integer List	Integer data	Creates a drop-down list box. To create entries into the list box, enter them in the Value text box, as described below.
Double	Double precision floating point data	Creates a static text box in which numeric values can be entered. To create an entry in the text box, enter it in the Value text box, as described below.
Boolean	Boolean	Creates a check box that can be turned on or off. To set its initial value, enter it in the Value text box, as described below.

- In the Value box, enter the following:

If the data type is	Enter
Integer List	Each entry for the drop-down list on a separate line. Entries appear in the drop-down list in the order they are entered in the Value box. The default entry is the first entry.
Double	A numeric value that will appear in the static text box.
Boolean	1 to turn on check box option; 0 to turn off check box option.

Once you insert a `dialogConstant` block, you can change its value through the Auto Dialog dialog box.

4.1.3.2 Encapsulating dialogConstant blocks in a compound block

You can insert `dialogConstant` blocks into existing compound blocks, or create a compound block from existing `dialogConstant` blocks. In addition, `dialogConstant` blocks can occupy multiple layers in a compound block and still appear in the Auto Dialog dialog box.

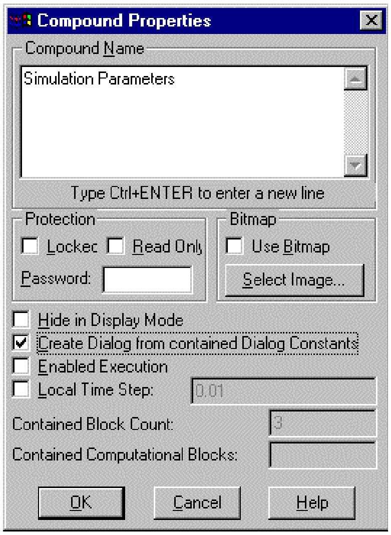
4.1.3.3 Invoking the Auto Dialog dialog box

To invoke the Auto Dialog dialog box for a compound block, you must activate the Create Auto Dialog Box From Contained dialogConstants options in the Compound Properties dialog box. When this option is activated, you can no longer drill into the compound block. If there are no `dialogConstants` in a compound block, the Create Auto Dialog Box From Contained dialogConstants option is dimmed.

➢ To switch between displaying the Auto Dialog dialog box from a compound block and drilling into the compound block:

1. Choose Edit > Block Properties.
2. Click on the compound block.

The Compound Properties dialog box appears.



Compound Properties

Compound Name: Simulation Parameters

Type Ctrl+ENTER to enter a new line

Protection: □ Lockec □ Read Onl

Password:

Bitmap: □ Use Bitmap

□ Hide in Display Mode

☑ Create Dialog from contained Dialog Constants

□ Enabled Execution

□ Local Time Step: 0.01

Contained Block Count: 3

Contained Computational Blocks:

3. Do one of the following:

To	Do this
Display the Auto Dialog dialog box	Activate the Create Dialog From Contained Dialog Constants option.
Enable drilling into the compound block	De-activate the Create Dialog From Contained Dialog Constants option.

4. Click on the OK button, or press ENTER.

5. Click the right mouse button over the compound block.

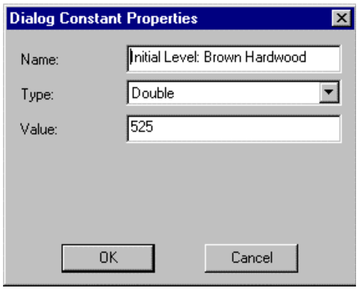
   If the Auto Dialog dialog box has been enabled, Commsim displays an Auto Dialog dialog box containing all the `dialogConstants` contained in the compound block. If the Auto Dialog dialog box has been disabled, Commsim displays the next lower level of the compound block.

4.1.3.4 Creating an Auto Dialog dialog box: an example

The following example steps you through the process of creating an Auto Dialog dialog box. In this exercise, you'll create an Auto Dialog dialog box in which you can control the value of a variable.

1. Insert a `dialogConstant` block into a block diagram.
2. Choose Edit > Block Properties and click on the `dialogConstant` block.

   The Dialog Constant Properties dialog box appears.
3. Create a static text like the one shown below.



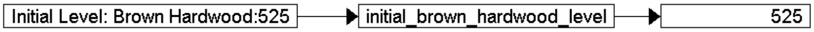
Dialog Constant Properties

Name: Initial Level: Brown Hardwood

Type: Double

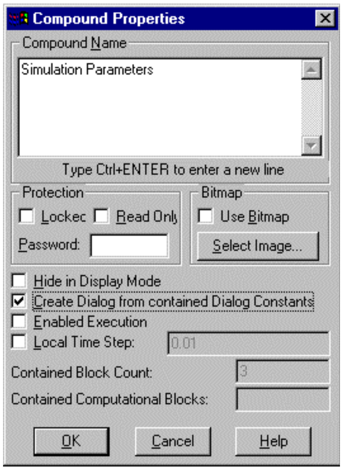
Value: 525

4. Click on the OK button, or press ENTER.
5. Feed the dialogConstant block through a `variable` named initial_brown_hardward_level block and into a `display` block.


Initial Level: Brown Hardwood:525
initial_brown_hardwood_level
525

6. Encapsulate the three blocks in a compound block named Simulation Parameters.
7. Choose Edit > Block Properties and click on the newly-created compound block.

   The following dialog box appears:



Compound Properties

Compound Name: Simulation Parameters

Type Ctrl+ENTER to enter a new line

Protection: □ Lockec □ Read Only

Password:

Bitmap: □ Use Bitmap

□ Hide in Display Mode

☑ Create Dialog from contained Dialog Constants

□ Enabled Execution

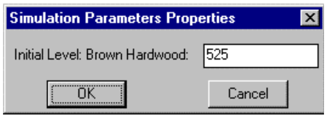
□ Local Time Step: 0.01

Contained Block Count: 3

Contained Computational Blocks:

8. Activate the Create Dialog From Contained Dialog Constants option, and click on the OK button, or press ENTER.
9. To change the value of the dialogConstant, choose Edit > Block Properties and click on the compound block.

   The following dialog box appears:

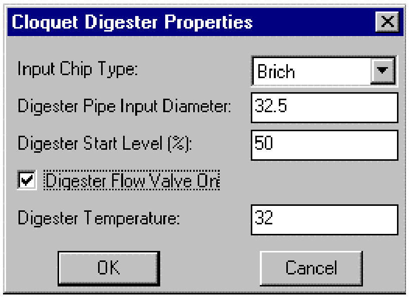

Simulation Parameters Properties

Initial Level: Brown Hardwood: 525

10. Enter a new value in the text box.

4.1.3.5 Creating more complex Auto Dialog dialog boxes

An Auto Dialog box can contain many `dialogConstant` block. For example, the Auto Dialog dialog box shown below contains five `dialogConstant` blocks of varying data types.


Cloquet Digester Properties

Input Chip Type: Brich

Digester Pipe Input Diameter: 32.5

Digester Start Level (%): 50

☑ Digester Flow Valve On

Digester Temperature: 32

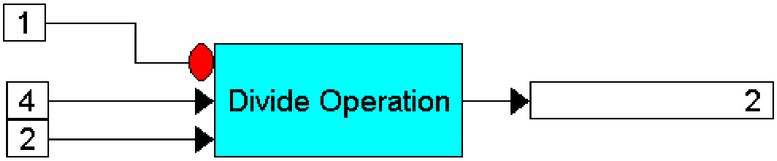
The `dialogConstants` displayed in the above dialog box are:

Name of dialogConstant	Data Type	Value
Input Chip Type	Integer List	Birch
Digester Pipe Input Diameter	Double	32.5
Digester Start Level (%)	Double	50
Digester OutFlow Valve On	Boolean	1
Digester Temperature	Double	32

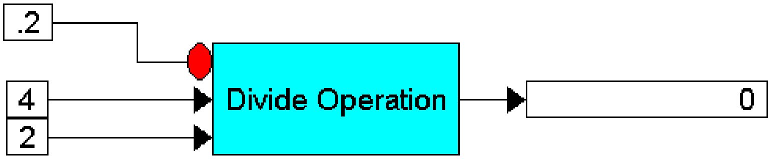
4.1.4 Triggering the execution of compound blocks

You can control when and whether Commsim executes a compound block by using an external Boolean trigger on the compound block. The trigger appears as a red ball attached to the input side of the compound block. When the input to the trigger is greater than or equal to one, the trigger is enabled and the blocks within the compound block are executed. Conversely, when the input to the trigger is less than one, the trigger is disabled and the blocks within the compound block are not executed. When there is no input to the trigger, the blocks within the compound block are not executed.

As a simple example, suppose you want to control when Commsim executes a divide operation. You can set up a diagram like the one shown below:



Here, the compound block performs a divide operation on the input signals. When a value of 1 is fed into the trigger, the divide operation is to be performed. Output from the compound block shows the quotient. If, however, the trigger input is .2, the divide operation is not performed, and the output from the compound block is 0.



With triggered compound blocks, you can speed up your simulation by encapsulating those portions of the diagram that are used infrequently in the compound block. You can then trigger the execution of the compound block only under specific conditions.

➢ To enable and disable a trigger:

1. Choose Edit > Block Properties.
2. Point to the compound block and click the mouse.
3. The Compound Properties dialog box appears.

4. Do one of the following:

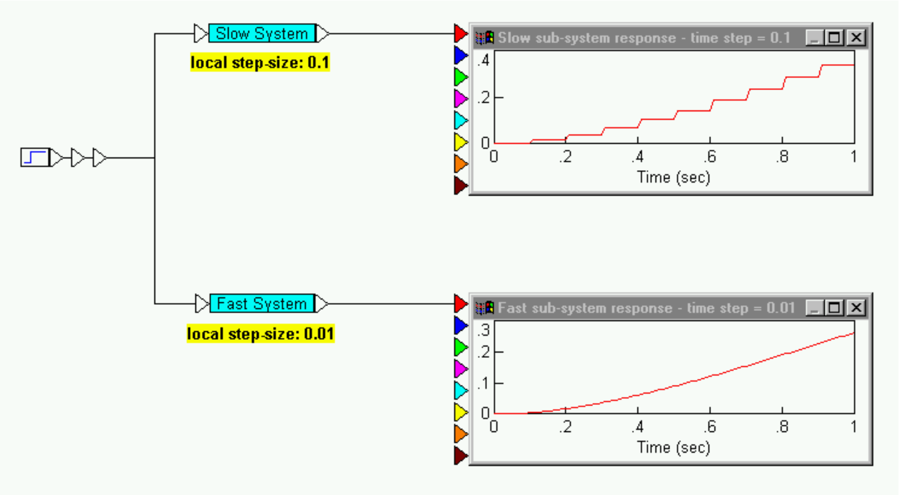
To	Do this
Enable the trigger	Activate the Enabled Execution parameter.
Disable the trigger	De-activate the Enabled Execution parameter.

5. Click on the OK button, or press ENTER.

4.1.5 Specifying a local time step

The Local Time Step parameter provides a general implementation of multi-rate simulation. When this parameter is activated, you can specify the time step for the compound block. Each compound block can run at an arbitrary time step. The time step is entered in the text box to the right of the parameter.

The following example demonstrates a multi-rate simulation. Here, separate simulation step sizes are specified and plotted for each subsystem: Slow System has a step size of 0.1, whereas Fast System has one of 0.01.



4.1.6 Tracking the number of blocks in a compound block

If you want to know how many blocks are in a given compound block, you can look in its Compound Properties dialog box. The dialog box lists the total number of blocks in the compound block, as well as the number of computational blocks.

4.1.7 Hiding compound blocks

You can selectively hide compound blocks while working in display mode.

➢ To hide compound blocks:

1. Choose Edit > Block Properties.
2. Point to the compound block you want hidden in display mode and click the mouse.
3. Activate the Hide In Display Mode parameter.
4. Click on the OK button, or press ENTER.
5. Activate View > Display Mode.

4.1.8 Configuring pictures on compound blocks

The pictures that can be configured on compound blocks are graphical images in .BMP file format. You can create them yourself or choose from the Commsim bitmap library, which resides in Commsim\BITMAP\DIAGRAM.

➢ To configure a picture on a compound block:

1. Choose Edit > Block Properties.
2. Point to the compound block on which the picture is to be configured and click the mouse.
3. Click on the Select Image button and choose the bitmap image to be configured on the block.
4. Click on the OK button, or press ENTER.
5. Click on the OK button, or press ENTER when the Compound Properties dialog box appears.

➢ To reset bitmaps:

Bitmaps may appear distorted in size under the following conditions:

- You open a block diagram and the current screen resolution is not set to the resolution when the bitmaps were originally created
- You change the font size in Commsim

The Edit > Reset Bitmap Scaling command will remove bitmap scaling.

- Choose Edit > Reset Bitmap Scaling.

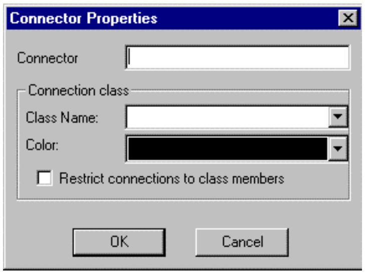
4.1.9 Labeling connector tabs on compound blocks

When you want to distinguish between the inputs and outputs on compound blocks, you can assign labels to their connector tabs. When the View > Connector Labels command is activated, connector labels are displayed on the block. In addition, when you drill into it, those labels appear next to the input and output connectors on the left and right side of the screen.

If you do not specify a connector label, the label defaults to the class name specified in the Connector Properties dialog box, if one is specified.

➢ To assign connector labels:

1. Point to the connector tab on the compound block you want to label. The pointer turns into an upward pointing arrow.
2. Double-click the mouse. The Connector Properties dialog box appears.



3. In the Connector box, enter a name.
4. Click on the OK button, or press ENTER.
5. If you want the label to appear on the block, activate the Connector Labels command in the View menu.

4.1.10 Dissolving a compound block

Use the Edit > Dissolve Compound Block command to de-encapsulate the blocks one level below the current level. When you execute Dissolve Compound Block, the blocks immediately below the current level move up to the current level. The blocks remain highlighted until the next command is executed to make it easier to recreate the compound block, in case you change your mind.

When you dissolve a compound block, Commsim maintains all internal wiring connections.

➢ To dissolve a compound block:

1. Choose Edit > Dissolve Compound Block.
2. Point to the compound block and click the right mouse button.
3. Click the mouse on empty screen space to exit this command.

4.1.11 Other things you can do with compound blocks

Commsim lets you do other things with compound blocks.

For information about	See
Coloring compound blocks	“10.2.2 Customizing other screen settings” on page 10-4
Protect compound blocks	“4.9.2 Protecting compound blocks” on page 4-34

4.2 Embedding blocks

With embedding, you can include information created in one Commsim block diagram, referred to as the *source diagram*, in one or more other block diagrams, referred to as the *destination diagrams*. Each time the source diagram changes, the changes are propagated in the destination diagrams.

When you embed a block diagram, a read-only version of the diagram is inserted into the destination diagram along with a link to the source diagram. You can drill into the embedded diagram just as you would a compound block. You can even override the read-only status of the embedded diagram and make changes directly to it. These changes, however, are not saved until you execute the File > Save Embedded Files command. The Save and Save All commands do not save edits made to embedded diagrams.

4.2.1 Setting up a diagram to be embedded

Before you can embed a diagram, check that its top level is a single compound block. If it's not, use the Edit > Create Compound Block compound to create one

If you want to restrict access to the `embed` block, apply the protection to the compound block.

4.2.2 Embedding a block diagram

Embedding a block diagram involves dragging an `embed` block into the work area and setting up the link to the source file.

➢ To embed a block diagram:

1. Open the destination diagram and move to the block diagram level where you want to insert an embedded block diagram.
2. Drag an `embed` block into the work area.
3. Choose Edit > Block Properties.
4. Click the mouse over the `embed` block.
5. In the File Name box, enter the name of the block diagram file to be embedded. If you do not see the file you want, click on the Select File button to search for it.
6. Click on the OK button, or press ENTER.

4.2.3 Editing an embedded block diagram

There are two ways to edit an embedded diagram:

- Open and edit the source file to which the embedded diagram is linked. When you edit a source file, all embedded diagrams linked to that source file are immediately updated to reflect the changes.
- Press CTRL+ALT+W to override the read-only status of the embedded diagram and make the edits directly to the diagram. To save the edits, make sure you execute the File > Save Embedded Files command.

4.2.4 Reconnecting an embedded block diagram

You may lose a link if you move or rename the source file. If this occurs, you must redirect the link to the appropriate location or file name.

➢ To reconnect a link:

1. Choose Edit > Block Properties.
2. Point to the `embed` block and click the mouse.
3. In the File Name box, enter the correct path or new file name. If you are unsure of the path or file name, click on the Select File button to search for the file you want.
4. Click on the OK button, or press ENTER.

4.2.5 Adding block diagrams

You can add another block diagram to the currently opened diagram using the File > Add command. After you've added the block diagram, some blocks and wires may overlap as a result of this operation; use the mouse and Edit menu commands to reposition them appropriately.

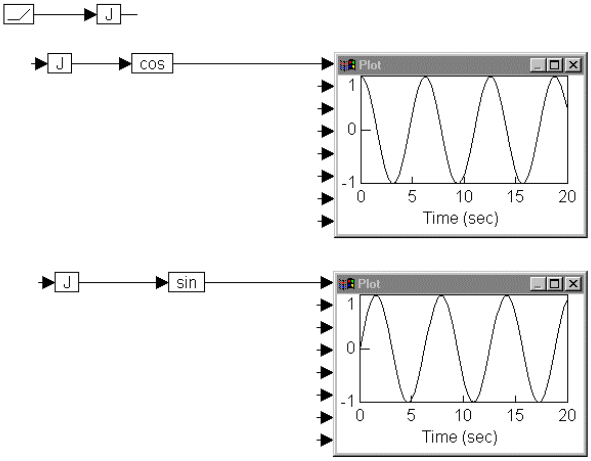
➢ To add a block diagram to the current diagram:

1. Open the block diagram into which you want to add another block diagram.
2. Choose File > Add.
3. In the File Name box, type or select the name of the block diagram you want to add. If you do not see the block diagram you want to add, select a new drive or directory.
4. Click on the OK button, or press ENTER. An empty rectangular box appears that represents the block diagram. The pointer is anchored to the box.
5. Move the box to where you want the block diagram added.
6. Click the mouse.

4.3 Using variables to pass signals

The `variable` block lets you name a signal and transmit it throughout your diagram without the use of wires.

Variables of the same name share signals. For example, in the diagram below, the variable *j* is used in three different locations:



The variable *j* in the upper part of the diagram is the declared variable. Only declared variables are allowed input signals. In addition, there can be only one declared variable of a given name. The other two *j* variables are referenced variables. Wires cannot be fed directly into referenced variables; they receive their input from the declared variable.

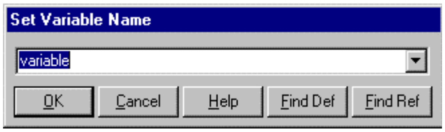
All variables can have any number of output signals.

4.3.1 Creating variables

The `variable` block is located under the Blocks menu in the Annotation category.

➢ To create a variable block:

1. Choose Edit > Block Properties.
2. Click the mouse over the `variable` block. The Set Variable Name dialog box appears.



3. Do one of the following:

To	Do this
Create a new variable	Enter a new name. To limit the scope of the variable, preface the name with a colon (:) character to make it local; two colon (::) characters to make it definition-scoped, or no colons to make it global. (For information on scoping variables, see the descriptions below.)
Reference an existing variable	Click on the DOWN ARROW and choose a name from the list. (For information on the scope of the variables, see the descriptions below.)

4. Click on the OK button, or press ENTER.

Naming a variable

It is never a good idea to name a `variable` block *-X,* or a number, like 1 or 2 or 3. Naming a variable *-X* leads to confusion with the `-x` block. Naming a variable a number leads to confusion with the `const` block.

4.3.2 Scoping variables

In Commsim, you can define which portions of the diagram can reference a variable by designating its scope. There are three types of scope: *diagram scope, definition and below scope*, and *level scope*.

- Diagram scope indicates that the variable can be referenced at any hierarchical level of the block diagram. Variables with diagram scope are referred to as *global variables*.
- Definition and below scope indicates that the variable can be referenced at the current hierarchical level, as well as all levels beneath it. To identify definition-scope variables, preface their names with two colon (::) characters.
- Level scope indicates that the variable can be referenced only at the current level of the block diagram. Variables with hierarchical level scope are referred to as *local variables*.

4.3.2.1 Level scope

A variable with level scope cannot be referenced outside of its current hierarchical level. Although level scoping is the most restrictive type of scope, it actually has several key advantages. By limiting the region over which variables can be referenced, you can construct sections of a diagram without worrying about whether your variable names conflict with other names used in other parts of the diagram. In addition, users reading your diagram will know immediately that the variables’ use is limited to a small region.

Variables with level scoping are prefaced with the colon (:) character.

4.3.2.2 Definition and below scope

Giving a variable definition and below scope allows the variable to be referenced not only at the hierarchical level of the diagram at which it was defined, but also at all the levels beneath it. For example, if compound block A contains a definition scope variable, all the sublevels in compound block A are able to use the variable.

By using definition-scoped `variables`, you can copy or add subsystems to an existing diagram without breaking or misdirecting references.

To give a variable definition and below scope, preface its name with two colon (::) characters.

4.3.2.3 Diagram scope

Variables with diagram scope are called global variables. Because global variables reference any part of a block diagram, you should exercise caution in your use of them. Global variables can make a block diagram hard to maintain because they increase the diagram’s complexity.

In addition, global variables increase the chances of a conflict in names between modules. For example, engineers working on different parts of a large project may choose the same name for different global variables. The problem won’t surface until each module is added to the master diagram.

As a rule of thumb, global variables should be used only when transmitting system-wide constants or signals that would be laborious or visually messy to represent as wires.

Making copies of global variables: When you make a copy of a compound block containing a global variable with wired input, Commsim renames the copied occurrence of the global variable in the following manner:

original-variable-block-name@unique-number

4.3.2.4 Finding variable definitions

There are two ways to find `variables` in your diagram:

- Use the Edit > Find command and activate the Match Variable Definitions Only option
- Click the right mouse button over the `variable` block and click on the Find Def button

4.3.2.5 Finding variable references

There are two ways to find `variables` in your diagram:

- Use the Edit > Find command and activate the Match Variables Only option
- Click the right mouse button over the `variable` block and click on the Find Ref button

4.3.3 Built-in variables

The following variable block names are built into Commsim:

Block name	Description
$firstPass	Generates an initial unit pulse on the first step of a simulation.
$lastPass	Generates a final pulse on the last step of a simulation.
$randomSeed	Accesses the random seed set in the dialog box for the Simulation Properties command.
$runCount	Holds the simulation iteration count for multiple simulation runs, such as Monte Carlo simulations and parameter sweeps.
$timeEnd	Returns the stop time of the simulation.
$timeStart	Returns the start time of the simulation.
$timeStep	Returns the step size of the simulation.

4.4 Using path aliases to reference files

As its name implies, a path alias is another name for all or part of the full specification of a file. You use path aliases to quickly insert frequently referenced files — for example, map files, import files, and bitmap image files. Rather than entering the complete file specification for each file, you can use path aliases to reference any part of the specification.

You can also use path aliases whenever automatically updating information would make maintaining your diagrams easier. For example, suppose an `animate` block referenced numerous bitmap images in C:\MYTEST\BMPS. If you moved the location of the bitmap images to C:\DIAGRAMS\BITMAPS, changing each file specification of each referenced bitmap would be a frustratingly long exercise. With path aliasing, you’d only have to update the path alias once for Commsim to correctly locate each bitmap image.

4.4.1 Types of path aliases

Commsim distinguishes between two types of path aliases: global and local. Global path aliases are saved in your Commsim.INI file and apply to all diagrams. Local path aliases, on the other hand, are saved with a specific diagram and apply only to that diagram.

4.4.2 Displaying conflicts between local and global path aliases

If you have both a local and global path alias of the same name, the local path alias takes precedence. To warn you if this occurs, use the Warn of Conflicting Local Alias Definitions option.

➢ To warn about conflicting path aliases:

1. Choose Edit > Preferences.
2. Click on the Preferences tab.
3. Activate the Warn of Conflicting Local Alias Definitions option.
4. Click on the OK button, or press ENTER.

4.4.3 Creating path aliases

You create path aliases using the Preferences command in the Edit menu. Global path aliases are specified under the Path Aliases tab; local path aliases are specified under the Local Aliases tab. When you click on either of these tabs, an Alias=Path window appears listing all existing global or local path aliases. New path aliases are entered in this window.

➢ To create a path alias:

1. Choose Edit > Preferences.
2. Do one of the following:

To create a	Click on this tab
Global path alias	Path Aliases
Local path alias	Local Path Aliases

3. In the Alias=path window, double-click the mouse over the ellipses. The pointer becomes an I-beam.

4. Enter the path alias in the following format:

   path-alias=pathname

   The path-alias cannot include commas (,), semi-colons (;), colons (:), or dollar signs ($). The pathname must adhere to the MS/DOS rules for drive, directory, and file specifications. The following table shows valid and invalid path aliases:

If you create this path alias	It is
BmpDir=C:\BITMAPS\ PUMP.BMP	Valid. This path alias points to a specific file
BmpDir=C:\BITMAPS	Valid; however, you must include the file specification prefaced with a backward slash when you use the path alias.
BmpDir=C:\BITMAPS\	Valid; however, you must include the file specification and enclose BmpDir in parentheses.
$BmpDir=C:\BITMAPS	Invalid; a dollar sign is not allowed in the path alias.

5. Click on the OK button, or press ENTER.

➢ To change a path alias:

1. Open the diagram to which the local path alias is to be applied.
2. Choose Edit > Preferences.
3. Do one of the following:

To edit a	Click on
Global path alias	Path Aliases
Local path alias	Local Path Aliases

4. In the Alias=path window, double-click the mouse over the ellipses. The pointer becomes an I-beam.
5. Edit the path alias accordingly. Refer to the above procedure for rules on entering path aliases.
6. Click on the OK button, or press ENTER.

4.4.4 Specifying path aliases in blocks

You can use global and local path aliases in any block that references a file. When you specify a path alias, prefix it with a dollar ($) sign. During simulation, when Commsim encounters a path alias, it expands the alias to its corresponding pathname.

➢ To specify a path alias in a block:

1. Choose Edit > Block Properties.
2. Click the mouse over the block to which a path alias will be applied.
3. In the Properties dialog box, position the insertion point in the text box in which you want to insert the path alias. (Typically, this is the File Name, Name, Bitmap Image, or Image box.)
4. Enter the path alias, prefaced with a dollar sign ($).

   Because the pathname associated with the path alias is not necessarily a complete file specification, you have may have to include the missing parts of the specification along with the path alias. For example, suppose you want to access the file C:\BITMAPS\PUMP.BMP. The following table shows what you specify based on how the path alias was originally created.

If the BmpDir path alias is defined as	You specify
BmpDir=C:	$BmpDir\BITMAPS\PUMP.BMP
BmpDir=C:\	$(BmpDir)BITMAPS\PUMP.BMP
BmpDir=C:\BITMAPS	$BmpDir\PUMP.BMP
BmpDir=C:\BITMAPS\	$(BmpDir)PUMP.BMP
BmpDir=C:\BITMAPS\PUMP	$BmpDir.BMP
BmpDir=C:\BITMAPS\PUMP.BMP	$BmpDir

   Note that the path alias must be enclosed in parentheses whenever a backslash (\) or period (.) does not separate it from the ensuing file specification.

   Default drives: If you do not define the drive, Commsim uses the current default drive.
5. Click on the OK button, or press ENTER.

4.4.5 Creating nested path aliases

A simple path alias expands to a specific pathname, for example, a drive, directory, and/or file. Frequently, however, you want to create a path alias that expands to another path alias. This is called nested path aliases. Nested path aliases make it easier to modify a path when the file, directory, or disk that is referenced moves or is renamed. For example, suppose you want to create a path alias to C:\BITMAPS\PUMP.BMP. You could create the following simple path alias:

```
PUMP=C:\BITMAPS\PUMP.BMP
```

If you moved the \BITMAPS directory to the D: drive, changing the path alias would be a simple exercise. But suppose you created path aliases for 100 bitmap files in C:\BITMAPS. Now you’d have to edit all 100 path aliases to point to D:. Using nested path aliases, as shown below, avoids this needless editing:

```
BMPDISK=C:
BMPS=$BMPDISK\BITMAPS
PUMP=$BMPS\PUMP.BMP
OPEN_VALVE=$BMPS\O_VALVE.BMP
CLOSED_VALVE=$BMPS\C_VALVE.BMP
SEALER=$BMPS\SEALER.BMP
.
.
```

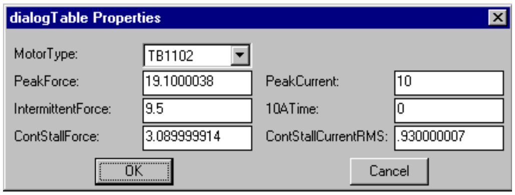
To redirect all the path aliases to the D: drive, simply edit the first path alias to read:

```
BMPDISK=D:
```

Similarly, if you moved all the bitmaps to a different directory, edit the second path alias accordingly.

4.5 Working with dialog tables

The `dialogTable` block lets you access lists of data specific to individual items in a single dialog box. These lists of data are stored in a data table. When you attach a data table to a `dialogTable` block, Commsim arranges and presents the information from the data lists in the `dialogTable`'s dialog box. For example, the dialog box below lists parametric data for a TB1102 motor:



dialogTable Properties

MotorType: TB1102

PeakForce: 19.1000038

PeakCurrent: 10

IntermittentForce: 9.5

10ATime: 0

ContStallForce: 3.089999914

ContStallCurrentRMS: .930000007

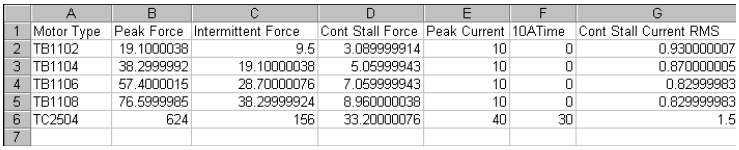
When you click on the DOWN ARROW in the MotorType box, you can select a different type of motor. When you do so, Commsim updates the dialog box with the appropriate data.

4.5.1 Understanding data tables

If you have used Microsoft Excel, Lotus 1-2-3, or any other spreadsheet program, you are probably familiar with data files. Data tables are similar to data files. Like data files, data tables are made up of a *header record* and *data records*.

A data table contains sets of information that describe a particular item. Each of these sets of information is called a *data record*. For example, one record in a list of motors contains all the information about a particular motor. The different types of information — such as, the motor type, its peak force, its intermittent force, and so on — are called *fields*. Each field must have a unique name. These *field names* are stored in the first row of cells in the data table. This row is called the *header record*. The first cell in the header record contains the *item identifier*. Once you attach the data table to the `dialogTable` block, you use the item identifier to choose which data record is to be displayed in the dialog box.

The following illustration shows an Excel data table for five different motors.



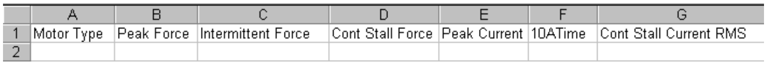
	A	B	C	D	E	F	G
1	Motor Type	Peak Force	Intermittent Force	Cont Stall Force	Peak Current	10ATime	Cont Stall Current RMS
2	TB1102	19.1000038	9.5	3.089999914	10	0	0.930000007
3	TB1104	38.2999992	19.10000038	5.05999943	10	0	0.870000005
4	TB1106	57.4000015	28.70000076	7.059999943	10	0	0.82999983
5	TB1108	76.5999985	38.29999924	8.960000038	10	0	0.829999983
6	TC2504	624	156	33.20000076	40	30	1.5
7							

The first field name in the header record — the item identifier — is Motor Types. The different types of motors are listed beneath the Motor Types column heading. The remaining field names in the header record are column headings for motor information. For example, the peak force for each motor type is listed in the column of cells labeled Peak Force. Each of the remaining rows of cells, the data records, contains a set of information for a particular motor. When you attach the data table to a `dialogTable` block, the field names in the header record are listed as parameter names in the dialog box.

4.5.2 Creating a data table in Excel

To create a data table, you use Excel and save the file in .CSV format. If you do not have Excel, you can use Notepad to create a .TXT file. For information on the rules governing Notepad-created data tables, see “4.5.3 Creating a data table with Notepad” on page 4-26.

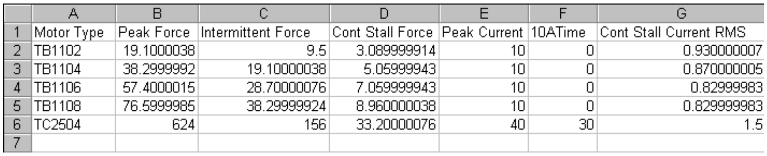
1. Start Excel. Excel opens a new empty workbook.
2. In the first row of cells, enter the field names for the data table. These field names are the header record.



	A	B	C	D	E	F	G
1	Motor Type	Peak Force	Intermittent Force	Cont Stall Force	Peak Current	10ATime	Cont Stall Current RMS
2							

   Note that the information in cell A1 must be the item identifier. In the above example, Motor Type is the item identifier.

3. Beginning in the first cell of the second row, enter the information the corresponds to the field name at the top of the column. Press the TAB key to move to the next cell.



	A	B	C	D	E	F	G
1	Motor Type	Peak Force	Intermittent Force	Cont Stall Force	Peak Current	10ATime	Cont Stall Current RMS
2	TB1102	19.1000038	9.5	3.089999914	10	0	0.930000007
3	TB1104	38.2999992	19.10000038	5.05999943	10	0	0.870000005
4	TB1106	57.4000015	28.70000076	7.059999943	10	0	0.82999983
5	TB1108	76.5999985	38.29999924	8.960000038	10	0	0.829999983
6	TC2504	624	156	33.20000076	40	30	1.5
7							

4. When you have finished entering all the data records, save the file as a .CSV file.
5. Exit Excel.

4.5.3 Creating a data table with Notepad

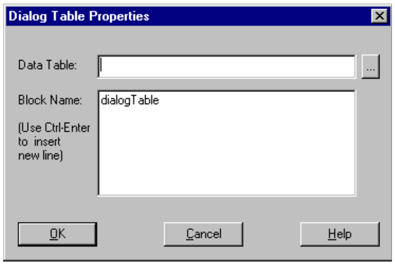
When you create a data table in Notepad, use the same format as described in “4.5.2 Creating a data table in Excel” on page 4-26. Then follow these additional rules:

- Separate each cell with a comma or space.
- Start each data record on a new line.
- Save the file in .TXT format.

4.5.4 Attaching a data table to a dialogTable block

Attaching a data table identifies the name and location of the file containing the data items and corresponding data records. To attach a data table to a `dialogTable` block, do the following:

1. Click the right mouse button over the `dialogTable` block. The following dialog box appears:



Dialog Table Properties

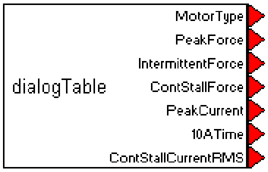
Data Table:

Block Name: dialogTable

(Use Ctrl-Enter to insert new line)

2. In the Data Table box, enter the complete path for the data table file, or click on the ... button to select the data table.
3. You can optionally enter a new label for the block in the Block Name box.
4. Click on the OK button, or press ENTER.

   Commsim displays the `dialogTable` block. Note that Commsim assigns output connectors to the block, which correspond to the field names in the header record of the data table.

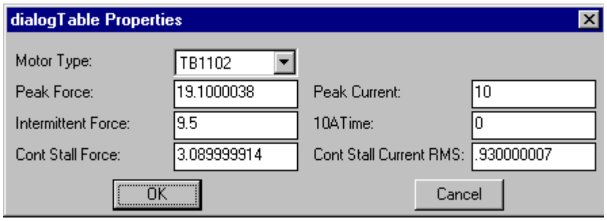


For this `dialogTable` block, the attached data table has seven field names in the header record.

4.5.5 Examining the attached data table

The Properties dialog box for the `dialogTable` block allows you to display and retrieve data records from the data table.

1. Click the right mouse button over the `dialogTable` block. The Properties dialog box appears, like the one shown below:



Motor Type: TB1102
Peak Force: 19.1000038
Intermittent Force: 9.5
Cont Stall Force: 3.089999914
Peak Current: 10
10ATime: 0
Cont Stall Current RMS: .930000007

   The field names in the header record are the parameter names in the dialog box. The information from the first data record is displayed in the text boxes.

2. To display another data record, click on the DOWN ARROW in the text box for the first parameter — in this case, Motor Type — and select the item whose data record you want displayed. Commsim updates the information in the remaining text boxes accordingly.

4.6 Comparing diagrams

To see the differences in two diagrams, you can use the Compare Diagrams command under the Tools menu. When you use this command and Commsim finds differences between the two diagrams, Commsim does the following:

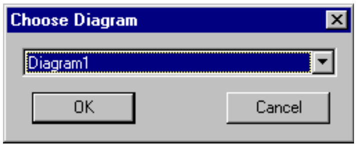
- Creates a read-only temporary diagram based on the active diagram.
- Highlights the blocks that are different in red.

To advance from one difference to the next, use the Edit > Goto Next Change command.

➢ To compare diagrams:

1. Open both diagrams that are to be compared.
2. Make sure one of the diagrams is your active diagram. Note that if Commsim detects differences, it creates a temporary read-only diagram based on the active diagram.

3. Choose Tools > Compare Diagrams. The Choose Diagram dialog box appears.



4. Click on the DOWN ARROW and select the diagram to be compared from the list of open diagrams.
5. Choose the OK button, or press ENTER.

   Commsim compares the diagrams, then displays a message indicating whether it detected differences in the two diagrams. If differences are detected, Commsim creates a temporary read-only Differences diagram. All the differences are highlighted in red.
6. Choose Edit > Goto Next Change command to move from one detected difference to the next. When you land on a difference, Commsim highlights the block in blue.

4.7 Using tags

You use tags to mark locations in a diagram that you want to find easily. Commsim marks the location with a green triangular tag in the upper left-hand corner of the current level of the diagram. The tag does not appear when you print the diagram.

➢ To insert a tag:

1. Go to the block diagram level that contains the information you want to tag.
2. Choose Edit > Toggle Tag, or press CTRL+F2.

➢ To remove a tag:

1. Press the F2 key to advance to the block diagram level that contains the tag.
2. Choose Edit > Toggle Tag, or press CTRL+F2.

➢ To go to a tagged location:

1. Choose Edit > Goto Tag, or press F2.

4.8 Tracking diagram progress

You can track diagram progress using the following two commands:

- **File > Diagram Information:** Maintains an edit history
- **File > Generate Report:** Generates a diagram report

4.8.1 Maintaining an edit history

The Diagram Information command in the File menu helps you keep track of important information about a block diagram as it is being developed. You can list the author's name and attach comments or an edit history to the block diagram. You can also identify the block diagram by a longer, more descriptive name. The name appears in File Open and File Add dialog boxes when you select its DOS file name.

The Diagram Information command also maintains statistics about the block diagram, including its DOS file name, its byte and block size, its last modification date, and the version of Commsim used to create it. Note that the Byte Size and Last Modified fields are not updated until you save the block diagram.

➢ To add or view diagram information:

1. Open the block diagram whose information is to be added to or viewed.
2. Choose File > Diagram Information.
3. You can add or change information in the Title, Author, and Comment boxes. The statistical information can be viewed, but not edited.
4. When you finish adding or viewing diagram information, click on the OK button, or press ENTER.

You can add or revise diagram information for the current block diagram at any time.

4.8.2 Generating a diagram report

A diagram report is a text file that lists statistical information about a particular diagram. You can include the following information in the report:

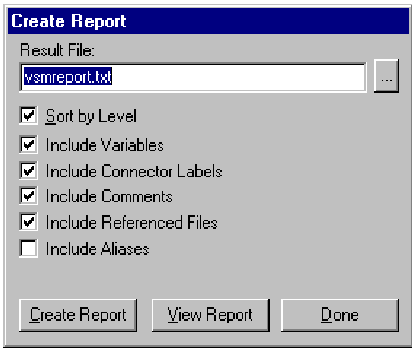
- Aliases used in diagram
- Variables used in diagram
- Connector labels specified in diagram
- Files referenced in diagram
- Comments specified in diagram

You also have the option of sorting the statistical information by diagram level.

After you generate the report, it is automatically saved to file. You can view the report on screen or you can retrieve the saved file later on.

➢ To create a diagram report:

1. Open the diagram for which you want to generate a report.
2. Choose File > Create Report. The Create Report dialog box appears.



Result File: vsmreport.txt

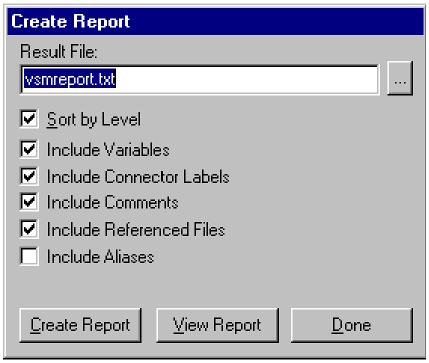
☑ Sort by Level
☑ Include Variables
☑ Include Connector Labels
☑ Include Comments
☑ Include Referenced Files
□ Include Aliases

3. Select the options you want. When you select Sort by Level, the other selected options are sorted by the level of the diagram in which they occur.
4. Click on Create Report.
5. The Report Generator creates the report and stores in it the file listed in the Result File box. The default name for this file is VSMREPORT.TXT.
6. Click on View Report.

   The generated report is displayed in Notepad according to the options you selected.

➢ To create a diagram report under a new name:

1. Open the diagram for which you want to generate a report.
2. Choose File > Create Report. The Create Report dialog box appears.



Result File: vsmreport.txt

☑ Sort by Level
☑ Include Variables
☑ Include Connector Labels
☑ Include Comments
☑ Include Referenced Files
□ Include Aliases

3. In the Result File box, enter a new name. To specify a different location for the file, click on the ... button.
4. Select the options you want.
5. Click on Create Report.
6. Click on View Report.

   The generated report is displayed in Notepad according to the options you selected.

4.9 Protecting your work

Commsim offers several levels of protection for your work:

- You can assign your block diagram a password to keep other users from opening the diagram. You can also request or require that they open the diagram in read-only mode.
- In large project development, where multiple users are working on the same diagram, you can assign password protection to particular parts of the diagram to prevent other users from viewing the information. You can also request or require that they view the information in read-only mode.

If you decide to use a password to restrict access, make sure to write it down exactly as you entered it — passwords are case sensitive — and store it in a safe place. Without the password, even you can’t access the information.

4.9.1 Protecting block diagrams

To assign a password to a block diagram and set options that control how much access other users have to the diagram, choose the Diagram Information command in the File menu.

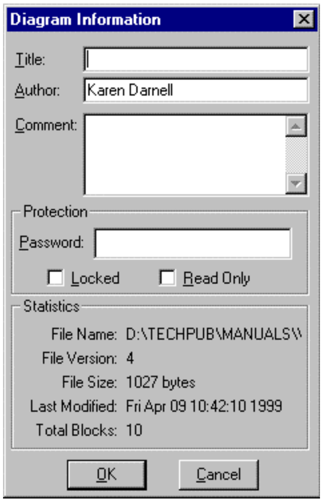


Diagram Information

Title:

Author: Karen Darnell

Comment:

Protection

Password:

☐ Locked ☐ Read Only

Statistics

File Name: D:\TECHPUB\MANUALS\\

File Version: 4

File Size: 1027 bytes

Last Modified: Fri Apr 09 10:42:10 1999

Total Blocks: 10

Password Protected: To prevent other users from opening the block diagram, type a password in the Password box and activate the Locked option. Only users who know the password can open the diagram and make changes to it.

Read-Only Password Protection: To allow other users to open the diagram, but prohibit them from making changes, type a password in the Password box and activate the Read Only option. Only users who know the password can open the diagram. Once opened, the diagram can be viewed, however, it cannot be changed.

Read-Only Requested Protection: To recommend, but not require, that other users only view a diagram without making changes to it, activate the Read Only check box. Although the diagram is opened in read-only mode, any user can de-activate the read-only protection and edit the diagram.

➢ To restrict access to a block diagram:

1. Open the block diagram to which restricted access is to be applied.
2. Choose File > Diagram Information.
3. Do one of the following:
   - To lock the diagram closed, enter a password in the Password box and activate the Locked parameter.
   - To make the block diagram read-only, enter a password in the Password box and

activate the Read Only parameter.

A password can contain up to 10 characters and can include any combination of letters and numbers. Commsim echoes an asterisk (*) for each character you type. Passwords are case sensitive.

4. Click on the OK button, or press ENTER.
5. Commsim asks you to re-enter the password for verification.

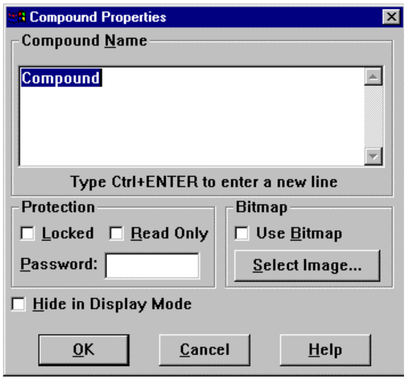
➢ To change or delete a password:

1. Choose File > Diagram Information command.
2. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.
3. Click on the OK button, or press ENTER.

   If you entered a new password, Commsim asks you to re-enter the new password for verification.

4.9.2 Protecting compound blocks

Restricting access to a compound block is similar to restricting access to a block diagram. You have the choice of password protection, read-only password protection, and read-only requested protection. You enter the level of protection in the Compound Properties dialog box.



Compound Properties

Compound Name

Compound

Type Ctrl+ENTER to enter a new line

Protection

□ Locked □ Read Only

Password:

Bitmap

□ Use Bitmap

□ Hide in Display Mode

➢ To restrict access to a compound block:

1. Choose Edit > Block Properties.

2. Point to the compound block you want protected and click the mouse.
3. Do one of the following:
   - To lock the compound block, activate the Locked parameter and enter a password in the Password box.
   - To make the compound block read-only, activate the Read Only parameter and enter a password in the Password box.

   A password can contain up to 10 characters and can include any combination of letters and numbers. Commsim echoes an asterisk (*) for each character you type. Passwords are case sensitive.
4. Click on the OK button, or press ENTER.
5. Commsim asks you to re-enter the password for verification.

➢ To change or delete a password:

1. Choose Edit > Block Properties.
2. Point to the compound block whose password you want to change and click the mouse.
3. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.
4. Click on the OK button, or press ENTER.

   If you entered a new password, Commsim asks you to re-enter the new password for verification.

➢ To edit a read-only compound block with password protection:

1. Choose Edit > Block Properties.
2. Point to the compound block that is read-only with password protection and click the mouse.
3. In the Password box, select the row of asterisks that represent the existing password and type in the correct password.
4. Click on the Read Only attribute to de-select it.
5. Click on the OK button, or press ENTER.

4.9.3 Protecting embed blocks

Password locking is a mechanism that prevents other users from drilling into and viewing the contents of an embedded diagram. Only users who know the password can unlock the `embed` block and view its contents.

Password locking is inherited from the compound block in the source file. In other words, you do not apply protection to the `embed` block itself, but rather to the compound block in the source file.

➢ To drill into a protected embed block:

1. Choose Edit > Block Properties.
2. Click the mouse over the `embed` block.
3. Enter the password.
4. Click on the OK button, or press ENTER.

➢ To protect an embedded block diagram:

1. Open the source file that contains the compound block to be protected.
2. Choose Edit > Block Properties.
3. Click the mouse over the compound block.
4. Activate the Locked parameter and enter a password in the Password box.

   A password can contain up to 10 characters and can include any combination of letters and numbers. Commsim echoes an asterisk (*) for each character you type. Passwords are case sensitive.

   If you do not enter a password, a user can subsequently unlock the `embed` block.
5. Click on the OK button, or press ENTER.
6. Commsim asks you to re-enter the password for verification.

➢ α To change or delete a password:

1. Open the source file that contains the compound block to whose password is be changed or deleted.
2. Choose Edit > Block Properties.
3. Point to the compound block and click the mouse.
4. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.
5. Click on the OK button, or press ENTER.

   If you entered a new password, Commsim asks you to re-enter the new password for verification.

Chapter 5
Designing Digital Filters

The following are described in this chapter.

5.1 Digital filter basics

A digital filter is a discrete time system that delivers an output, which is a modified version of its input.

Filters are the basic building blocks for most signal processing applications. They are typically used to extract or eliminate one or more constituent frequencies of an incoming signal.

Filters used for signal conditioning are usually designed from frequency response specifications, and are called frequency-selective filters. Frequency-selective filters operate by attenuating some frequency components of the input signal while allowing other components to pass through unchanged. For example, a low-pass filter attenuates all frequencies in the input signal that are above a specified frequency.

5.2 Filter operations

Filter operations can be represented mathematically by one or more difference equations. A general difference equation can be written as:

$$y(k) = \sum_{i=0}^{M} a_i x(k-i) - \sum_{j=1}^{N} b_j y(k-j)$$

This equation represents the relationship between the k^{th} sample of the output to the *N* previous values of the output, the *M* previous values of the input, and the current value of the input. If all the coefficients b_j are zero, the resulting filter is called a non-recursive or Finite Impulse Response (FIR) filter. Recursive filters are also known as Infinite Impulse Response (IIR) filters.

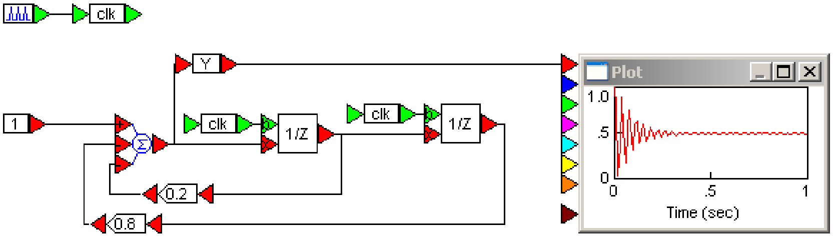
In FIR filters, the output is simply the weighted sum of the current and previous inputs. In contrast, in IIR filters, the output is the weighted sum of the current and previous inputs, and the previous outputs.

5.3 Time domain filters with tapped delay

Consider a filter described by the following recursive difference equation:

$y(k) = x(k) - 0.2y(k-1) - 0.8y(k-2)$

You can easily specify and implement this filter in time domain using `unitDelay` blocks. The filter input is *x*(*k*) and the filter output is *y*(*k*). The intermediate states are *y*(*k*-1) and *y*(*k*-2). The filter can be implemented as:



The Time Between Pulses parameter for the `pulseTrain` block is set to 0.01. Note that the simulation time step must be less than or equal to 0.01. An arbitrary value of 1 is assigned to input *x*.

5.4 Time domain filters with transfer functions

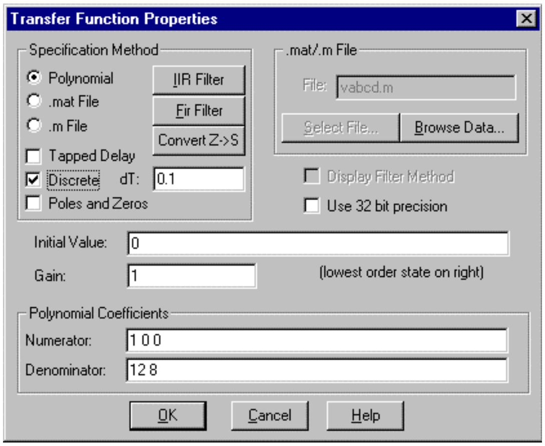
Filters can also be implemented in the time domain using the `transferFunction` block. For example, consider again the difference equation:

$y(k) = x(k) - 0.2y(k\text{-}1) - 0.8y(k\text{-}2)$

You can represent it in the form of a transfer function as:

$$\frac{Y(z)}{R(z)} = \frac{z^2}{z^2 + 0.2z + 0.8}$$

You can then implement the transfer function in block diagram form using the `transferFunction` block:



Transfer Function Properties

Specification Method

● Polynomial
○ .mat File
○ .m File
□ Tapped Delay
☑ Discrete dT: 0.1
□ Poles and Zeros

.mat/.m File

File: vabcd.m

□ Display Filter Method
□ Use 32 bit precision

Initial Value: 0

Gain: 1

(lowest order state on right)

Polynomial Coefficients

Numerator: 1 0 0

Denominator: 1 2 8

In the Transfer Function Properties dialog box, activate Discrete and set the value of dT to be greater than or equal to the simulation time step.

5.5 Frequency domain filter implementation

The dual nature of time and frequency domains means a filter in the time domain can be equivalently implemented in the frequency domain. Depending on the application, however, one domain is usually more convenient to work in than the other.

A recursive IIR filter can be implemented in the frequency domain by taking the product of the frequency domain equivalents of the input sequence and the filter.

$$y(k) = IDFT(Y(\omega)) = IDFT\left(x(\omega) \cdot \frac{H_a(\omega)}{H_b(\omega)} \right)$$

Here, $X(\omega)$ and $Y(\omega)$ are the Discrete Fourier Transforms (DFT) of the input and the output sequences respectively, and *IDFT* represents the Inverse Discrete Fourier Transform operation. $H_a(\omega)$ and $H_b(\omega)$ are the DFTs of the filter coefficients a_i and b_j, respectively, as given by the following difference equation:

$$y(k) = \sum_{i=0}^{M} a_i x(k-i) - \sum_{j=1}^{N} b_j y(k-j)$$

The DFT’s $H_a(\omega)$ and $H_b(\omega)$ must be of the same length as $X(\omega)$ and $Y(\omega)$. To accomplish this, the filter coefficients must be zero-padded appropriately. Consequently, the frequency domain implementation is computationally inefficient and will not be discussed further.

5.6 Comparison of FIR and IIR filters

The non-recursive FIR filter has a finite memory due to the finite number of delays that can be realized in a practical implementation. FIR filters usually have superior phase characteristics. To obtain sharp cut-off characteristics, FIR filters need to be of high order.

On the other hand, a recursive IIR filter has infinite memory due to its dependence on all prior outputs. Moreover, it generally requires a significantly lower number of elements to obtain a specific cut-off characteristic. The phase characteristics of IIR filter, however, are inferior to those of FIR filters.

5.7 Interactive filter design with the transferFunction block

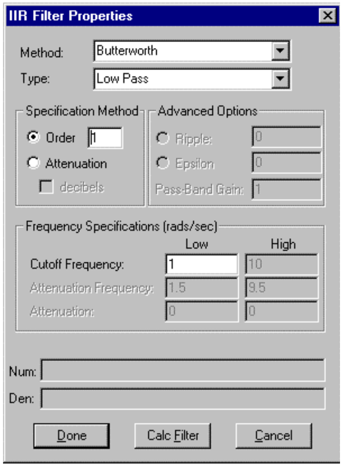
The first step in the digital filter design process is to specify the characteristics that you desire. The more fundamental specification would be the difference equation that is to be satisfied. Such specifications may arise directly from requirements in a signal processing problem.

However, much more common are the specifications that arise when you want to process a continuous time signal digitally, and you expect the digital filter to approximate the performance of an analog filter.

Using the `transferFunction` block you can design either IIR filters using analog prototypes or FIR filters.

5.8 IIR filter design

IIR filter design is the design of digital filters using Bessel, Chebyshev, Butterworth, or Inverse Chebyshev analog prototypes. To set up an IIR filter, click on the IIR Filter command button in the Transfer Function Properties dialog box.



IIR Filter Properties

Method: Butterworth

Type: Low Pass

Specification Method

● Order 1

○ Attenuation

□ decibels

Advanced Options

○ Ripple: 0

○ Epsilon: 0

Pass-Band Gain: 1

Frequency Specifications (rads/sec)

	Low	High
Cutoff Frequency:	1	10
Attenuation Frequency:	1.5	9.5
Attenuation:	0	0

Num:

Den:

5.8.1 Using the IIR Filter Properties dialog box

The IIR Filter Properties dialog box lets you constrain the design through either filter order or goodness-of-fit. The analog prototypes can be subsequently converted into a digital filter using bilinear transformation.

Method: You can choose from four analog filter methods, as described below.

- **Bessel:** Bessel filters are designed using Bessel polynomials. The Bessel filters are characterized by the property that the group delay is maximally flat at the origin of the *s*-plane. The step response of the Bessel filters exhibits very low overshoot and both the magnitude and impulse response exhibit gaussian decay as the filter order is increased.
- **Butterworth:** Butterworth filters are characterized by the property that the magnitude characteristic is maximally flat at the origin of the s-plane. This means that all the existing derivatives of the magnitude response are zero at the origin. Butterworth Low Pass filters are all-pole designs and have an attenuation of 3 dB at the critical frequency. The filter order completely specifies the filter and can either be explicitly provided or determined from the attenuation frequency and the attenuation level desired.
- **Chebyshev and Inverse Chebyshev:** Chebyshev filters are characterized by the property that the peak magnitude of the approximation error is minimized over a prescribed band of

frequencies. The magnitude is equi-ripple over the band of frequencies. For example, the magnitude oscillates between the maxima and minima of equal amplitude.

For the Chebyshev filters, the band of the frequencies over which the error is minimized is the Pass Band. For Inverse Chebyshev filters, the error is minimized over the Stop Band. The optimality property of the Chebyshev filters guarantees that no other all-pole filter offers equal or better performance in both the Pass and Stop bands. Inverse Chebyshev filters exhibit monotonic behavior in the Pass Band (maximally flat around the zero frequency) and equi-ripple behavior in the Stop Band. The Low Pass filter has poles in the left half of the s-plane and zeros on the imaginary axis.

Type: Indicates the band pass filter type.

Specification method: The following table describes how the specification method relates to the analog filter prototypes.

Filter	Notes
Bessel	You only need to specify the order.
Butterworth	If you specify the order, Commsim determines the attenuation. If you specify the attenuation, Commsim determines the order.
Chebyshev	If you specify the order, then the order and epsilon define the filter. The attenuation is fixed once a particular order and epsilon are chosen. If you specify attenuation, Commsim determines the order based on the attenuation and epsilon. Commsim determines the order such that the attenuation and epsilon specifications are met.
Inverse Chebyshev	Whether you specify the order or epsilon, the attenuation needs to be specified. If you specify the order, Commsim computes the epsilon based upon the attenuation and the attenuation level desired. In general, as the attenuation desired for a fixed-order filter increases, the corresponding epsilon also increases. This property could be exploited to yield very narrow band filters by specifying an extremely high attenuation, along with a narrow band. If you specify epsilon, Commsim determines the order based upon the attenuation desired. Commsim determines the order such that the attenuation and epsilon specifications are met.

The order of the filter that is generated is twice the order of the filter specified. For example, if you enter 2 in the Order box for a Band Pass filter, the filter generated will have an order of 4.

Cut-off Frequency: The low and high cut-off frequencies in the Frequency Specification box define the band edges. For Low Pass and High Pass filter types, there is only one cut-off frequency. For Band Pass and Band Stop filters, the low and high frequencies are both cut-off frequencies.

Attenuation and Attenuation Frequency: The attenuation characteristics of the filter are defined by:

- **Low and high attenuation frequencies:** The attenuation frequencies are set by the Attenuation Frequency (Low) and Attenuation Frequency (High) parameters. The values you enter indicate the frequency at which the specified attenuation level is reached.
- **Low and high attenuation levels:** The attenuation levels are set by the Attenuation (Low) and Attenuation (High) parameters. The values you enter indicate the amount by which you desire to suppress the level. An attenuation level of 100 equals a magnitude of 1/100.

For example, a Band Pass filter with band edges specified at 100 and 1000, an attenuation level of 10, and attenuation frequencies of 20 (low) and 100 (high) means that the filter gain is 0.1 at 80 and 0.05 at 1100. The epsilon (ε) is a measure of the attenuation level reached by the filter's magnitude characteristics at the critical frequency. Attenuation level at the critical frequency is given by:

Advanced Settings: The Epsilon and Ripple parameters provide two alternate ways of specifying the behavior of a Chebyshev filter.

There is a fluctuation (or ripple) in the amount (or attenuation gain) of the Band Pass and Band Stop. The filter order affects the size of the ripple, and the filter can be tuned to minimize that ripple.

Epsilon refers to the error between the ideal filter and the actual filter, regardless of the ripple. Minimizing the epsilon provides a best fit filter.

$$\frac{1}{\sqrt{1+\varepsilon^2}}$$

The ripple is the attenuation level at the critical frequency. Defining the epsilon completely defines the ripple.

5.8.1.1 Setting the frequency units

Frequency units can be specified in either radians per second or hertz. You set the frequency unit in the dialog box for the Simulate > Simulation Properties command.

5.8.1.2 Generating an IIR filter

When you generate an IIR filter, Commsim calculates the polynomial coefficients for the transfer function with the desired frequency characteristics.

➢ To generate an IIR filter:

1. Click on the Calc Filter command button to calculate the filter coefficients. The coefficients will be displayed in the Num (numerator) and Den (denominator) boxes.
2. Click on the Done button to close the IIR Filter Setup dialog box and transfer the filter numerator and denominator coefficients to the Transfer Function Setup dialog box.

5.9 FIR filter design

Commsim uses the Remez Multiple Exchange algorithm to design FIR filters. FIR filters in discrete time are realized as all-zero filters and are characterized by a finite impulse response in the time domain. Because they are all-zero filters, they are particularly well suited to efficient computation by tapped delay.

FIR filter design is typically executed in the frequency domain for convenience. The filter has the desired magnitude specifications and a linear phase characteristic.

Differentiators and Hilbert transformers

You can also design differentiators and Hilbert transformers using the Remez Multiple Exchange algorithm.

Differentiators are characterized by an approximate linear magnitude response over the desired frequency range.

Hilbert transformers are characterized by a flat magnitude response and a phase of 90^{o} over the specified frequency range. Frequency characteristics of an ideal Hilbert transformer are:

$$H(e^{j\omega}) = -j \quad 0 \le \omega \le \frac{\pi}{T}$$

$$= +j \quad \frac{\pi}{T} \le \omega \le \frac{2\pi}{T}$$

where ω is the angular frequency and *T* is the sampling time period.

5.9.1 Discrete and continuous FIR filter design

The discrete time filter design problem is treated as a weighted Chebyshev approximation problem and is solved using the Remez Multiple Exchange algorithm to compute the filter coefficients. The algorithm builds a discrete time representation of the filter.

In Commsim, the Discrete parameter in the Transfer Function Setup dialog box controls whether the generated FIR filters are discrete or continuous. When you design a discrete FIR filter, you must also specify a time step in the dT box.

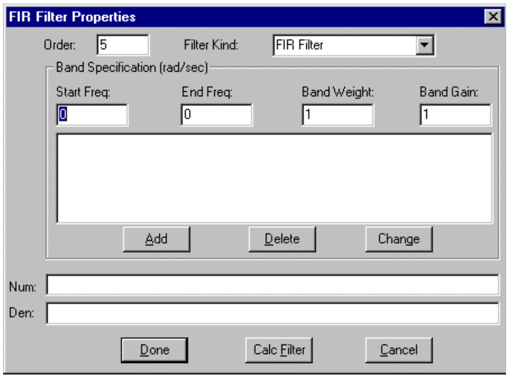
To implement a continuous FIR filter, de-activate the Discrete parameter. In this case, the filter is initially designed as a discrete time filter. Bilinear transformation is subsequently used to produce a continuous time equivalent. For more information on the Remez algorithm, see *Theory and Application of Digital Signal Processing* (Prentice Hall).

Tapped delay implementation

Tapped delay is a method of transfer function implementation that has linear computational and storage requirements with respect to model order. Because most FIR filters have a tendency to be high order, it makes sense to design FIR filters with tapped delay implementation. To do so, activate Tapped Delay in the Transfer Function Properties dialog box.

5.9.2 Using the FIR Filter Properties dialog box

When you click on the FIR Filter command button in the Transfer Function Setup dialog box, the FIR Filter Setup dialog box is opened.



Order: 5

Filter Kind: FIR Filter

Start Freq: 0

End Freq: 0

Band Weight: 1

Band Gain: 1

Order: The value specified in the Order box defines the filter order. Typically, higher orders yield better approximations.

Filter Kind: Indicates the type of filter to be generated. Your choices are FIR, differentiator, and Hilbert transformation.

Band specification: Band specifications describe the frequency bands magnitude response characteristics of the filter. The following rules must be observed when entering band specifications:

- Frequencies are specified in hertz for discrete and continuous filters. To specify in hertz, choose the Simulate > Simulation Properties command, then click on the Preferences tab and activate the Hertz option.
- For discrete filters, the frequency specified must be lower than the Nyquist frequency.
- For continuous filters, infinite frequency is indicated using the reserve word “inf.”

Start Freq and End Freq: Defines the lower and upper cut-off frequencies for each band.

Band Weight: Dictates the relative amounts of error allowed for each band. Higher weight values of a particular frequency band reflect higher sensitivity to error, where error is perceived as the difference between the actual and desired filter response. At least one band must have a weight of 1. For each of the other bands, you can use a higher or lower weight depending on the relative error that can be tolerated.

An equal weight of 1 on all bands indicates that the maximum absolute error on all bands is the same. A weight of 10 on one band and a weight of 1 on other bands implies that the former band has a maximum approximation error that is ten times less than that of the other bands.

Band Gain: Defines the desired frequency response magnitude for each band.

➢ To add a band:

1. Enter the band specification and click on the Add button.

   The band information is added to the list box. Each row in the list box corresponds to a single band. For FIR filters, if the number of bands increases, the filter order must be increased correspondingly, to maintain the same approximation error. For differentiator and Hilbert transformers, the number of bands is limited to one. The gain on the differentiator implies the gain achieved at the end frequency. The weight in either case is optimally adjusted to give the best error characteristics.

➢ To delete a band:

1. Select the band to be deleted from the list box and click on the Delete button.

- ➢ To change a band’s specifications:

1. Select the band to be changed from the list box.

   The band’s data appears in the edit boxes.
2. Make the desired changes.
3. Click on the Change button. The band data is modified in the list box to reflect the changes.

5.9.2.1 Generating an FIR filter

The Calc Filter command button generates the appropriate filter coefficients. Before computing the filter coefficients, the algorithm computes the maximum approximation error. This error is usually referred to as delta (δ) and is defined as the weighted difference between the actual and the desired magnitude response. A band with a weight of one will have delta as its absolute approximation error, while a band with a weight of 10 will have its absolute error 0.1 times δ. The value of δ is displayed in the message box.

- ➢ To generate an FIR filter:

1. Click on the Calc Filter button. The coefficients are displayed in the Num (numerator) and Den (denominator) boxes. If the delta displayed is too large, increase the order of the filter and re-calculate the filter.
2. Click on the Done button to close the FIR Filter Properties dialog box and transfer the filter numerator and denominator coefficients to the Transfer Function Properties dialog box.

Chapter 6
Working with Other Applications

The following are described in this chapter.

6.1 Importing basics

Commsim uses the `import` block to import information from many different file types generated by other applications. These include data files (.DAT), MatLab and MatLab-like files (.MAT and .M), and 8-bit or 16-bit sound files (.WAV).

The `import` block reads data points from the specified input file into the system model and translates them into scalar, vector, or matrix output signals. The `import` block can receive up to 50 scalar inputs and an unlimited number of vector or matrix input. The data can be either fixed interval or asynchronous.

The `import` block is particularly useful for comparing experimental data with simulated results and for inserting trial control data from an external source.

6.1.1 Setting up the input file

The input file can contain a header line to describe the separation of data points. The following table describes the header line format:

For this type of interval	Use this format
Fixed interval	#I=start-time, end-time, increment
Asynchronous interval	#T=number (time-column)

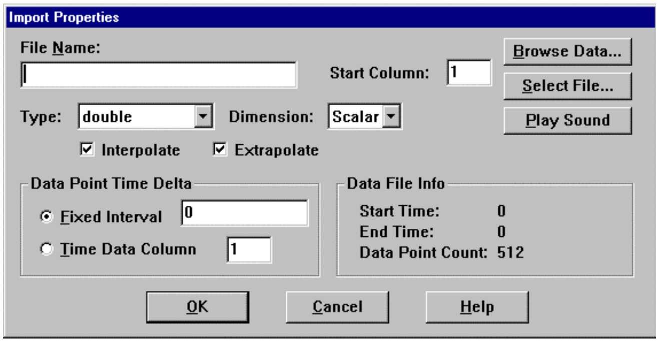
6.1.2 Importing data

Importing data involves dragging an `import` block into the work area and setting up the block to reference the input file.

➢ To import data:

1. From the Signal Producer category in the Blocks menu, drag an `import` block into the work area.
2. Choose Edit > Block Properties.
3. Click the mouse over the `import` block.

   The Import Properties dialog box appears.



File Name:

Start Column: 1

Type: double

Dimension: Scalar

☑ Interpolate ☑ Extrapolate

Data Point Time Delta

● Fixed Interval 0

○ Time Data Column 1

Data File Info

Start Time: 0

End Time: 0

Data Point Count: 512

4. Select the import parameters. (See the descriptions below for more information about each parameter.)
5. Click on the OK button, or press ENTER.

6.1.3 Using the Import Properties dialog box

The Import Properties dialog box provides the following options:

File Name: Indicates the file to be used as input to Commsim. You can specify .DAT, .M, .MAT, or .WAV files. When you specify a .WAV file, you can play the sounds you imported by clicking on the Play Sound button.

If you do not know the name or location of the input file, click on the Select File button to locate and choose a file.

To browse or edit the input file, click on the Browse Data button after you select a file.

Start Column: In a multi-column file, you can choose the column that corresponds to the top-most connector tab. The default value 1 corresponds to the first column.

Type: Indicates the type of data to be imported.

Dimension: Controls the dimensionality of the output signals. The choices are scalar, vector, and matrix.

Interpolate: Interpolates between two data points, instead of using the last known data value. Thus, if the data point is 5 at t_1, and 15 at t_2, then at $t_{1.5}$, the data point is 10 with interpolation, and 5 with no interpolation.

Extrapolate: Infers the next unknown data point based on the difference between the last two known data points.

Data Point Time Delta: Indicates the time interval between data points in the input file. If the input file was generated by Commsim using the `export` block, Commsim automatically reads the time interval information from the file header and sets the parameter accordingly. You have the following choices:

- **Fixed Interval:** Indicates that data points occur in fixed intervals. Enter the interval in the corresponding box. This is the default setting.
- **Time Data Column**: Indicates that data points occur in irregular time intervals. Enter the column containing the time data points in the corresponding box. Valid column numbers are 1 through 16.

Data File Info: Provides read-only information about the imported data. The Start Time and End Time fields indicate when Commsim starts and stops recording data. The Data Point Count field indicates the maximum number of data points to be read into Commsim. If the input file was generated in Commsim using the `export` block, Commsim automatically reads the data point count from the file header and sets the field accordingly. The maximum number of data points that can be read into Commsim is 250 million.

6.2 Exporting basics

The `export` block writes signals to an output file in .DAT, .M, .MAT, or .WAV file format. The `export` block can send up to 50 scalar outputs and an unlimited number of vector or matrix output. The output file can subsequently be used as input to Commsim or to a variety of other programs, such as MatLab and Microsoft Excel. The following information is written to the file:

- Data points that represent signal values. Data points are stored as ASCII text.
- Time interval information that applies to the data points.

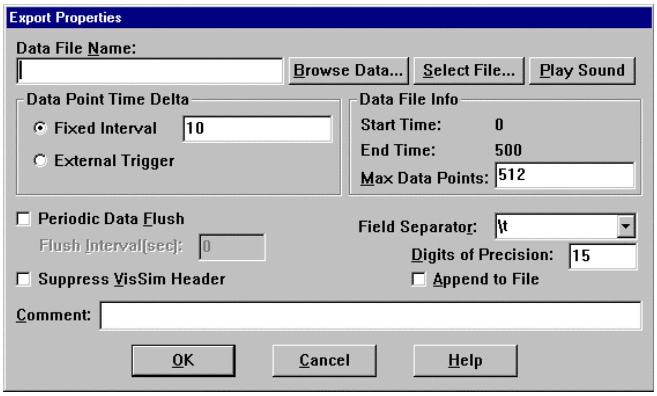
6.2.1 Exporting data

Exporting data involves dragging an `export` block into the work area and setting up the block to reference the output file.

➢ To export data:

1. From the Signal Consumers category in the Blocks menu, drag an `export` block into the work area.
2. Choose Edit > Block Properties.
3. Click the mouse over the `export` block.

   The Export Properties dialog box appears.



Export Properties

Data File Name:

Data Point Time Delta
- ● Fixed Interval: 10
- ○ External Trigger

Data File Info
- Start Time: 0
- End Time: 500
- Max Data Points: 512

□ Periodic Data Flush
Flush Interval(sec): 0

Field Separator: \t
Digits of Precision: 15

□ Suppress VisSim Header
□ Append to File

Comment:

4. Select the export parameters. (See the descriptions below for more information about each parameter.)
5. Click on the OK button, or press ENTER.

6.2.2 Using the Export Properties dialog box

The Export Properties dialog box provides the following options:

Data File Name: Indicates the name of the export file into which data points are to be written. You can type the file name directly into this box or select one using the Select File button. If you do not specify a data file, Commsim writes the data points to a file using the same name as your current block diagram. Commsim applies the .DAT extension to the file and stores it in your current directory.

You can export data in .DAT, .M, .MAP, .MAT, and .WAV file formats. The following special considerations apply to map files and wave files:

- If you want to create an output file to be used as input to the `map` block, you must specify the .MAP extension.
- You can create 8-bit and 16-bit sound files. You specify the bit count in the Digits of Precision box. Provided you have the appropriate hardware configuration and software drivers installed, you can preview the sound by clicking on the Play Sound button.

If you click on the Browse Data button, the file specified in the Data File Name box is opened for you to examine or edit.

Data Point Time Delta: Controls how Commsim writes the time interval information to the data file. You have the following choices:

- **Fixed Interval:** Indicates that data points occur in fixed intervals. The default interval used will be taken from the simulation step size. You can specify a different interval, however, it should be a multiple of the simulation step size, because the `export` block does not interpolate. Data is only exported at integral multiples of the simulation step size. This automatic adjustment is invisible when it occurs, which means it is not reflected in either the `export` block's dialog box or the data file header. You can see the adjustment only when you open the data file.

  If you import the output file into a simulation, you should edit the file header to reflect the interval at which the data was actually exported. The `import` block will interpolate as needed, retaining the timing of the original simulation run. Use the Browse Data button to open the data file for editing. The format of the data header file should be as follows:

  #I = *start-time, end-time, increment*

  The default is Fixed Interval.

  - **External Trigger:** Indicates that data will be recorded based on the state of the external trigger input. When External Trigger is activated, Commsim adds a round input connector tab to the `export` block. A zero value on the trigger inhibits data recording. A value of 1 causes a data point to be recorded. Valid column numbers are 1 through 16, inclusive.

Periodic Data Flush and Flush Interval: When activated, Periodic Data Flush writes the data in the export buffers to the specified data file at intervals established with Flush Interval. When you choose a flush interval, the data is to be written to disk at the specified interval and only enough memory will be allocated to handle the flush buffer (and not the whole file). If you specify 0 for the Max Data Points option, Commsim automatically calculates the buffer required by the flush interval.

Suppress Commsim Header: Suppresses writing the data header to the export file. Suppressing the data header may be necessary if the export file is to be imported into a software product other than Commsim.

The header information indicates whether the data is fixed or variable interval; the valid time range over which the data is collected; the actual fixed interval; and the time column for variable interval data. The following formats are used:

Fixed Interval	#I = *start-time, end-time, increment*
Variable Interval	#T= *number (time-column)*

Field Separator: Specifies the column separation character in the export file, which allows for compatibility with other applications. Recognized column separators are tabs, new lines, spaces, commas, semi-colons, and colons.

Digits of Precision: Specifies (for .DAT, .M, .MAT files) the maximum number of significant digits printed regardless of the decimal point. The default is 15.

For .WAV files, use Digits Of Precision to indicate whether the sound file is 8-bit or 16-bit. Enter 8 for 8-bit sound files or 16 for 16-bit sound files.

Append to File: Appends the exported data to the end of a specified file, instead of re-writing the file at the start of each new simulation run. This parameter is useful for multi-run applications, such as data acquisition, Monte Carlo simulations, and neural network training

Comment: Specifies a comment that is placed at the beginning of the exported data file. A comment is limited to 180 characters.

Data File Info: Provides read-only information about the export file. The Start Time and End Time fields indicate when Commsim starts and stops writing data points to the export file. These settings are obtained from the current settings of Range Start and Range End in the Simulation Properties dialog box.

The Max Data Points field indicates the maximum number of data points to be written to the export file. The default is 512 data points. The maximum number of data points that can be written to file is 250 million.

6.3 Commsim-MatLab interface basics

The Commsim-MatLab interface allows you to access MatLab to perform matrix calculations and transfer variables between the two applications. Commsim provides three interface blocks:

- **The `MatLab Expression` block**, which invokes MatLab to calculate a general matrix expression.
- **The `MatLab Read Variable` block**, which reads a variable from the MatLab workspace into Commsim.
- **The `MatLab Write Variable` block**, which writes a variable into the MatLab workspace.

To use the Commsim-MatLab interface, MatLab version 5+ must be installed on your computer. All the blocks are located under the Blocks > MatLab Interface.

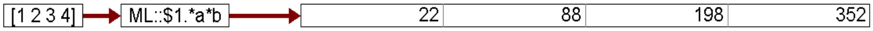
6.3.1 Evaluating MatLab expressions

The `MatLab Expression` block evaluates MatLab expressions using the MatLab engine. The `MatLab Expression` block accepts any number of matrix inputs and produces a single matrix output.

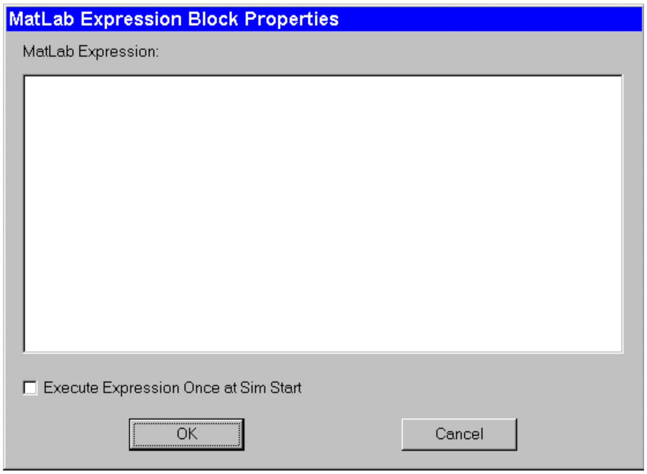
With MatLab expressions, you can more easily execute matrix operations. To illustrate a simple element multiply using the `MatLab Expression` block, consider the following example:

```
>> a = [1 2 3 4]

a =

          1 2 3 4

>> b = 22

b =

           22
```

Here, two matrices a and b are defined in the MatLab workspace. To multiply the matrices by a four element Commsim matrix ([1 2 3 4]), you set up your diagram as shown below:



Setting up a MatLab expression: The MatLab Expression Block Properties dialog box lets you set up your expression:



□ Execute Expression Once at Sim Start

MatLab Expression: Indicates a MatLab expression. A MatLab expression consists of one or more MatLab tokens. You enter expressions according to the syntax rules for the MatLab language. If you're unfamiliar with the language, refer to the MatLab documentation.

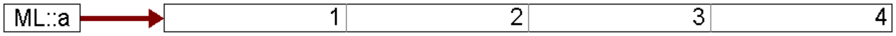
The `MatLab Expression` block accepts any number of matrix inputs and produces a single matrix output. You reference the inputs in the expression using the notation $1 for input 1, $2 for input 2, and so on.

The output that Commsim presents on the `expression` output connector is the MatLab variable ans.

Execute Expression Once at Sim Start: Gets the expression result from MatLab for the first time step.

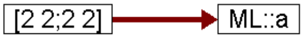
6.3.2 Reading and writing MatLab variables

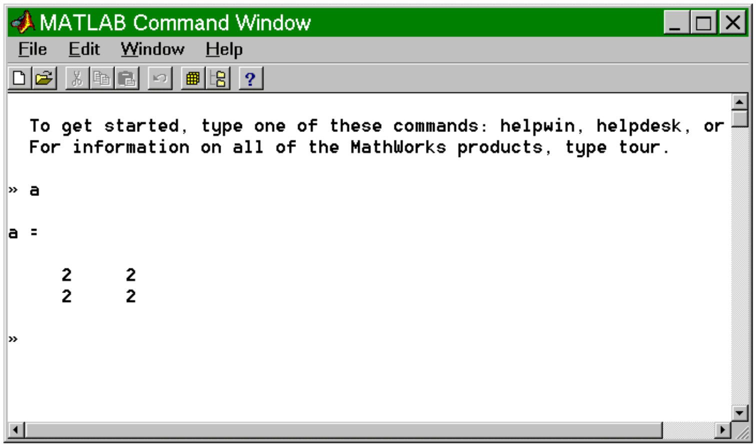
To read and write MatLab variables, use the `MatLab Read Variable` and `MatLab Write Variable` blocks. For example:



Here, the MatLab variable *a* is read into Commsim. The `display` block shows that *a* is a 1-by-4 matrix.

To write a variable to MatLab:





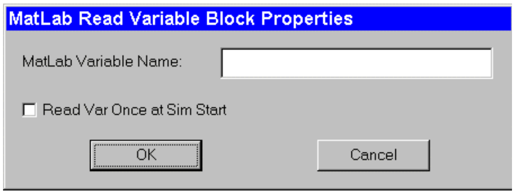
```
  To get started, type one of these commands: helpwin, helpdesk, or
  For information on all of the MathWorks products, type tour.

» a

a =

     2     2
     2     2

»
```

This time, the variable *a* is a 2-by-2 matrix, as shown in the MatLab command window.

➢ To read MatLab variables into Commsim:

1. Choose Blocks > MatLab Interface.
2. Click on the `MatLab Read Variable` block.
3. Click the mouse in the workarea.
4. Choose Edit > Block Properties and click on the `MatLab Read Variable` block.

   The MatLab Read Variable Block Properties dialog box appears.



MatLab Read Variable Block Properties

MatLab Variable Name:

□ Read Var Once at Sim Start

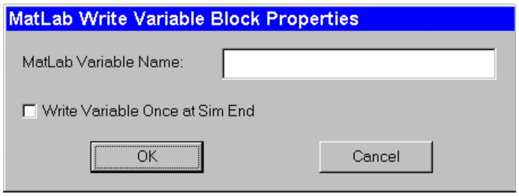
5. Do the following:
   - In the MatLab Variable Name box, enter the name of the variable to be read into Commsim.
   - Activate the Read Var Once at Sim Start option to retrieve the variable value at the first time step of the simulation.
   - Click on the OK button, or press ENTER.

6. To verify that the data has been properly passed to Commsim, do the following:
   - Feed the `MatLab Read Variable` block into a display block.
   - Choose Simulate > Go, or press the ▶ button

➢ To write variables into MatLab:

1. Choose Blocks > MatLab Interface.
2. Click on the `MatLab Write Variable` block.
3. Click the mouse in the workarea.
4. Choose Edit > Block Properties and click on the `MatLab Write Variable` block.

   The MatLab Write Variable Block Properties dialog box appears.



MatLab Write Variable Block Properties

MatLab Variable Name:

□ Write Variable Once at Sim End

5. Do the following:
   - In the MatLab Variable Name box, enter the name of the variable to be written to MatLab.
   - Activate the Write Variable Once at Sim End option if you want to send the variable value at the first time step of the simulation.
   - Click on the OK button, or press ENTER.
6. Feed matrix input into the `MatLab Write Variable` block.
7. Choose Simulate > Go, or press the ▶ button.
8. To verify that the variable has been passed to MatLab, do the following:
   - Go to the MatLab command window.
   - Type the name of the variable or “who”.

6.4 ActiveX basics

Commsim offers the `ActiveX read` and `ActiveX write` blocks for real-time, interprocess communication. Using these blocks, you can exchange data with other applications that support ActiveX. The `ActiveX read` block receives data from the Active X container; the `ActiveX write` block sends data to an ActiveX container.

Required files

To register properly, the Commsim ActiveX control requires the following files:

MFC42.DLL

MSVCRT.DLL

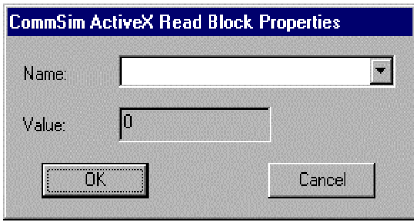
OLEAUT32.DLL

VSMIPCD.DLL

The first three files are located in your Windows system directory; the last file is located in the \Commsim7 directory.

6.4.1 Using the ActiveX read block

The `ActiveX read` block links source information in an application file to a Commsim block diagram. The source application must be an ActiveX container.



CommSim ActiveX Read Block Properties

Name:

Value: 0

Name: Indicates the name of the ActiveX control. Click on the DOWN ARROW to choose from existing names.

Value: Indicates the current value for the ActiveX control. This is a read-only box.

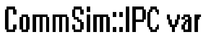
➢ To create an ActiveX link to read data into Commsim:

Follow this general procedure when you want to read data into Commsim. Because each application may handle ActiveX controls differently, see that application’s documentation for specific instructions.

After you have established your link between the application and Commsim, run the simulation to begin transmitting data.

1. In the application, insert the Commsim ActiveX control or OLE object.

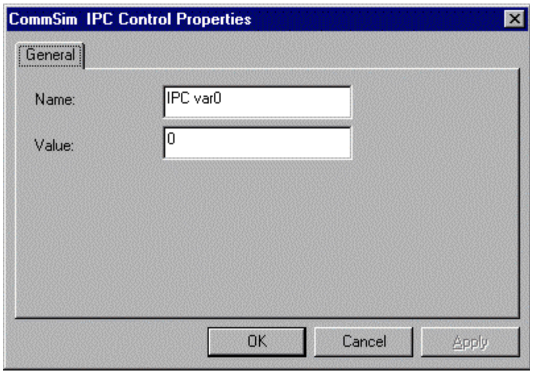
The following element appears in the workspace:



CommSim::IPC var

2. To manipulate the data:
   - Start Microsoft Visual Basic. In many applications, you can start Visual Basic from the application. For example, in Microsoft Excel 97, click on Tools > Macro > Visual Basic Editor.
   - Write the appropriate subroutine to manipulate data that is written into the ActiveX container.
3. Return to the application and invoke the Commsim ActiveX control properties dialog box. For most applications, just click the right mouse button over Commsim::IPC var and select Commsim IPC Control Object; then click on Properties.

   The Commsim IPC Control Properties dialog box appears.



CommSim IPC Control Properties

General

Name: IPC var0

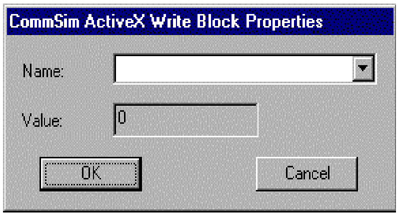
Value: 0

4. Do the following:
   - In the Name box, enter the name of the ActiveX control.
   - In the value box, enter a value for the control. Typically, the value is set dynamically using Microsoft Visual Basic.
   - Click on the OK button, or press ENTER.
5. Start Commsim and open the appropriate block diagram.
6. Click on Blocks > Real Time > ActiveX read.
7. Click the mouse where you want the `ActiveX read` block to be inserted.
8. Choose Edit > Block Properties.
9. Click on the `ActiveX read` block.
10. In the Name box, click on the DOWN ARROW and select the name of the ActiveX control that you specified in the corresponding Commsim IPC Control Properties dialog box (step 3).

11. Click on the OK button, or press ENTER.
12. Run the simulation.

6.4.2 Using the ActiveX write block

The `ActiveX write` block links source information in a Commsim block diagram to another application. The destination application must be an ActiveX container.



CommSim ActiveX Write Block Properties

Name:

Value: 0

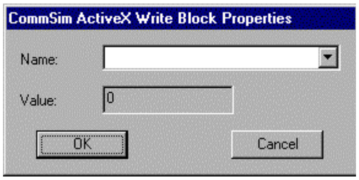
Name: Indicates the name of the ActiveX control. Click on the DOWN ARROW to select from a list of active names, or type in a new name.

Value: Indicates a value for the signal. Typically, the value is set dynamically by the application.

➢ To create an ActiveX link to write data to an ActiveX container:

1. Start Commsim and open the appropriate block diagram.
2. Click on Blocks > Real Time > ActiveX write.
3. Click the mouse where you want the `ActiveX write` block to be inserted.
4. Choose Edit > Block Properties.
5. Click on the `ActiveX write` block.

   The Commsim ActiveX Write Block Properties dialog box appears.



CommSim ActiveX Write Block Properties

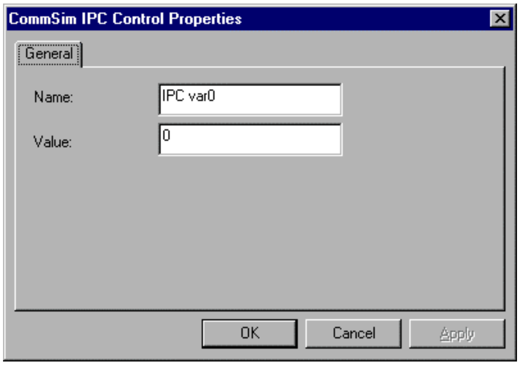
Name:

Value: 0

6. Do the following:
   - In the Name box, click on the DOWN ARROW and enter the name of the ActiveX control.
   - Click on the OK button, or press ENTER.

7. Switch to the destination application and open the file in which you want to create a link.
8. Insert the Commsim ActiveX control or OLE object.
9. Invoke the Commsim IPC Control Properties dialog box. For most applications, you click the right mouse button over Commsim::IPC var and select Commsim IPC Control Object; then click on Properties.

   The Commsim IPC Control Properties dialog box appears.



CommSim IPC Control Properties

Name: IPC var0

Value: 0

10. Do the following:
    - In the Name box, enter the name of the ActiveX control that correspond to the ActiveX control name in Commsim (see step 6).
    - In the value box, do not type anything. Typically, the value is set dynamically using Microsoft Visual Basic.
    - Click on the OK button, or press ENTER.
11. To manipulate the data:
    - Start Microsoft Visual Basic. In many applications, you can start Visual Basic from the application. For example, in Microsoft Excel 97, click on Tools > Macro > Visual Basic Editor.
    - Write the appropriate subroutine to manipulate data that is written into the ActiveX container. For information on writing subroutines, see the Microsoft Visual Basic documentation, go to the Object Browser for a list of available functions, and see “6.4.3 Commsim-specific ActiveX functions” on page 6-16.
12. Return to Commsim and run the simulation.

6.4.3 Commsim-specific ActiveX functions

Commsim supports two functions for ActiveX design:

- getlpcValue ()
- setlpcValue ()

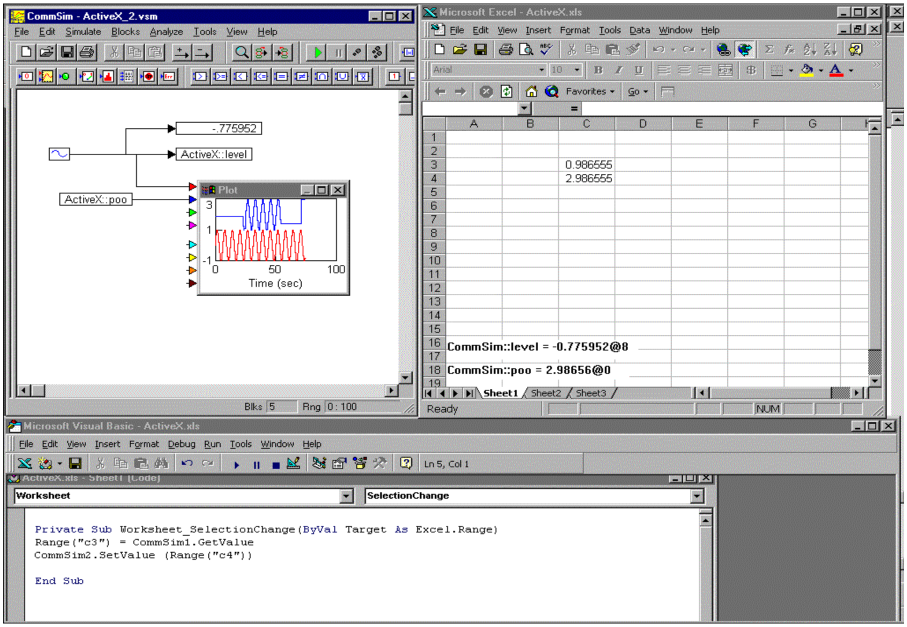
The getlpcValue function gets the 64-bit floating point value associated with the named data item. The setlpcValue function sets the 64-bit floating point value associated with the named data item.

6.4.4 Example

The following example demonstrates how to send data to Excel, manipulate the data and return it to Commsim in real time.

In the Commsim window, the `ActiveX write` block named *level* writes a sinusoidal signal to Excel. The value is displayed in cell C3. Cell C4 displays the value of C3 plus 2. The sum is sent back to Commsim through the `ActiveX read` block named *poo*.

A subroutine, shown in the Microsoft Visual Basic window, gets the data from *level* and displays it in cell C3; then writes the data from C4 to *poo*. This simple subroutine only updates the value of C4 and sends the data to Commsim each time you click the mouse on a different cell.


CommSim - ActiveX_2.vsm
File Edit Simulate Blocks Analyze Tools View Help
-.775952
ActiveX::level
ActiveX::poo
Plot
3
1
-1
0
50
100
Time (sec)
Blks 5
Rng 0 : 100
Microsoft Excel - ActiveX.xls
File Edit View Insert Format Tools Data Window Help
Arial
10
Favorites
Go
Sheet1
Sheet2
Sheet3
Ready
NUM
Microsoft Visual Basic - ActiveX.xls
File Edit View Insert Format Debug Run Tools Window Help
Ln 5, Col 1
ActiveX.xls - Sheet1 (Code)
Worksheet
SelectionChange

	A	B	C	D	E	F	G
1							
2							
3			0.986555				
4			2.986555				
16	CommSim::level = -0.775952@8						
18	CommSim::poo = 2.98656@0						

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
Range("c3") = CommSim1.GetValue
CommSim2.SetValue (Range("c4"))

End Sub
```

6.5 DDE basics

By creating dynamic data exchange (DDE) links, you can share information in one file with several other files, and you need only maintain the original file; the other files are updated automatically. For example, if you store data in a Microsoft Excel spreadsheet, you can use that data in a Commsim block diagram. When you update the spreadsheet, Commsim automatically updates the data in the block diagram when you run a simulation.

You create DDE links by copying a selection from one application (referred to as the source or server) and pasting it into another one (referred to as the destination or client) using the Paste Link or Paste Special command. Before you can create a link, the source file must be saved to disk.

Commsim offers three blocks for creating DDE links:

- **The `DDEsend` block,** which links source information in a Commsim block diagram to another application, such as a Microsoft Excel or Visual Basic file.
- **The `DDEreceive` block,** which links source information in an application file to a Commsim block diagram.

- **The DDE block,** which establishes a two-way link: it acts as both the source (sender) and destination (receiver). For example, a DDE block can send data to a Visual Basic program to work on, and then receive the updated data back from Visual Basic.

You can create DDE links only between Commsim and other Windows applications that support DDE linking. Some applications do support DDE links, but do not support creating the links by copying and pasting selections. When this is the case, you can still create a link by entering the link address directly to both the source and destination files.

6.5.1 Creating an app-to-Commsim link with DDEreceive

Follow this procedure when the source information for the link is contained in an application other than Commsim.

➢ To create a DDE link from an application into Commsim:

1. In the application, select the information you want linked to your block diagram, and from the Edit menu, choose the Copy command.

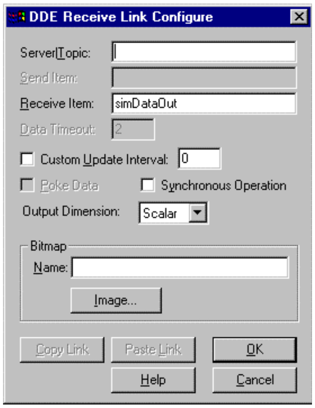
   The selected information is copied to the Clipboard.

2. Switch to Commsim and open the block diagram in which you want to create a DDE link.
3. Do one of the following:
   - Choose Edit > Paste Link.
   - Position the pointer where you want the DDEreceive block to appear and click the mouse.

   -Or-

   - From the Blocks menu under DDE, drag a DDEreceive block into the work area.
   - Choose Edit > Block Properties and click the mouse over the DDEreceive block.

4. The DDE Receive Link Configure dialog box appears.



Server|Topic:

Send Item:

Receive Item: simDataOut

Data Timeout: 2

☐ Custom Update Interval: 0

☐ Poke Data ☐ Synchronous Operation

Output Dimension: Scalar

Bitmap

Name:

5. Click on the Paste Link button and choose the options you want. (For information on the options, see the descriptions below.)
6. Click on the OK button, or press ENTER.
7. Choose Simulate > Go to update the link.

6.5.1.1 Using the DDE Receive Link Configure dialog box

The DDE Receive Link Configure dialog box provides the following options:

Server|Topic: Indicates the name of the source application (server) and the type of information (topic). For example, Excel|FOO.XLS indicates an Excel spreadsheet named FOO.

If the source application supports Copy Link, Commsim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the source application name and topic name directly to this box. Use the same names that the source file uses as its server and topic names. Separate the names with a pipe (|) character.

Send Item: This option does not apply to the `DDEreceive` block.

Receive Item: Indicates a name that references cells, cell ranges, values, or a field of data in the source file. For example, R1C1 references the information in the cell occupying row 1, column 1 of an Excel spreadsheet.

If the source application supports Copy Link, Commsim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the same name that the source file uses as its item name.

Data Timeout: This option does not apply to the `DDEreceive` block.

Custom Update Interval: Indicates how often the `DDEreceive` block requests information from the linked application. If you enter the value 1, `DDEreceive` requests information once per sec; if you enter 10, `DDEreceive` requests information once every 10 sec; and so on. If you do not enter a value, `DDEreceive` updates at each time step of the simulation by default.

Poke Data: This option does not apply to the `DDEreceive` block.

Synchronous Operation: Suspends the simulation until the `DDEreceive` block receives a message with updated data.

The `DDEreceive` block has a buffer that contains the current value of the block. If the block is not synchronous, at every time step, `DDEreceive` supplies whatever value is in its buffer. When Synchronous Operation is turned on and the `DDEreceive` block has not received updated data since the last time step, `DDEreceive` waits until it receives a new message with updated data.

Output Dimension: Controls the dimensionality of the data exiting the `DDEreceive` block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the `DDEreceive` block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

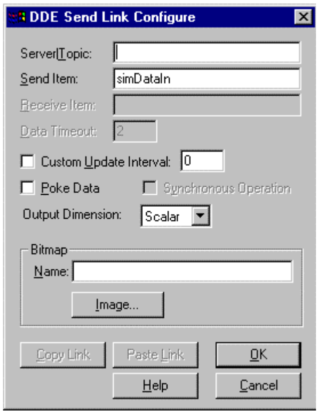
6.5.2 Creating a Commsim-to-app link with DDEsend

Follow this procedure when the source information for the link is contained in a block diagram.

➢ To create a DDE link from a Commsim block to another application:

1. In Commsim, wire a `DDEsend` block to the output of the block that contains the information you want linked to another application.
2. Choose Edit > Block Properties and click the mouse over the `DDEsend` block.

The DDE Send Link Configure dialog box appears.



DDE Send Link Configure

Server|Topic:

Send Item: simDataIn

Receive Item:

Data Timeout: 2

□ Custom Update Interval: 0

□ Poke Data □ Synchronous Operation

Output Dimension: Scalar

Bitmap

Name:

3. In the Send Item box, enter a name. The default name is simDataIn.

   Note: When the block diagram contains multiple links to other applications (that is, the diagram contains more than one `DDEsend` block), the name you enter in the Send Item box must be unique to that block diagram. If it's not unique, Commsim will not pass the correct information to the application.

4. Choose the options you want. (For information on the options, see the descriptions below.)
5. Click on the Copy Link button.
6. Click on the OK button, or press ENTER.
7. Switch to the destination application and open the file in which you want to create a link.
8. Position the insertion point where you want to insert the information.
9. Choose Edit > Paste Link.

   Note: Some applications have a Paste Special command instead of a Paste Link command. Refer to the application's documentation for information on linking.

10. Switch back to the block diagram, and choose Simulate > Go to update the link.

6.5.2.1 Using the DDE Send Link Configure dialog box

The DDE Send Link Configure dialog box provides the following options:

Server|Topic: Indicates the name of the source application (server) and the type of source information (topic). This parameter defaults to Commsim|*name-of-block-diagram*. The server name must always be Commsim.

Send Item: Indicates a name for the source information. The destination file uses this name in its item field. To maintain multiple DDE links from a single block diagram, the name you enter must be unique.

The information in this box defaults to simDataIn.

Receive Item: This option does not apply to the `DDEsend` block.

Data Timeout: This option does not apply to the `DDEsend` block.

Custom Update Interval: Overrides the time step interval for sending data to the destination application. If you enter the value 1, `DDEsend` sends data once per sec; if you enter 10, `DDEsend` sends data once every 10 sec; and so on. If you do not enter a value, `DDEsend` sends data at each time step of the simulation. You can use Custom Update Interval only when Poke Data is activated.

Poke Data: Sends data to the destination application at every time step, regardless of whether it is ready to receive the data. When Poke Data is not activated, data is sent only when the destination application requests it.

You can override the time step interval for sending data with the Custom Update Interval.

Output Dimensions: Controls the dimensionality of the data entering the `DDEsend` block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the `DDEsend` block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

6.5.3 Creating a two-way link with DDE

The `DDE` block is a combination of the `DDEreceive` block and `DDEsend` block: the `DDE` block can both send and receive information. As a server, the `DDE` block passes source information to another application to work on. As a client, the `DDE` block receives updated information back from the application.

➢ To create a two-way DDE link:

Use this procedure when you want Commsim to fill in the name of the server and topic pair of the destination application.

1. Create the link to pass information from the application to Commsim:
   - In the application, select the information you want linked to your Commsim block diagram, and choose Edit > Copy.

     The information is copied to the Clipboard.
   - Switch to Commsim and open the block diagram in which you want to link the copied information.
   - From the Blocks menu under DDE, drag a `DDE` block into the work area.
   - Choose Edit > Block Properties and click the mouse over the `DDE` block.

     The DDE Link Configure dialog box appears.

- Choose the Paste Link button and the additional options you want. (For information on the options, see the descriptions below.)
- Click on the OK button, or press ENTER.

2. Create the link to pass information from Commsim back to the application:

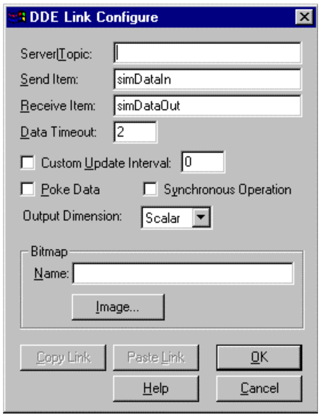
- In Commsim, wire the block (containing the information you want linked to the other application) to the DDE block.
- Choose Edit > Block Properties and click the mouse over the DDE block.

  The DDE Link Configure dialog box appears.
- In the Send Item box, enter a unique name.
- Choose the Copy Link button and the additional options you want. (For information on the options, see the descriptions below.)
- Click on the OK button, or press ENTER.
- Switch back to the application file.
- Move the insertion point to where you want to insert the information.
- Choose Edit > Paste Link.

Note: Some applications have a Paste Special command instead of a Paste Link command. Refer to the application's documentation for information on linking.

6.5.3.1 Using the DDE Link Configure dialog box

The DDE Link Configure dialog box provides options for establishing a DDE link, specifying the source information, indicating the time-out interval, and more.



Server|Topic:

Send Item: simDataIn

Receive Item: simDataOut

Data Timeout: 2

□ Custom Update Interval: 0

□ Poke Data □ Synchronous Operation

Output Dimension: Scalar

Name:

Server|Topic: Indicates the name of the application (server) and the type of information (topic) to which you’re establishing a link. For example, VBDDE|NNET sends data to and receives data from the application called VBDDE on the NNET topic. Use the pipe (|) character to separate the server from the topic.

Send Item: Indicates a name for the source information. The destination file uses this name for its item field. To maintain multiple DDE links from a single block diagram, the name you enter must be unique.

The information in this box defaults to simDataIn.

Receive Item: Indicates a name that references cells, cell ranges, values, or a field of data in the source file. For example, R1C1 references the information in the cell occupying row 1, column 1 of an Excel spreadsheet.

If the source application supports Copy Link, Commsim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the same name that the source file uses as its item name.

Custom Update Interval: Overrides the time step interval for sending and receiving information. If you enter 1, DDE requests and sends information once per sec; if you enter 10, DDE requests and sends information once every 10 sec; and so on. If you do not enter a value, DDE updates at each time step of the simulation.

Data Timeout: Indicates the time, in sec, that Commsim will wait to receive simulation time step data from the client. The default is two sec.

Poke Data: This option does not apply to the DDE block.

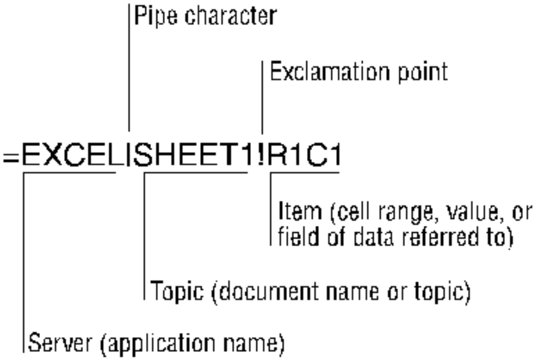
Synchronous Operation: This option does not apply to the DDE block.

Output Dimension: Controls the dimensionality of the data entering and exiting the block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the DDE block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

6.5.4 DDE links with applications that do not support Copy Link and Paste Link

A DDE link consists of a three-part link address contained in both the source (server) and destination (client) files. An example of such an address is shown below:


Pipe character
Exclamation point
=EXCEL|SHEET1!R1C1
Item (cell range, value, or field of data referred to)
Topic (document name or topic)
Server (application name)

Most applications, including Commsim, automatically create the link address using the Copy Link and Paste Link commands. If, however, the application with which you're linking supports DDE but not the Copy Link and Paste Link commands, you can still create a DDE link by typing the link address directly into the source and destination files. Just make sure that the server, topic, and item names are the same in source and destination files.

Refer to the descriptions of the `DDE`, `DDEreceive`, and `DDEsend` blocks earlier in this chapter for information on how to enter these fields directly into the blocks. Refer to the documentation for the other application for entering link addresses.

6.6 Look-up table basics

To create a look-up table, Commsim relies on the `map` block. There are two ways to set up a `map` block:

- Insert a map block directly into your diagram and choose the appropriate set-up parameters from its Properties dialog box
- Use the Look-up Table Wizard

The Look-up Table Wizard automates the process of setting up a `map` block by guiding you through the steps for specifying a data source, activating interpolation or extrapolation, and choosing the data file’s dimensionality.

The purpose of the Look-up Table Wizard and `map` block is to allow you to encapsulate nonlinear behaviors through direct measurements. You can, for example, use laboratory data or a manufacturer's component performance data directly in a simulation.

6.6.1 Specifying the data source

Whether you use the Look-up Table Wizard or set up the `map` block manually, you must create the data source, typically referred to as a map file. A map file is a multi-column ASCII data file from which the input signals are mapped to the desired output domain. Signals can be separated by commas, spaces, tabs, vertical bars, colons, semicolons, or slashes.

A map file can be a one-, two-, or three-dimensional file. The following rules apply to each file:

- **One-dimensional map files:** A one-dimensional map file has one independent variable, but can have from one to 16 dependent variable outputs. The first column is an independent variable range. The numbers in the independent variable column must be either increasing in order or decreasing in order, but not both. Each additional data column you supply in the map file yields an additional dependent variable. Use the Edit > Add Connector command to add an output connector for each dependent variable column in the resulting `map` block.

  A one-dimensional matrix is limited to 8,000 rows. Lines that are prefaced with a semi-colon (;) are treated as comments.

- **Two-dimensional map files:** A two-dimensional map file has two independent variables and one dependent variable output. The first row contains the domain points for the first independent variable; and the first column (excluding the column member in row 1) represents the second independent variable. The position (1,1) must be left blank.

  Like one-dimensional mapping, the independent variable values must be either monotically increasing or decreasing.

  A two-dimensional matrix is limited to 90 rows by 90 columns (or, a maximum of 89 * 89 data points).

  Lines that are prefaced with a semi-colon (;) are treated as comments.

  An example of a two-dimensional map file is shown below.

	10	**11**	**20**	**25**
-5	-5	-2	1	20
2	2	5	7	10
3	3	7	8	5
4	4	9	10	2
5	5	11	15	-5

In the above matrix, the first row represents the domain points of the first independent variable, and the first column represents the domain points of the second independent variable. The entries represent the dependent variable values at the corresponding values of independent variables 1 and 2. For example, for x_1 = 10, x_2 = 2, the output is 2; for x_1 = 10.5, x_2 = 2.5, the output is 4.25.

Three-dimensional map files: A three-dimensional map file has three independent variables and one dependent variable output. The format of the first seven lines is as follows:

Line	Format
1	Starts with #3D
2	Indicates the size of dimension 1
3	Indicates the interpolation points of dimension 1
4	Indicates the size of dimension 2
5	Indicates the interpolation of dimension 2
6	Indicates the size of dimension 3
7	Indicates the interpolation of dimension 3

Lines 8 through n are elements of dimension 3 matrices of (dimension 1 columns) * (dimension 2 rows). Lines that are prefaced with double hyphens (--), double slashes (//), or commas (,) are treated as comments.

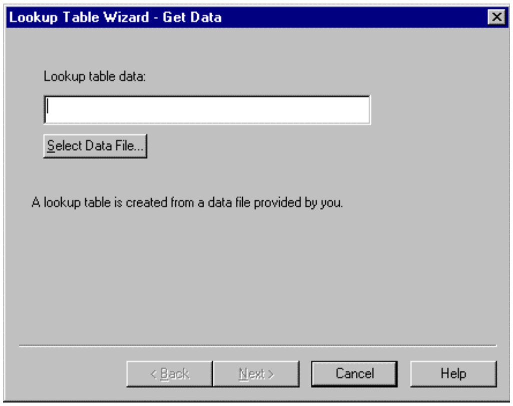
Dependent variables are linearly interpolated for independent variable values between map points, and linearly extrapolated for values beyond the bounds of the table using the last two points in the table. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.

6.6.2 Using the Look-up Table Wizard

➢ To create a look-up table:

1. Choose Tools > Import > Lookup Table.

   The Get Data dialog box appears.



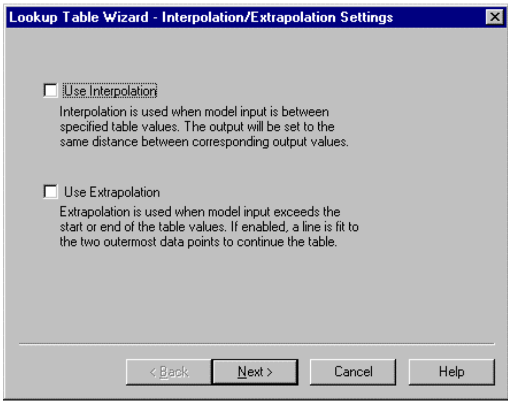
Lookup Table Wizard - Get Data

Lookup table data:

A lookup table is created from a data file provided by you.

2. In the Lookup Table Data box, enter the name of the map file to be used as input. You can type the name directly into the text box, or select one using the Select Data File button.
3. Click on the Next button, or press ENTER.

   The Interpolation/Extrapolation dialog box appears.



Lookup Table Wizard - Interpolation/Extrapolation Settings

☐ Use Interpolation

Interpolation is used when model input is between specified table values. The output will be set to the same distance between corresponding output values.

☐ Use Extrapolation

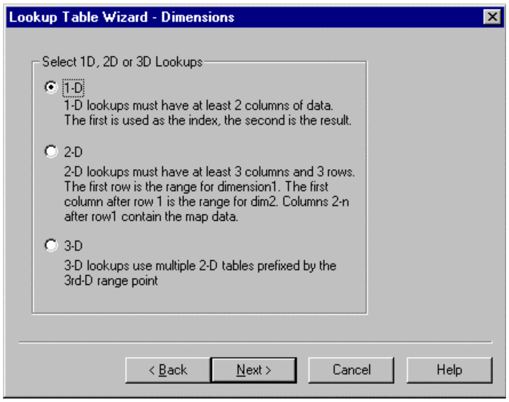
Extrapolation is used when model input exceeds the start or end of the table values. If enabled, a line is fit to the two outermost data points to continue the table.

4. Under Interpolation Settings, make the desired selections.

If you select	This occurs
Interpolation	Allows dependent variables to be linearly interpolated for independent variable values between data points.
Extrapolation	Allows dependent variables to be linearly extrapolated for values beyond the bounds of the table using the last two data points in the table.

5. Click on the Next button, or press ENTER.

   The Map Dimensions dialog box appears.



Lookup Table Wizard - Dimensions

Select 1D, 2D or 3D Lookups

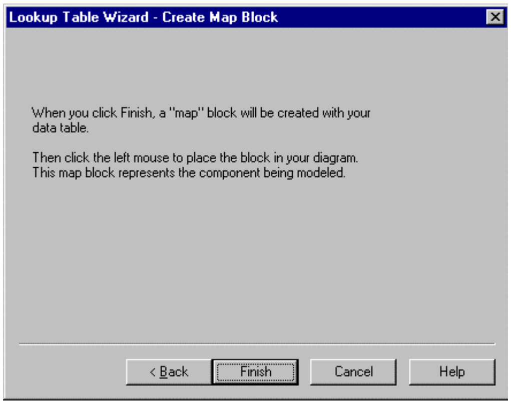
● 1-D
1-D lookups must have at least 2 columns of data. The first is used as the index, the second is the result.

○ 2-D
2-D lookups must have at least 3 columns and 3 rows. The first row is the range for dimension1. The first column after row 1 is the range for dim2. Columns 2-n after row1 contain the map data.

○ 3-D
3-D lookups use multiple 2-D tables prefixed by the 3rd-D range point

6. Choose the dimensionality of the map file.
7. Click on the Next button, or press ENTER.

   The Create Map Block dialog box appears.



Lookup Table Wizard - Create Map Block

When you click Finish, a "map" block will be created with your data table.

Then click the left mouse to place the block in your diagram. This map block represents the component being modeled.

8. Click on the Finish button, or press ENTER.

Commsim displays the current block diagram. Attached to the pointer is a dashed rectangle representing the newly-created `map` block.

9. Click the mouse to insert the `map` block in your diagram.

6.6.3 Using the map block

You can set up a look-up table or change an existing table without using the Look-up Table Wizard. You can alternatively insert a `map` block directly into your diagram and specify the data source, map dimensionality, and whether data is interpolated or extrapolated through the block's Properties dialog box.

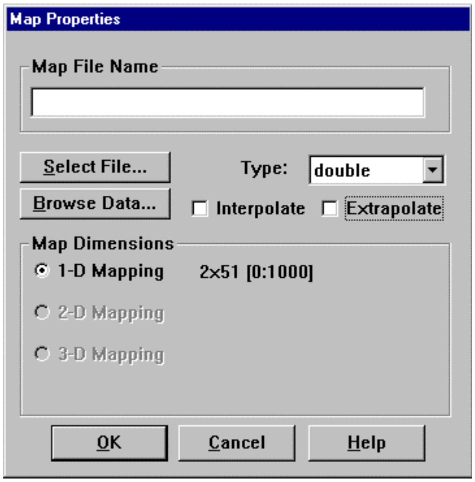
➢ To insert a map block:

1. Choose Blocks > Nonlinear.
2. Click on the `map` block.
3. Click in the work area where you want to insert the `map` block.

➢ To set up, view, or change map block parameters:

1. Choose Edit > Block Properties.
2. Click the mouse over the map block.

   The Map Properties dialog box appears.



Type: double

☐ Interpolate ☐ Extrapolate

● 1-D Mapping 2x51 [0:1000]

○ 2-D Mapping

○ 3-D Mapping

3. In the Map File Name box, enter the name of the map file. You can type in a file name directly into this box or select one using the Select File button. To open the specified file with the default text editor, click on the Browse Data button

4. Under Map Dimensions, select the dimensionality of the map file.
   - **1-D Mapping:** Indicates 1-D mapping capability.

     The numbers to the right of the 1-D Mapping parameter refer to the dimensionality and range of the map vector. For example, 10x1[1:100] represents a 1-D table with 10 elements ranging from 1 to 100.

     Lines that begin with a semi-colon (;) are treated as comments.
   - **2-D Mapping:** Provides simultaneous mapping for two independent variables.

     Lines that begin with a semi-colon (;) are treated as comments.

     The numbers to the right of the 2-D Mapping parameter refer to the dimensionality and range of the map vector. For example, 10x50[10:20, -10:10] represents a 2-D table with 10 columns and 50 rows, where the minimum column is 10, the maximum column is 20, the minimum row is -10, and the maximum row is 10.
   - **3-D Mapping:** Provides simultaneous mapping of three independent variables. Lines that begin with a double hyphen (--), semi-colon (;), or double slash (//) are treated as comments.
5. In the Type box, click on the down arrow and select the type of data read in from the map file.
6. Select one, both, or neither of the following options, depending on how you want the data interpreted.

Choose this	**To do this**
Interpolate	Allows dependent variables to be linearly interpolated for independent variable values between data points. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.
Extrapolate	Allows dependent variables to be linearly extrapolated for values beyond the bounds of the table using the last two data points in the table. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.

7. Click on the OK button or press ENTER.

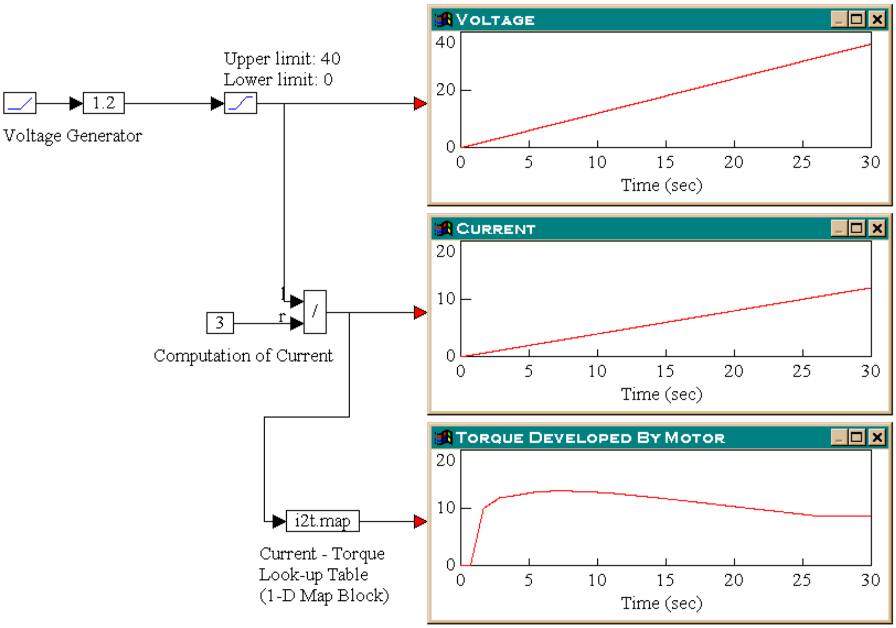
6.6.4 Examples

6.6.4.1 1-D look-up table

Consider a hypothetical electrical motor that accepts DC input voltage in the range of 0 to 40 V. Furthermore, assume that the current drawn by the motor is equivalent to that of an ideal 3Ω resistor. The motor manufacturer has specified the following current-torque curve for the motor:

Current (A)	Torque (N-M)
0.	0.
.3	0.
.68	10.
1.15	11.8
2.16	12.77
2.86	13.04
3.7	12.86
4.36	12.66
5.74	11.84
6.73	11.18
10.5	8.62
11.5	8.62

Assuming that the voltage is applied at the rate of 1.2 *t*, where *t* is time in sec. This system can be realized as shown below.



To generate the voltage, wire a `ramp` block, used to generate simulation time *t*, to a `gain` block set to 1.2. The output of the `gain` block passes through a `limit` block with its lower and upper limits set to 0 and 40, respectively. The output of the `limit` block is the voltage applied to the motor and is monitored in the upper plot.

To compute the current, divide the output of the `limit` block is by a constant value 3. The current is monitored in the middle plot.

A `map` block points to the data file I2T.MAP, with two columns of data containing the current-torque curve for the motor. (The sample data used in this example is shown in the table above.)

The `map` block monitors the input value and compares it with the data in the first column. For example, if the input value is 3, the `map` block recognizes that the input is between the two points 2.86 and 3.7 in the input column. The `map` block performs a linear interpolation between the corresponding values in the second column, namely 13.04 and 12.86. Consequently, for an input of 3, the output of the `map` block is:

$$13.04 + (3 - 2.86) * ((12.86 - 13.04) / (3.7 - 2.86))$$

which is equal to 13.01.

6.6.4.2 2-D look-up table

Using the same hypothetical motor described above, make the following assumptions:

- The torque developed by the motor is a function of the current, as well as the operating temperature of the motor.
- The current-temperature-torque has the following profile:

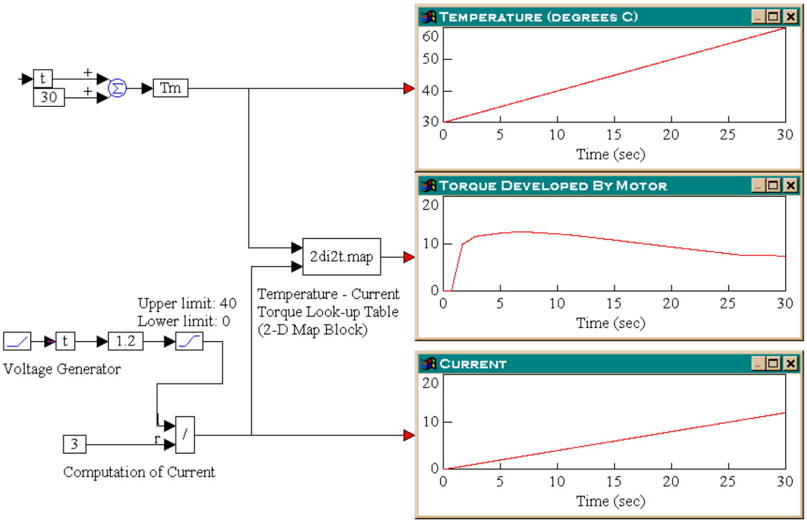
	Torque (Temperature Dependent) (N-M)			
Current (A)	Torque at 30° C	Torque at 40° C	Torque at 50° C	Torque at 60° C
0.	0.	0.	0.	0.
0.3	0.	0.	0.	0.
0.68	10	9.5	9.10	8.7
1.15	11.8	11.21	10.74	10.27
2.16	12.77	12.13	11.62	11.19
2.86	13.04	12.39	11.87	11.35
3.7	12.86	12.22	11.70	11.18
4.36	12.66	12.03	11.52	11.01
5.74	11.84	11.25	10.77	10.30
6.73	11.18	10.62	10.17	9.73
10.5	8.62	8.19	7.84	7.50
11.5	8.62	8.19	7.84	7.50

- The motor temperature profile is given by $T_m = (30 + t)^0$ C, where T_m is the motor temperature and *t* is time in seconds.
- A data file named 2DI2T.MAP is formatted as shown below:

	30.	**40.**	**50.**	**60.**
0.	0.	0.	0.	0.
0.3	0.	0.	0.	0.
0.68	10	9.5	9.10	8.7
1.15	11.8	11.21	10.74	10.27
2.16	12.77	12.13	11.62	11.19
2.86	13.04	12.39	11.87	11.35
3.7	12.86	12.22	11.70	11.18
4.36	12.66	12.03	11.52	11.01
5.74	11.84	11.25	10.77	10.30
6.73	11.18	10.62	10.17	9.73
10.5	8.62	8.19	7.84	7.50
11.5	8.62	8.19	7.84	7.50

The values of the temperature are entered in row 1, starting with column 2. The values of current are entered in column 1, starting with row 2. The values of the current and the temperature are shown in bold type for clarity.

Using a 2D `map` block, the system simulation can be realized as:



In addition to the blocks used in Example 1, a `variable` *t*, defined as simulation time, is connected to the output of the `ramp` block. A `map` block is used to access 2DI2T.MAP. Because this data file is a 2-D look-up table, the `map` block accepts two inputs: temperature (the independent variable in the first row) and current (the independent variable in the first column).

To generate the temperature profile, a `const` block of 30 is added to `variable` *t*, fed through another `variable` *Tm*, and monitored in the top `plot` block.

As before, the outputs of the `map` block, and the `/` block are monitored to observe the profiles of the motor torque and current, respectively.

During simulation, when the temperature is ≥30^{o} C and <40^{o} C, the second column of data is used to generate the torque profile. Similarly, for temperatures that are ≥40^{o} C and <50^{o} C or ≥50^{o} C and <60^{o} C, data in the third and fourth columns is used respectively.

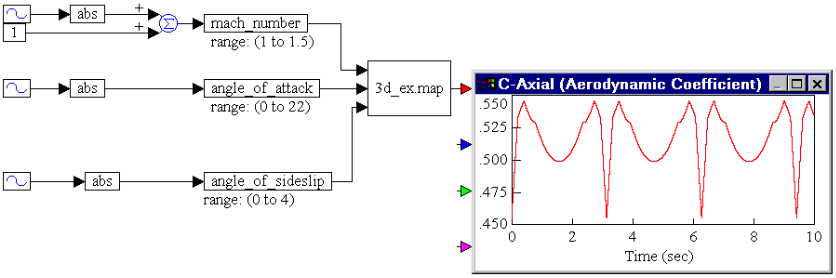
6.6.4.3 3-D look-up table

The structure and usage of a Commsim diagram that includes a 3-D look-up table is very similar to a 2-D look-up table. The major difference is in the specification of the data file to be used by the 3-D `map` block. As an example, consider the following data file:

```
#3D_EX.MAP

--Table_Caxial_0 5 10 3

--Mach No. breakpoints

5

   1.05   1.10   1.20   1.35   1.50

--Angle_of_Attack breakpoints

10

   4.00   6.00   8.00  10.00  12.00  14.00  16.00  18.00  20.00
22.00

--Angle_of_Sideslip breakpoints

3

   0.00   2.00   4.00

--

-- Angle_of_Sideslip =   0.00


   0.493   0.533   0.550   0.529   0.496

   0.492   0.532   0.549   0.529   0.497

   0.491   0.530   0.547   0.529   0.497

   0.488   0.528   0.545   0.529   0.497

   0.485   0.525   0.541   0.529   0.497

   0.481   0.520   0.536   0.529   0.497

   0.474   0.513   0.528   0.530   0.498

   0.465   0.504   0.517   0.530   0.498

   0.452   0.488   0.482   0.530   0.498

   0.430   0.439   0.444   0.531   0.499

--

-- Angle_of_Sideslip =   2.00

   0.493   0.533   0.550   0.529   0.496
```

```
                  0.492   0.532   0.549   0.529   0.497
                  0.490   0.530   0.547   0.529   0.497
                  0.488   0.528   0.544   0.529   0.497
                  0.485   0.525   0.541   0.529   0.497
                  0.480   0.520   0.535   0.529   0.497
                  0.474   0.513   0.528   0.530   0.498
                  0.465   0.503   0.516   0.530   0.498
                  0.451   0.487   0.480   0.530   0.498
                  0.429   0.437   0.442   0.531   0.499
               --
               -- Angle_of_Sideslip =   4.00
                  0.493   0.532   0.549   0.529   0.496
                  0.491   0.531   0.548   0.529   0.497
                  0.490   0.530   0.546   0.529   0.497
                  0.487   0.527   0.543   0.529   0.497
                  0.484   0.524   0.540   0.529   0.497
                  0.479   0.519   0.534   0.529   0.497
                  0.473   0.512   0.527   0.530   0.498
                  0.463   0.502   0.514   0.530   0.498
                  0.449   0.485   0.476   0.530   0.498
                  0.426   0.433   0.437   0.531   0.499
--
```

This data corresponds to one of the aerodynamic coefficients of a projectile in motion, traveling at speeds ranging from 1 to 1.5 mach. The value of the coefficient varies with three parameters: mach number, angle of attack, and angle of sideslip. Assuming sinusoidal variations in all three parameters, a diagram that uses this data file can be realized as shown below.



Three `sin` blocks produce sinusoidal variations of amplitudes 0.5, 22, and 4 for the three `variables` *mach_number, angle_of_attack*, and *angle_of_slideslip*. Three `abs` blocks ensure that the values attained by the `variables` are strictly positive. A constant value of 1 is used to obtain a variation in the range (1, 1.5) for *mach_number.*

The outputs of the three `variable` blocks are fed into a `map` block that points to the map file 3D_EX.MAP, whose contents are shown above. The resulting value of the C-axial aerodynamic coefficient are shown in the `plot` block.

6.7 Commsim - Mathcad 2000 interface basics

The primary purpose of embedding a Mathcad document is to run a simulation in Commsim that depends on one or more Mathcad calculations. These calculations could be performed:

- To provide initial conditions for a Commsim simulation (evaluated once at the beginning)
- As a part of the simulation (evaluated at every simulation step)
- Using the end results of a Commsim simulation as inputs (evaluated only once at the end)

6.7.1 Using the in*x* and out*x* keywords

The fundamental means of setting up the exchange of data between a Commsim simulation and an embedded Mathcad document is the use of the keywords in*x*, and out*x* in the embedded Mathcad document.

- The keywords in0, in1, in2,… refer to the input connector tabs on the an embedded Mathcad document, where in0 corresponds to the top input connector.
- The keywords out0, out1, out2,… refer to the output connector tabs on the an embedded Mathcad document, where out0 corresponds to the top output connector.

Note that all keywords must be lowercase.

6.7.2 Embedding Mathcad documents

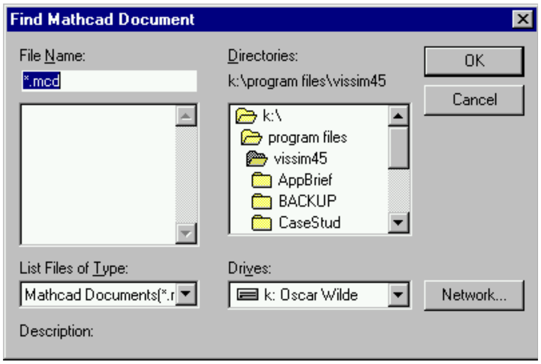
You can embed existing objects from files or insert new blank objects and create the information right in your diagram. To do so, you use the Insert Mathcad Object command in the Tools menu.

The following procedures explain how to embed objects in a Commsim diagram. For a step-by-step example on embedding a Mathcad document in a Commsim diagram, see “6.7.8 Examples” on page 6-45.

➢ To embed an existing Mathcad document:

1. Choose Tools > Insert Mathcad Object > From File.

   The Find Mathcad Document dialog appears.



File Name: *.mcd

Directories: k:\program files\vissim45

k:\
program files
vissim45
AppBrief
BACKUP
CaseStud

List Files of Type: Mathcad Documents(*.r

Drives: k: Oscar Wilde

Description:

2. Find and select the Mathcad document (.MCD) you want to embed in your Commsim diagram.
3. Click on the OK button, or press ENTER.

➢ To embed a new Mathcad document:

1. Choose Tools > Insert Mathcad Object >New.

   Commsim opens a Mathcad window in which to create an embedded Mathcad document.
2. Create the document. For information on using Mathcad, see your Mathcad documentation.
3. Click outside the Mathcad document to return to Commsim.

6.7.3 Editing and updating Mathcad documents

You can edit an embedded Mathcad document directly from Commsim, or you can invoke Mathcad as a separate, stand-alone application. To update the embedded Mathcad document and return to the Commsim environment, simply click outside the Mathcad document.

➢ To edit an embedded Mathcad document directly from Commsim:

1. Double-click on the embedded Mathcad document.

   -Or-

   Click the right mouse button on the embedded Mathcad document.

   From the drop-down menu, choose Mathcad Object > Edit.

   Commsim opens a Mathcad window with the Mathcad document.
2. Make the changes you want. For information on using Mathcad, see your Mathcad documentation.
3. Click outside the Mathcad document to update the Mathcad document and return to Commsim.

➢ To invoke Mathcad as a stand-alone application:

1. Click the right mouse button on the embedded Mathcad document.
2. From the drop-down menu, choose Mathcad Object > Open.

   Commsim opens a Mathcad window with the Mathcad document.
3. Make the changes you want. For information on using Mathcad, see your Mathcad documentation.
4. To update the Mathcad document and return to Commsim, do the following:
   - Choose File > Update.
   - Choose File > Exit and Return to.

6.7.4 Scaling and cropping embedded Mathcad documents

You can resize an embedded Mathcad document by scaling or cropping it. When you scale an embedded Mathcad document, the text within the block is also scaled. When the dimensions of the embedded Mathcad document are much larger than the text, you can adjust the size of the block by cropping its sides. The size of the text within the block is not affected by this action.

- ➢ To scale an embedded Mathcad document manually:

1. Click on the embedded Mathcad document.
2. Position the pointer over one of the handles on the block.
3. Hold down the mouse button and drag until the block is the size you want.
4. Click outside the Mathcad document to return to Commsim.

- ➢ To scale an embedded Mathcad document using the Mathcad Properties dialog box:

1. Click right mouse button over the embedded Mathcad document.
2. In the drop-down menu, choose Properties.
3. The Mathcad Properties dialog box appears.
4. Click on the View tab.
5. To scale the block with respect to its original size, activate the Relative To Original Size option.
6. In the Scale box, select the scaling factor.
7. Click on the Apply button to preview the new size.
8. Click on the OK button to make the change permanent.

- ➢ To crop an embedded Mathcad document manually:

1. Double-click on embedded Mathcad document.
2. Position the pointer over one of the handles on the block.
3. Hold down the mouse button and drag until the block is the size you want.
4. Click outside the Mathcad document to return to Commsim.

- ➢ To reset the size of an embedded Mathcad document:

You can use this method when you cropped the embedded Mathcad document or when you scaled in manually.

1. Click the right mouse button on the embedded Mathcad document.
2. From the drop-down menu, choose Reset Size.

6.7.5 Examining the properties of an embedded Mathcad document

The Mathcad Properties dialog box lists the size of your embedded Mathcad document, and allows you to control the general appearance of the document in your Commsim diagram. You can also use the Mathcad Properties dialog box to scale the corresponding embedded document.

- To access general information about an embedded Mathcad document:

  You can display the size and location of the embedded Mathcad document.

  1. Click right mouse button over the embedded Mathcad document.
  2. In the drop-down menu, choose Properties.
  3. The Mathcad Properties dialog box appears.
  4. Click on the General tab.

- To control the appearance of an embedded Mathcad document:

  You can display the size and location of the embedded Mathcad document.

  1. Click right mouse button over the embedded Mathcad document.
  2. In the drop-down menu, choose Properties.
  3. The Mathcad Properties dialog box appears.
  4. Click on the View tab.
  5. Do the following:

To	Do this
Display the contents of the embedded Mathcad document	Activate the Display As Editable Information option.
Display a Mathcad icon	Activate the Display As Icon option. Click on the Change Icon button to select the type of icon to be displayed.

  6. Click on the OK button.

6.7.6 Changing number of connector tabs on an embedded Mathcad document

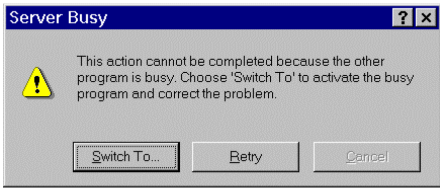
To add or delete connectors to an embedded Mathcad document, follow the standard procedures for adding or deleting connectors on Commsim blocks.

6.7.7 Troubleshooting

This section contains information that may help you work through problems you encounter.

6.7.7.1 Server Busy message

Occasionally, when Commsim loads the Mathcad interface OLE control for the first time, you may experience a delay and the appearance of the following message:



Server Busy

This action cannot be completed because the other program is busy. Choose 'Switch To' to activate the busy program and correct the problem.

6.7.7.2 Incorrect results

The most common reason for getting unexpected results when using an embedded Mathcad document is due to input/output mismatch caused by incorrect numbering.

When you are entering an expression or equation in an embedded Mathcad document, it is important to remember that the input and output keywords begin with the index of 0, and not 1. In other words, the first (that is, top) input connector on the embedded Mathcad document is represented in the Mathcad expressions by the keyword in0; the second input by the keyword in1, and so on. Similarly on the output side, the first (that is, top) output of the embedded Mathcad document takes the value defined by the keyword out0; the second output takes the value of out1, and so on.

6.7.8 Examples

This section describes:

- A step-by-step procedure to perform a basic arithmetic operation in Mathcad.
- A pre-configured room temperature control system using Mathcad.

6.7.8.1 Setting up simple calculations in embedded Mathcad documents

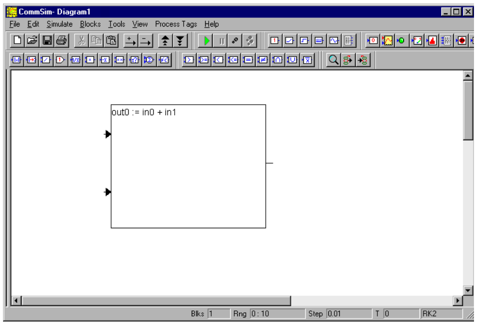
This example passes two values to Mathcad. Mathcad performs a simple arithmetic operation and returns the result to Commsim. This example is included in the …/ MATHCAD_EXAMPLES directory and is named SIMPLE_EXAMPLE.VSM.

1. Start Commsim.
2. Choose Tools > Insert Mathcad Object > New.

   Commsim creates a Mathcad object window. The environment automatically changes to the standard Mathcad editing work-area.
3. In the Mathcad object window, type the following expression:

   out0 := in0 + in1

   Important: The keywords out0, in0, and in1 must be lowercase.
4. Click outside the Mathcad object window to return to the Commsim editing environment. The menus automatically change back to Commsim menus.
5. Attach one more input connector tab to the embedded Mathcad document.

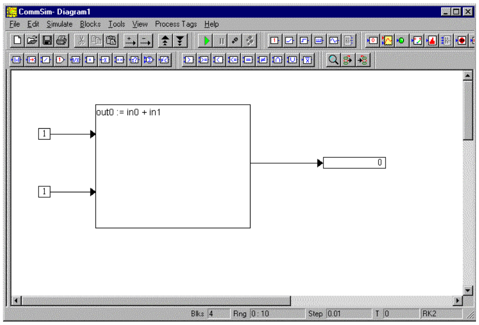


out0 := in0 + in1

Blks 1 Rng 0 : 10 Step 0.01 T 0 RK2

6. Wire two `const` blocks (Blocks > Signal Producer > `const`) to the inputs on the embedded Mathcad document and set their parameter values to 3 and 4; then wire a

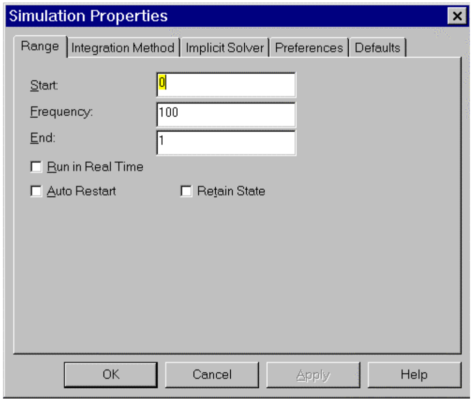
numerical `display` block (Blocks > Signal Consumer > `display`) to the output on the embedded Mathcad document to see the result.



7. Choose Simulate > Simulation Properties.

   The Simulation Properties dialog box appears.

8. Set the Start and End values to 0 and 1, respectively.



Simulation Properties

Range | Integration Method | Implicit Solver | Preferences | Defaults

Start: 0

Frequency: 100

End: 1

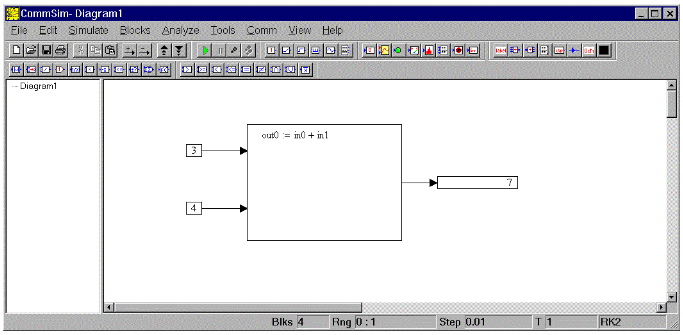
□ Run in Real Time

□ Auto Restart   □ Retain State

9. Click on the OK button, or press ENTER.

10. Choose Simulate > Go, or click on the Run button in the toolbar. You can alternatively press the F5 key on your keyboard.

Commsim passes the values of the constants you have selected to Mathcad via the embedded Mathcad document, obtains the computed output value, and displays it in the `display` block.



out0 := in0 + in1

Blks 4 Rng 0 : 1 Step 0.01 T 1 RK2

6.7.8.2 Room temperature control: ROOMCTRL_MC.VSM

This model simulates the closed-loop temperature control of a room, with calculations being performed simultaneously in Commsim/Comm and Mathcad based on a bi-directional exchange of computational results at every simulation step.

The controller itself is a simple yet realistic ON-OFF controller with deadband. Most commercial and domestic room temperature controllers use this implementation. In this simulation, the deadband is implemented in an embedded Mathcad document, and the rest of the simulation including numerical integration and dynamic room model with randomly varying number of occupants, is implemented in Commsim.

6.8 WMF basics

One of the most common image file formats for the PC is the Windows Metafile Format (WMF). WMF files are vector files that look good when printed on high resolution devices.

When Commsim saves a diagram as a WMF file, it uses the number of colors available to your monitor. In general, you can use the generated WMF file in any application that recognizes this file format; however, in some cases, if you have set the size of your monitor's color palette to greater than 256 colors, you may experience trouble printing or displaying the WMF file from the other application. In the unlikely event that this occurs, simply change the size of you color palette to 256 colors and resave the diagram.

6.8.1 Saving a diagram as a WMF file

When you save a Commsim diagram as a WMF file, Commsim saves only the current level of the diagram and strips away common window elements (including title bars, menu bars, and scroll bars). You should also keep in mind that the WMF file is a picture of the diagram; therefore, none of the block diagram information is preserved.

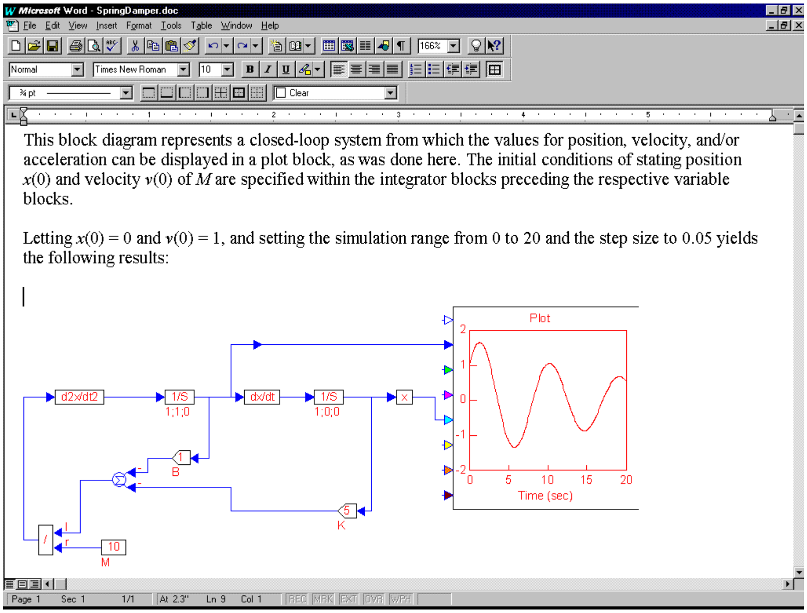
➢ To save a Commsim diagram as a WMF file:

1. Go to the level of the diagram that you want saved as a WMF file.
2. Choose File > Save As Metafile.
3. In the Save In box, enter or select the directory in which the WMF file is to be saved.
4. In the File Name box, enter a name for the WMF file.
5. Click on the OK button, or press ENTER.

6.8.2 Inserting a WMF file in another application

To insert a WMF file in another application, you use a command like Import, Insert, or Picture. For example, to insert a WMF file in a Word document, you use the Picture command under the Insert menu. Similarly, to insert a WMF file in a Pagemaker document, you use the Place command under the File menu. Because each application has its own unique method for inserting WMF files, you must read that application's documentation for its particular procedure.

Below is an example of a WMF file of a Commsim diagram that has been inserted in a Word document.



This block diagram represents a closed-loop system from which the values for position, velocity, and/or acceleration can be displayed in a plot block, as was done here. The initial conditions of stating position *x*(0) and velocity *v*(0) of *M* are specified within the integrator blocks preceding the respective variable blocks.

Letting *x*(0) = 0 and *v*(0) = 1, and setting the simulation range from 0 to 20 and the step size to 0.05 yields the following results:

Chapter 7
Wireless Lan

The following are described in this chapter.

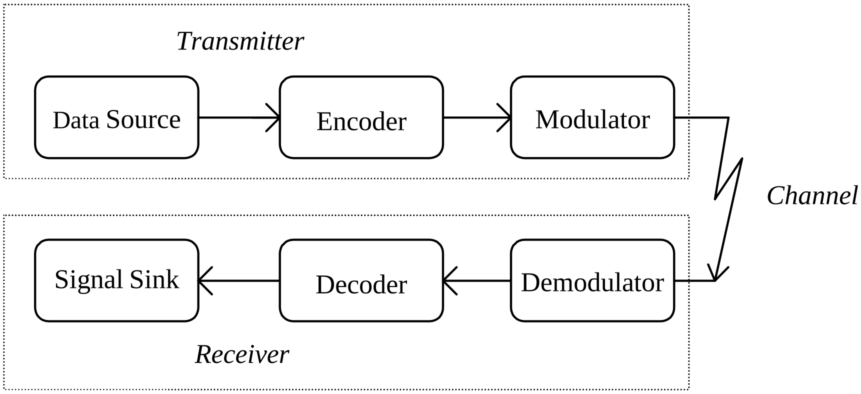
The WLAN module extends the capabilities of the Commsim physical layer simulation environment by including support for the Bluetooth and 802.11a and 802.11b wireless standards. Both standards are commonly used in the design of Wireless Local Area Networks (WLAN). In particular, the 802.11b standard is quickly gaining acceptance in the implementation of “Wi-Fi” networking components.

Many thanks to Jian Sun and Dr. Matt Valenti of West Virginia University for collaborating in the development of this Wireless module.

7.1 Using Commsim WLAN

7.1.1 Wireless system communication elements

The physical layer of a typical wireless link includes, at a minimum, the elements depicted in the following figure. Many of the blocks required to model these elements are already provided by the standard Commsim block set, but several more complex functions found in wireless systems cannot easily be simulated using the standard Comm block set. The WLAN module fills this gap by including additional blocks specific to the Bluetooth and IEEE 802.11 standards.



The new blocks included in the Wireless LAN module provide for the most part Encoder/Decoder and Modulator/Demodulator functions. A full listing of these blocks is provided in the next section.

7.1.2 Wireless blocks

The blocks included in the Wireless LAN module are organized in five categories according to the specific Standard they support:

- Bluetooth
- 802.11 (1 Mbps and 2 Mbps modes)
- 802.11a (OFDM modulation at 6 ~ 54 Mbps)
- 802.11b (CCK modulation at 5.5 and 11 Mbps)
- Generic Wireless

Bluetooth

`GFSK, Hop Generator, Scrambler, Short Hamming Encoder, Short Hamming Decoder`

802.11

`Barker Sequence, CRC-16, Descrambler, GFSK-2, GFSK-4, Hop Generator, Scrambler`

802.11a

`Convolutional Encoder, Depuncture, Interleaver, Puncture, Scrambler, Viterbi Decoder, OFDM Demodulator, OFDM Modulator, OFDM Pilot Extract, OFDM Pilot Map, OFDM Vector Demodulator, OFDM Vector Modulator`

802.11b

`High Rate Hop Generator, CCK Baseband Modulator, CCK Demodulator`

Generic Wireless

`Frequency Hop`

7.1.3 Bluetooth Overview

The Bluetooth standard provides wireless data and/or voice communication between multiple peripherals located in close proximity of each other (typical range is 10 meters). It can be operated in point-to-point or point-to-multipoint modes, and exhibits a channel symbol rate of 1 Mbps, with a maximum payload rate of 723 kbps. A Bluetooth network is also referred to as a “piconet”.

In each piconet one Bluetooth unit acts as the master and the remaining units act as slaves, with up to seven active slave units being allowed. Access to the channel is controlled by the master. A Time-Division Duplex (TDD) communication scheme is used , in which packets are exchanged between master and slave in alternating fashion.

Bluetooth operates in the 2.4 GHz ISM (Industrial Scientific Medicine) microwave band, and employs a frequency hopping scheme spanning 79 channels (1 MHz spacing) and with a hopping rate of 1.6 kHz. The pseudo-random hopping pattern is derived from a combination of the master clock time and the unique device address of each unit.

The modulation used in Bluetooth is Gaussian Frequency Shift Keying (GFSK) with a BT value of 0.5. A “1” is represented by a positive frequency shift and a “0” is represented by a negative frequency shift. The modulation index must be in the range of 0.28 – 0.35, which corresponds to a frequency deviation of 140 ~ 175 kHz.

The Bluetooth channel is organized into time slots of 625 us in duration. The start of each packet must be aligned with the beginning of a new time slot, and a packet may extend for up to five time slots in duration. The master always starts a transmission in even-numbered time slots, while the slave always starts its transmissions in odd-numbered time slots. Each packet is transmitted on a different hop frequency.

For more details on the Bluetooth specification, please visit the official Bluetooth web site at:

www.bluetooth.com

7.1.4 802.11 Overview

The IEEE 802.11 standard was developed to provide Local Area Network (LAN) services in a wireless communications environment. The standard supports operation in the 2.4 GHz ISM band using either a Frequency Hopped Spread Spectrum (FHSS) approach or Direct Sequence Spread Spectrum (DSSS).

The FHSS format uses binary or 4-ary GFSK modulation (BT= 0.5) and supports data rates of 1 Mbps and 2 Mbps respectively. In the US, the set of operating transmit and receive channels for the FHSS specification includes 79 frequencies, with center frequencies ranging from 2402 MHz to 2480 MHz. Channels are spaced at 1 MHz intervals. In the US, the FHSS hopping patterns are grouped into 3 sets, each including 26 patterns.

The DSSS format also provides 1 and 2 Mbps data rates, but employs differential PSK baseband modulation formats. The 1 Mbps mode uses Differential Binary Phase Shift Keying (DBPSK), and the 2 Mbps mode uses Differential Quadrature Phase Shift Keying (DQPSK). The modulated baseband signals (both 1 Msps) are then spread using a chip rate of 11 MHz (i.e. there are 11 chips per symbol). The symbols are spread using an 11-chip Barker sequence, whose start time is aligned with the start of each symbol. In the US, 11 operating channels are available for the 802.11 DSSS specification, with center frequencies ranging from 2412 to 2462 MHz.

More details on the 802.11 standard may be obtained directly from the IEEE. The 802.11 specification is available for free download from the following URL:

http://standards.ieee.org/getieee802/

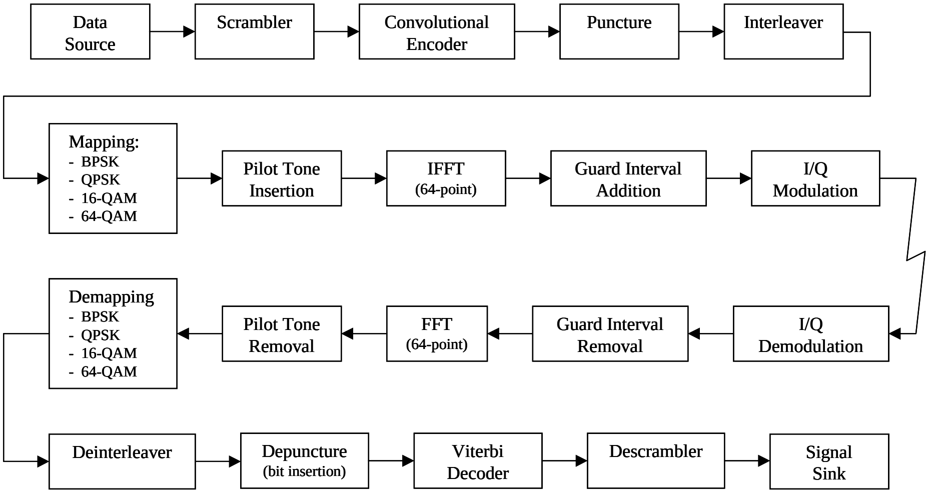
7.1.5 802.11a Overview

The IEEE 802.11a standard is an extension to the original 802.11 standard devised to significantly increase the system data rate. Unlike the 802.11b extension, 802.11a is not compatible with the original 802.11 frequency plan (2.4 GHz band), and operates instead in the 5 ~ 6 GHz band.

The 802.11a standard uses Orthogonal Frequency Division Multiplexing (OFDM) modulation and supports data rates ranging from 6 Mbps to 54 Mbps. The OFDM modulation uses 52 subcarriers that are modulated using BPSK, QPSK, 16-QAM, or 64-QAM. Convolutional coding is used for Forward Error Correction (FEC), at coding rates of 1/2, 2/3, or 3/4.

In the US, the set of operating frequencies for the OFDM specification includes 12 channels, with center frequencies ranging from 5180 MHz to 5805 MHz.

A simplified block diagram of an 802.11a physical layer link is shown in the following figure.



Simplified 802.11a Physical Layer Block Diagram

More details on the 802.11a specification may be obtained directly from the IEEE. The 802.11a specification is available for free download from the following URL:

http://standards.ieee.org/getieee802/

7.1.6 802.11b Overview

The IEEE 802.11b standard (also known as “Wi-Fi”) provides a high speed physical layer extension to the original 802.11 DSSS specification. It operates in the 2.4 GHz ISM band, and remains compatible with the original 802.11 frequency plan (see above).

The 802.11b standard extends the capabilities of 802.11 by adding Complementary Code Keying (CCK) modulation at rates of 5.5 and 11 Mbps. As with 802.11, a chipping rate of 11 MHz is used. For compatibility purposes, the packet’s preamble and headers are still transmitted using DBPSK and DQPSK at 1 Mbps and 2 Mbps.

An optional Packet Binary Convolutional Coding (PBCC) mode, providing a data rate of 5.5 Mbps or 11 Mbps, is also specified in the standard. The PBCC mode is not yet included in the WLAN module, but is planned for future releases.

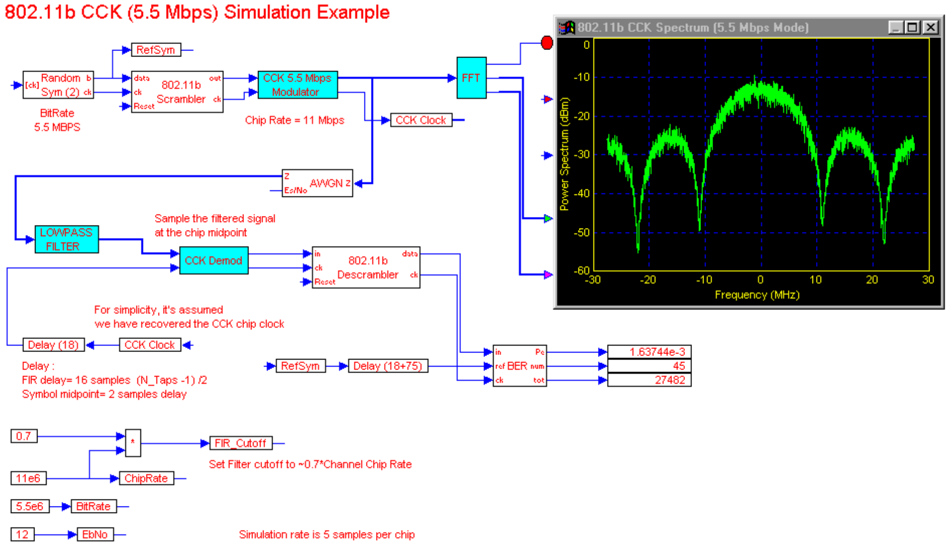
The 802.11b standard also includes an optional Channel Agility mode, in which the signal can frequency-hop across a subset of the available DSSS channels.

More details on the 802.11b specification may be obtained directly from the IEEE. The 802.11b specification is available for free download from the following URL:

http://standards.ieee.org/getieee802/

7.1.7 Sample wireless simulation

An example of a communication system simulation using 802.11 components is shown below. The example diagram “80211b_CCK.vsm” is located in the “Wireless” folder. The actual example file may differ slightly from the diagram that follows.


802.11b CCK (5.5 Mbps) Simulation Example
RefSym
[ck] Random Sym (2) b ck
data ck Reset 802.11b Scrambler out ck
CCK 5.5 Mbps Modulator
FFT
BitRate 5.5 MBPS
Chip Rate = 11 Mbps
CCK Clock
z Es/No AWGN z
Sample the filtered signal at the chip midpoint
LOWPASS FILTER
CCK Demod
in ck Reset 802.11b Descrambler data ck
For simplicity, it's assumed we have recovered the CCK chip clock
Delay (18)
CCK Clock
Delay : FIR delay= 16 samples (N_Taps -1) /2 Symbol midpoint= 2 samples delay
RefSym
Delay (18+75)
in ref ck BER Pe num tot
1.63744e-3
45
27482
0.7
*
FIR_Cutoff
Set Filter cutoff to ~0.7*Channel Chip Rate
11e6
ChipRate
5.5e6
BitRate
12
EbNo
Simulation rate is 5 samples per chip
802.11b CCK Spectrum (5.5 Mbps Mode)
Power Spectrum (dBm)
0
-10
-20
-30
-40
-50
-60
-30
-20
-10
0
10
20
30
Frequency (MHz)

This example focuses on the 802.11b 5.5 Mbps CCK modulation format. The diagram displays the modulated CCK spectrum and computes a Bit Error Rate (BER) operating point assuming a Gaussian noise channel.

CCK modulation, as defined in the IEEE 802.11b specification, is used to provide a high rate mode (operating at either 5.5 Mbps or 11 Mbps) that is compatible with the pre-existing lower rate data modes (1 and 2 Mbps) of the 802.11 standard.

7.2 WLAN Block Set

7.2.1 BLUETOOTH BLOCKS

7.2.1.1 Bluetooth Hop Generator

This block generates a Bluetooth frequency-hopping pattern and a Master Clock output. Different hop patterns are generated for the Master and Slave, and a toggle output is provided to indicate which time slot (master or slave) is active. The Bluetooth Master Clock has a cycle length of 2^{28} (28 bit shift register), which corresponds to 2^{27} time slots as hops occur only on every other clock pulse. Master and Slave take turns transmitting, with the Master using even hop slot numbers and the Slave odd ones. The default time slot duration is 625 µs.

This block supports both the 79 and 23 hop specifications. The hop sequence is derived from the current Bluetooth Master Clock value and the Device Address, as described in Section 11 of the Bluetooth Specification. The default Bluetooth hop rate is 1.6 kHz, which corresponds to a Master clock pulse rate of 3.2 kHz (hops occur on every other clock pulse).

This block support both internal and external timing. When in external mode, the user must provide both the Master Clock counter value and pulse train. This block outputs a hop pattern consisting of channel numbers in the range of [0, 78] or [0, 22] depending on the Hop Mode selection. This block should be followed by a Frequency Hop block to implement the actual signal hopping.

x_1 = Optional Master Clock counter value

x_2 = Optional Master Clock pulse train

y_1 = Master Hop Channel # [0, 78] or [0, 22]

y_2 = Slave Hop Channel # [0, 78] or [0, 22]

y_3 = Master/Slave Slot Flag (0= Master; 1= Slave)

y_4 = Master Clock counter value

y_5 = Master Clock pulse train

Device Address

Specifies the device address in hex. This value is used to select the hop sequence generated by the block.

Timing Mode

Internal

Indicates internal clock timing. The hop rate, start time, and Master Clock initial value must be specified.

External

Indicates external timing. An external clock and Master Clock counter value must be provided to the block.

Hop Mode

79 Hops

The block generates a hop sequence using 79 possible channels. This mode is used in the US and most of Europe.

23 Hops

The block generates a hop sequence using 23 possible channels. This mode is use in Japan, Spain and France.

Init Master Clock

Specifies the initial counter setting for Bluetooth Master Clock. This parameter is only available when in Internal Timing mode.

Hop Rate

Specifies the hop rate for the block in hops per second. Note that the output clock rate will be twice this rate. This parameter is only available when in Internal Timing mode. The default hop rate is 1.6 kHz.

Start Time

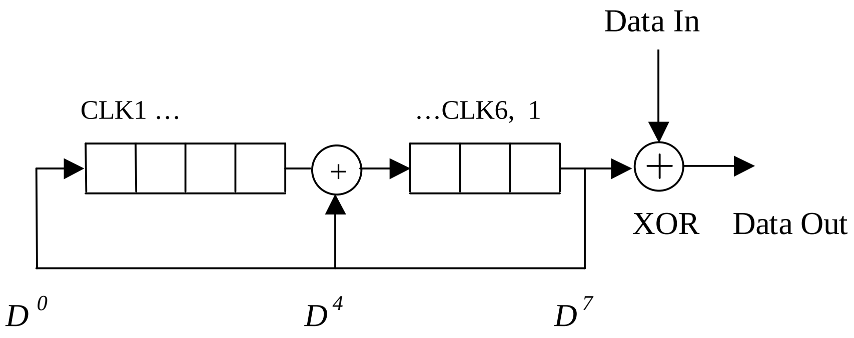
Specifies a starting time for the hop sequence and clock output. This parameter is only available when in Internal Timing mode.

7.2.1.2 Bluetooth Scrambler

This block provides a data scrambling and descrambling function as specified by the Bluetooth standard. The input data is XOR'ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized for each transmission (when a pulse is presented at the frame clock input) by using bits 1-6 of the current Bluetooth Master Clock and extending them with an MSB value of 1.

The generator polynomial for the feedback shift register is:

$$p(D) = D^7 + D^4 + 1$$



The user must provide an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired. The use must also supply the Bluetooth Master Clock value (essentially the output of a binary counter), which can be obtained from a Bluetooth Hop Generator block.

The Bluetooth Scrambler block is used for both data scrambling and unscrambling.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Frame clock pulse

x_4 = Bluetooth Master Clock value

y_1 = Output data

y_2 = Output data clock pulses

This block does not have any internal parameters.

7.2.1.3 GFSK Modulator

This block implements a Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK modulation, the digital information is transmitted by shifting the carrier frequency between two states. A “1” is represented by a positive frequency shift and a “0” is represented by a negative frequency shift.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the Bluetooth specification for the 1 Mbps mode, which specifies a BT value of 0.5 and a frequency deviation range of +/- 140 ~ 175 kHz.

x = Input data signal (binary [0, 1])

y = Complex output signal [Re, Im]

7.2.1.4 Shortened Hamming Decoder

This block implements a decoder for the shortened (15, 10) Hamming code as defined in the Bluetooth standard. The encoder matrix for this code is shown below. This code achieves a code rate of 2/3 and has the ability of correcting one bit error in a code word. The code is systematic and the parity-check matrix is shown below.

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

The decoding process includes computing the syndrome of the input vector and the parity-check matrix, locating any errors and correcting the error when possible. Since the code is systematic, the information bits reside in the first 10 bits of the codeword and the parity bits occupy the last 5 bits.

This block does not include any parameters. It accepts as input a coded vector of size 15 and outputs a decoded vector of size 10 elements.

x_1 = Input frame clock pulse

x_2 = Input coded vector (size 15)

y_1 = Output frame clock pulse

y_2 = Decoded output vector (size 10)

This block does not have any internal parameters.

7.2.1.5 Shortened Hamming Encoder

This block implements an encoder for the shortened (15, 10) Hamming code as defined in the Bluetooth standard. The encoder matrix for this code is shown below. This code achieves a code rate of 2/3 and has the ability of correcting one bit error in a code word. The code is systematic and is described by the parity-check matrix *H* shown below.

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

The encoding process involves computing the parity bits for the given input word and appending these to the input to create the coded output word. Since the code is systematic, the information bits are preserved in the output codeword and reside in its first 10 bits. The last 5 bits represent the parity bits.

This block does not include any parameters. It accepts as input a data vector of size 10 and outputs an encoded vector of size 15 elements.

x_1 = Input frame clock pulse

x_2 = Input data vector (size 10)

y_1 = Output frame clock pulse

y_2 = Encoded output vector (size 15)

This block does not have any internal parameters.

7.2.2 802.11 BLOCKS

7.2.2.1 Barker Sequence

This block generates a repeating Barker sequence. Barker codes are a family of pseudo-random sequences with quasi-ideal cross-correlation properties. Barker codes are often used in CDMA systems and are also part of the 802.11 2.4 GHz DSSS specification. The Barker code used in the 802.11 specification uses N= 11.

The user can select the length of the Barker code, as well as the symbol rate and an initial delay. The following illustrates the output pattern for each available length:

$N = 3$	1 1 0
$N = 4$	1 1 0 1
$N = 5$	1 1 1 0 1
$N = 7$	1 1 1 0 0 1 0
$N = 11$	1 0 1 1 0 1 1 1 0 0 0
$N = 13$	1 1 1 1 1 0 0 1 1 0 1 0 1

This block can accept an external clock or operate from an internal source. A clock value greater than 0.5 is considered high.

x_1 = Optional external clock

y_1 = Barker code sequence

y_2 = Output clock pulses

Sequence Length

Specifies the length *N* of Barker code. Available lengths include 3, 4, 5, 7, 11 and 13. The default value for the 802.11 specification is 11.

Sequence Offset

Specifies a starting offset for the Barker sequence. Valid range is 0 to *N* -1.

Output Mode

Bilevel

The signal amplitudes associated with the output sequence are {-1, 1}.

Binary

The signal amplitudes associated with the output sequence are {0, 1}.

Timing

Internal

Indicates internal clock timing. The bit rate and start time need to be specified.

External

Indicates external timing. An external clock must be provided at the *x*1 input.

Bit Rate

Specifies the Barker sequence bit rate in bits per second. This parameter is only available when in Internal Timing mode.

Start Time

Specifies a start time, in seconds, for the Barker sequence. This parameter is only available when in Internal Timing mode.

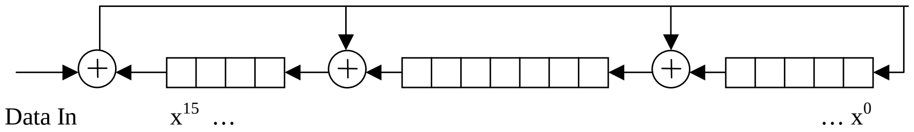
7.2.2.2 CRC-16 Generator

This block computes the CCITT-16 CRC (Cyclic Redundancy Code) based on an input binary serial data stream. At each input clock pulse, the block will update the CRC output vector, which is comprised of 16 elements (x^{15}, x^{14}, … x^{1}, x^{0}). The output can be provided as is, or flipped (one’s complement mode).

A CRC is commonly used to verify that a data sequence is received error free by including it’s “state” at the end of a data packet. If the CRC computed at the receiver (based on the received data) matches the CRC sent by the transmitter, then it’s highly likely that the block was received correctly.

The CRC shift register can be reset to its starting value (all 1's) by providing a pulse on the Reset input. The first element of the output vector is the MSB of the CRC (x^{15} cell). The generator polynomial and internal shift register structure are shown below.

$$p(x) = x^{16} + x^{12} + x^{5} + 1$$



x_1 = Binary data [0, 1]

x_2 = Data clock (pulse train)

x_3 = Reset (pulse)

y_1 = CRC output vector (size 16)

Output Mode

Regular

The block outputs the internal shift register state as is.

One's Complement

The block outputs the one's complement of internal shift register state (all 0's are flipped to 1's and all 1's are flipped to 0). This mode is used in the 802.11 specification.

7.2.2.3 802.11 Hop Generator

This block generates an 802.11 frequency-hopping pattern. This block supports both the 79 and 23 hop specifications, depending on the specified Operating Region. For specific details on the 802.11 hop sequences, please refer to Section 14.6.4 – 14.6.8 of the 802.11 specification.

This block outputs a hopping pattern represented by a hop index (for driving a Frequency Hop block) and a Channel number for information purposes. The hop index range is [0, 78] or [0, 22] depending on the Hop Mode selection. This block support both internal and external timing. When in external mode, the user must provide a pulse train indicating when the next hop slot is starting.

Note: the output Hop Index specifies the hop operating frequency using the standard 1 or 2 Mbps (Low Rate) channel assignments beginning with index value 0. For example, Channel

#37 (centered at 2437 MHz) will be output as Hop Index #35, since channel numbers start with Channel #2 (2402 MHz).

This block should be followed by a Frequency Hop block to implement the actual hopping (use the Hop Index output as the drive signal).

x_1 = Pattern selection input

x_2 = Optional external hop clock (pulse train)

y_1 = Hop Index # [0, 78] or [0, 22]

y_2 = Output hop clock pulse

y_3 = Hop Channel # [2, 80] or [2, 24]

Region

Specifies the 802.11 operating geographical region. Choices include North America/Canada, Europe, Japan, Spain and France.

Timing Mode

Internal

Indicates internal clock timing.

External

Indicates external timing. An external clock signal must be provided.

Initial Hop Index

Specifies the initial internal counter value for the 802.11 hopping pattern.

Hop Rate

Specifies the hop rate for the block in hops per second. This parameter is only available when in Internal Timing mode. The default hop rate is TBD kHz.

Start Time

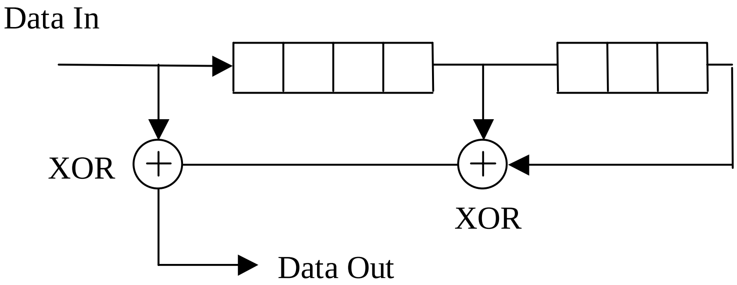
Specifies a starting time for the hop sequence. This parameter is only available when in Internal Timing mode.

7.2.2.4 802.11 Descrambler

This block provides a data descrambling function as specified in the 802.11 and 802.11b standards. The input data is XOR'ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized whenever an external pulse is presented at the frame clock input. Depending on the operating mode – either long or short PLCP preamble – the shift register is initialized with either hex 0x6C (1101100) or 0x1B (0011011) respectively, with the LSB corresponding to the rightmost shift register cell. The feed-through implementation on the scrambler and descrambler is self-synchronizing, which does not require any knowledge of the initial Tx shift register state for receive processing.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



The user must supply an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Reset clock pulse

y_1 = Output data

y_2 = Output data clock pulses

Register Init Value

Specifies the initial value of the shift register in hex format. Since the descrambler is self-synchronizing, providing an initial value simply provides a means of achieving an immediate synchronization with the transmitter, instead of having to wait for seven clock cycles.

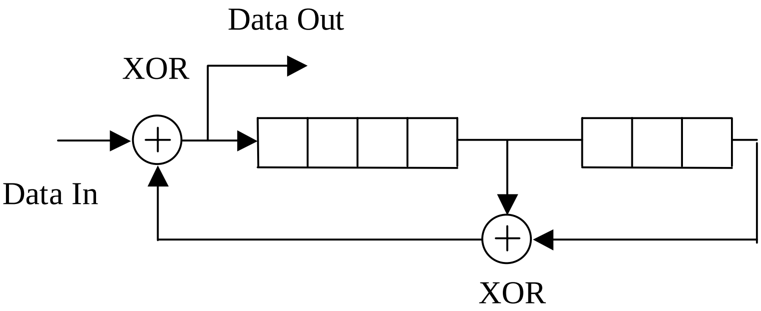
7.2.2.5 802.11 Scrambler

This block provides a data scrambling function as specified in the 802.11 and 802.11b standards. The input data is XOR’ed with the output sequence from a feedback shift register of size $N = 7$. The shift register is re-initialized whenever an external pulse is presented at the frame clock input. Depending on the operating mode – either long or short PLCP preamble –

the shift register is initialized with either hex 0x6C (1101100) or 0x1B (0011011) respectively, with the LSB corresponding to the rightmost shift register cell.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



The user must supply an input bit stream and clock (pulse train) and periodically a frame clock pulse to re-initialize the internal shift register when desired.

x_1 = Input data

x_2 = Input data clock pulses

x_3 = Frame clock pulse

y_1 = Output data

y_2 = Output data clock pulses

Register Initialization

Long Preamble

The register is initialized to hex 0x6C as specified in the 802.11 specification for use with the long preamble.

Short Preamble

The register is initialized to hex 0x1B as specified in the 802.11 specification for use with the short preamble.

7.2.2.6 GFSK-2 Modulator

This block implements a two level Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK-2 modulation, the digital information is transmitted by shifting the carrier frequency between two states. The 802.11 GFSK-2 mode specifies the following frequency deviations:

Input Symbol	*Carrier Deviation*
1	1/2 x h2 x *R* = 160 kHz (for h2 = 0.32 and *R* = 1 Msps)
0	-1/2 x h2 x *R* = -160 kHz

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the 802.11 specification for the 1 Mbps mode, which specifies a BT value of 0.5 and a nominal frequency deviation +/- 160 kHz.

x = Input data signal (binary [0, 1])

y = Complex output signal [Re, Im]

7.2.2.7 GFSK-4 Modulator

This block implements a four level Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK-4 modulation, the digital information is transmitted by shifting the carrier frequency between four states. The 802.11 GFSK-4 mode specifies the following frequency deviation values depending on the input data symbol (first received bit is the LSB):

Input Symbol	*Carrier Deviation*
1 0	3/2 x h4 x *R* = 216 kHz (for h4= 0.144 and *R* = 1 Msps)
1 1	1/2 x h4 x *R* = 72 kHz
0 1	-1/2 x h4 x *R* = -72 kHz
0 0	-3/2 x h4 x *R* = -216 kHz

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of these blocks may need to be adjusted for proper operation, depending on the chosen data rate and simulation rate. The default parameters for the BT product, symbol rate, and FM deviation value reflect the 802.11 specification for the 2 Mbps mode, which specifies a BT value of 0.5 and the frequency deviations shown above.

x = Input data signal (symbol value [0, 1, 2, 3])

y = Complex output signal [Re, Im]

7.2.3 802.11a BLOCKS

7.2.3.1 802.11a Convolutional Encoder

This implements an 802.11a rate ½ convolutional encoder. The block's settings allow specification of the generator coefficients and constraint length, but the default values correspond to those of the 802.11a specification (k=7, G1= 133, G2= 171 octal). All other code rates other than rate ½ are achieved by puncturing the output of this block (see Puncture block).

At each input clock pulse, the blocks internal shift register is shifted to the right and a new input data bit is read. The convolutional encoder's A and B outputs are then updated.

x_1 = Input data bits [0, 1]

x_2 = Input clock (pulse train)

y_1 = A output

y_2 = B output

y_3 = Output clock pulse

Generator #1

Specifies the value of the first generator coefficient in octal format. The default is 133, which corresponds to "1011011" in binary.

Generator #2

Specifies the value of the first generator coefficient in octal format. The default is 171, which corresponds to "1111001" in binary.

Constraint Length

Specifies the length of the internal shift register in bits.

7.2.3.2 802.11a Depuncture

This block converts a received serial bit stream into pairs (A B) of data suitable for use by the 802.11a Viterbi Decoder block. For code rates other than rate ½, this block also provides a depuncture function, which inserts dummy bits (erasures) in the locations where bits were "stolen" at the encoder side.

Knowledge of the output symbol rate (A and B pairs) is required to so that the block can properly output the depunctured bit pairs. The output data stream is always re-synched with the input serial clock at the start of each new pattern period (every 2 input clock pulses for a rate 1/2 code; every 3 input clock pulses for rate 2/3; and every 4 input clock pulses for rate 3/4). For further details on code puncturing see the description of the 802.11a Puncture block.

x_1 = Input bits

x_2 = Input clock (pulse train)

y_1 = A output

y_2 = B output

y_3 = Output clock (pulse train)

Code Rate

Specifies the desired code rate as Rate 1/2, Rate 2/3 or Rate 3/4.

Output Symbol Rate

Specifies the output symbol rate (A B pairs) so that the block can properly output the depunctured bit pairs. This value should match the symbol rate used at the encoder block.

Erasure Value

Specifies the value to be used for all dummy bits inserted. This value is typically set to 0 as the soft decision Viterbi block that follows is usually set to operate with bi-level data [-1, +1].

7.2.3.3 802.11a Puncture

This block is used to convert the output of the 802.11a Convolutional Encoder into a serial bit stream and also implement puncturing when the selected code rate is not rate ½. Puncturing is implemented by “stealing” specific coded bits from the encoder output (i.e. not transmitting such bits). At the receiver, these stolen bit slots are filled with dummy bit values.

Knowledge of the input symbol rate (clock rate) is required to so that the block can derive the appropriate output serial timing. This block does not require that the output bit interval be comprised of an integral number of simulation samples. As a result, the duration of individual output bits can differ slightly from bit to bit within the underlying “pattern period”, which is defined as the periodic time interval (in input clock cycles) required to implement each specific puncture pattern. The output stream is always re-synched with the input clock at the start of each new pattern period.

At each input clock new A and B value are read, and represented herein by increasing subscripts (i.e. A_0 B_0, A_1 B_1, A_2 B_2). The following describes the puncture pattern for each code rate (i.e. which bits are output).

Rate 1/2:

Pattern Period= 1 clock cycle Output pattern: A_0 B_0

Rate 2/3:

Pattern Period= 2 clock cycles Output pattern: A_0 B_0 A_1

Rate 3/4:

Pattern Period= 3 clock cycles   Output pattern: A_0 B_0 A_1 B_2

x_1 = Input A value

x_2 = Input B value

x_3 = Input clock pulse

y_1 = Serial punctured output

y_2 = Output clock (pulse train)

Code Rate

Specifies the desired code rate as Rate 1/2, Rate 2/3 or Rate 3/4.

Input Symbol Rate

Specifies the approximate input symbol rate (A B pairs) so that the block can properly compute the output serial timing for the selected code rate.

7.2.3.4 802.11a Interleaver

This implements an 802.11a Interleaver or Deinterleaver. The block size of the interleaver is set to correspond to the number of coded bits in a single OFDM frame, depending on the intended rate of the link per Section 17.3.5.6 of the 802.11a specification. This block is normally inserted after the convolutional encoding process.

This block basically implements a “block type” interleaver, and introduces a delay equal to the block size (i.e. number of coded bits in the OFDM frame). The block size is set as follows depending on the selected Rate Mode:

6 Mbps, 9 Mbps:	48 bits
12 Mbps, 18 Mbps:	96 bits
24 Mbps, 36 Mbps:	192 bits
48 Mbps, 54 Mbps:	288 bits

x_1 = Input data

x_2 = Input clock (pulse train)

y_1 = Output data

y_2 = Output clock (pulse train)

802.11a Rate Mode

Specifies the data rate of the 802.11a link being simulated (not necessarily the simulation rate).

Mode

Interleave

The block operates as an interleaver.

Deinterleave

The block operates as a deinterleaver.

7.2.3.5 802.11a Scrambler

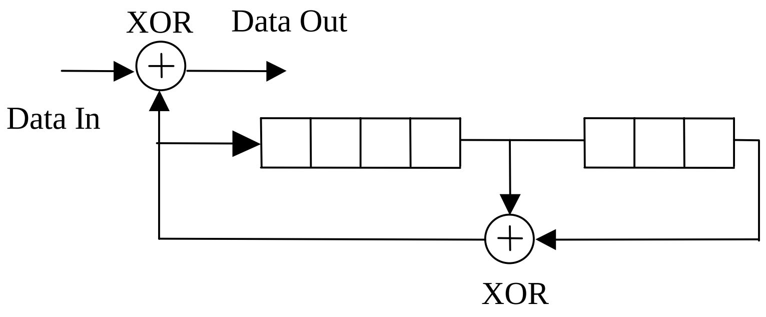
This block provides a data scrambling/descrambling function as specified in the 802.11a standard. The input bits are XOR'ed with the output sequence of a feedback shift register of size $N = 7$, which produces a repeating pattern of length 127.

The internal shift register is re-initialized whenever an external pulse is presented at the reset input. Depending on the operating mode – either internal or external initialization – the shift register is then initialized with either a fixed hex value or an externally provided value, where the LSB corresponds to the rightmost shift register cell in the diagram below. The initialization value is typically set to a pseudo-random non-zero value.

Note: when this block is used to scramble the 802.11a OFDM pilot tones, the shift register is typically initialized to all 1's.

The generator polynomial for the feedback shift register is:

$$p(x) = x^7 + x^4 + 1$$



x_1 = Input bits

x_2 = Input clock (pulse train)

x_3 = Register initialization value (external mode only)

x_4 = Reset input (pulse)

y_1 = Output data

y_2 = Output clock (pulse train)

Register Initialization

External

The register is initialized to the current value presented at the external “Init” input.

Internal

The register is initialized to the hex value specified in the Register Init Value parameter field.

Register Init Value

Specifies the reset value for the internal shift register when in Internal register initialization mode. This value is specified in hex format.

7.2.3.6   802.11a Viterbi Decoder

This block implements a rate ½ soft decision Viterbi Decoder compatible with the 802.11a specification. This block is used to decode a convolutionally encoded bit stream. Parameters include the encoder constraint length (*k*), the trellis truncation length *M*, the number of quantization bits, and the generator coefficients. An external Metric file must also be specified.

This block accepts soft A and B ouputs from a 802.11a Depuncture block, which should be in bilevel format (i.e. +/- 1 without any noise).

x_1 = A Input ~[-1, +1]

x_2 = B Input ~[-1, +1]

x_3 = Input clock (pulse train)

y_1 = Decoded bit stream (0, 1)

y_2 = Ouput clock (pulse train)

y_3 = Best path metric

Generator #1

Specifies the value of the first generator coefficient in octal format. The default is 133, which corresponds to “1011011” in binary.

Generator #2

Specifies the value of the first generator coefficient in octal format. The default is 171, which corresponds to “1111001” in binary.

Constraint Length

Specifies the constraint length *k* of the associated encoder.

Trellis Truncation Length

Specifies the trellis truncation length *M*. The maximum allowed value of *M* is 96.

Quantization Bits

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

Select File

Opens the Select File dialog box for selecting the desired Metric file.

Browse File

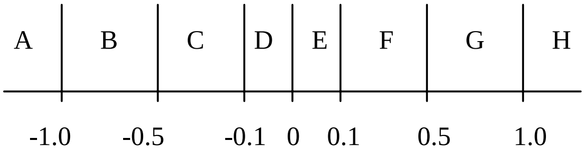
Opens the selected Metric file using Notepad.

Metric File Path

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

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

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



Example Metric File

Viterbi Decoder metric file for use with erasures (0), Quant bits = 3

3,	0.0,	7.0
-1.0,	1.0,	6.0
-0.5,	2.0,	5.0
-0.1,	3.0,	3.0
0,	3.0,	3.0
0.1,	5.0,	2.0
0.5,	6.0,	1.0
1.0,	7.0,	0.0

For example:
for an input $0.5 \le x < 1.0$,
the metric for a “0” bit is “6.0”,
and the metric for a “1” bit is “1.0”.

Note: In the above example, values in the range of [-0.1, 0.1], which includes the erasure value (dummy bit) 0, are given an equal metric.

7.2.3.7 OFDM Demodulator

This block demodulates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal back to a group of data and pilot subcarriers, as specified in the IEEE 802.11a standard. For more details on OFDM, please refer to the OFDM Modulator block description.

The OFDM signal is demodulated by applying a forward FFT to the received I and Q complex baseband inputs. An external trigger pulse is used to signal the beginning of each received frame to the block.

Since a Guard Interval (GI) may be used in the transmission of the OFDM frame, this block allows the user to specify an FFT wraparound value to be applied to the received vector prior to performing the FFT. This feature is useful when the active samples used by the OFDM Demodulator include samples from the GI (instead of just FFT segment samples). Since the GI is generated from a cyclic shift of the FFT segment, this feature allows the OFDM Demodulator block to recover the proper IFFT samples.

The OFDM Demodulator block accepts a baseband OFDM modulated signal in I and Q component format. Other inputs include a sampling clock (pulse train) and a trigger used to signal the beginning of each OFDM frame. Block outputs include a frame clock and two data vectors representing the Real and Imaginary components of the recovered “subcarrier”data. This block can be followed by an OFDM Pilot Extract block to separate-out the data and pilot subcarriers.

x_1 = Beginning of frame trigger

x_2 = I baseband input

x_3 = Q baseband input

x_4 = Input sampling clock (pulses)

y_1 = Frame clock pulse

y_2 = Real output subcarrier vector (size 2^N)

y_3 = Imaginary output subcarrier vector (size 2^N)

FFT Size

Specifies the size of the forward FFT (power of two) to be performed on the received data. Valid range is from 16 to 64K. This value should match the IFFT size used by the transmitter.

FFT Wraparound Offset

Specifies a cyclic offset to be applied to the received data prior to performing the FFT operation. This parameter is useful when samples from the GI are included in the received frame.

7.2.3.8 OFDM Modulator

This block generates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a specification.

In OFDM modulation, the output is obtained by applying an Inverse FFT (typically a power of two) to an input vector of complex IQ points corresponding to data and pilot “subcarriers”. To reduce ISI, the inverse FFT output is often padded with an optional guard interval representing a cyclic shift of the inverse FFT output itself. The resulting OFDM “symbol” is then treated as a baseband time domain signal and used for transmission.

This block accepts two input vectors representing the Real and Imaginary components of the subcarrier data. An OFDM Pilot Map block can be used to merge the data subcarriers with any pilot or null subcarriers. The input vectors are read when a frame clock pulse is detected. Additional inputs include a guard interval selector, an extended FFT period flag, and an optional external clock for the output data. Block outputs include time domain baseband I and Q outputs representing the OFDM symbol, and an output sample clock.

The external Guard Interval (GI) selection input allows the user to vary the guard interval size during the simulation. The following describes the GI corresponding to each allowed input value:

0 –> No GI
1 –>GI = FFT Size / 2
2 –> GI = FFT Size / 4
3 –> GI = FFT Size / 8
4 –> GI = FFT Size / 16
5 –> GI = FFT Size / 32

The size of the output frame is equal to GI + FFT size, unless the extended FFT input is set high. The optional extended FFT period connector is used to signal the OFDM Modulator to output the FFT segment twice (i.e. the output equals GI + FFT + FFT).

The OFDM Modulator also supports windowing between successive OFDM frames to reduce spectral sidelobes. The user can specify an overlap ranging from 0 to 3 samples. The window function follows a $\sin^2$ rolloff profile as specified in the 802.11a specification.

In this Commsim implementation a windowed overlap is generated by extending the GI at the beginning of the frame by a few samples, and carrying over the last few samples of the FFT section to the next output frame. For example, if an overlap of 2 samples is used, the GI length is increased by two samples (to which a rolloff is applied) and overlapped with the last two samples (also weighted by a rolloff) from the previous frame's FFT section. This implementation reduces simulation delay and internal buffering needs.

x_1 = Input Frame clock pulse

x_2 = Real input vector (size 2^N)

x_3 = Imaginary input vector (size 2^N)

x_4 = Guard Interval selector {0, 1, 2, 3, 4, 5}

x_5 = Extended FFT period flag (High = extend FFT period)

x_6 = Optional external sample clock

y_1 = I signal output

y_2 = Q signal output

y_3 = Output clock pulses

Timing

Internal

Indicates internal clock timing per the specified output rate.

External

Indicates external timing. An external clock must be provided at the *x*6 input.

Use Simulation Sample Rate

When selected, this mode forces the OFDM Modulator block to use the current simulation sample rate as its output rate.

Output Rate

Specifies the output rate for the OFDM output samples. This parameter is only available when in Internal Timing mode. The default value is set to the current simulation rate.

FFT Size

Specifies the size of the inverse FFT (power of two) to be performed. Valid range is from 16 to 64K.

Overlap Size

Specifies an overlap between successive OFDM output frames. Specifying an overlap results in a windowing operation and provides a smoother transition from frame to frame. Valid range is from 0 to 3 samples.

7.2.3.9 OFDM Pilot Extract

This block decomposes the Re/Im output of an OFDM Demodulator into two groups of subcarrier vectors (Data and Pilot) according to a user defined mapping file.

The subcarrier mapping file should match the one used by the OFDM transmitter, and serves to specify the ordering in the received vector of the data, pilot and any null subcarriers (which are discarded).

The user must specify the number of data and pilot subcarriers, as well as the input vector size, which must be a power of two (FFT size). The user can also specify the starting position within the input vector for element #0 in the map file as either 0 or N/2, where N is the FFT size. The Data and Pilot subcarrier values are written in sequential order to their respective outputs according to their order of appearance in the Map File.

For more details on the Map File format, please refer to the OFDM Pilot Map block description.

x_1 = Input Frame clock pulse

x_2 = Real output vector from OFDM Demod (size 2^N)

x_3 = Imaginary output vector from OFDM Demod (size 2^N)

y_1 = Frame clock pulse

y_2 = Data subcarrier I vector

y_3 = Data subcarrier Q vector

y_4 = Pilot subcarrier I vector

y_5 = Pilot subcarrier Q vector

Input Insertion Point

Begin at N/2 Location

Forces an N/2 circular shift in the input vector, where N is the size of the FFT used by the OFDM Demodulator. Element zero in the Map File is read from the N/2 input vector location.

Begin at 0 Location

Element zero in the Map File is read from the 0 input vector location.

Number of Data Subcarriers

Specifies the size of the Data subcarrier I and Q output vectors.

Number of Data Subcarriers

Specifies the size of the Pilot subcarrier I and Q output vectors.

Data Subcarrier Weight

Specifies the weight applied to the I and Q Data subcarriers in the modulator. The received subcarrier is scaled (divided) by this value.

Pilot Subcarrier Weight

Specifies the weight applied to the I and Q Pilot subcarriers in the modulator. The received subcarrier is scaled (divided) by this value.

FFT Size

Specifies the block's input vector size *N*. This value should match the size of the FFT used by the preceding OFDM Demodulator, as well as the Map File's number of entries. Valid range is from 16 to 64K.

Select File

Launches a dialog box for selecting the desired Subcarrier Map file.

Browse File

Opens the selected Subcarrier Map file using Notepad.

Subcarrier Mapping File

Specifies the path and filename for the Subcarrier Map file. The format of the generated file is shown below. The keyword identifier specifies the subcarrier type to be used for each element. Allowed keywords are: DATA, PILOT, and NULL. Valid entry separators are commas, tabs and blank space.

Header line (for storing user defined data)	
element #0	*keyword*
element #1	*keyword*
…	
element #N-1	*keyword*

7.2.3.10 OFDM Pilot Map

This block generates a composite Re/Im pair of “subcarrier” vectors for use by an OFDM Modulator or Vector OFDM Modulator block.

This block accepts two pairs of input vectors (each with Real and Imaginary components) representing data subcarriers and pilot subcarriers. A user defined data mapping file is used to specify the ordering in the output vector of the data, pilot and any null subcarriers.

The user must specify the number of data and pilot subcarriers, as well as the output vector size, which must be a power of two (FFT size). The user can also specify the starting position within the output vector for the input data as either 0 or N/2, where N is the FFT size. The Data and Pilot subcarrier values are read in sequential order from their respective input ports and placed in the output vector according to the output ordering specified by the Map File.

The format of the “Map File” is as follows. The number of entries in the file should match the intended OFDM FFT size. A one line header is assumed. Thereafter, the first entry of each row represents the output vector element #, and is followed by a keyword identifying whether that location is to contain a data subcarrier (DATA), a pilot subcarrier (PILOT), or a null subcarrier (NULL). Any additional characters on the line are ignored. If the “Begin at N/2” option is selected, then the output vector is cyclically shifted by N/2 before being output (i.e. the entry for the #0 element will be output at the N/2 vector location.

x_1 = Input Frame clock pulse

x_2 = Data subcarrier I vector

x_3 = Data subcarrier Q vector

x_4 = Pilot subcarrier I vector

x_5 = Pilot subcarrier Q vector

y_1 = Frame clock pulse

y_2 = Output vector (Real component) (size 2^N)

y_3 = Output vector (Imaginary component) (size 2^N)

Input Insertion Point

Begin at N/2 Location

Forces an N/2 circular shift in the output vector, where N is the size of the FFT used by the OFDM Modulator. Element zero from the Map File is output in the N/2 output vector location.

Begin at 0 Location

Element zero from the Map File is output in the 0 output vector location.

Number of Data Subcarriers

Specifies the size of the Data subcarrier I and Q input vectors.

Number of Data Subcarriers

Specifies the size of the Pilot subcarrier I and Q input vectors.

Data Subcarrier Weight

Specifies a weight to be applied to the I and Q Data subcarriers.

Pilot Subcarrier Weight

Specifies a weight to be applied to the I and Q Pilot subcarriers.

FFT Size

Specifies the block's output vector size *N*. This value should match the size of the FFT used by the OFDM Modulator that will follow, as well as the Map File's number of entries. Valid range is from 16 to 64K.

Select File

Launches a dialog box for selecting the desired Subcarrier Map file.

Browse File

Opens the selected Subcarrier Map file using Notepad.

Subcarrier Mapping File

Specifies the path and filename for the Subcarrier Map file. The format of the generated file is shown below. The keyword identifier specifies the subcarrier type to be used for each element. Allowed keywords are: DATA, PILOT, and NULL. Valid entry separators are commas, tabs and blank space.

Header line (for storing user defined data)	
element #0	*keyword*
element #1	*keyword*
…	
element #N-1	*keyword*

7.2.3.11 OFDM Vector Demodulator

This block demodulates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a specification. For more details on OFDM, please refer to the OFDM Modulator block description.

This block differs from the regular OFDM Demodulator block in that it uses vector inputs for the I and Q received signals, instead of scalars. The OFDM signal is demodulated by applying a forward FFT to the received I and Q complex baseband samples. An external frame clock pulse is used to signal each new OFDM symbol.

Since a Guard Interval (GI) may be used in the transmission of the OFDM symbol, this block allows the user to specify a limited FFT offset value to be applied to the received vector prior to performing the FFT. This feature is useful when the input vector used by the Vector OFDM Demodulator includes some GI samples (instead of just FFT segment samples). Since the GI is generated from a cyclic shift of the FFT segment, this feature allows the Vector OFDM Demodulator block to recover the proper IFFT samples.

The Vector OFDM Demodulator block accepts a baseband OFDM modulated signal in I and Q vector format and a frame (symbol) clock. Block outputs include a frame clock and two data vectors representing the Real and Imaginary components of the recovered “subcarrier”data. This block can be followed by an OFDM Pilot Extract block to separate-out the data and pilot subcarriers.

x_1 = Input frame clock

x_2 = I baseband vector (size 2^N)

x_3 = Q baseband vector (size 2^N)

y_1 = Frame clock pulse

y_2 = Real output data vector (size 2^N)

y_3 = Imaginary output data vector (size 2^N)

FFT Size

Specifies the size of the forward FFT (power of two) to be performed on the received data. Valid range is from 16 to 64K. This value should match the IFFT size used by the transmitter.

FFT Offset

Specifies a cyclic offset to be applied to the received data vector to performing the FFT operation. This parameter is useful when samples from the GI are included in the received frame. Valid range is 0 to 3.

7.2.3.12 OFDM Vector Modulator

This block generates a baseband Orthogonal Frequency Division Multiplexed (OFDM) modulated signal, as used in the IEEE 802.11a specification.

This block differs from the regular OFDM Modulator block in that it uses vector outputs for the I and Q signals, instead of scalars. This version of the block does not support varying the Guard Interval (GI) size during a simulation, nor does it support the extended FFT mode. For more details on OFDM, please refer to the OFDM Modulator block description.

This block accepts two input vectors representing the Real and Imaginary components of the subcarrier data. An OFDM Pilot Map block can be used to merge the data subcarriers with any pilot or null subcarriers. The input vectors are read when a frame clock pulse is detected. Block outputs include I and Q vector OFDM outputs, and an output frame clock. The size of the output vector is equal to GI + FFT size. The GI size can range from none to 1/32 of the FFT size.

The Vector OFDM Modulator supports windowing between successive OFDM frames to reduce spectral sidelobes. The user can specify an overlap ranging from 0 to 3 samples. The window function follows a $\sin^2$ rolloff profile as specified in the 802.11a specification.

In this Commsim implementation, a windowed overlap is generated by extending the GI at the beginning of the frame by a few samples, and carrying over the last few samples of the FFT section to the next output frame. For example, if an overlap of 2 samples is used, the GI length is increased by two samples (to which a rolloff is applied) and overlapped with the last two samples (also weighted by a rolloff) from the previous frame's FFT section. This implementation reduces simulation delay and internal buffering needs.

x_1 = Input Frame clock pulse

x_2 = Real input vector (size 2^N)

x_3 = Imaginary input vector (size 2^N)

y_1 = Output frame clock pulse

y_1 = I signal output

y_2 = I signal output vector (size GI + FFT)

y_3 = Q signal output vector (size GI + FFT)

FFT Size

Specifies the size of the inverse FFT (power of two) to be performed. Valid range is from 16 to 64K.

Overlap Size

Specifies an overlap between successive OFDM output frames. Specifying an overlap results in a windowing operation and provides a smoother transition from frame to frame. Valid range is from 0 to 3 samples.

Guard Interval Size

Specifies the size of the desired Guard Interval as a fraction of the IFFT size. Choices include: None, ½, ¼, 1/8, 1/16 and 1/32 of the IFFT size.

7.2.4 802.11b BLOCKS

7.2.4.1 802.11b High Rate Hop Generator

This block generates an 802.11b frequency-hopping pattern optionally used with the High Rate modes (5.5 and 11 Mbps). This block supports both the Set 1 and Set 2 channel selections for either North America or Europe (except France and Spain).

This block supports either internal or external hop timing. When in external mode, the user must provide an input pulse train indicating when the next hop slot is starting. This block outputs a hop pattern consisting of a hop index (for driving a Frequency Hop block) and either a High Rate or Low Rate channel number as appropriate for information purposes. A High/ Low flag is used to characterize the channel number as this can vary during the hop process. High Rate channels can range from 1 to 13, while Low Rate channels range from 2 to 80.

North America Set 1

Set 1 hop channels include High Rate Channels 1, 6 and 11.

For Set 1 only two hop patterns are specified:

Pattern#1: 1, 6, 11, 1, 6, 11…

Pattern#2: 1, 11, 6, 1, 11, 6…

North America Set 2

Set 2 hop channels include High Rate Channels 1, 3, 5, 7, 9 and 11.

For Set 2 the input pattern number AND the current Hop Index determine the next hop channel number according to section 18.4.6.7.3 of the 802.11b specification.

Europe Set 1 (except France and Spain)

Set 1 hop channels include High Rate Channels 1, 7 and 13.

For Set 1 only two hop patterns are specified:

Pattern#1: 1, 7, 13, 1, 7, 13…

Pattern#2: 1, 13, 7, 1, 13, 7…

Europe Set 2 (except France and Spain)

Set 2 hop channels include High Rate Channels 1, 3, 5, 7, 9, 11 and 13.

For Set 2 the input pattern number AND the current Hop Index determine the next hop channel number according to section 18.4.6.7.3 of the 802.11b specification.

This block should be followed by a Frequency Hop block to implement the actual hopping (use the Hop Index output as the drive signal).

Note: the output Hop Index specifies the hop operating frequency using the Low Rate (1, 2 Mbps) channel assignments beginning with index value 0. For example High Rate channel #6, centered at 2437 MHz and corresponding to Low Rate Channel #37, will be output as Hop Index #35. The offset value of two is due to the fact that the 802.11 channel numbers start with Channel #2 (2402 MHz).

x_1 = Pattern number selection input

x_2 = Optional external hop clock (pulse train)

y_1 = Hop Index # [0, 78]

y_2 = Output hop clock pulse

y_3 = Hop Channel # (see notes above)

y_4 = High/Low Rate flag

Region

Specifies the 802.11b operating geographical region. Choices include North America/ Canada, and Europe (except Spain and France).

Timing Mode

Internal

Indicates internal hop timing.

External

Indicates external timing. An external clock signal (pulse train) must be provided to indicate each hop.

Hop Set

Specifies which hop set to use; either Set 1 or Set 2.

Initial Hop Index

Specifies the initial internal counter value for the 802.11b hopping pattern.

Hop Rate

Specifies the hop rate for the block in hops per second. This parameter is only available when in Internal Timing mode. The default hop rate is TBD kHz.

Start Time

Specifies a starting time for the hop sequence. This parameter is only available when in Internal Timing mode.

7.2.4.2 CCK Demodulator

This block provides a symbol detection function for Complementary Code Keying (CCK) modulated signals. CCK modulation, as defined in the IEEE 802.11b specification, is used to provide a High Rate mode operating at either 5.5 Mbps or 11 Mbps, which is compatible with the pre-existing lower rate data modes (1 and 2 Mbps) of the 802.11 standard. The CCK chips are modulated using QPSK. For additional details on CCK modulation, see the description of the CCK Modulator.

This block accepts a complex baseband input and a chip clock, and produces a decoded bits vector of size 4 or 8 (depending on the CCK mode). The received sequence is compared internally with the CCK waveform set, and the one with maximum correlation is selected as the best estimation of the transmitted signal.

Each time eight new chips are received, a new output vector and a frame clock pulse are presented at the block’s output. An external reset capability for the internal chip counter is also provided for synchronization purposes.

x_1 = Complex baseband input (Re, Im)

x_2 = Chip clock (pulse train)

x_3 = Reference Phase {0, 1, 2, 3}

x_4 = Ref. Phase initialization strobe (pulse)

x_5 = Frame counter Reset

y_1 = Output frame clock pulse

y_2 = Output data bits vector (size 4 or 8)

y_3 = Internal “C” chip vector (size 8) {0, 1, 2, 3}

CCK Mode

(8, 4) CCK

Specifies (8, 4) CCK mode, as used in the IEEE 802.11b 5.5 Mbps rate.

(8, 8) CCK

Specifies (8, 8) CCK mode, as used in the IEEE 802.11b 11 Mbps rate.

7.2.4.3 CCK Modulator

This block generates a complex baseband modulated Complementary Code Keying (CCK) signal. CCK modulation, as defined in the IEEE 802.11b specification, is used to provide a High Rate mode operating at either 5.5 Mbps or 11 Mbps, and which is compatible with the pre-existing lower rate data modes (1 and 2 Mbps) of the 802.11 standard. CCK constructs orthogonality or semi-orthogonality from the input signal vector using an expanded Hadamard transform. The CCK chips are modulated using QPSK.

The block's function can be broken down into two main functions. The first is a mapping (encoding) of the input data vector (size 4 or 8 bits) into an eight element CCK chip vector (available as an external output). The second is the conversion of this "C vector" into a baseband time domain complex IQ signal.

For the 5.5 Mbps specification, 4 input data bits are mapped to 8 CCK chips. In this case, herein referred to as (8, 4) CCK modulation, the input data words are $\{d_0, d_1, d_2, d_3\}$, where d_0 comes first in time.

CCK modulation maps the data bits into phases $\{\varphi_0, \varphi_1, \varphi_2, \varphi_3\}$ by pairs. The first pair (d_0, d_1) controls the value of φ_0, which is mapped differentially according to the previous data words, with the phase shift defined in Table 1. The remaining phase terms are defined as

$\varphi_1 = d2 \times \pi + \frac{\pi}{2}$, $\varphi_2 = 0$, and $\varphi_3 = d3 \times \pi$.

Dibit pattern (d0, d1)	Even symbols Phase change (+jω)	Odd symbols Phase change (+jω)
00	0	π
01	π/2	3π/2
11	π	0
10	3π/2	π/2

Table 1. DQPSK encoding table

For the 11 Mbps specification, 8 input data bits are mapped to 8 CCK chips. In this case, herein referred to as (8, 8) CCK modulation, the input data words are $\{d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$, where d_0 comes first in time. The pair (d_0, d_1) is mapped differentially according to the previous data words as in the CCK (8, 4) case. The remaining pairs (d_i, d_{i+1}) are mapped to the remaining phase terms as defined in Table 2.

Bit pairs (d_i ,d_{i+1})	Phase
00	0
01	π/2
10	π
11	3π/2

Table 2. QPSK encoding table

The resulting four phase terms are then combined as follows to obtain the actual CCK modulated code word, which is comprised of eight “chips”. These 8 chips represent the output of the CCK Encoder block.

$$\begin{aligned}\vec{c} &= \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\} \\ &= \{e^{j(\varphi_1+\varphi_2+\varphi_3+\varphi_4)}, e^{j(\varphi_1+\varphi_3+\varphi_4)}, e^{j(\varphi_1+\varphi_2+\varphi_4)}, -e^{j(\varphi_1+\varphi_4)}, \\ &\quad e^{j(\varphi_1+\varphi_2+\varphi_3)}, e^{j(\varphi_1+\varphi_3)}, -e^{j(\varphi_1+\varphi_2)}, e^{j\varphi_1}\}\end{aligned}$$

This block accepts an input vector of either 4 or 8 data bits and outputs a QPSK modulated baseband signal. An auxiliary output is also provided to reveal the values of the eight CCK output chips (C vector), which can assume one of four possible phase states {0, 1, 2, 3}. If desired, the modulated IQ output signal can be translated to a specific carrier frequency by using an IQ Modulator block.

x_1 = Input frame clock (pulse train)

x_2 = Input bits data vector (size 4 or 8)

x_3 = Reference Phase {0, 1, 2, 3}

x_4 = Ref. Phase initialization strobe (pulse)

x_5 = External chip clock (pulse train) [optional]

y_1 = I signal output

y_2 = Q signal output

y_3 = Chip clock (pulse train)

y_4 = Internal "C" chip vector {0, 1, 2, 3}

CCK Mode

(8, 4) CCK

Specifies (8, 4) CCK mode, as used in the IEEE 802.11b 5.5 Mbps rate.

(8, 8) CCK

Specifies (8, 8) CCK mode, as used in the IEEE 802.11b 11 Mbps rate.

Timing

Internal

Indicates internal clock timing per the specified chip rate.

External

Indicates external timing. An external clock must be provided at the *x*5 input.

Chip Output Rate

Specifies the output chip rate in bits per second. This parameter is only available when in Internal Timing mode. The 802.11b default is 11 MHz.

7.2.5 GENERIC WIRELESS BLOCKS

7.2.5.1 Frequency Hop

This block imparts frequency hopping to a complex baseband input signal according to the external channel selection input. This block does not affect the amplitude of the input signal and merely provides a frequency translation operation.

x_1 = Input signal (complex)

x_2 = Hop channel selection input

y_1 = Frequency hopped output (complex)

$$f_{out} = x_1 \cdot (f_{\min} + x_2 \cdot \Delta f)$$

Number of Hops

Specifies the number *N* of available hop channels. This value is used internally for range checking purposes only.

Lowest Hop Channel

Specifies the carrier frequency corresponding to the lowest hop channel in *hertz*. This corresponds to input channel # 0.

Hop Carrier Spacing

Specifies the frequency spacing Δf between adjacent hop channels in *hertz*.

Initial Phase

Specifies the initial carrier phase in *degrees*. This option is only available when in Continuous Phase mode.

Phase Mode

Continuous

Indicates that the signal phase is continuous across channel hops. This simulates the use of a VCO or NCO in generating the output signal.

Discontinuous

Indicates that the signal phase is not continuous across channel hops. This simulates the use of multiple free running oscillators to generate each of the hopping channel carriers.

Chapter 8
Comm Block Set

The following are described in this chapter.

The Comm menu lists the communication blocks provided with Commsim. When you click on the Comm menu, most of the items that appear have a filled triangle (s) 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.

8.1 Channels category

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

8.1.1 AWGN (Complex or Real)

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

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

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

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

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

y_2 = Current run's E_s/N_o value

Number of Runs

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

Symbol Rate

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

Real Average Complex Signal Power
Real Average Signal Power

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

Ref. Power Units

dBm in 50 Ohms

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

Watts in 1 Ohm

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

Run *x* E_s/N_o

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

8.1.2 Binary Symmetric Channel

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

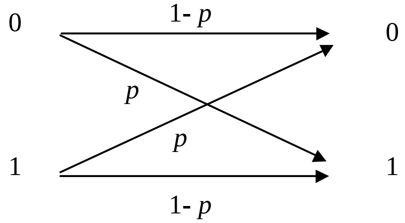
The `Binary Symmetric Channel` block accepts a real signal as input and outputs a real signal.

x_1 = Input signal (binary)

x_2 = Input data clock (impulse train)

y_1 = Output signal (0, 1)

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



Channel Error Prob.

Indicates the probability of a bit flip p. The range for this value is from 0.9999 to 3.1×10^{-8}.

8.1.3 Jakes Mobile

This block implements the Jakes fading channel model, which is commonly used in the mobile communications industry. This model approximates a Rayleigh fading process via summation of multiple complex sinusoids, as indicated below. Block parameters include the maximum Doppler frequency and the desired number of additional terms (N_0) to be used.

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

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

x = Complex input signal [Re, Im]

y = Complex output Signal [Re, Im]

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

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

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

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

Number of Terms

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

Doppler Frequency

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

8.1.4 Mobile Fading

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

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

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

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

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

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

where f_m is the doppler shift frequency

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

Number of Taps

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

Doppler Shift

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

8.1.5 Multipath

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

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

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

$$\begin{aligned} \text{where} \quad \tau_k &= \text{delay of path } k \\ \alpha_k &= \text{relative weight for path } k \\ \phi_k &= \text{phase rotation for path } k \\ N &= \text{number of paths} \end{aligned}$$

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.

8.1.6 Propagation Loss

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

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

x_1 = Input signal

x_2 = Path distance (in External mode)

y_1 = Attenuated output signal

y_2 = Attenuation in dB

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

Path Distance

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

Frequency

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

Distance Mode

External

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

Internal

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

Distance Units

kilometers

Indicates that the path distance is specified in kilometers.

miles

Indicates that the path distance is specified in miles.

8.1.7 Rice/Rayleigh Fading

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

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

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

x = Complex input signal [Re, Im]

y = Complex output signal [Re, Im]

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

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

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

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

Rice Factor (*r*)

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

RMS Doppler Spread (B_d)

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

RMS Fade Loss (L_{dB})

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

8.1.8 Rummler Multipath

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

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

x = Complex input signal [Re, Im]

y = Complex output Signal [Re, Im]

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

where τ = reflected path delay
α = overall path attenuation
β = reflected path relative gain
f_0 = fade null frequency

Reflected Path Delay

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

Shape Parameter

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

Fade Null Frequency

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

Overall Gain Term

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

8.1.9 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.

x_1 = Complex input signal [Re, Im]

x_2 = Reference average power level

y = Complex output signal [Re, Im]

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

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

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

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

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

Operating Point

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

Saturation Gain

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

Average Input Power

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

Average Power Mode

External
Internal

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

Alpha_a

Indicates the amplitude gain coefficient. See equation above.

Beta_a

Indicates the amplitude gain coefficient. See equation above.

Alpha_phi

Indicates the phase rotation coefficient. See equation above.

Beta_phi

Indicates the phase rotation coefficient. See equation above.

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

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

8.1.10 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 Commsim `map` block. The user must also provide an external reference power level indicating the tube's saturation level (not in dB).

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

x_1 = Complex input signal [Re, Im]

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

y = Complex output signal [Re, Im]

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

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

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

8.1.11 Vector AWGN

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

x_1 = Input signal vector (Real elements)

x_2 = Frame clock (impulse)

y_1 = Output signal vector

y_1 = Frame clock (impulse)

y_3 = Current run's E_s/N_o value

Number of Runs

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

Vector Size

Specifies the size of the input and output vectors.

Ref. Average Signal Power

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

Ref. Power Units

dBm in 50 Ohms

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

Watts in 1 Ohm

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

Run *x* E_s/N_o

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

8.2 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`.

8.2.1 Complex Addition

This block performs complex addition.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 + x_2$$

8.2.2 Complex Conjugate

This block outputs the complex conjugate of the input.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = x*$$

8.2.3 Complex Division

This block performs complex division.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 \div x_2$$

8.2.4 Complex Inverse

This block outputs the inverse of the complex input.

x = Complex input [Re, Im]

y = Complex output [Re, Im]

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

8.2.5 Complex Multiplication

This block performs complex multiplication.

x_1 = Complex signal #1 [Re, Im]

x_2 = Complex signal #2 [Re, Im]

y = Complex output [Re, Im]

$$y = x_1 \cdot x_2$$

8.2.6 Complex Power

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

x = Complex input [Re, Im]

y = Complex output [Re, Im]

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

Exponent

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

8.2.7 Complex Square Root

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

x = Complex input [Re, Im]

y = Complex output [Re, Im]

$$y = \sqrt{x}$$

8.2.8 Complex to Mag/Phase

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

x = Complex vector input [Re, Im]

y_1 = Complex magnitude

y_2 = Complex Phase

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

8.2.9 Complex to Real/Imag

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

x = Complex vector input [Re, Im]

y_1 = Real part

y_2 = Imaginary part

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

8.2.10 Mag/Phase to Complex

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

x_1 = Complex magnitude

x_2 = Complex phase

y = Complex vector output [Re, Im]

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

8.2.11 Real/Imag to Complex

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

x_1 = Real part

x_2 = Imaginary part

y = Complex vector output [Re, Im]

$$y = [x_1, x_2]$$

8.3 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`.

8.3.1 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 whever 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, π/4-DQPSK, D8PSK, D16PSK and D32PSK.

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y_1 = Output symbol number

x_2 = Output clock (impulse train)

DPSK Type

Specifies the desired DPSK modulation type. Supported modes include DBPSK, DQPSK, π/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.

8.3.2 FM Demodulator

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

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

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

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

x = Complex FM modulated signal

y = Demodulated signal

FM Deviation Index

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

Offset

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

Offset = Carrier Freq / FM deviation index

Overflow Value

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

8.3.3 IQ Detector

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

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

x_1 = IQ Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y = Output symbol number

Constellation Size

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

Select File

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

Browse File

Opens the selected constellation map file using Notepad.

File Path

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

File header *(can be anything)*

Symbol number *I output* *Q output*

...

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

```
Arbitrary IQ Constellation Map File                  (Header line)
0    1.2       1
1    -0.9      1.1
2    -1        -1.15
                                                (blank lines are OK)
3    -1.05     -0.95
```

The above example illustrates a QPSK constellation with slight imbalance.

8.3.4 PPM Demodulator

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

x = Baseband modulated signal

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

Number of Levels

Indicates the number *N* of possible symbol values.

Pulse Width

Indicates the width of the rectangular pulse in seconds.

Symbol Rate

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

Frame Start Delay

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

8.3.5 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.

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

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

y = Output symbol number

PSK Type

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

Constellation Rotation

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

Channel Rotation

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

Select File

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

Browse File

Opens the selected PSK constellation map file using Notepad.

PSK File Path

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

8.3.6 QAM/PAM Detector

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

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

x_1 = Complex input signal [Re, Im]

x_2 = Sample clock (impulse train)

x_3 = External constellation spacing reference [Optional]

y = Output symbol number

QAM Type

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

Constellation Rotation

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

Channel Rotation

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

Constellation Spacing

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

Amplitude Reference

Internal

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

External

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

Select File

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

Browse File

Opens the selected QAM constellation map file using Notepad.

QAM File Path

Specifies the 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.

8.4 Digital category

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

8.4.1 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 viceversa. 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.

8.4.2 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 wheneve 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).

8.4.3 Bits to Symbol

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

x_1 = Input bit stream #1

.. ...

x_n = Input bit stream #*n*

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

Bit Order

LSB First

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

MSB First

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

Number of Input Bits

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

8.4.4 Buffer

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

The output vector can be made to update following each input data clock pulse, or wait until an entire frame of data is available. Once the specified number of values (*N*) has been read, the block outputs a write strobe (frame clock) to signify that the output vector is fully updated. Once this occurs, new input values will start to overwrite the oldest values in the internal buffer in cyclic fashion. The buffer is initialized to all zeros at the start of a simulation.

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

x_1 = Input serial data

x_2 = Input clock (impulse train)

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

y_1 = Frame Clock (impulse)

y_2 = Output data vector (size *N*)

Buffer Ordering

FIFO

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

LIFO

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

Output Mode

Sequencial

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.

8.4.5 D Flip Flop

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

x_1 = D input

x_2 = Clock (rectangular)

y_1 = Flip flop output

y_2 = Complemented output

Edge Mode

Rising Edge

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

Falling Edge

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

Initial State

Specifies the initial state of the flip flop.

Threshold

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

8.4.6 Divide by N

This block implements a digital divide by N function. An internal counter is used to produce an output clock transition for every *N* detected input clock transition. Block parameters include the divide ratio, initial delay, and clock threshold voltage.

The `Divide by N` block works on the rising and falling edges of the input waveform, and can accept either an impulse train or a rectangular pulse train as its input. 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.

x = Input clock

y = Divided output clock

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 which 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.

8.4.7 JK Flip Flop

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

x_1 = J input

x_2 = K input

x_3 = Clock (rectangular)

y_1 = Flip flop output

y_2 = Complemented output

Edge Mode

Rising Edge

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

Falling Edge

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

Initial State

Specifies the initial state of the flip flop.

Threshold

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

8.4.8 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.

8.4.9 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 your responsibility. It is recommended that each output bit be represented by an integer number of simulation steps.

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

x_2 = Input symbol clock (impulse train)

y_1 = Output serial bit stream

y_2 = Output bit clock (impulse train)

Bit Order

LSB First

Indicates that y_1 is considered the LSB.

MSB First

Indicates that y_1 is considered the MSB.

Bits per Symbol

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

Output Bit Rate

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

8.4.10 Queue

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

x_1 = Data in

x_2 = Input clock (impulse train)

x_3 = Output clock (impulse train)

y_1 = Data out

y_2 = Queue count

y_3 = Overflow flag (see below)

Queue Size

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

Empty Output

Specifies the value to output when the queue is empty.

Queue Type

FIFO

Specifies that the queue operates as a FIFO queue.

LIFO

Specifies that the queue operates as a LIFO queue.

The overflow flag output behaves as follows:

Flag = 0:	Normal condition
Flag = 1:	An overflow has occurred during the simulation
Flag = 2:	Overflow in progress

8.4.11 Serial to Parallel

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

x_1 = Input serial bit

n = number of bits

x_2 = Input bit clock (impulse train)

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

y_2 = Output data clock (impulse train)

Bit Order

LSB First

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

MSB First

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

Bits per Symbol

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

8.4.12 State Machine

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

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

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

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

y_1 = Current state [0, *L*-1]

y_2 = Output clock (impulse train)

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

Number of Inputs

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

Number of Outputs

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

Number of States

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

Select File

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

Browse File

Opens the selected state machine map file using Notepad.

State Map File Path

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

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

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

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

```
V.32 Trellis Map File           ( 1st header line)
16    2   8                     ( 16 inputs, 2 outputs, 8 states)
state    input      new state     out #1      out #2      ( 2nd header line)
0         0         0             -4          1
0        1          0             0           -3
...      ...        ...           ...         ...
0        15         2             -2          1
1        0          2             -4          1
...      ...        ...           ...         ...
7        15         7             -1          4
```

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

8.4.13 Symbol to Bits

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

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

y_1 = Output bit stream #1

.. ...

y_n = Output bit stream #*n*

Bit Order

LSB First

Indicates that y_1 is considered the LSB.

MSB First

Indicates that y_1 is considered the MSB.

Number of Output Bits

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

Unbuffer

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

x_1 = Frame Clock (impulse)

x_2 = Input data vector (size N)

x_3 = External clock [optional] (impulse train)

y_1 = Output serial data

y_2 = Output clock (impulse train)

Output Mode

FIFO

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

LIFO

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

Output Timing

Internal

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

External

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

Buffer Size

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

Cyclic Output

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

Output Rate

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

8.5 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)`.

8.5.1 Block Interleaver

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

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

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

x_1 = Input data stream

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

y_1 = Output data stream

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

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

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

Mode

Interleave

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

Deinterleave

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

Rows

Specifies the number of rows used in the block.

Columns

Specifies the number of columns used in the block.

8.5.2 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. Generator coefficients should be computed appropriately. The first output bit comes from generator coefficient #1.

The `Convolutional Encoder` block requires that the simulation step divides evenly into both the input and output (coded) bit duration.

x_1 = Input data bit stream

x_2 = Input data clock (impulse train)

y_1 = Coded output bit stream

y_2 = Output data clock (impulse train)

No. Information Bits

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

No. Coded Bits

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

Constraint Length

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

Input Bit Rate

Specifies the bit rate of the incoming data in hertz.

Generator Coefficients

Specifies the value of each generator coefficient in octal format.

8.5.3 Convolutional Interleaver

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

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

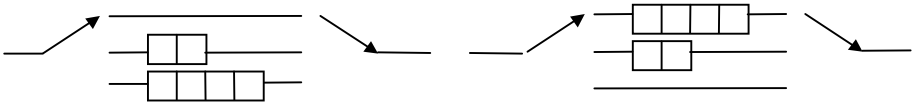
The `Convolutional Interleaver` block accepts a real value and outputs a real value.

x_1 = Input data stream

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

y_1 = Output data stream

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



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.

Row Increment

Specifies the cell increment from row to row.

8.5.4 Depuncture

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

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

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

x_1 = Frame clock (impulse)

x_2 = Input data vector (size M)

y_1 = Frame clock (impulse)

y_2 = Output data vector (size N)

Number of Encoder Outputs

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

Output Vector Size

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

Compute Input Vector Size

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

Select File

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

Browse File

Opens the selected Puncture Pattern file using Notepad.

Puncture Pattern File Path

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

Header line (can be anything)

# of entries *(period of pattern)*

mask value #1

mask value #2, mask value #3

etc…

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

8.5.5 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 which differ by only one bit.

For example, this block will map the input values below

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

to the following

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

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

x = Input value

y = Gray encoded integer

8.5.6 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 which differ by only one bit.

For example, this block will map the input values

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

to the following

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

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

x = Gray encoded value

y = Output value

8.5.7 Hamming Decoder

This block implements a Hamming decoder. 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 coded vector of size n and outputs a data vector of size k. The input is assumed to be in systematic form (data bits followed by parity bits). An external clock is used to signal when each decoding operation should take place.

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

x_1 = Input coded vector (size n)

x_2 = Input clock (pulse)

y_1 = Output data vector (size k)

y_2 = Output clock (pulse)

Hamming Code Size

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

8.5.8 Hamming Encoder

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

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

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

x_1 = Input data vector (size *k*)

x_2 = Input clock (pulse)

y_1 = Coded output vector (size *n*)

y_2 = Output clock (pulse)

Hamming Code Size

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

8.5.9 Puncture

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

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

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

x_1 = Frame clock (impulse)

x_2 = Input data vector (size N)

y_1 = Frame clock (impulse)

y_2 = Output data vector (size M)

Number of Encoder Outputs

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

Input Vector Size

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

Compute Output Vector Size

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

Select File

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

Browse File

Opens the selected Puncture Pattern file using Notepad.

Puncture Pattern File Path

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

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

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

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

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

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

8.5.10 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 nonbinary symbols from a Galois Field (GF) and are systematic. In an RS(n, k) code, n - k parity symbols are added at the end of the k input symbols, resulting in a total of *n* output symbols. An RS(*n*, *k*) code is capable of correcting up to (*n* - *k*)/2 errors.

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

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

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

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

x_2 = Input clock (pulse)

y_1 = Decoded output vector (size *k*)

y_2 = Output clock (pulse)

y_3 = Error count (-1 indicates RS failure)

Symbol Size

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

Information Symbols

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

Coded Symbols

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

Advanced Settings

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

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

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

where "@" represents a lower case alpha.

First root of generator polynomial

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

Power of alpha used for roots of generator polynomial

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

The primitive polynomials used by the RS Decoder block are shown below:

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

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

8.5.11 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 nonbinary symbols from a Galois Field (GF) and are systematic. In an RS(*n*, *k*) code, *n* - *k* parity symbols are added at the end of the *k* input symbols, resulting in a total of *n* output symbols. An RS(*n*, *k*) code is capable of correcting up to (*n* - *k*)/2 errors.

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

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

x_1 = Input data vector (size *k*)

x_2 = Input clock (pulse)

y_1 = Coded output vector (size *n*)

y_2 = Output clock (pulse)

Symbol Size

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

Information Symbols

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

Coded Symbols

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

Advanced Settings

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

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

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

where "@" represents a lowercase alpha.

First root of generator polynomial

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

Power of alpha used for roots of generator polynomial

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

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

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

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

8.5.12 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.

8.5.13 Trellis Encoder

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

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

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

x_1 = Input symbol number

x_2 = Input data clock (impulse train)

y_1 = I channel output

y_2 = Q channel output

y_3 = Output data clock

Number of Input Bits

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

Number of Output Bits

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

Number of States

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

Trellis File Path

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

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

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

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

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

V.32 Trellis Map File						(*1st header line)*
4	5	8				*(4 input bits, 5 output bits, 8 states)*
state		input	new state	I out	Q out	(*2nd header line)*
0		0	0	-4	1	
0		1	0	0	-3	
...		...	...	...	...	
0		15	2	-2	1	
1		0	2	-4	1	
...		...	...	...	...	
7		15	7	-1	4	

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

8.5.14 Viterbi Decoder (Hard)

This block implements a hard 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 output bit rate, and the generator coefficients.

The `Viterbi Decoder (Hard)` block requires the simulation step size to divide evenly into both the input (coded) and output bit duration.

This block takes as input a binary stream and outputs a binary stream. 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.

x_1 = Input coded bit stream (0, 1)

x_2 = Input coded data clock (impulse train)

y_1 = Decoded bit stream (0, 1)

y_2 = Decoded data clock (impulse train)

y_3 = Best path metric (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 of the output decoded data in hertz.

Generator Coefficients

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

8.5.15 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.

The `Viterbi Decoder (Soft)` block requires the simulation step size to divide evenly into both the input and output (coded) bit duration.

This block accepts a real value and outputs a binary stream.

x_1 = Input coded data stream

x_2 = Input coded data clock (impulse train)

y_1 = Decoded bit stream (0, 1)

y_2 = Decoded data clock (impulse train)

y_3 = Best path metric

No. Information Bits

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

No. Coded Output Bits

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

Constraint Length

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

Trellis Truncation Length

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

Quantization Bits

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

Output Bit Rate

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

Generator Coefficients

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

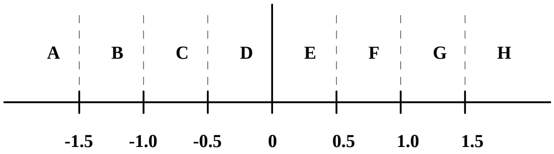
Metric File Path

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

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

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

Example Metric File



```
Viterbi Decoder Metric File, Q = 3
3,      7.0,    0.0
-1.5,   6.0,    1.0
-1.0,   5.0,    2.0
-0.5,   4.0,    3.0
0,      3.0,    4.0
0.5,    2.0,    5.0
1.0,    1.0,    6.0
1.5,    0.0,    7.0
```

For example:
for an input $0.5 \le x < 1.0$,
the metric for a “0” bit is “2.0”,
and the metric for a “1” bit is “5.0”.

8.6 Estimators category

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

8.6.1 Average Power (Complex or Real)

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

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

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

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

y = Power estimate

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

N = window size or # of samples since reset

k,i = simulation step indices

Load

1 Ohm

Specifies a load resistance of 1 Ohm.

50 Ohms

Specifies a load resistance of 50 Ohms.

Output Units

dBm

Watts

dBW

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

Mode

Sliding

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

Running

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

Window Size

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

Shift Reg. Initial Value

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

8.6.2 BER Curve Control

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

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

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

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

x_1 = Current E_s/N_o level

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

y_1 = Trigger for BER plot

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

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

Number of Runs

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

Mode

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

Bit Error Rate

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

Symbol Error Rate

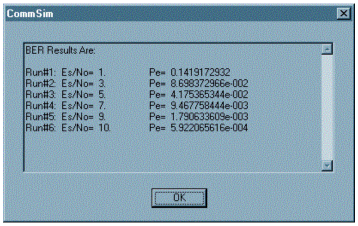
Forces the use of the Es/No label in the results summary. Use this setting when providing a reference Es/No input to the block.

Duration

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

Show Results

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



```
BER Results Are:

Run#1:  Es/No= 1.          Pe= 0.1419172932
Run#2:  Es/No= 3.          Pe= 8.698372966e-002
Run#3:  Es/No= 5.          Pe= 4.175365344e-002
Run#4:  Es/No= 7.          Pe= 9.467758444e-003
Run#5:  Es/No= 9.          Pe= 1.790633609e-003
Run#6:  Es/No= 10.         Pe= 5.922065616e-004
```

Suppress Result Notification

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

8.6.3 BER Control (# Errors)

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

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

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

x_1 = Current E_s/N_o level

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

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

y_1 = Trigger for BER plot

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

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

Number of Runs

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

Mode

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

Bit Error Rate

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

Symbol Error Rate

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

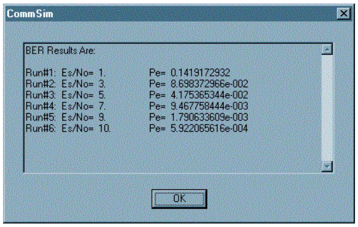
Max Errors

Specifies the desired maximum number of errors for each run.

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

Show Results

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



```
BER Results Are:

Run#1:  Es/No= 1.          Pe=  0.1419172932
Run#2:  Es/No= 3.          Pe=  8.698372966e-002
Run#3:  Es/No= 5.          Pe=  4.175365344e-002
Run#4:  Es/No= 7.          Pe=  9.467758444e-003
Run#5:  Es/No= 9.          Pe=  1.790633609e-003
Run#6:  Es/No= 10.         Pe=  5.922065616e-004
```

Suppress Result Notification

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

8.6.4 Bit/Symbol Error Rate

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

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

x_1 = Recovered data stream

x_2 = Reference data stream

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

y_1 = Error rate (symbol or bit)

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

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

Count Start Delay

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

Output Mode

Bit Error Rate

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

Symbol Error Rate

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

Bits per Symbol

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

8.6.5 Correlation

This block performs a correlation between two real signals, referred to as the recovered signal and reference signal. The correlation is performed over a variable window size. Two modes are available: standard correlation and gated. In standard correlation mode, the two signals are continuously shifted through the correlation window. In gated mode, the recovered signal is still continuously shifted, but the reference signal is only shifted when the external gate signal is low. The latter mode can be used to implement a matched filter or convolution by sliding in a desired waveform and then locking the control gate.

x_1 = Recovered signal

x_2 = Reference signal

x_3 = External gate control (0, 1)

y = Correlation output

$$y_{[k]} = \sum_{i=0}^{N-1} \left(x_{1[k-i]} \cdot x_{2[j]} \right) \qquad j = \begin{cases} k-i & \text{sliding mode} \\ j_0 - 1 - i & \text{gated mode} \end{cases}$$

N = window size

i, j, k = simulation step indices

j_0 = gate high 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 recovered signal. When the gate voltage becomes high, the reference signal is no longer shifted and a sliding correlation with the incoming recovered 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 correlation buffer in simulation steps.

8.6.6 Delay Estimator

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

x_1 = Simulation output signal

x_2 = Reference signal

y_1 = Delay estimate

y_2 = Result flag (0, 1)

Window Size

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

Max. 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.

8.6.7 Event Time

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

x = Input signal

y = Time of occurrence

Event Mode

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

Threshold Value

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

8.6.8 Mean

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

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

x_1 = Input signal

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

y = Mean estimate

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

N = window size or number of samples since reset

k,i = simulation step indices

Mode

Sliding Window

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

Running

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

Window Size

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

Shift Reg. Initial Value

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

8.6.9 Median

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

x = Input data

y = Median output

Window Size

Specifies the size N of the sliding window.

Shift Register Initial Value

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

8.6.10 Variance

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

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

x_1 = Input signal

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

y_1 = Variance estimate

y_2 = Mean estimate (optional)

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

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

N = window size or number of samples since reset

k,i = simulation step indices

Mode

Sliding Window

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

Running

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

Window Size

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

Initial Value

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

8.6.11 Weighted Mean

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

x_1 = Input signal

x_1 = Input weight (w in formula below)

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

y = Weighted mean

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

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

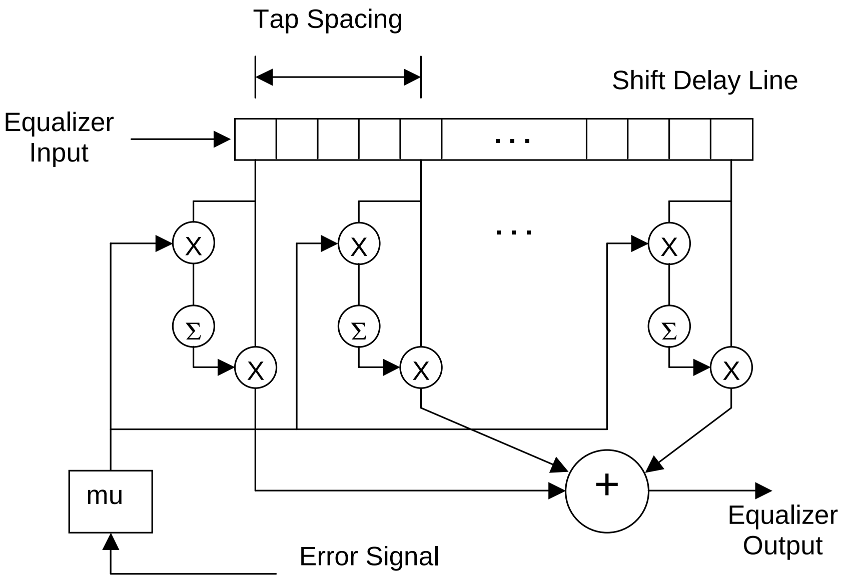
Specifies the size N of the sliding window buffer in simulation steps. This parameter is available only when sliding mode is activated.

8.7 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`.

8.7.1 Adaptive Equalizer (Complex or Real)

Two versions of this block exist, one complex and the other real. This block implements a conventional or fractionally-spaced adaptive equalizer suitable for use with both analog and digital waveforms. Key block parameters include the total number of taps *N*, the number of taps per symbol, initial tap values, and the convergence coefficient “mu”. The basic equalizer structure is shown below. Each cell in the equalizer’s internal shift delay line corresponds to a single simulation step. The tap spacing is a derived internal parameter obtained by looking at the input symbol rate and the desired number of taps per symbol. This block requires that the tap spacing be an integer number of simulation time steps. This can be achieved my making the simulation rate an integer multiple of the symbol rate times the number of taps per symbol.



This block uses a Least Mean Square (LMS) convergence algorithm to adapt the tap values with the goal of minimizing the average error. The user is responsible for providing a suitable error input to the block, for example the error vector between a received point in the IQ plane and the closest constellation point. The error value is only read whenever an internal/external clock pulse occurs. Due to internal delays within the block, the error signal can involve 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" 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.

When implementing a fractionally spaced equalizer (e.g. more than 1 tap per symbol) only the first tap of each "subgroup" (comprised of taps corresponding to the same symbol) is initialized. This is done to synchronizes the input/output clock (at 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.

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

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

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

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

y_1 = Output signal

y_2 = Clock output (impulse train)

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

Number of Taps

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

Taps per Symbol

Specifies the number of taps per symbol period.

Mu

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

Symbol Rate

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

Input Time Delay

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

Initialized Tap Value

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

Initialized Tap Offset

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

Show Taps

Displays the current values of the equalizer internal taps.

Timing

External

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

Internal

Specifies that the equalizer 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.

8.7.2 File FIR Filter

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

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

x = Input signal

y = Filtered output signal

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.

8.7.3 FIR Filter

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

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

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

x = Input signal

y = Filtered output signal

Number of Taps

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

Cutoff Freq 1

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

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

Cutoff Freq 2

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

Window Type

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

Beta

Specifies the shape parameter associated with the Kaiser window.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Rolloff Factor

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

Filter Type

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

Normalize LPF to 0 dB at dc

Specifies that the filter impulse response is to be normalized so that the dc gain is unity (0 dB). 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 Commsim 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 Commsim filter viewer, which allows you to review the filter's impulse response, gain response, phase response, and group delay response.

8.7.4 IIR Filter

This block implements an Infinite Impulse Response (IIR) filtering. You can choose from several well-known analog filter prototypes, including Butterworth and Chebyshev designs. The desired filter is implemented using bilinear transformation to map the *s*-domain analog design to the digital *z*-domain.

The `IIR` block differs from a discrete-time `transferFunction` block (listed under the Linear Systems category in the Blocks menu) in that the simulation time step is updated automatically prior to each run.

The `IIR` block accepts a real signal and outputs a real signal. You should verify that a stable impulse response is obtained, especially when specifying a very narrowband filter (<1% Fs) that is also of high filter order. The View Response button can be helpful for this purpose.

x = Input signal

y = Filtered output signal

Filter Method

Indicates the filter design method to be used. You can choose from Butterworth, Chebyshev Type I, Chebyshev II (Inverse Chebyshev), and Bessel.

Cutoff Freq 1

Specifies the desired cutoff frequency for Lowpass and Highpass filter types, or specifies the desired lower cutoff frequency for Bandpass and Bandstop filter types.

Cutoff Freq 2

Specifies the upper cutoff frequency for Bandpass and Bandstop filter types.

Passband Specification

Epsilon

Indicates that the filter response at the cutoff frequency is determined from the value of Epsilon. A value of 1 for Epsilon, corresponds to an attenuation of 0.5.

Ripple

Indicates that the filter response at the cutoff frequency is determined from the value of Ripple. Ripple is a positive value, expressed in decibels, and corresponds to the desired attenuation at the cutoff frequency.

Stopband Atten.

Specifies the stopband attenuation for the desired filter in decibels. This parameter only applies to Chebyshev Type II filters. Its value must exceed the Ripple value.

Units

Hertz

Indicates that cutoff frequency values are in hertz.

Radians/Sec

Indicates that cutoff frequency values are in radians/second.

Filter Order

Indicates the desired filter order. Valid range is 1 to 20. If you select either the bandpass and 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 Commsim Filter Viewer, which allows you to review the filter’s impulse response, gain response, phase response, and group delay response.

8.7.5 MagPhase Filter

This block implements an arbitrary complex FIR filter based on user-specified magnitude and phase responses supplied via an external file. The magnitude response is specified in decibels, while the phase response may be provided in either degrees, radians, or as a group delay. The filter is realized using the overlap-save method. The input signal is mapped to the frequency domain via FFT, multiplied by the specified frequency response, and then mapped back to the time domain via IFFT. The size of the FFT is twice that of the equivalent complex FIR filter tap length.

The input file should include points from $-f_s/2$ to $+f_s/2$ (0 to $+f_s/2$ for a real filter) in increasing order, but the frequency spacing need not be uniform. Linear interpolation is used to compute intermediate points as required. When a single-sided response is provided (real case), the routine internally creates a mirror image of the response (with opposite sign phase response) for the negative portion of the spectrum. In the event that the frequency range specified by the input file does not include the entire $-f_s/2$ to $+f_s/2$ range, values outside the specified range are linearly extrapolated based on the closest two specified frequency points.

An implementation delay equal to the equivalent FIR filter length plus one (in simulation steps) is experienced when using this block. This is in addition to any filter delay due to its response.

This block accepts a complex signal and outputs a complex signal. It also outputs the internal interpolated response used by the FFT routine. This is provided by the second output connector in vector form [3x1], and may be used to drive a `plot` block configured in XY mode.

x = Complex input signal [Re, Im]

y_1 = Filtered complex output signal [Re, Im]

y_2 = Interpolated response used by FFT routine [mag, phase, freq]

Equivalent FIR Filter Tap Length

Specifies the length of the filter's impulse response. This value must be a power of two as it determines the size of the internal FFT computations.

Phase Units

Degrees

Indicates that the file phase response is specified in units of degrees.

Group Delay

Indicates that the file phase response is specified as a group delay response in seconds.

Radians

Indicates that the file phase response is specified in units of radians.

Filter File Data

[0, fs/2]

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

[-fs/2, +fs/2]

Used when the input file includes data over the range of [-fs/2, +fs/2] (complex filter).

Add Linear Phase

Automatically adds linear phase to the input phase specification. The amount of linear phase added corresponds to a delay of ½ the FIR filter's impulse response duration. This option is useful when the input file phase specification represents the filter's deviation from linear phase.

Normalize Phase

Automatically normalizes the internal phase response (derived from the group delay data) so that the phase response is zero at the zero frequency point. This setting only applies when the Group Delay specification method is chosen.

Select File

Opens the Select File dialog box for selecting a filter frequency response file.

Browse File

Opens the selected filter frequency response file using Notepad.

Filter File Path

Specifies the DOS path to the desired filter frequency response file. The format of the file is described below:

File header (anything)

number of entries (n)

frequency point #1, magnitude, phase

frequency point #2, magnitude, phase

...

frequency point #n, magnitude, phase

Entries are to be provided in increasing frequency order and should cover the range from -f_S/2 to +f_S/2 or 0 to +f_S/2. Entries may be separated by commas, blank space, or tabs. The maximum allowed line length is 100 characters. The expected units are hertz for frequency, decibels for magnitude, degrees or radians for phase, and seconds for group delay.

Care should be taken to ensure that the correct amount of linear phase is built into the phase specification. Alternatively the Add Linear Phase box may be checked. Since the filter’s delay is typically equal to ½ the specified FIR filter impulse response length, the required amount of linear phase can be obtained as follows:

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

8.7.6 Pulse Shaping Filter

This block implements pulse shaping using a *finite impulse response* (FIR) approach. It allows the use of a variety of windowing shapes, as well as Nyquist type pulse shapes, such as the raised cosine and root raised cosine forms. This block also supports Gaussian pulse shapes.

The `Pulse Shaping Filter` block expects an impulse pulse train representing the input symbol values. If a rectangular input signal is provided instead (e.g. NRZ data), an inverse sinc function can be applied to convert, internally to the block, the rectangular pulse train into an impulse one.

The `Pulse Shaping Filter` block normally introduces a delay equal to $N / 2$ simulation steps, where N is the number of filter taps (assuming the filter sampling frequency equals the simulation rate). The number of taps is equal to the pulse span interval times the number of samples per symbol. This block accepts a real signal and outputs a real signal.

x = Input signal (impulse train or rectangular pulses)

y = Pulse shaped output

Pulse Span

Specifies the span of the pulse shaping filter in units of Symbol Periods (T). Note: The filter tap at the upper span boundary is usually omitted. For example, in the case of a 4T pulse span, the filter taps will range from [-2/T, 2/T - Δt], where Δt is the simulation time step.

Samples per Symbol

Specifies the number of samples per symbol associated with the input data signal.

Rolloff Factor

Specifies the rolloff factor (alpha) associated with the raised cosine or root raised cosine filter types. The valid range is from 0 to 1, with 0 corresponding to a truncated sin(x)/x impulse response.

Beta

Specifies the shape parameter associated with the Kaiser window.

BT Product

Specifies the BT product value associated with the Gaussian filter type.

Filter Type

Indicates the desired pulse shaping filter type. Click on the DOWN ARROW to select from a list of filter types. Available types are: window only, raised cosine, root raised cosine, and Gaussian. A Rolloff Factor must be supplied when the Raised Cosine or Root Raised Cosine filters are selected, and a BT Product value for the Gaussian case.

Window Type

Specifies the window function to be used in the realization of the FIR filter. If the Kaiser window is selected, a value for the Kaiser Beta parameter must also be supplied.

Apply Inverse Sinc

When selected, an inverse sinc function is cascaded with the pulse shaping FIR response. This option should be used whenever a rectangular pulse train, rather than an impulsive one, is used at the block’s input.

Show Taps

Displays the current FIR Pulse Shaping Filter tap values. Taps are shown in order of increasing delay. Once the taps are displayed, the values can be saved to a user-specified file by pressing the Save To File button. If the Use FIR File Format box is checked, the values are saved in a format compatible with the `File FIR` block.

View Response

Displays the current FIR Pulse Shaping Filter response by launching the Commsim Filter Viewer.

8.7.7 Sampling File FIR Filter

This block implements a sampling Finite Impulse Response (FIR) filter based on user-supplied tap values provided in a file. Up to 32,767 taps can be specified. The block will sample the input signal at the specified rate and propagate these values through its internal shift register. The `Sampling File FIR` output signal can be either held or interpolated when the filter sampling rate is below the simulation rate. A filter delay adjustment is provided to allow fine tuning of the sampling instant.

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

x = Input signal

y_1 = Filtered output signal

y_2 = Filter sampling clock (impulse train)

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.

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.

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.

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.

8.7.8 Sampling FIR Filter

This block implements a sampling Finite Impulse Response (FIR) filter. The block will sample the input signal at the specified rate and propagate these values through its internal shift register. The `Sampling FIR` output signal can be either held or interpolated when the filter sampling rate is below the simulation rate. A filter delay adjustment is provided to allow fine tuning of the sampling instant. The `Sampling FIR` block employs the windowing method for filter design and allows you to implement lowpass, highpass, bandpass, bandstop, raised cosine, root raised cosine, Hilbert and Gaussian filters with your choice of window function.

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

This block accepts a real signal and outputs a real signal.

x = Input signal

y_1 = Filtered output signal

y_2 = Filter sampling clock (impulse train)

Number of Taps

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

Cutoff Freq 1

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

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

Cutoff Freq 2

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

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 filter impulse response is to be normalized so that the dc gain is unity (0 dB). 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.

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 between computed output points.

Show Taps

Displays the current FIR Filter tap values in the Commsim 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.

8.8 Instruments category

Blocks in the Instruments categories include `BER Curve Display`, `Oscilloscope Display`, `Sprectrum Analyzer Display (Complex)`, and `Spectrum Analyzer Display (Real)`.

8.8.1 BER Curve Display

This pre-wired set of blocks provides quick access to a BER curve display. It includes a `BER Co`ntrol block (Estimators), and a pre-configured `plot` block.

8.8.2 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.

8.8.3 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.

8.8.4 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.

8.9 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.

8.9.1 AM Modulator

This block performs *double-sideband amplitude modulation* (DSB-AM) of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `AM` block belongs to the family of analog modulators. In AM, the information is transmitted by varying the carrier signal amplitude according to the input signal level. The carrier frequency remains constant. This block accepts an analog signal as its input.

x = Input signal

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

y_2 = Carrier phase (rad) [optional]

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

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude *A* when the input is 0. This value is specified in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

Modulation Factor

Specifies the AM factor *m*. It controls the extent of carrier amplitude deviation according to the equation shown above.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the carrier phase.

Wrapped [0, 2pi]

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

8.9.2 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, π/4-DQPSK, D8PSK, D16PSK and D32PSK.

x_1 = Input symbol number (integer [0 … *N* -1], where *N* is the constellation size)

x_2= Input data clock (impulse train)

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

y_2 = Unmodulated carrier phase (rad)

$$y_1(t) = Ae^{j(2\pi f_c + \phi + \sum_{k=0}^{n} \theta_{d[k]})} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi \qquad \theta_{d[k]}(x) \text{ = k}th\text{ phase transition}$$

The phase transition value θ_d is obtained from the input symbol value according to the specified map file.

DPSK Type

Specifies the desired DPSK modulation type. Supported modes include DBPSK, DQPSK, π/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 correspond to greater power in the quadrature axis than in the in-phase axis.

Phase Imbalance

Specifies the phase imbalance of the modulator in degrees as a deviation from ideal. A positive value correspond to a clockwise rotation of the Q axis relative to the I axis. For example, 10 degrees imbalance implies an angle of 80 degrees between the I and Q axes, instead of the ideal 90 degrees.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

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

Select File

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

Browse File

Opens the selected DPSK phase transition map file using Notepad.

DPSK File Path

Specifies the DOS path to the desired DPSK phase transition map file.

The file format uses a “keyword” followed by a listing of input symbols and corresponding “normalized” delta-phase values, where unity (1) corresponds to the constellation spacing (e.g. 90 degrees for DQPSK, and 45 degrees for Pi4DQPSK or D8PSK). Valid modulation keywords are *dbpsk*, *dqpsk*, *dpi4qpsk*, *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, π/4-DQPSK and D8PSK

```
dbpsk
0    0          (0 deg)
1    1          (180 deg)

dqpsk
0    0          (0)
1    1          (90)
2   -1          (-90)
3    2          (180)

dpi4qpsk
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)
```

8.9.3 FM Modulator

This block performs *frequency modulation* (FM) of the input signal based on the selected block settings. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `FM` block belongs to the family of analog modulators. In FM, the information is transmitted by varying the carrier frequency according to the input signal level. The carrier amplitude remains constant.

The `FM` block takes an analog signal as its input.

x = Input signal

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

y_2 = Phase of complex modulated signal (optional)

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

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

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Carrier Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

FM Deviation

Specifies the FM deviation index β. It controls the extent of carrier frequency variation according to above equation, and is specified in hertz/volt.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase.

Wrapped [0, 2pi]

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

8.9.4 FSK Modulator

This block performs *frequency shift keying* (FSK) modulation of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `FSK` block belongs to the family of digital modulators. In FSK modulation, the information is transmitted by varying the carrier frequency between N frequency settings depending on the input signal level. The carrier amplitude remains constant.

The `FSK` block accepts a symbol number as its input.

x = Input symbol number (integer [0 … N -1], where N is the number of tones)

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

y_2 = Phase of complex modulated signal (optional)

$$f_{out} = f_{min} + x \cdot \Delta f$$

Number of Tones

Specifies the number N of available tones.

Lowest Frequency

Specifies the carrier frequency corresponding to the lowest desired output tone in hertz. This corresponds to input symbol # 0.

Frequency Spacing

Specifies the frequency spacing Δf between adjacent FSK tones in hertz.

Amplitude

Specifies the carrier single-sided peak amplitude in volts.

Initial Phase

Specifies the initial carrier phase in degrees. This option is only available when you select continuous phase mode.

Phase Mode

Continuous

Indicates that the signal phase is continuous across FSK tone transitions. This simulates the use of a *voltage controlled oscillator* (VCO) or NCO in generating the output FSK signal.

Discontinuous

Indicates that the signal phase is not continuous across FSK tone transitions. This simulates the use of multiple free running oscillators to generate the output signal.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase. This mode is only available when the Phase Mode is set to “Continuous”.

Wrapped [0, 2pi]

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

8.9.5 GFSK Modulator

This block implements a Gaussian Frequency Shift Keying (GFSK) modulator as a compound block. In GFSK modulation, the digital information is transmitted by shifting the carrier frequency between two states.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of each of these blocks may need to be specified for proper operation, in addition to setting the BT product, symbol rate, and FM deviation global parameters. The default settings reflect a Bluetooth implementation.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

8.9.6 GMSK Modulator

This block implements a Gaussian Minimum Shift Keying (GMSK) modulator as a compound block. In GMSK modulation, the digital information is transmitted by shifting the carrier frequency between two states with a frequency offset of +/- 0.25 the symbol rate. This represents a special case of GFSK.

This block employs a Gaussian FIR Filter and an FM Modulator as its internal components. The internal parameters of each of these blocks may need to be specified for proper operation, in addition to setting the BT product and the symbol rate global parameters.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

8.9.7 IQ Modulator

This block performs generic Amplitude Phase Modulation given a pair of analog I/Q signal inputs. Two versions of this block are provided: one producing a Complex output and the other producing a Real output. The block can be used to produce either digital or analog modulation. The information is transmitted by varying both the carrier amplitude and phase according to the complex input signal. Block parameters include the carrier frequency, initial phase, and amplitude scaling factor. This block takes two parallel analog signals as its inputs.

x_1 = I channel input

x_2 = Q channel input

y_1 = Modulated signal ([Re, Im] for Complex)

y_2 = Unmodulated carrier phase (rad) [optional]

$$y_1(t) = A(x_1 + jx_2)e^{j(2\pi f_c t + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in *hertz*. It may be set to zero when working in complex envelope representation.

Amplitude Factor

Specifies the carrier amplitude scaling factor A applied to the input (I,Q) vector.

Initial Phase

Specifies the initial phase θ of the modulating carrier in *degrees*.

8.9.8 MSK Modulator

This block performs *minimum shift keying* (MSK) modulation of the input signals based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `MSK` block belongs to the family of digital modulators. MSK modulation is similar to SQPSK modulation, except that sinusoidal pulse shaping is applied to the data signal prior to modulation.

MSK modulation can also be viewed as a form of FSK modulation with tones at

$$f_c \pm \frac{R}{2}$$

where R is the symbol rate.

MSK modulation results in lower sidelobe energy levels than both QPSK and SQPSK modulation. Since the Q data is delayed half a symbol (within the block), it is preferable to select a simulation step size that yields an even number of simulation steps per symbol period.

The `MSK` block accepts two binary signals as its input: I and Q data, respectively.

x_1 = I channel data (binary)

x_2 = Q channel data (binary)

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

y_2 = Unmodulated carrier phase (rad) (optional)

$$y_1(t) = Ad_I \cos(\pi f_s(t-\tau))\cos(2\pi f_c(t-\tau)) + jAd_Q \sin(\pi f_s(t-\tau))\sin(2\pi f_c(t-\tau))$$

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

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

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Constellation Rotation

Specifies the constellation rotation θ in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at π/4 radians.

Data Rate

Specifies the data rate f_s in symbols/second.

Data Start Time

Specifies the start time τ of the input data in seconds.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

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

8.9.9 PM Modulator

This block performs *phase modulation* (PM) of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `PM` block belongs to the family of analog modulators. In PM, the information is transmitted by varying the carrier phase according to the input signal level. The carrier amplitude remains constant.

The `PM` block accepts an analog signal as its input.

x = Input signal

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

y_2 = Phase of complex modulated signal (optional)

$$y_1(t) = Ad^{\,j(2\pi f_c t + \beta x + \phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Carrier Frequency

Specifies the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Initial Phase

Specifies the initial carrier phase θ in degrees.

Modulation Index

Specifies the PM index β. It controls the extent of carrier phase variation according to the above equation, and is specified in radians/volt.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the modulated carrier phase.

Wrapped [0, 2pi]

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

8.9.10 PPM Modulator

This block performs PPM of the input signal based on the selected modulation parameters. In PPM, the information is transmitted by varying the occurrence of a rectangular pulse within a pre-defined symbol frame. The location of the pulse is proportional to the input signal level. Pulse spacing is automatically calculated, and no portion of the pulse ever occurs beyond the symbol frame boundaries.

The `PPM` block belongs to the family of digital modulators.

The `PPM` block accepts a symbol number as its input, and outputs a baseband real signal. The input is rounded to the closest allowed symbol number.

x = Input symbol number (integer [0 … *N* -1], where *N* is the number of levels)

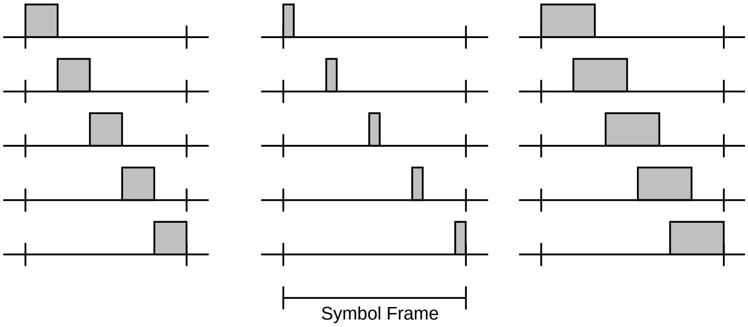
y = Baseband modulated signal

Number of Levels

Indicates the number *N* of possible input symbol values.

Pulse Width

Specifies the width of the rectangular pulse in seconds. The pulse width must be less than the symbol period, but may greater than, equal to, or less than the symbol period divided by the number of pulse positions. All the cases shown below are valid examples of the various output pulse positions for a five-level scheme.



Symbol Rate

Specifies the symbol rate (pulse rate) in symbols/second.

Pulse Amplitude

Specifies the amplitude of the rectangular pulse. This value is specified in volts.

Data Frame Start

Specifies the start time of the first data frame in seconds.

8.9.11 PSK Modulator

This block performs *phase shift keying* (PSK) modulation of the input signal based on the selected modulation parameters. In PSK modulation, the digital information is transmitted by varying the carrier phase between known phase states. The carrier amplitude remains constant. Two versions of this block are provided: one producing a complex output and the other producing a real output. The following constellations are available: BPSK, QPSK, 8-PSK, 16-PSK, and 32-PSK.

This block belongs to the family of digital modulators. It accepts as its input a binary signal (BPSK only) or a symbol number and maps it to the constellation point specified in the PSK map file.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

x_2 = Input clock (impulse train)

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

y_2 = Unmodulated carrier phase (rad)

$$y_1(t) = Ae^{f(2\pi f_c t+\theta_d+\phi)} \qquad \phi = \frac{\pi\theta_r}{180}$$

$$y_2(t) = 2\pi f_c t + \theta \qquad \theta_d(x) = \text{data phase}$$

PSK Type

Specifies the PSK modulation type. Choices include BPSK QPSK 8-PSK 16-PSKor 32-PSK Each modulation scheme is described below in more detail.

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude A in volts.

Constellation Rotation

Specifies the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at 0 rad for BPSK, and at π/n rad for all others, where n is the constellation size.

Gain Imbalance

Specifies the gain imbalance (Q relative to I) of the modulator in units of dBs. A positive value correspond to greater power in the quadrature axis than in the in-phase axis.

Phase Imbalance

Specifies the phase imbalance of the modulator in degrees as a deviation from ideal. A positive value correspond to a clockwise rotation of the Q axis relative to the I axis. For example, 10 degrees imbalance implies an angle of 80 degrees between the I and Q axes, instead of the ideal 90 degrees.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

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

Select File

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

Browse File

Opens the selected PSK constellation map file using Notepad.

PSK File Path

Specifies the DOS path to the desired PSK constellation map file. The format of the map file is described below:

File header (anything)

modulation keyword

symbol # for 1st constellation point, symbol # for 2nd constellation point

...

symbol # for last constellation point

next modulation keyword [optional]

symbol # ...[optional]

Valid modulation keywords are *bpsk*, *qpsk*, *8psk*, *16psk* and *32psk*. They must be specified in lowercase letters. After the modulation keyword, data can be arranged as desired starting on the next line. Valid data delimiters are commas, blank spaces, tabs, and carriage returns. The maximum allowed line length is 100 characters.

Constellation point numbering starts from the positive *I* axis and proceeds counterclockwise. Each map file may contain multiple modulation mappings: one for each PSK type. Below is an example of a Gray encoded 8-PSK map file, illustrating the use of various data delimiters. Gray coding makes neighboring constellation points differ by only 1 bit.

PSK Map File: Gray Coded Mapping

8psk

0 1 3, 2

6

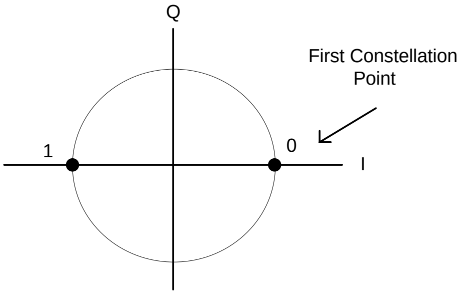
7, 5 4

8.9.11.1 BPSK

This option performs BPSK modulation of the input signal based on the specified block parameters. In BPSK modulation, the digital information is transmitted by varying the carrier phase between two states spaced by π rad.

The `BPSK` block accepts a binary signal {0, 1} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).

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



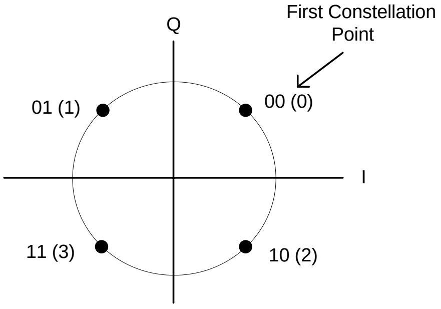
BPSK

8.9.11.2 QPSK

This option performs *quadrature phase shift keying* (QPSK) modulation of the input signal based on the specified block parameters. In QPSK modulation, the digital information is transmitted by varying the carrier phase among four states equally spaced at π/2 rad increments.

The `QPSK` block accepts a symbol number {0, 1, 2, 3} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file PSK_GRAY.DAT).

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



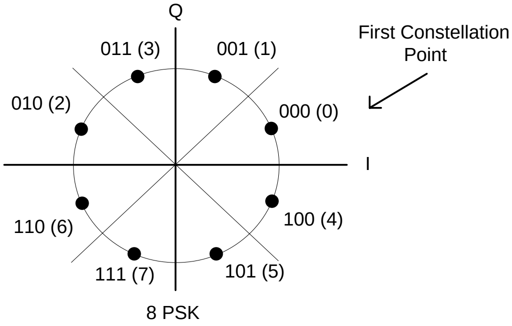
QPSK

The default first constellation point is at π/4 radians.

8.9.11.3 8-PSK

This option performs *eight phase shift keying* (8-PSK) modulation of the input signal based on the specified block parameters. In 8-PSK modulation, the digital information is transmitted by varying the carrier phase among eight states equally spaced at π/4 rad increments.

The `8-PSK` block accepts a symbol number {0, 1, 2, ..., 7} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).

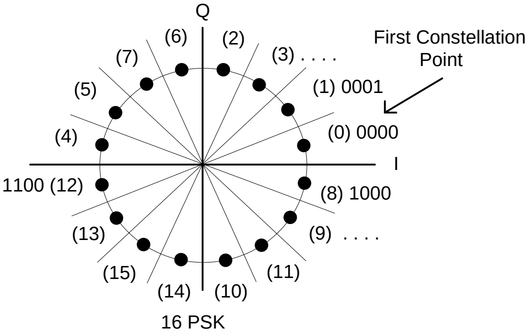


The default first constellation point is at π/8 radians.

8.9.11.4 16-PSK

This option performs *16 phase shift keying* (16-PSK) modulation of the input signal based on the specified block parameters. In 16-PSK modulation the digital information is transmitted by varying the carrier phase among 16 states equally spaced at π/8 rad increments.

The `16-PSK` block accepts a symbol number {0, 1, 2, ..., 15} as its input and maps it to the constellation point specified in the PSK map file. The following example assumes the default PSK map file (PSK_GRAY.DAT).



8.9.11.5 32-PSK

This option performs *32 phase shift keying* (32-PSK) modulation of the input signal based on the specified block parameters. In 32-PSK modulation the digital information is transmitted by varying the carrier phase among 32 states equally spaced at π/16 rad increments.

The 32-PSK block accepts a symbol number {0, 1, 2, ..., 31} as its input and maps it to the constellation point specified in the PSK map file.

The default first constellation point is at π/32 rad.

8.9.12 QAM/PAM Modulator

This block performs *quadrature amplitude modulation* (QAM) or *pulse amplitude modulation* (PAM), depending on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output. The following constellations are available: 16-QAM, 32-QAM, 64-QAM, 256-QAM, 4-PAM, and 8-PAM.

The QAM/PAM block belongs to the family of digital modulators. This block accepts a symbol number as its input and maps it to the constellation point specified in the QAM/PAM map file.

x_1 = Input symbol number (integer [0 … N -1], where N is the constellation size)

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

y_2 = Unmodulated carrier phase (rad) (optional)

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

$$y_2(t) = 2\pi f_c t + \phi \qquad \begin{aligned} \theta_d(x) &= \text{data phase} \\ \mathrm{A_d}(x) &= \text{data amplitude} \end{aligned}$$

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.

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Constellation Spacing

Indicates the spacing between adjacent constellation points α in volts.

Constellation Rotation

Indicates the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at (α/2, α/2) in the (I, Q) plane for QAM signals and (α/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 correspond 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.

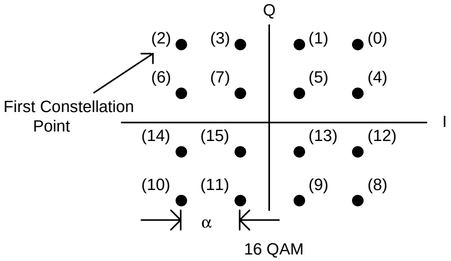
```
QAM Map File: Gray Coded Mapping
8pam
0 1 3 2 6 7 5 4

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.

8.9.12.1 16-QAM

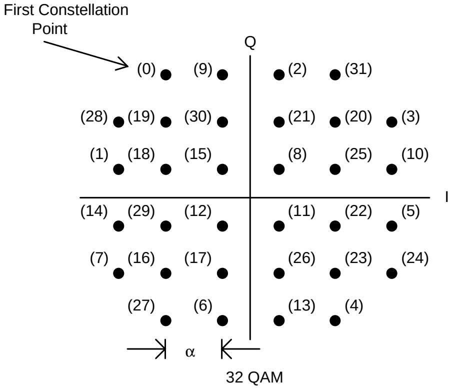
This block performs *16-level quadrature amplitude modulation* (16-QAM) of the input signal based on the selected modulation parameters. In 16-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 16 equally spaced states 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.



8.9.12.2 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).

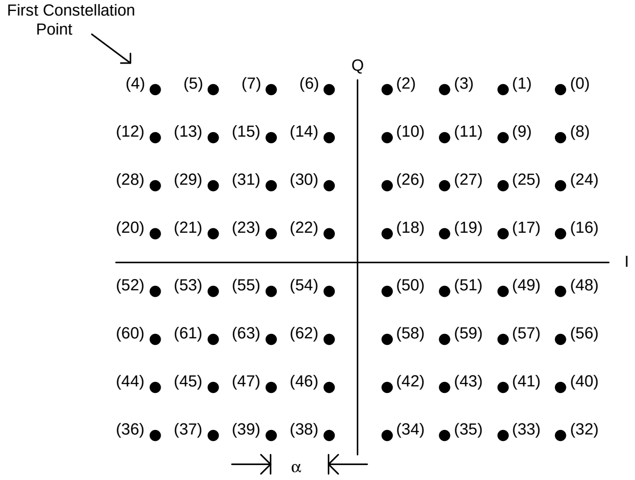


Note: For true V.32, the above constellation must be rotated +45^{o}.

8.9.12.3 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.



8.9.12.4 128-QAM

This block performs *128-level quadrature amplitude modulation* (128-QAM) of the input signal based on the selected modulation parameters. In 128-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 128 distinct states in the (I, Q) plane.

The `128-QAM` block accepts a symbol number (0, 1, 2, ..., 127) as its input and maps it to the constellation point specified in the QAM map file. The default QAM map file does not contain a 128-QAM constellation. A sample 128-QAM map file (QAM128.DAT), which is not Gray encoded, is included for reference.

8.9.12.5 256-QAM

This block performs *256-level quadrature amplitude modulation* (256-QAM) of the input signal based on the selected modulation parameters. In 256-QAM, the digital information is transmitted by varying both the carrier amplitude and phase. The resulting constellation is composed of 256 equally spaced states in the (I, Q) plane.

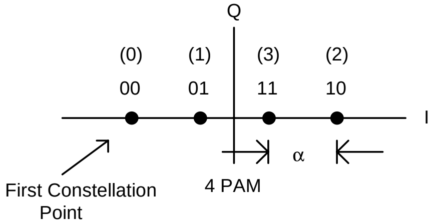
The `256-QAM` block accepts a symbol number {0, 1, 2, ..., 255} as its input and maps it to the constellation point specified in the QAM map file. The default QAM map file (QAM_GRAY.DAT) implements a Gray-coded 256-QAM constellation (not shown).

8.9.12.6 4-PAM

This block performs *four-level pulse amplitude modulation* (4-PAM) of the input signal based on the selected modulation parameters. The modulation scheme is also known as *amplitude shift keying* (ASK). In 4-PAM, the digital information is transmitted by varying the carrier amplitude among four equally spaced amplitude states. The carrier phase remains constant.

The `4-PAM` block accepts as input a symbol number {0, 1, 2, 3} and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM/PAM map file (QAM_GRAY.DAT).

$$A_d = \begin{cases} -3 & x \le 0.5 \\ -1 & 0.5 < x \le 1.5 \\ +3 & 1.5 < x \le 2.5 \\ +1 & x > 2.5 \end{cases} \qquad \theta_d$$

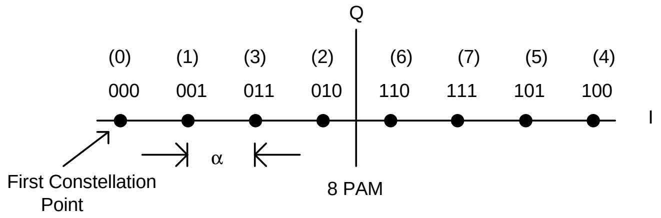


8.9.12.7 8-PAM

This block performs *eight-level pulse amplitude modulation* (8-PAM) of the input signal based on the selected modulation parameters. This modulation scheme is also known as ASK. In 8-PAM, the digital information is transmitted by varying the carrier amplitude among eight equally spaced amplitude states. The carrier phase remains constant.

The `8-PAM` block accepts a symbol number {0, 1, ..., 7} as its input and maps it to the constellation point specified in the QAM map file. The example shown below assumes the default QAM/PAM map file (QAM_GRAY.DAT).

$$A_d = \begin{cases} -7 & x \le 0.5 \\ -5 & 0.5 < x \le 1.5 \\ -1 & 1.5 < x \le 2.5 \\ -3 & x > 2.5 \end{cases} \qquad A_d = \begin{cases} +7 & 3.5 < x \le 4.5 \\ +5 & 4.5 < x \le 5.5 \\ +1 & 5.5 < x \le 6.5 \\ +3 & x > 6.5 \end{cases} \qquad \theta_d$$



8.9.12.8 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.

8.9.13 SQPSK Modulator

This block performs *staggered quadrature phase shift keying* (SQPSK) modulation of the input signal based on the selected modulation parameters. Two versions of this block are provided: one producing a complex output and the other producing a real output.

The `SQPSK` block belongs to the family of digital modulators, and is also known as *offset QPSK* (OQPSK). In SQPSK modulation, the digital information is transmitted by varying the carrier phase among four states equally spaced at π/2 rad increments. The carrier amplitude remains constant. The data on the Q channel input is delayed ½ symbol duration relative to the I channel data. This ensures that at any given time, only one of the two data channels (I or Q) may undergo a transition. As a result, the modulated signal phase never changes by more than π/2 rad, and the modulated spectrum exhibits lower sidelobes than ordinary QPSK. Since the Q data is delayed ½a symbol (within the block), it is preferable to select a simulation step size that yields an even number of simulation steps per symbol period.

The `SQPSK` block accepts two binary data streams as its input (I and Q data, respectively) and uses Gray encoding in its mapping. A real-to-integer conversion (rounding) is performed on the inputs. To demodulate SQPSK, a regular QPSK detector can be used.

x_1 = I channel data (binary)

x_2 = Q channel data (binary)

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

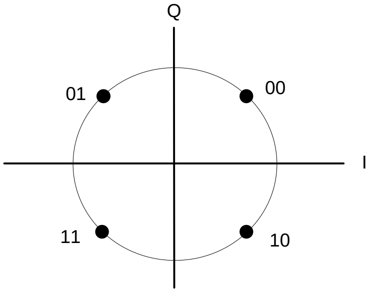
y_2 = Unmodulated carrier phase (rad) (optional)

$$y_1(t) = Ae^{j(2\pi f_c t + \theta_d + \phi)} \qquad \phi = \frac{\pi\theta_r}{180}$$

$$y_2(t) = 2\pi f_c t + \phi \qquad \theta_d(x_1, x_2) = \text{data phase}$$

T = symbol duration

$$\theta_d = \begin{cases} \pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 \le 0.5 \\ 3\pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 > 0.5 \\ -\pi/4 & x_1 > 0.5 \text{ and } \tilde{x}_2 \le 0.5 \\ -3\pi/4 & x_1 \le 0.5 \text{ and } \tilde{x}_2 > 0.5 \end{cases} \qquad \text{where} \quad \tilde{x}_2 = x_2(t - T/2)$$



SQPSK

Carrier Frequency

Indicates the output carrier frequency f_c in hertz. It may be set to 0 when working in complex envelope representation.

Amplitude

Specifies the carrier single-sided peak amplitude *A* in volts.

Constellation Rotation

Specifies the constellation rotation θ_r in degrees from the default setting. Positive values correspond to counterclockwise rotation. The default first constellation point is at π/4 rad.

Phase Output Mode

Unwrapped

Specifies the unwrapped output mode for the unmodulated carrier phase.

Wrapped [0, 2pi]

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

8.10 Multirate Support category

Blocks in the Multirate Support category include `Clock Edge`, `Clock Extend`, and `Interpolator`.

8.10.1 Clock Edge

This block is used to “latch” and propagate a clock pulse generated from a slower rate “locally stepped compound block”. Without the use of this block, it’s possible for a clock’s “high” state to span multiple simulation time steps across such a boundary, resulting in the erroneous perception of multiple clock pulses having occurred.

This block operates by forcing a clock’s “high” state to last for only one simulation sample irrespective of the actual duration of the input pulse. Once a high state is detected, the occurrence of a low state is required to reset the latch. This block should be placed just outside of the slower locally stepped compound block (output side).

x = Input clock signal

y = Output clock signal

8.10.2 Clock Extend

This block is used to propagate a clock pulse into a slower rate “locally stepped compound block”. Without the use of this block, it’s possible for clock pulses to be “dropped” across such a boundary.

This block operates by extending a clock’s “high” state across multiple simulation steps so that the slower running block can detect it. For this block to operate properly, it’s important to accurately specify the local clock rate of the compound block that follows. This block should be placed just outside of the slower locally stepped compound block (input side).

x = Input clock signal

y = Output clock signal

Local Sim Rate of Next Block

Specifies the simulation rate in hertz of the locally stepped compound block that follows.

8.10.3 Interpolator

This block provides linear waveform interpolation for a signal passing between differing sample rate regions. This block is best suited for analog waveforms and is especially useful when the sample rates in question are not integer multiples of each other (i.e. the sample times of the two blocks do not generally coincide).

Use of this block avoids the “staircase” effect that can otherwise occur on analog signals when crossing from one sample rate region to another. Such an effect occurs when two or more sample times at the faster simulation rate occur between sample times at the lower rate. This block also provides added accuracy when going.

This block introduces a single time step delay to the output at the sample rate of the input signal , e.g. the main sim rate for a High-to-Low case and the local rate for a Low-to-High case.

x = Input signal

y = Output interpolated signal

Local Sim Rate of Next Block

Specifies the simulation rate in hertz of the locally stepped compound block that follows (high to low case) or precedes (low to high case) the `Interpolator` block.

Mode

High to Low

Use this setting when going from a higher sample rate environment to a lower sample rate environment.

Low to High

Use this setting when going from a lower sample rate environment to a higher sample rate environment.

8.11 Operator category

Blocks in the Operators category include `A/D Converter`, `Compander`, `Complex FFT/IFFT`, `Conversions`, `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.`

8.11.1 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).

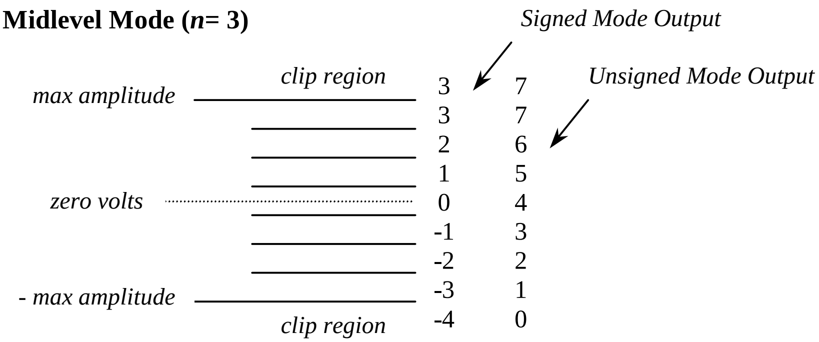
The input signal is assumed to vary over a range of +/- *Max Amplitude* volts centered about a zero volts level. To sample a non-zero centered waveform, please add an apprpriate offset to the input signal.

Two options exist for handling conversion of the zero input level: midlevel and offset. In the *midlevel* mode any input value in the range [-0.5Δq, 0.5Δq] is converted to level 0 (assuming signed output mode), where Δq is a quantization level. In the *offset* mode, a positive value in the range [0, Δq] is converted to level 0, while negative values in the range [-Δq, 0-] are converted to level -1 (assuming signed output mode). The distinction between the two modes is further illustrated below for clarity.

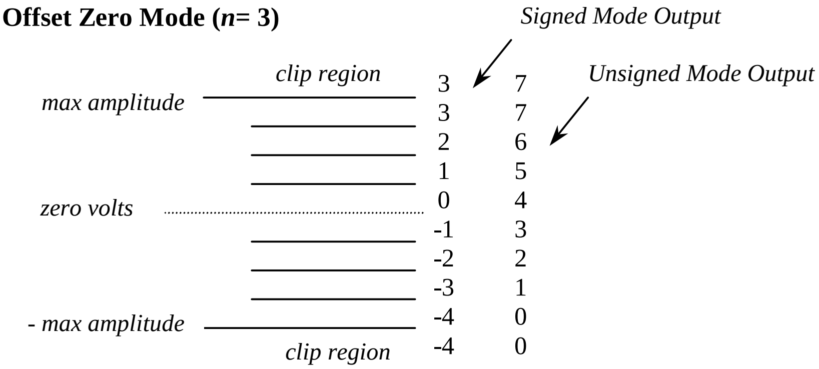
x = Input signal

y = Quantized integer output

Midlevel Mode (*n*= 3)



Offset Zero Mode (*n*= 3)



Number of Bits

Specifies the number of bits *n* of quantizer resolution.

Max Amplitude

Indicates the signal amplitude in volts corresponding to the positive and negative clip input levels.

Zero Level

Midlevel

A 0 input voltage level 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 range [-maxAmp, maxAmp] is divided into (2^n - 1) levels, e.g. 255 levels for an 8 bit ADC. Please refer to the previous figure for additional clarification.

Offset

A 0 input voltage level 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 range [-maxAmp, maxAmp] is divided into 2^n levels, e.g. 256 levels for an 8 bit ADC. Please refer to the previous figure for additional clarification.

Output Mode

Unsigned

Indicates that the quantized integer output will be in the range [0, 2^n- 1].

Signed

Indicates that the quantized integer output will be in the range [-$2^{(n-1)}$, $2^{(n-1)}$ - 1].

8.11.2 Compander

This block implements signal companding and its inverse. Companding refers to a process of nonlinear amplitude compression usually performed prior to A/D quantization of a signal. This process is employed when it is desirable to obtain a nonlinear quantization of the original signal. You can specify either μ-law or *A*-law companding, as described below. A standard value used in μ-law companding is μ=255, while for *A*-law companding *A*=87.56 is often used. Since the compander expects values in the range [-1, 1], the input signal is normalized by the Max Value parameter.

x = Input signal

y = Companded output

In the equations below $x = \frac{x_1}{x_{\max}}$

$$y = \operatorname{sgn}(x) \cdot \frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad |x| \le 1 \qquad \mu - \text{law companding}$$

$$y = \begin{cases} \operatorname{sgn}(x) \cdot \dfrac{1+\ln A|x|}{1+\ln A} & \dfrac{1}{A} < |x \le 1| \\ \operatorname{sgn}(x) \cdot \dfrac{A|x|}{1+\ln A} & 0 \le |x| \le \dfrac{1}{A} \end{cases} \qquad A - \text{law companding}$$

Compander Mode

Compress

Indicates that the block is operating in compression mode.

Expand

Indicates that the block is operating in expansion mode. This operation is the inverse of compression.

Compander Type

μ-Law

Indicates μ-law companding.

A-Law

Indicates *A*-law companding.

Max Value

Indicates the maximum allowed magnitude (x_{max}) of the input signal. This value is used to normalize the input to the range [-1, 1]. When in expand mode, Max Value scales the output.

A Value

Indicates the value of *A* for *A*-law companding. The value of *A* greater than or equal to 1. A value of *A* equal to 1 indicates no compression.

μ Value

Indicates the value of μ for μ-law companding. The value of μ must be greater than 0.

8.11.3 Complex Exponential

This block outputs the complex exponential of the input signal.

$$y(t) = e^{j\,x(t)}$$

x = Phase angle (rad)

y = Complex output [Re, Im]

8.11.4 Complex FFT/IFFT

This block computes either the forward *fast Fourier transform* (FFT) or inverse IFFT of the input complex signal. You may select from a variety of window functions in performing the FFT. The *N* point FFT/IFFT operation is single shot and is started by an external trigger. Results are viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line and *x*-axis output (either frequency or time) are provided for driving the `plot` block. FFT results are presented over the range [-*fs*/2, *fs*/2], where *fs* is the simulation sampling rate in hertz.

In order to view the entire result, the simulation time range should extend at least 2x*N* simulation steps from where the external trigger is applied. When the output is represented in dBm or dBm/Hz, the output is limited to -300 dBm.

x_1 = Input trigger (high > 0.5)

x_2 = Real part (or magnitude) of complex input

x_3 = Imaginary part (or phase) of complex input

y_1 = Output trigger (0 or 1) for plot block

y_2 = Real part (or magnitude) of complex output

y_3 = Imaginary part (or phase) of complex output

y_4 = *X*-axis drive signal: Frequency (*x*Hz) in FFT mode or time (seconds) in IFFT mode

FFT Mode

Forward FFT

Inverse FFT

Specifies either an FFT or IFFT operation.

FFT Window Type

Selects the desired window function to be used in computing the FFT. This parameter is not applicable to the IFFT operation.

FFT Representation

Mag / Phase

Indicates that the FFT is represented by magnitude and phase components.

Real / Imaginary

Indicates that the FFT is represented by real and imaginary components.

FFT Size

Specifies the size of the FFT or IFFT operation. Valid range is 8 to 8 Meg.

Unwrap Phase

Specifies that the computed phase response should be unwrapped. When not selected, the output phase is restricted to [-π, +π].

Remove Linear Phase

Specifies that linear phase, based on the provided delay input, should be removed from the FFT phase response result.

Delay

Specifies the delay, in seconds, corresponding to the amount of linear phase to be removed from the phase result. This setting only applies when the Remove Linear Phase option is selected.

Number of FFT Averages

Specifies the number of consecutive FFTs (each of *N* points), which are averaged before producing the output spectrum.

Output Freq. Units (FFT only)

Specifies the frequency units for the *x*-axis drive signal (y_4). Choices include hertz, kilohertz, megahertz, and gigahertz.

Output Mode (FFT only)

Standard

Indicates unscaled FFT output. This mode is most practical when viewing a filter response, as a level of 1 corresponds to unity gain.

Power Spectrum

Indicates that the FFT result represents the signal’s power spectrum in dBm. The output value represents total energy per bin. A load of 50 Ohms is assumed.

Spectral Density

Indicates that the FFT result represents the signal’s power spectral density in dBm/hertz. A load of 50 Ohms is assumed.

8.11.5 Conversions

This block implements many common conversions. The desired conversion is selected by choosing the appropriate radio button in the block's setup dialog box. Two versions of this block are available, one for scalar signals and another for vector inputs.

Decibels to Power

The block converts a decibel input to a power value.

x = Input value in decibels

y = Power output value

$$y = 10^{(x/10)}$$

Decibels to Real

The block converts a decibel input to a real number.

x = Input value in decibels

y = Output real value

$$y = 10^{(x/20)}$$

Degrees to Radians

The block converts from degrees to radians.

x = Input value in degrees

y = Output value in radians

$$y = \frac{x\pi}{180}$$

Hertz to Rad/sec

The block converts from hertz to radians/second.

x = Input value in hertz

y = Output in radians/second

$$y = 2\pi x$$

Mag/Phase to Real/Im

The block converts a complex input represented in magnitude/phase format to real/imaginary format. This option is not available in the vector version of the block.

x_1 = Input complex magnitude

x_2 = Input complex phase

y_1 = Real part of complex output

y_2 = Imaginary part of complex output

$$y_1 = x_1 \cos(x_2) \qquad y_2 = x_1 \sin(x_2)$$

Power to Decibels

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

x = Input power value

y = Output in decibels

$$y = 10\log(x) \qquad x > 0$$

Radians to Degrees

The block converts from radians to degrees.

x = Input value in radians

y = Output value in degrees

$$y = \frac{180x}{\pi}$$

Rad/sec to Hertz

The block converts from radians/second to hertz.

x = Input value in radians/second

y = Output in hertz

$$y = \frac{x}{2\pi}$$

Real to dB

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

x = Input real value

y = Output in decibels

$$y = 20\log(x) \qquad x > 0$$

Real/Im to Mag/Phase

The block converts a complex input represented in real/imaginary format to magnitude/phase format. A four-quadrant arctangent function is used, which returns values in the range of -π to π. If both x_1 and x_2 are 0, zero phase is returned. This option is not available in the vector version of the block.

x_1 = Real part of complex input

x_2 = Imaginary part of complex input

y_1 = Complex magnitude

y_2 = Complex phase

$$y_1 = \sqrt{(x_1)^2 + (x_2)^2} \qquad y_2 = \operatorname{atan}(x_2, x_1)$$

8.11.6 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 = Complex input signal [Re, Im]

y = Delayed version of complex input signal [Re, Im]

$$y_{[n]} = x_{[n-k]} \qquad k = \text{delay size} \qquad n = \text{simulation step index}$$

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.

8.11.7 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.

x = Input signal

y = Delayed version of input signal

$y_{[n]} = x_{[n-k]}$ $k = \text{delay size}$ $n = \text{simulation step index}$

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.

8.11.8 Gain (dB)

This block lets you specify a gain value in decibels.

x = Input signal

y = Scaled output value

$$y = x \cdot 10^{(gain/20)}$$

Gain

Indicates the gain value in decibels. This value may be positive or negative.

8.11.9 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.

8.11.10 Integrate & Dump (Complex or Real)

These blocks implement integrate and dump operations on the input signal. The input signal is continuously integrated and the integral output is periodically dumped and reset to a specified value. The dump rate can be specified internally or through an external clock.

When one of these blocks is used to demodulate a baseband phase modulated signal, it should be followed by the appropriate detector block for the modulation used. These blocks accept and output either a real signal or a complex signal depending on the selected version of the block.

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

x_2 = Optional external clock (0, 1) (impulse train)

y_1 = Integrator output ([Re, Im] for complex)

y_2 = Output clock (impulse train)

Reset Value

Indicates the value to which the integral resets when dumped. This parameter is also used as the integrator initial condition at simulation start.

Scale Factor

Indicates the output scaling factor. This parameter is usually set to the inverse of the data rate so as to cancel out the integrator Δt scaling.

Dump Timing

Internal

Indicates that the Dump Rate and Initial Delay (prior to the first dump) are specified internally.

External

Indicates that an outside dump clock is supplied at the clock input port (x_2).

Suppress First Output Clock

When selected, the first output clock pulse is suppressed when in external clock mode. The first external clock pulse still dumps and resets the internal I&D buffer. This option makes it easier to achieve frame synchronization in some diagrams by making the first output pulse from the I&D block correspond with the first valid dumped value.

Dump Rate

Specifies the block's dump rate in hertz. This option is only available when internal dump timing mode is selected.

Initial Delay

Specifies the initial delay in seconds to the beginning of the first integration period. The first dump event will occur at t = Initial Delay + Dump Period. This parameter is only available when internal dump timing mode is selected.

Output Mode

Continuous

Indicates that the output is the instantaneous integrator output.

Held

Indicates that the output is held constant between dump clock pulses.

Integration Method

Euler

Specifies the Euler integration method (forward difference).

Trapezoidal

Specifies the trapezoidal integration method.

Backward Diff.

Specifies the backwards difference integration method.

Max Index

This block returns the index of the largest input signal. The block can be configured to accept up to 16 inputs. A typical application of this block is in the creation of M-ary decision circuits (e.g. detection of MFSK). Should two or more inputs share the largest value, the output index will be that of the lowest input connection in the set.

x_1 = Input #1

x_2 = Input #2

…

x_n = Input #*N*

y = Output index value (either {0,1,.., *N*-1} or {1, 2, .. *N*})

Number of Inputs

Specifies the number of input connections *N*. Valid range is 2 to 16.

Index Mode

Start at Zero

Specifies the output range to be {0,1,.., *N*-1}.

Start at One

Specifies the output range to be {1, 2, .. *N*}.

8.11.11 Modulo

This block performs either real-valued or integer modulo operations. The output represents the remainder after integer division of the input value by the specified modulo value. The result will fall in the range [0, mod val) when specifying a positive modulo value, and (mod val, 0] when specifying a negative modulo value.

When in integer modulo mode, the input value undergoes a tolerance adjustment and is then truncated to an integer prior to the modulo operation. Also, the input value and the modulo value must fall within the accepted range for signed long variables.

x = Input signal

y = Remainder from modulo operation

Modulo

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

Modulo Mode

Integer

Indicates integer modulo operation.

Real

Indicates real modulo operation.

Tolerance

Indicates the value added to the input signal before performing the modulo operation. This step is necessary due to possible floating-point rounding errors. For example, after an arbitrary operation, an expected integer value of 5 might be represented in floating point notation as 4.99999...9.

This value is only applicable to integer mode.

The default tolerance value is 1e-6.

8.11.12 Oscilloscope

This block emulates the use of a two input channel oscilloscope. You can specify either input as the trigger source, specify falling or rising edge triggering, and set the trigger threshold level. The overall time span is also user defined. The output is viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line, the *A* and *B* channel traces, and time axis outputs are provided for driving the `plot` block.

Once enabled by an external start pulse, the `Oscilloscope` block monitors the input signal for a threshold crossing, i.e. a trigger event. Once triggered (and the specified trigger delay has elapsed) it reads in *N* data points on both channels, where *N* is based on the desired time span, and then displays both traces all at once as a vector. Upon completing each trace, the `Oscilloscope` block resets itself and awaits the next trigger event before repeating the above cycle.

x_1 = Start pulse (high > 0.5)

x_2 = *A* Channel input

x_3 = *B* Channel input

y_1 = Output trigger (0 or 1) for plot block

y_2 = *A* Channel vector trace output

y_3 = *B* Channel vector trace output

y_4 = Time axis for plot's *X*-axis

Trigger Channel

A Channel

Specifies Channel *A* is to be used for triggering the display.

B Channel

Specifies Channel *B* is to be used for triggering the display.

Triggering Mode

Rising Edge

Forces the displayed trace to start following a rising crossing of the specified threshold level.

Falling Edge

Forces the displayed trace to start following a falling crossing of the specified threshold level.

Time Span

Specifies the desired time span in seconds for the display.

Threshold

Specifies the voltage level (in volts) used for triggering the display.

Trigger Delay

Specifies the time delay (in seconds) to wait after a trigger event before starting the trace.

Number of Sweeps per Update

Specifies the number of traces to overlay on the display. Note: to avoid the appearance of retrace lines in the display, the Plot's “Line Type” should be configured for “points” mode when this setting is > 1.

8.11.13 Phase Rotate

This block multiplies the complex input by the complex exponential $e^{j\phi}$, which results in a phase rotation of ϕ radians.

x_1 = Complex input signal [Re, Im]

x_2 = Rotation angle (radians)

y = Complex output signal [Re, Im]

$$y = x_1(\cos x_2 + j \sin x_2)$$

8.11.14 Phase Unwrap

This block performs a phase unwrapping operation on the input signal which is assumed to be represented in the range of [-180, +180] degrees or [-π, +π] radians. The block operates by comparing the input phase value to a weighted average of the previous several points. When a phase jump in excess of 180 degrees is detected, a phase wrap is declared on the input, and a value of 360 degrees is added to (or subtracted from) the output waveform. For this block to work properly, the phase variations from point to point should not be excessive (<45 degrees).

x = Input phase signal

y = Unwrapped phase signal

Units

Degrees

Indicates the input phase signal is specified in *degrees*.

Radians

Indicates the input phase signal is specified in radians.

8.11.15 Polynomial

This block computes *g(x)* for *g( )* up to a fifth order polynomial.

x = Input signal

y = Output signal *g(x)*

$$g(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

x^0 Coefficient

Indicates the coefficient of the x^0 term (*f* in the above formula).

x^1 Coefficient

Indicates the coefficient of the x^1 term (*e* in the above formula).

x^2 Coefficient

Indicates the coefficient of the x^2 term (*d* in the above formula).

x^3 Coefficient

Indicates the coefficient of the x^3 term (*c* in the above formula).

x^4 Coefficient

Indicates the coefficient of the x^4 term (*b* in the above formula).

x^5 Coefficient

Indicates the coefficient of the x^5 term (*a* in the above formula).

Show Formula

When selected, the polynomial formula is displayed as part of the block's name.

Decimal Digits

Specifies the number of decimal digits to use for the polynomial coefficients in the displayed block name (if the Show Formula checkbox is selected).

8.11.16 Spectrum (Complex or Real)

This block outputs the complex power spectrum of the input signal. The spectrum can be continuously updated (once started by the external trigger) or produced at user-defined intervals (again, using the external trigger). Results are viewed using a `plot` block configured in XY mode with an external trigger. An output trigger line and x-axis output are provided for driving the `plot` block.

Two versions of this block exist: one complex and the other real. The real version of the block only outputs the positive side of the spectrum (since it's mirror imaged), but includes all signal energy, including that from the negative side of the spectrum. This is equivalent to doubling the FFT result at all points except dc and *fs*/2. So, for example, the complex spectrum of a -10 dBm real sinusoid of frequency f_o results in two peaks of -13 dBm at -f_o and +f_o , while the real spectrum results in a single peak of -10 dBm at f_o.

Once triggered, the `Spectrum` block reads in *N* data points, perform an *N* point FFT, and then outputs the FFT result all at once as a vector. If you have selected to average multiple FFTs, then an averaging step is also performed prior to outputting the result. You may also select from a variety of window functions when performing the underlying FFT. Once completed, the `Spectrum` block either immediately reads in the next *N* data points (continuous mode) or waits until the next input trigger before repeating the above cycle.

Spectral results are presented over the range [-*fs*/2, *fs*/2] for the complex version, and over the range of [0, *fs*/2] for the real version, where *fs* is the simulation sampling rate in *hertz*. The output spectrum can be output in watts, dBm, or dBm/Hz. When in dBm or dBm/Hz mode, the output is limited to -300 dBm.

Note: When using this block to measure the power level of individual tones it is recommended to use the “dBm” units setting. Conversely, when wanting to measure noise levels, it is recommended to use the “dBm/Hz” units setting.

x_1 = Input trigger (high > 0.5)

x_2 = Complex input (real for real version of block)

y_1 = Output trigger (0 or 1) for `plot` block

y_2 = Magnitude (or Real) output

y_3 = Phase (or Imaginary) output

y_4 = Frequency (*x*Hz) for plot’s x-axis

Trigger Mode

Continuous

Once a result has been output, the block immediately reads in more data.

Triggered

Once a result has been output, the block will wait for a new trigger before reading in new data.

FFT Window Type

Selects the desired window function to be used in computing the underlying FFT.

Spectral Output

Mag / Phase

Indicates that the result is represented by magnitude and phase components.

Real / Imaginary

Indicates that the result is represented by real and imaginary components.

FFT Size

Specifies the size *N* of the underlying FFT computation. Valid range is from 8 to 8 Meg.

Unwrap Phase

Specifies that the computed phase output should be unwrapped. When not selected, the output phase is restricted to [-π, +π].

Remove Linear Phase

Specifies that linear phase, based on the provided delay input, should be removed from the phase output.

Delay

Specifies the delay, in seconds, corresponding to the amount of linear phase to be removed from the phase result. This setting only applies when the Remove Linear Phase option is selected.

Number of FFT Averages

Specifies the number of consecutive power spectra (each of *N* points), which are averaged before producing the final result.

Output Freq. Units

Specifies the frequency units for the *x*-axis drive signal (y_4). Choices include hertz, kilohertz, megahertz, and gigahertz.

Power Spectrum Units

Watts

The spectral output is represented in Watts.

dBm

The spectral output is represented in dBm. The output values represent total energy per bin.

dBm/Hz

The spectral output is represented as a power spectral density in dBm/Hz.

dBm/MHz

The spectral output is represented as a power spectral density in dBm/MHz.

Load

1 Ohm

The spectral output is referenced to a 1 ohm load.

50 Ohms

The spectral output is referenced to a 50 ohms load.

8.11.17 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 inpulse train (zero padded).

x = Input signal

y = Sampled output

Decimate by

Indicates the decimation factor. If, for example, it is set to 5, then every fifth simulation step is sampled and posted to the output port.

Offset

Indicates the offset from the first simulation step for applying the sampling. When set to 0, the sampling starts with the first simulation step.

Output Mode

Held Output

Indicates that the output signal is held constant (last decimated value) between sampled points.

Pulsed Output

Indicates that the output signal is 0 between sampled points.

8.11.18 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 both real/imaginary format or magnitude/phase for the FFT data.

The `Vector FFT` block expects two input vectors at all times. If only real data is to be used, a 0 input vector of size *N* must be supplied as the imaginary input (this can be done using the `Vector Constant` block). Not all outputs have to be connected.

x_1 = Frame clock (high > 0.5) (impulse)

x_2 = Input signal vector (real or magnitude component; size *N*)

x_3 = Input signal vector (imaginary or phase component; size N)

y_1 = Frame clock (impulse)

y_2 = Output signal vector (real or magnitude component; size N)

y_3 = Output signal vector (imaginary or phase component; size N)

y_4 = Output signal vector (frequency or time output vector; size N)

FFT Mode

Forward FFT

Inverse FFT

Specifies either an FFT or IFFT operation.

FFT Window Type

Selects the desired window function to be used in computing the FFT. This parameter is not applicable to the IFFT operation.

FFT Representation

Mag / Phase

Indicates that the FFT data is represented by magnitude and phase components.

Real / Imaginary

Indicates that the FFT data is represented by real and imaginary components.

FFT Size

Specifies the size *N* of the FFT or IFFT operation. The valid range is from 8 to 8 Meg.

Output Freq. Units (FFT only)

Specifies the frequency units for the x-axis drive signal (y4). The choices include hertz, kilohertz, megahertz, and gigahertz.

Use Simulation Time Step

When checked, the time step (or frequency resolution) of the input data is assumed to be the current simulation settings.

Sampling Frequency

Specifies an alternate sampling frequency to associate with the vector data, when use of the current simulation time step value is not desired. This parameter only affects the scaling of the Frequency or Time output vector (y4), which is provided primarily for plotting purposes (x-axis drive signal).

8.12 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)`.

8.12.1 Charge Pump

This block implements a first order active lag-lead loop filter for use in a second order digital PLL. The UP and DOWN inputs are assumed to be binary and represent the output error signals from a type-3 or type-4 digital phase/frequency detector. The output is used to drive a `VCO` block.

It is important to ensure that the Phase Detector Gain and VCO Gain parameters actually match the values used in the connected blocks. The approximate PLL noise loop bandwidth, as well as the computed transfer function coefficients (tau1, tau2), are displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency.

x_1 = UP error signal

x_2 = DOWN error signal

y = Output signal (VCO drive)

Charge Pump transfer function (Laplace format):

$$F(s) = \frac{s\tau_2 + 1}{s\tau_1}$$

Natural Frequency

Indicates the natural frequency (ω_n) of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase or frequency step.

Loop Bandwidth

Indicates the approximate noise loop bandwidth of the PLL in hertz. This represents the double sided bandwidth (3 dB) over which the PLL is able to follow frequency variations in the input signal. The loop bandwidth parameter provides an alternative method to specifying the PLL natural frequency for describing the response time of the PLL.

Damping Factor

Specifies the damping factor of the PLL. The default value is 0.7071068, which yields a critically damped second order loop. As this value is decreased, the PLL response becomes underdamped. Increasing the value creates an overdamped response.

VCO Gain

Indicates the gain (K_o) of the connected VCO in (radians/sec)/volt.

Detector Gain

Indicates the gain (K_d) of the connected Phase Detector in radians/volt. 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/π and approximately 1/2π 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.

8.12.2 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 Commsim's compound blocks, including the PLLs and Costas Loop.

A `Loop Filter (2nd Order PLL)` block is able to track a phase or frequency step, but not Doppler rate. For this, you need to use the `Loop Filter (3rd Order PLL)` block, as described in the next section. Because `Loop Filter (2nd Order PLL)` blocks are more stable than `Loop Filter (3rd Order PLL),` you should use a second order PLL unless Doppler rate is a factor.

It is important to ensure that the Detector Gain and VCO Gain parameters actually match the values used in the connected blocks. The PLL noise loop bandwidth, as well as the computed transfer function coefficients (tau1, tau2), are displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency. Also, for good PLL operation when using the passive loop type, the loop gain (K_oK_d) should be much larger than the natural frequency (ω_n).

x = Input drive signal

y = Output signal (VCO drive)

Loop filter transfer function (Laplace format):

$$F(s) = \frac{s\tau_2 + 1}{s\tau_1} \qquad \text{(active)} \qquad\qquad F(s) = \frac{s\tau_2 + 1}{s\tau_1 + 1} \qquad \text{(passive)}$$

Natural Frequency

Indicates the natural frequency (ω_n) of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase or frequency step.

Loop Bandwidth

Indicates the approximate noise loop bandwidth of the PLL in hertz. This represents the double sided bandwidth (3 dB) over which the PLL is able to follow frequency variations in the input signal. The loop bandwidth parameter provides an alternative method to specifying the PLL natural frequency for describing the response time of the PLL.

Damping Factor

Specifies the damping factor of the PLL. The default value is 0.7071068, which yields a critically damped second order loop. As this value is decreased, the PLL response becomes underdamped. Increasing the value creates an overdamped response.

VCO Gain

Indicates the gain K_o of the connected VCO in (radians/second)/volt.

Detector Gain

Indicates the gain K_d of the connected phase detector in radians/volt. 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(θ_ε) $\simeq$ θ_ε 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.

8.12.3 Loop Filter (3rd Order PLL)

This block implements a Wiener Optimal second order loop filter for use in a third order PLL. A third order PLL is able to track a frequency ramp (Doppler rate). The input is assumed to be an error signal from a phase detector. The output is typically used to drive a VCO block. Both input and output are expressed in volts.

It is important to ensure that the Phase Detector Gain and VCO Gain parameters match the values used in the connected blocks. The PLL Noise Loop Bandwidth is displayed in the dialog box based on the values of the other parameters. In order for the PLL to operate properly, the simulation sampling frequency must be much larger than the PLL natural frequency.

x = Input drive signal

y = Output signal (VCO drive)

Loop filter transfer function (Laplace format):

$$F(s) = \frac{\omega_3^3 + 2\omega_3^2 s + 2\omega_3 s^2}{K_o K_d s^2}$$

Natural Frequency

Indicates the natural frequency ω_3 of the PLL in radians/second. This parameter is not to be confused with the VCO center frequency. The natural frequency determines the response time of the PLL to a phase, frequency, or frequency ramp step.

VCO Gain

Indicates the gain K_o of the connected VCO in (radians/second)/volt.

Detector Gain

Indicates the gain K_d of the connected phase detector in radians/volt. A phase detector gain value of 1 (default) indicates that an input error signal of 0.1 V represents a 0.1 rad phase error. Commonly used phase detectors actually output $\sin(\theta_\varepsilon) \simeq \theta_\varepsilon$ for small error angles only. The gain value is based on assuming a small error angle.

Update

This button updates the Noise Bandwidth value based on any change made to the Natural Frequency parameter. The values are not permanently stored until you click on the OK button.

8.12.4 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.

x_1 = Reference input

x_2 = VCO input

y = Error signal [0, 1]

Threshold

Specifies the voltage level above which the input is considered high.

8.12.5 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.

x_1 = Reference input

x_2 = VCO input

y_1 = UP error signal [0, 1]

y_2 = DOWN error signal [0, 1]

Initial State

Specifies the initial state of the phase detector internal flip-flop.

High Threshold

Specifies the voltage level above which a rising input becomes high.

Low Threshold

Specifies the voltage level below which a falling input becomes low.

Edge Mode

Rising Edge

Specifies that the phase detector operates using rising clock edges.

Falling Edge

Specifies that the phase detector operates using falling clock edges.

8.12.6 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.

x_1 = Reference input

x_2 = VCO input

y_1 = UP error signal [0, 1]

y_2 = DOWN error signal [0, 1]

High Threshold

Specifies the voltage level above which a rising input becomes high.

Low Threshold

Specifies the voltage level below which a falling input becomes low.

8.13 RF category

Blocks in the RF category include `Amplifier`, `Coupler`, `Double Balanced Mixer`, `Switch`, `Splitter/Combiner`, and `Variable Attenuator.`

8.13.1 Amplifier

This block implements a nonlinear RF amplifier. Block parameters include the amplifier small signal gain, the 1 dB compression point, second and third 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. 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. Once saturation is reached, the amplifier output remains constant.

x = Input signal

y = Amplifier output signal

$$y = ax + bx^2 + cx^3 + dx^4 + ex^5 + noise \qquad (\textit{until saturation})$$

Gain

Indicates the small signal gain of the amplifier in decibels.

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 intermods would equal the power in the fundamental. The value of IP2 is typically 20 ~ 25 dB above the 1 dB compression point.

Output Bias

Positive

Specifies that the output dc bias due to the IP2 term is positive.

Negative

Specifies that the output dc bias due to the IP2 term is negative.

IP3

Specifies the theoretical output 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.

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.

Noise Figure

When the Add Noise option is selected, specifies the equivalent input noise figure of the amplifier in decibels.

Add Noise

When this option is selected, white noise is added to the output according to the specified Noise Figure and Gain.

8.13.2 Attenuator

This block implements a passive RF attenuator. Block parameters include the attenuator loss in decibels and the physical temperature of the device. The `Attenuator` block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x = Input signal

y = Attenuated output signal

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

Loss

Indicates the loss of the device in decibels. This parameter is specified as a positive value.

Physical Temperature

When the Add Noise option is selected, specifies the physical temperature of the attenuator in degrees Kelvin. The default value is 290 K.

Add Noise

When this option is selected, white noise is added to the output according to the specified Loss and Physical Temperature parameters.

8.13.3 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.

<u>Input Coupling</u>	**<u>Output Coupling</u>**
x_1 = Primary input signal	x = Input signal
x_2 = Coupled signal	y_1 = Primary output
y = Output signal	y_2 = Coupled output

<u>Input Coupling</u>

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

Output Coupling

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

$$y_2 = x \cdot 10^{(-coupledLoss/20)}$$

Direct Path Loss

Specifies the direct path loss of the coupler in decibels. This parameter is specified as a positive value.

Coupled Loss

Specifies the coupled path loss of the coupler in decibels. This parameter is specified as a positive value.

Noise Figure

When the Add Noise option is selected, specifies the equivalent input noise figure of the coupler in decibels. This value is typically set to the same value as the Direct Path Loss.

Coupling Mode

Input Coupling

Configures the device as an input coupler.

Output Coupling

Configures the device as an output coupler.

Add Noise

When this option is selected, white noise is added to the primary output according to the specified Noise Figure.

8.13.4 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, conversion loss, LO power and harmonic levels, isolation, dc bias, 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 feedthrough at the IF output.

x_1 = RF input

x_2 = LO input

y = IF output

$$y = k \cdot \left(x_1 + ax_1^3 + bx_1^5\right) \cdot \left(x_2 + cx_2^3 + dx_2^5\right) + noise \qquad \textit{(linear region)}$$

$$y = k \cdot \left(\pm\, satAmpOutput\right) \cdot \left(x_2 + cx_2^3 + dx_2^5\right) + noise \qquad \textit{(saturated region)}$$

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.

8.13.5 Splitter/Combiner

This block models an RF splitter or combiner. Block parameters include the splitter mode, number of connections, additional path loss, and noise figure of the device. The block can also be modeled as a noiseless device. A 50 Ohm impedance is assumed.

Splitter Mode	**Combiner Mode**
x = Input signal	x_1 = Input signal #1
y_1 = Output #1	...
...	x_n = Input signal #n
y_n = Output #n	y = Combined output

Split 0

$$y_1 = y_2 = ... = y_n = x\sqrt{10} \cdot^{(-loss/20)} + noise$$

Split 180

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

Combiner

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

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.

8.13.6 Switch

This block models an RF switch. Block parameters include the switch sense, switch loss, isolation, and noise figure of the device. The block can also be modeled as a noiseless device and/or as having perfect isolation. A 50 Ohm impedance is assumed. The path selector input determines which input (or output) is active.

Input Switch

x_1 = Input path selector {1,2,..,n}

x_2 = Input signal #1

...

x_n = Input signal #n

y = Output signal

Output Switch

x_1 = Output path selector {1,2,..,n}

x_2 = Input signal

y_1 = Output signal #1

...

y_n = Output signal #n

Input Switch

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

Output Switch

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

$$y_i = x \cdot 10^{(-isolation/20)} \qquad i \neq sel$$

Number of Connections

Specifies the number of inputs (input switch mode) or outputs (output switch mode) for the device. Valid range is 2 to 8.

Switch Loss

Specifies the loss through the switch in decibels. This parameter is specified as a positive value.

Isolation

Specifies the isolation between inputs or outputs for the switch in decibels. This parameter is specified as a positive value. This parameter is enabled only when Perfect Isolation is not selected.

Noise Figure

Specifies the equivalent input noise figure of the switch in decibels. This value is typically set to the same value as the Switch Loss.

Switch Mode

Input Switch

Indicates that there are N inputs and 1 output.

Output Switch

Indicates that there is 1 input and N outputs.

Perfect Isolation

Assumes perfect isolation between the switch inputs or outputs.

Add Noise

Adds white noise to the primary output according to the specified Noise Figure.

8.13.7 Variable Attenuator

This block implements a passive variable attenuator. The attenuation is controlled via external input, and can therefore be varied during the simulation. Noise is added to the output based on the specified physical temperature and the current loss value. The block can also be modeled as noiseless. A 50 Ohm impedance is assumed.

x_1 = Input signal

x_2 = Loss in dB (≥ 0)

y = Attenuated output signal

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

Phys. Temperature

Specifies the physical temperature of the attenuator in degrees Kelvin when the Add Noise option is selected. The default value is 290 K.

Add Noise

Adds white noise to the output according to the specified Loss and Physical Temperature.

8.14 Signal Sources category

Blocks in the Signal Sources category include `Complex Tone`, `File Data`, `Frequency Sweep`, `Impulse`, `Impulse Train`, `Noise`, `PN Sequence`, `Rectangular Pulses`, `Random Seed`, `Random Symbols`, `Sinusoid`, `Spectral Mask`, `Vector Constant`, `Walsh Sequence`, and `Waveform Generator.`

8.14.1 Complex Tone

This block generates a rotating complex phasor according to the selected block parameters. The internal generator phase (in radians) is also available as an output.

y_1 = Complex output signal [Re, Im]

y_2 = Generator phase (rad) [Optional]

$$y_1(t) = Ae^{j(2\pi f_c t+\phi)} \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Frequency

Indicates the frequency f_c, in hertz, of the complex tone.

Amplitude / Power

Indicates the amplitude A, in volts, of the complex tone, or depending on the Units setting, the complex power of the tone in milli-decibels (50 Ohms).

Initial Phase

Indicates the starting phase θ of the complex tone. This value is specified in degrees.

Units

Volts

Indicates that the signal amplitude is specified in volts.

dBm

Indicates that the signal complex power is specified in milli-decibels (50 Ohms load).

8.14.2 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.

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.

Size of Header

Specifies the size of the file's header. This only applies to binary data files.

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 indicate 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.

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.

8.14.3 Frequency Sweep

This block generates a frequency sweep according to the selected block parameters. The sweep start and stop frequencies, its duration, and the initial phase are specified. After the sweep is completed, a new sweep is started. This block outputs a real signal.

y = Sweep signal output

Start Frequency

Indicates the starting frequency of the sweep signal. This value is specified in hertz.

Stop Frequency

Indicates the stop frequency of the sweep signal. This value is specified in hertz.

Sweep Duration

Indicates the duration of the repeating sweep signal. This value is specified in seconds.

Amplitude

Indicates the amplitude of the sweep signal. This value is specified in volts.

Initial Phase

Indicates the starting phase of the sweep signal (referenced to the sine function). This value is specified in degrees.

8.14.4 Impulse

This block generates a single impulse of the specified magnitude at the specified time. If the specified impulse time is between simulation steps, the impulse will occur at the next simulation step. No pulse will occur if the time specified is prior to the simulation start time.

y = Output signal

Impulse Time

Specifies the occurrence time, in seconds, of the impulse.

Amplitude

Specifies the amplitude of the impulse in volts.

8.14.5 Impulse Train

This block generates an impulse train given the specified parameters, which include the sequence start time, frequency and amplitude.

y = Output pulse train

Pulse Frequency

Specifies the frequency, in hertz, of the impulse train. The inverse of this value is the pulse repetition period.

Amplitude

Specifies, in volts, the peak amplitude of the impulse.

Start Time

Used to specify, in seconds, the start time of the first output pulse.

8.14.6 Noise

This block generates Gaussian random noise according to the specified noise density or noise temperature parameter. The simulation sampling frequency is automatically taken into account when the noise is generated. The block supports both 1 Ohm and 50 Ohms load impedances.

y = Output noise signal

8.14.7 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.

8.14.8 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.

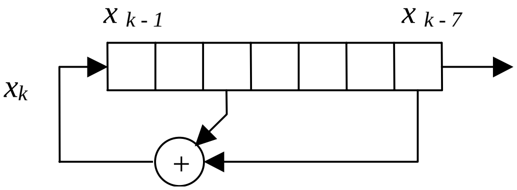
Two different forms are used to describe PN sequences in the literature, and there are also two separate forms for describing the generator polynomial. It's important to note that both forms will produce the same output sequence, but for a given initial state of the shift register they will differ in their output (different locations within the same pattern).

The first form, implemented by Commsim, defines the PN sequence as a recursion formula involving previous sequence outputs. This form uses the modulo 2 sum of the contents of one or more delay stages to create a single feedback value that is inserted in the leftmost location of the shift register. An example is shown below.

Form I Example:

n = 7 PN length = 127

$$x_k = x_{k-7} \oplus x_{k-3} \qquad \text{where } \oplus \text{ represents modulo 2 addition}$$



Generator coefficient = 211 octal (10001001)

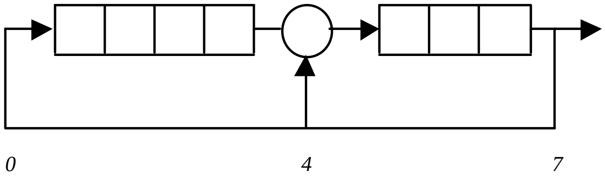
The generator coefficient value is obtained by specifying a "1" for each term where a feedback connection occurs, and writing the result as an octal number. In this form the LSB of the generator coefficient is <u>always</u> set to "1" by convention, and corresponds to the *x* (*k - 0)* term. Since in 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 <u>multiple</u> locations within the shift register and modulo 2 additions are performed with their contents.

Form II Example:

n = 7 PN length = 127

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



Generator polynomial = 221 octal (10010001)

Note that the generator coefficient value is <u>different</u> from before and is obtained by specifying a "1" for each x^k term that appears in the polynomial (with the x^0 term corresponding to the LSB of the generator coefficient value), and writing the result as an octal number. In this case the x^0, x^4, and x^7 terms are used, and so the first, fifth and eighth bits are set in the generator coefficient. The shift register contents represent the "state" of the generator, with the LSB corresponding to the rightmost shift register position.

A *Form II* expression or order *n* can be converted to a *Form I* expression by executing the following steps :

1) Discard the x^n term

2) Convert each remaining x^k term (including the "1", which is really x^0) to a corresponding $x\ (n - k)$ entry in the *Form I* expression.

x = Optional external clock (impulse train)

y_1 = Pseudo-random sequence output (binary or bilevel)

y_2 = Output clock (impulse train)

Shift Register Size

Specifies the order *n* of the PN sequence and determines its length. The sequence length is 2^n - 1 (2^n for augmented sequences), where *n* is the shift register size. Valid range is from 2 to 28.

Sequence Offset

Determines the starting position of the PN sequence. The offset value is used to advance the shift register from its known starting point state. Valid range is from 0 to 2^n - 1, or 32,767.

Initial State

Specifies the initial state of the internal shift register in octal format. The LSB represents the initial output, while the remaining bits represent the next *n*-1 outputs. For example, 56 octal is 101110.

Generator Coeff.

Specifies the generator code for the PN sequence in octal format. The default code for a given order *n* can be obtained by selecting the Use Default Generator Coefficient option. Note that this value is always odd.

Zero Augmented Sequence

Instructs the PN generator to augment the output sequence with an extra 0. The extra 0 is inserted after *n*-1 consecutive 0s are encountered in the output sequence.

Timing

External

Indicates external timing. An external clock must be provided at the x_1 input.

Internal

Indicates internal clock timing. The Bit Rate and Start Time need to be specified.

Output Mode

Bilevel

Indicates that the signal amplitudes associated with output the sequence are {-1, 1}.

Binary

Indicates that the signal amplitudes associated with output the sequence are {0, 1}.

Use Default Generator Coefficient

Loads the default generator coefficient for each order *n*. The default code represents the minimal weight primitive polynomial.

Bit Rate

Specifies the PN sequence bit rate in bits per second. This parameter is only available when internal timing mode is selected.

Start Time

Specifies a start time, in seconds, for the PN sequence. This parameter is only available when internal timing mode is selected.

8.14.9 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.

Seed

Specifies the random number seed to be used by the Comm blocks.

Reset Seed on Auto Restart

Forces the Commsim 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.

8.14.10Random 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.

8.14.11Rectangular 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.

8.14.12 Sinusoid

This block generates a sine or cosine wave specified in hertz according to the selected block parameters. The signal phase is also available in radians.

y_1 = Sine or cosine output (real)

y_2 = Sine generator phase (rad) (optional)

$$y_1(t) = A\sin(2\pi f_c t + \phi) \quad \text{or} \quad A\cos(2\pi f_c t + \phi) \qquad \phi = \frac{\pi\theta}{180}$$

$$y_2(t) = 2\pi f_c t + \phi$$

Frequency

Indicates the sinusoidal frequency f_c in hertz.

Amplitude

Power

Indicates the amplitude A, in volts, of the sinusoidal waveform, or depending on the Units setting, the power of the signal in milli-decibels (50 Ohms).

Initial Phase

Indicates the starting phase θ of the sinusoidal output. This value is specified in degrees.

Units

Volts

Indicates that the signal amplitude is specified in volts.

dBm

Indicates that the signal power is specified in milli-decibels (50 Ohms load).

Output Mode

Cosine

Indicates that the output waveform is a cosine.

Sine

Indicates that the output waveform is a sine.

8.14.13 Spectral Mask

This block outputs a user-defined spectral mask. Its purpose is to let you overlay an FCC mask onto a power spectrum being generated by the simulation environment. The mask is viewed using a `plot` block configured in XY mode with an external trigger. Both the mask information (amplitude) and frequency axis data are output as data vectors of size *N*, where *N* is a user-defined power of two. This block is meant to be used in conjunction with the `Spectrum Analyzer` block.

Once triggered, the `Spectral Mask` block reads in an external frequency vector (of size *N*+1), or an internally generated data set, and uses the specified look-up table to generate a piecewise linear graphical representation of the spectral mask. The look-up table data may be specified over the range of [0, *fs*/2] or [-*fs*/2, *fs*/2], and should include all corners of the mask. The `Spectral Mask` block automatically interpolates (and extrapolates if necessary) the table values to generate the output graph. Data points in the mask file need not be provided in uniform increments, but are required to be in ascending order. Data points should be specified in decibels.

x_1 = Input trigger (high > 0.5) (pulse)

x_2 = Input frequency vector [optional] (size *N*+1)

y_1 = Spectral mask vector for plot block (size *N*+1)

y_2 = Frequency vector for plot block (if needed)

Mask Data Range

[0, fs/2]

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

[-fs/2, +fs/2]

Used when the input file includes data points over the range of [-fs/2, +fs/2].

Frequency Source

External

Specifies that the frequency axis data points are to be provided via the external vector input.

Internal

Specifies that the frequency axis data points are to be computed internally based on the simulation time step value and the value of *N*.

Spectrum Size

Specifies the reference FFT size (*N*) used by the corresponding `Spectrum Analyzer` block. The valid range is 8 to 16,384. The actual vector output will be *N*+1 to account for both endpoints at -*fs*/2 and *fs*/2.

Select File

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

Browse File

Opens the selected mask file using Notepad.

Mask File Path

Specifies the DOS path to the desired spectral mask definition file. Data points are to be provided in increasing order, and are to be arranged in two columns. The second line in the file is used to specify the number of total entries in the file. Thereafter, the first column specifies each data point’s frequency in hertz, and the second column the corresponding mask level in *dB*. The format of the Spectral Mask definition file is further described below:

File header (anything)

number of entries

frequency point #1, mask value #1

frequency point #2, mask value #2

...

Allowed data delimiters are commas, blank space, and tabs. The maximum allowed line length is 100 characters.

8.14.14Vector Constant

This block produces a user specified column vector that outputs the same constant value for all its elements.

y = Output signal vector [size *N*]

Vector Size

Specifies the size *N* of the output column vector (range is 1 to 1,048,576).

Constant Value

Specifies the output value for all the vector elements.

8.14.15VCO (Complex or Real)

This block implements a VCO. Two versions of this block are provided: one producing a complex output and the other producing a real output. When the input drive signal is 0, the VCO block outputs a tone at the specified center frequency. With a non-zero input, the output frequency deviates from the center frequency depending on the magnitude of the drive signal and the specified VCO gain.

x = Input drive signal

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

y_2 = Accumulated phase (rad) (optional)

$$y_1(t) = Ae^{j\theta(t)} \qquad y_2(t) = \theta(t)$$

$$\theta(t) + \int_0^t \left(2\pi f_c + x_1(\tau)K_o\right)d\tau + \phi$$

where:

f_c = translation frequency A = carrier amplitude

K_o = VCO gain ϕ = initial phase (radians)

Center Frequency

Indicates the VCO center frequency in hertz. The value may be set to 0 or even a negative frequency.

Amplitude

Indicates the amplitude of the output tone (single-sided peak amplitude). This value is specified in volts.

Initial Phase

Indicates the starting phase of the output complex tone. This value is specified in degrees.

VCO Gain

Indicates the gain of the VCO in (radians/second)/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.

8.14.16 Walsh Sequence

This block generates a repeating Walsh binary sequence, typically used in CDMA systems. Walsh sequences represent a family of orthogonal sequences, and are constructed from Hadamard matrices. The user can specify the sequence length (*N*) and row (*K*) of the sequence (*K* ,*N*) to be output. The output bit rate, and an initial delay can also be specified. This block can accept an external clock. A clock value greater than 0.5 is considered high.

The desired Walsh sequence is selected by specifying the "row #" of the associated Hadamard matrix. The row value can be specified as either a fixed value or via an external input.

x_1 = Optional external clock (impulse train)

x_2 = Optional external row selector

y_1 = Walsh sequence output

y_2 = Output bit clock (impulse train)

y_3 = Frame clock (indicates beginning of new row) (impulse)

$$H_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad H_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & \overline{H_N} \end{bmatrix}$$

Matrix Output Row

Specifies which row (*K*) of the (*N* x *N*) Hadamard matrix H_N is to be used as the Walsh output sequence. Valid range is 0 to *N*-1, where *N* is the sequence length. This entry is disabled when in external row selection mode.

Sequence Offset

Specifies a starting offset for the Walsh sequence. Valid range is 0 to *N*-1.

Sequence Length

Specifies the Walsh sequence length *N*. This value is a power of two, and has a valid range of 2 to 256.

Row Selection

Fixed (Internal)

The Walsh sequence row is fixed and corresponds to the Matrix Output Row parameter setting.

External Input

The Walsh sequence row is selected via the external control input. Valid input range is 0 to *N*-1.

Output Mode

Bilevel

The signal amplitudes associated with output the sequence are {-1, 1}.

Binary

The signal amplitudes associated with output the sequence are {0, 1}.

Timing

Internal

Indicates internal clock timing. The bit rate and start time need to be specified.

External

Indicates external timing. An external clock must be provided at the *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.

8.14.17Waveform Generator

This block implements a generic waveform generator capable of producing the following waveform types: square wave, triangle wave or sawtooth wave.

y_1 = Output waveform

y_2 = Output clock (impulse train)

Waveform Type

Square Wave

Outputs a square wave at the specified frequency.

Triangle Wave

Outputs a triangle wave at the specified frequency.

Sawtooth Wave

Outputs a sawtooth wave at the specified frequency.

Waveform Frequency

Specifies the waveform output rate in hertz.

Peak-to-Peak Amplitude

Specifies the peak-to-peak amplitude in *volts* of the waveform. A 1V p-p value with an offset of zero will produce a waveform in the range of [-0.5, +0.5] V.

Offset

Specifies an offset value in volts for the output waveform.

Start Time

Specifies the starting time, in seconds, for the output waveform.

8.15 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.`

8.15.1 Matrix to Vector

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

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

x_2 = Input Matrix [size *M* x *N*]

y_1 = Output clock (pulse)

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

Number of Output Vectors

Specifies the number *N* of desired output column vectors, and need not be the same as the number of columns in the input matrix (may be smaller). The maximum value for *N* is 16 or the number of columns in the matrix, whichever is smaller. This value must be specified numerically (global variable not allowed).

8.15.2 SubVector

This block extracts a *column* vector of size *N* from a larger or equal sized *column* vector of size *L* This block produces an updated output each time a “high” input clock is present. When desiring to extract a subvector from a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

x_1 = Input clock (high > 0.5) (pulse)

x_2 = Input column vector [size *L*]

y = Output column vector [size *N*]

Output Vector Size

Specifies the length *N* of the desired output column subvector. This value must be less or equal to the size of the input column vector.

Offset

Specifies the starting location within the input vector of the output subvector. The valid range is 0 to *N*-1.

8.15.3 Vector Bits to Symbol

This block combines adjacent elements of a column vector (containing binary bits) into a smaller column vector composed of symbols. This block produces an updated output each time a "high" input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

The input vector length must be evenly divisible by the number of bits per symbol *K*. This block operates by combining successive groups of *K* bits into a corresponding sequence of output vector symbols.

Example: Input Vector = $[1, 0, 1 , 1, 1, 0 , 0, 0, 1]^T$

Output Vector = $[5, 6, 1]^T$ 3 bits/symbol, MSB mode

x_1 = Input clock (high = 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.

8.15.4 Vector Demux

This block demultiplexes elements from a column vector into 2 or more smaller column vectors. This block produces an updated output each time a "high" input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

The output vectors can be forced to all be the same size through the use of padding when the number of output vectors *N* does not divide evenly into the input size vector *M*. Otherwise, one or more of the output vectors may be larger than others by one element. This block operates by stuffing elements from the input vector into each output vector in round robin fashion.

Example:	Vector A = [a1, a2, a3, a4, a5]^T	
w/o padding -	Output #1 = [a1, a3, a5]^T	Output #2 = [a2, a4]^T
w/ padding -	Output #1 = [a1, a3, a5]^T	Output #2 = [a2, a4, pad]^T

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

x_2 = Input column vector [size M]

y_1 = Output clock (pulse)

$y_{2 \ldots N+1}$ = 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.

8.15.5 Vector Merge

This block appends a column vector of size M to another column vector of size L. The output vector will be of size $N = L + M$. The `Vector Merge` block automatically reads the size of the input vectors and computes the corresponding output vector size. Among other uses, the `Vector Merge` block can be used to zero pad a non power-or-two sized vector so that it can be used by the `Vector FFT` block.

This block produces an updated output each time a “high” input clock is present. This block has no internal parameters.

x_1 = Input clock (high > 0.5) (pulse)

x_2 = Input column vector #1 [size L]

x_3 = Input column vector #2 [size M]

y = Output signal vector [size N]

8.15.6 Vector Mux

This block multiplexes elements from two or more column vectors into a new vector. This block produces an updated output each time a “high” input clock is present. When desiring to operate on *row* vectors, first convert all row vectors to column vectors by using a matrix transpose block.

All input vectors must be of the same size L. The output size will then be a vector of length N x L, where N is the number of input vectors. The block operates by taking elements from each input vector in round robin fashion.

Example: Vector A = [a1, a2, a3]^T Vector B = [b1, b2, b3]^T

Output = [a1, b1, a2, b2, a3, b3]^T

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

$x_{2 \ldots N+1}$ = Input column vectors [size L]

y_1 = Output clock (pulse)

y_2 = Output column vector [size M=N x L]

Number of Input Vectors

Specifies the number N of input column vectors. The maximum value for N is 16. This value must be specified numerically (global variable not allowed).

8.15.7 Vector Symbol to Bits

This block decomposes a column vector of symbol values into a larger column vector composed of binary bits. This block produces an updated output each time a “high” input clock is present. When desiring to operate on a *row* vector, first convert the row vector to a column vector by using a matrix transpose block.

This block operates by breaking up each symbol into its underlying binary bits and then outputting the K least significant bits. The output order of the K output bits is controlled by the MSB / LSB selection.

Example: Input Vector = [5, 2, 11]^T 3 bits/symbol, MSB mode

Output Vector = [1, 0, 1 , 0, 1, 0 , 0, 1, 1]^T

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

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

x_2 = Input column vector [size M]

y_1 = Output clock (pulse)

y_2 = Output column vector [size M / K]

Number of Bits per Symbol

Specifies the number of output bits per symbol *K*.

Bit Order

LSB First

Indicates that the first element in each output group of binary bits corresponds to the least significant bit of the input symbol value.

LSB Last

Indicates that the last element in each output group of binary bits corresponds to the least significant bit of the input symbol value.

8.15.8 Vector to Matrix

This block assembles two or more column vectors into a matrix. This block produces an updated output each time a “high” input clock is present. When desiring to operate on *row* vectors, first convert all row vectors to column vectors by using a matrix transpose block.

All input vectors must be of the same size *M*. The output size will then be a matrix of size *M* x *N*, where *N* is the number of input vectors.

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

$x_{2\ldots N+1}$ = Input column vectors [size *M*]

y_1 = Output clock (pulse)

y_2 = Output matrix [size *M* x *N*]

Number of Input Vectors

Specifies the number *N* of input column vectors. The maximum value for *N* is 16. This value must be specified numerically (global variable not allowed).

Chapter 9
Core Block Reference

The following are described in this chapter.

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

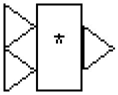
To make it easier to find blocks in this chapter, they are presented in alphabetical order, regardless of their block category. For most blocks, a mathematical function is included. The following table translates the symbols that may appear in the function:

Symbol	What it represents
A	Amplitude
e	Naperian constant
dt	Derivative with respect to time
lb	Lower bound
mod	Modulus
s	Laplacian operator

Symbol	What it represents
t	Time
ub	Upper bound
ω	Frequency
x	Input signal
y	Output signal

Input signals are represented as x_1, x_2, ...x_n, where x_1 represents the topmost signal entering the block. When n is omitted, x_1 is assumed. Output signals are represented as y_1, y_2, ...y_n, where y_1 represents the topmost signal exiting the block. When n is omitted, y_1 is assumed.

9.1 * (multiply)



$$y = x_1 * x_2 * ... * x_n$$

Block Category: Arithmetic

The * block produces the product of the input signals. Inputs can be scalars or vectors.

Commsim assigns ones to all unconnected inputs.

Multiplying vectors and matrices

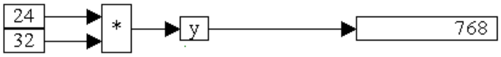
To perform a single value summation of an element-by-element multiply of two vectors, use the `dotProduct` block.

To multiply two matrices, use the `multiply` block.

Examples

1. Multiplication of two scalar inputs

Consider the equation $y = 24 * 32$, which can be realized as:



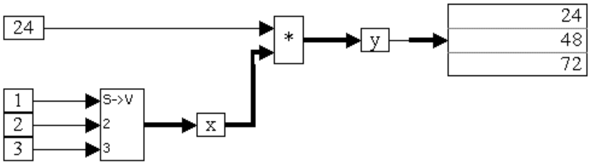
Two `const` blocks provide the values 24 and 32. When connected to a `*` block, the product is 768.

2. Multiplication of a scalar and a vector

Consider the equation

$y = 24\,\mathbf{x}$

where **x** = [1 2 3]. This equation can be realized as shown below.



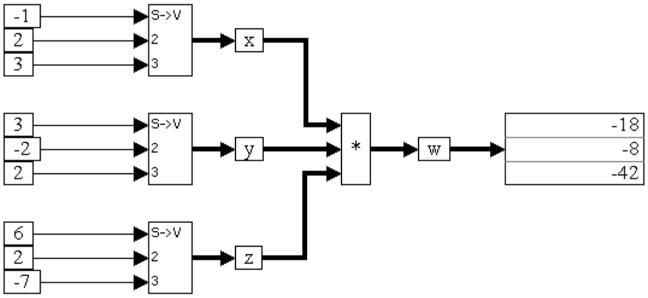
A `scalarToVec` block creates a three-element vector from the constant values 1, 2, and 3. When the simulation runs, the `*` block multiplies all the elements of the incoming vector line with the constant value 24.

3. Multiplication of vectors

Consider the equation:

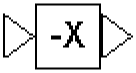
w = **x y z**

where **x** = [-1 2 3], **y** = [3 -2 2], and **z** = [6 2 -7]. This equation can be realized as:



When the simulation runs, the * block performs an element-by-element multiplication operation on the incoming vectors. For example, **w**(1) = **x**(1) * **y**(1) * **z**(1), **w**(2) = **x**(2) * **y**(2) * **z**(2), and so on.

9.2 -X (negate)



$$y = -x$$

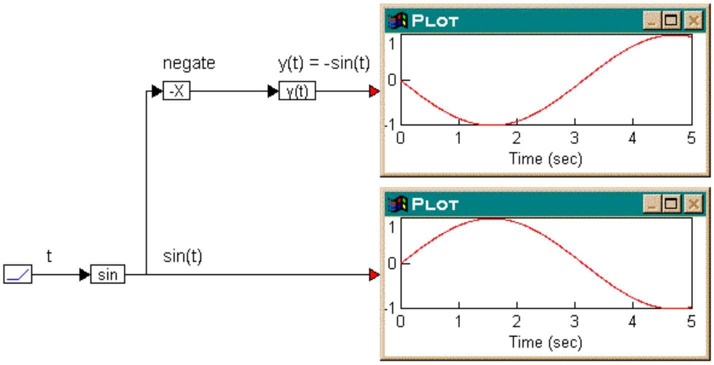
Block Category: Arithmetic

The `-X` block negates the input signal. Input can be scalar, vector, or matrix.

Examples

1. Negation of a scalar

Consider the equation $y(t)$ = - sin(t), which can be realized as:



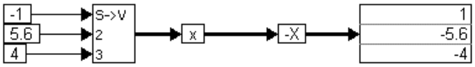
A `ramp` block is used to access simulation time t, a sin block generates sin(t), and a -`X` block converts sin(t) to -sin(t). Both sin(t) and $y(t)$ are plotted for comparison.

2. Negation of a vector

Consider the equation:

z = -**x**

where **x** = [-1 5.6 4]. This equation can be realized as:



A `scalarToVector` block creates a three-element vector from the constant values -1, 5.6, and 4. When the simulation runs, the `-X` block performs an element-by-element negate operation on the incoming vector.

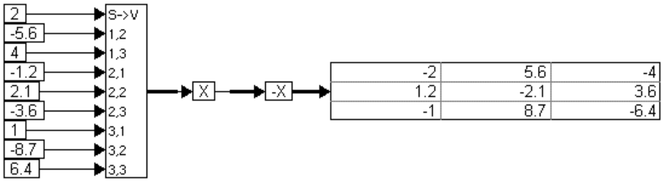
3. Negation of a matrix

Consider the equation:

Z = -**X**

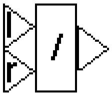
$$\mathbf{X} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

This equation can be realized as:



When the simulation runs, the `-X` block performs an element-by-element negate operation on the incoming matrix.

9.3 / (divide)



$$y = \frac{x_1}{x_2}$$

The `/` block produces the quotient of the input signals. The inputs can be scalars or vectors. On the connector tabs, “l” represents the numerator x_1 and “r” represents the denominator x_2. If x_1 is unconnected, Commsim feeds it a zero. If x_2 is equal to 0 or unconnected, Commsim displays a “Divide by 0” message and highlights the offending block in red.

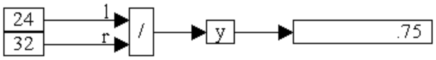
Performing matrix inversions

To perform matrix inversions, use the `invert` block.

Examples

1. Division of two scalar inputs

Consider the equation y = 24/32, which can be realized as:

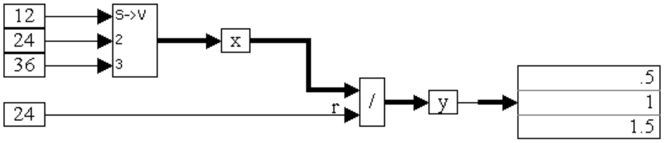


2. Division of a vector by a scalar

Consider the equation:

y = **x**/24

where **x** = [12 24 36]. This equation can be realized as:



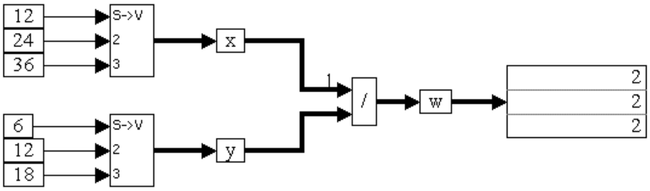
When the simulation runs, the / block divides each element of vector **x** with the constant value of 24.

3. Division of vectors

Consider the equation:

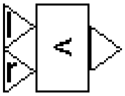
w = **x**/**y**

where **x** = [12 24 36] and **y** = [6 12 18]. This equation can be realized as:



When the simulation runs, the / block performs an element-by-element division operation on the incoming vectors. For example, **w**(1) = **x**(1)/**y**(1), **w**(2) = **x**(2)/**y**(2), and so on.

9.4 < (less than)



$$: \begin{cases} 1 \text{if } x_1 < x_2 \\ 0 \text{if } x_1 \geq x_2 \end{cases}$$

Block Category: Boolean

The < block produces an output signal of 1 if and only if input signal x_1 is less than input signal x_2. Otherwise, the output is 0. On the connector tabs, “l” represents x_1 and “r” represents x_2.

If you click the right mouse button over the < block, the Boolean block menu appears allowing you to assign a different function to the block.

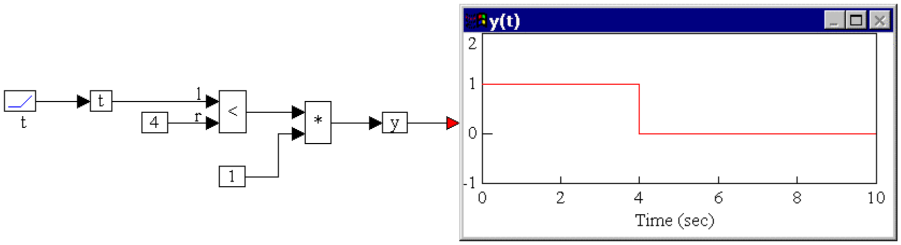
Examples

1. Simple if-then-else construct

Consider a variable *y* such that:

If *t* < 4 then *y* = 1; else *y* = 0

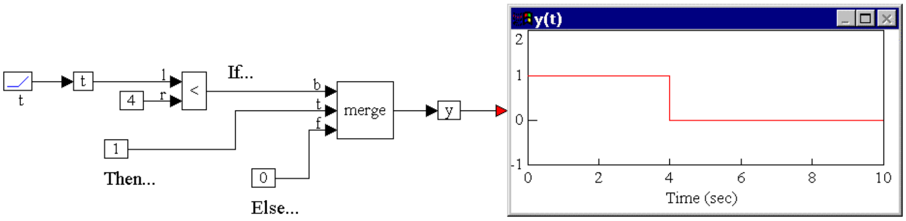
Assume that *t* is simulation time. This system can be realized as:



By multiplying a constant value 1 with the output of the < block, *y* is guaranteed to assume a value of 0 until the inequality is true. When the inequality is true, *y* assumes a value equal to the output of the * block.

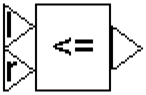
2. Modified if-then-else construct

The previous example can also be realized as:



The key difference in implementation is the use of a `merge` block rather than a * block. The `merge` block explicitly depicts the if-then-else structure; the * block is a shortcut and can lead to confusion.

9.5 <= (less than or equal to)



$$y = \begin{cases} 1 & \text{if } x_1 \le x_2 \\ 0 & \text{if } x_1 > x_2 \end{cases}$$

Block Category: Boolean

The <= block produces an output signal of 1 if and only if input signal x_1 is less than or equal to input signal x_2. Otherwise, the output is 0. On the connector tabs, “l” represents x_1 and “r” represents x_2. The <= block accepts two scalar inputs.

If you click the right mouse button over the <= block, the Boolean block menu appears allowing you to assign a different function to the block.

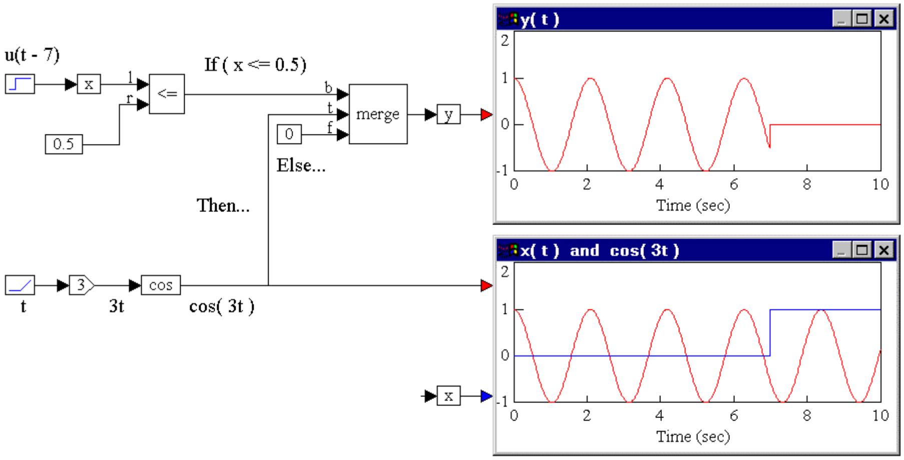
Examples

1. Simple if-then-else construct

Consider a variable *y* such that:

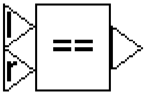
If $x \le 0.5$ then $y = \cos(3t)$; else $y = 0$

where *t* is simulation time. Let *x* be a unit step delayed by 7 sec, represented as *u*(*t* - 7). This system can be realized as shown below.



Until the onset of the step input at $t = 7$ sec, the Boolean inequality $x \le 0.5$ evaluates to true, and *y* takes on a value of cos(3*t*). At $t = 7$ sec, the Boolean inequality evaluates to false and remains false for the duration of the simulation. Consequently, from this point onwards, *y* takes on the value of 0. The lower `plot` block monitors the outputs of the `cos` and `variable` *x* blocks.

9.6 == (equal to)



$$y = \begin{cases} 1 \text{ if } x_1 = x_2 \\ 0 \text{ if } x_1 \neq x_2 \end{cases}$$

Block Category: Boolean

The == block is useful for evaluating the Boolean == equality. This block accepts two scalar inputs labeled “l” (for x_1)and “r” (for x_2). The output of the == block is 1 if and only if input “l” is identically equal to input “r;” otherwise, the output is zero.

Boolean equality comparisons of floating point variables and non-integer constants

As with programming in any language, it is generally not a good idea to perform Boolean equality comparisons involving floating point variables and non-integer constants. These types of comparisons should be converted to Boolean inequality comparisons. (For example, {If position is equal to π, then…}) should be converted to {If position is greater than or equal to < π rounded off>, then…}.) The reason for this is because a floating point variable, such as position, is rarely exactly equal to a non-zero non-integer value, particularly if it is obtained by solving one or more equations.

If you click the right mouse button over the == block, the Boolean block menu appears allowing you to assign a different function to the block.

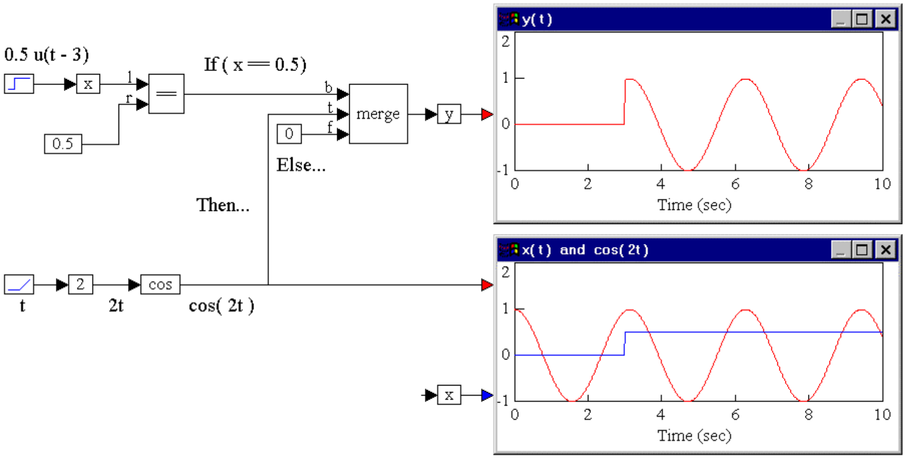
Examples

1. Comparing constants

Consider a variable *y* such that:

If *x* = 0.5 then *y* = *cos*(*2t*); else *y* = 0

where *t* is simulation time. Let *x* be a step function of amplitude 0.5, delayed by 3 sec. This is usually represented as 0.5 *u*(*t* - 3). This system can be realized as:



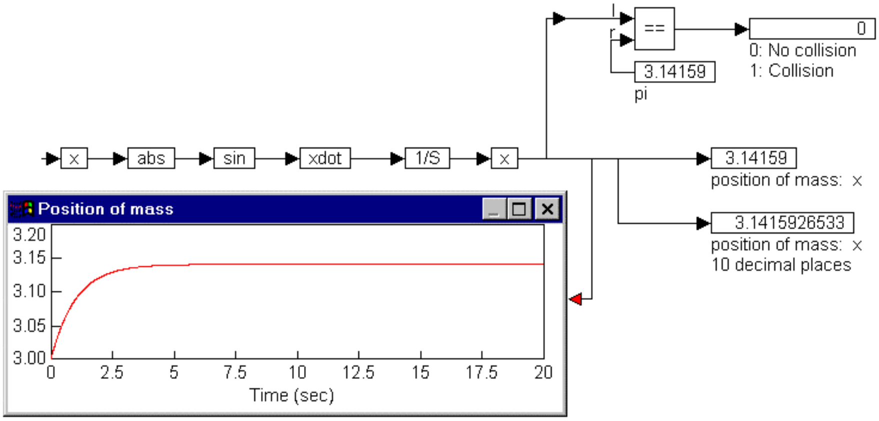
Until the onset of the step input at *t* = 3 sec, the Boolean equality x == 0.5 evaluates to false, and *y* takes on a value of 0. At *t* = 3 sec, the Boolean equality evaluates to true, and remains true for the duration of the simulation. Consequently, from this point onwards, *y* takes on the value of cos(2*t*). The lower `plot` block is used to monitor the outputs of the `cos` block and the `variable` *x*.

2. Comparing a floating point variable with a non-integer constant

In a collision detection problem, if position *x* of a mass in motion is equal to π, then a collision is assumed to have occurred with an immovable wall that is located at *x* = π. Furthermore, the position of the mass is assumed to be given by the solution of the following first order differential equation:

$$\dot{x} = \sin(|x|)$$

The initial condition is assumed to be *x*(0) = 3.0. It is tempting to realize this system as:



From the result shown in the `plot` block, at around *t* = 7 sec, the mass arrives at the boundary located at π. However, the collision detection logic, using an == block that compares *x* with a constant value of π, never detects the collision. This is because the final mass position, as obtained from the output of the `integrator`, is 3.141592653, which is not equal to 3.14159.

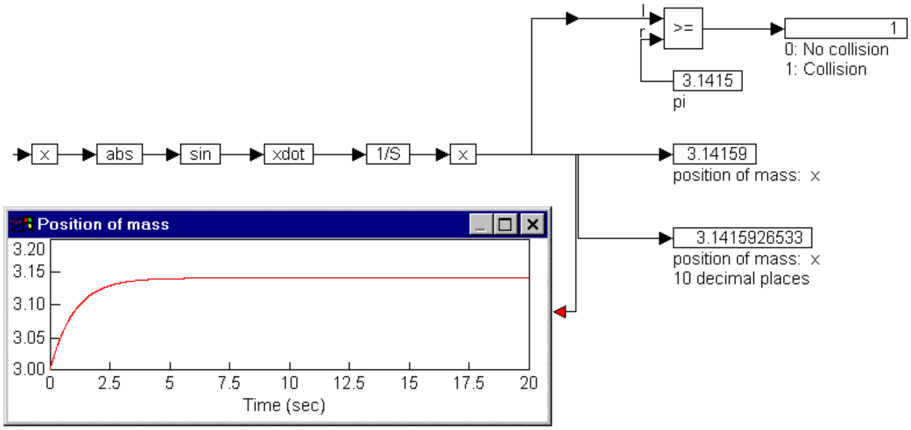
It is clear from the `plot` block, that for all practical purposes, the mass collided with the wall around *t* = 7 sec. To capture this reality in the simulation, convert the Boolean equality comparison:

If *x* = 3.14159 then…

to a Boolean inequality comparison:

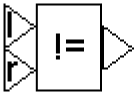
If *x* ≥ 3.1415 then…

After reducing the `const` block to four decimal places with no round-off, the system can be realized as:



Except for replacing the == block with the >= block, this diagram is similar to the previous one. In this case, the collision detection logic detects a collision around t = 8 sec. Obviously, the time at which the collision is detected depends on the number of decimal places retained for the π approximation.

9.7 != (not equal to)



$$y = \begin{cases} 1 \text{ if } x_1 \neq x_2 \\ 0 \text{ if } x_1 = x_2 \end{cases}$$

Block Category: Boolean

The != block produces an output signal of 1 if and only if the two scalar input signals are not equal. On the connector tabs, "l" represents x_1 and "r" represents x_2.

If you click the right mouse button over the != block, the Boolean block menu appears allowing you to assign a different function to the block.

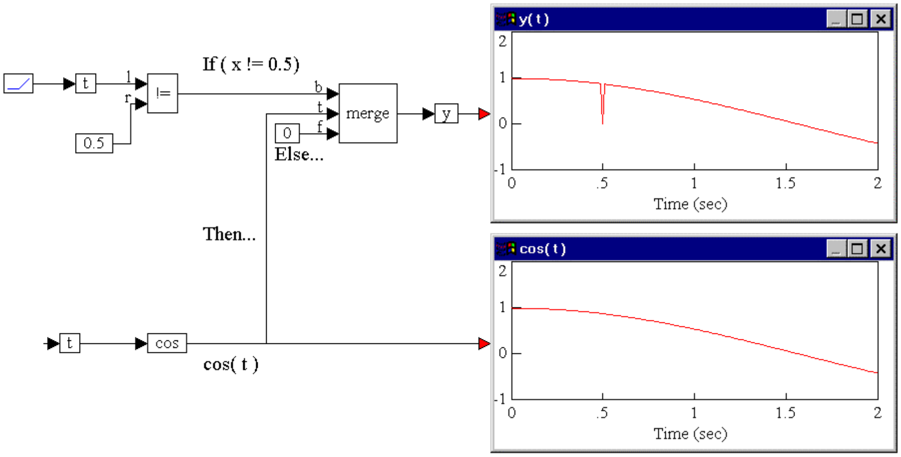
Examples

1. Comparing constants

Consider a variable *y* such that:

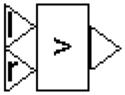
If $t \neq 0.5$ then $y = \cos(t)$; else $y = 0$

where *t* is simulation time. This system can be realized as shown on below.



Until the value of *t* reaches 0.5, the Boolean inequality $t \neq 0.5$ evaluates to true, and *y* takes on a value of cos(*t*). At $t = 0.5$ sec, the Boolean inequality evaluates to false, and at the very next time step, returns to true, and remains true for the duration of the simulation. Consequently, at the moment $t = 0.5$ sec, *y* takes on the value of 0, and at every other point, *y* is equal to cos(*t*).

9.8 > (greater than)



$$y = \begin{cases} 1 & \text{if } x_1 > x_2 \\ 0 & \text{if } x_1 \leq x_2 \end{cases}$$

Block Category: Boolean

The > block is useful in evaluating the Boolean > inequality. It accepts two scalar inputs, labeled “l” and “r.” The output of the > block is 1 if and only if input “l” > input “r;” otherwise the output is zero.

If you click the right mouse button over the > block, the Boolean block menu appears allowing you to assign a different function to the block.

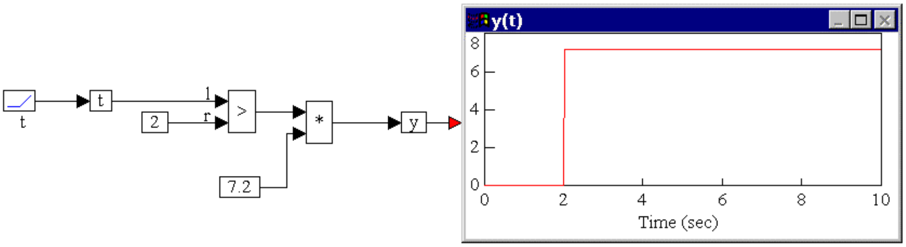
Examples

1. Simple if-then-else construct

Consider a variable *y* such that:

If *t* > 2 then *y* = 7.2; else *y* = 0

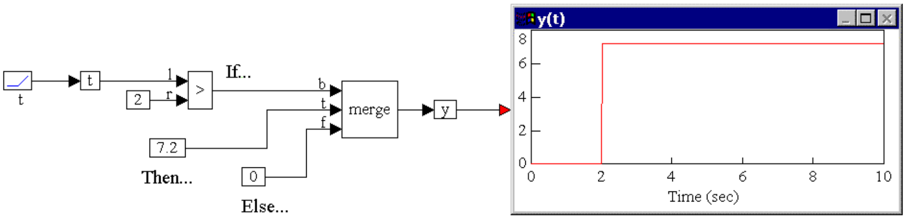
Assume that *t* is simulation time. This system can be realized as:



By multiplying a constant value of 7.2 with the output of the > block, *y* is guaranteed to assume a value of 0 until the inequality is true. When the inequality is true, *y* assumes a value equal to the output of the * block.

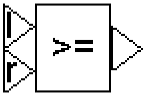
2. Modified if-then-else construct

Using the above equation, it can also be realized as:



The key difference in implementation is the use of a `merge` block rather than a * block. The `merge` block explicitly depicts the if-then-else structure, whereas the * block is a shortcut and can lead to confusion.

9.9 >= (greater than or equal to)



$$y = \begin{cases} 1 \text{ if} x_1 \geq x_2 \\ 0 \text{ if} x_1 < x_2 \end{cases}$$

Block Category: Boolean

The >= block produces an output signal of 1 if and only if input signal x_1 is greater than or equal to input signal x_2. Otherwise, the output is 0. On the connector tabs, “l” represents x_1 and “r” represents x_2. The >= block accepts two scalar inputs.

If you click the right mouse button over the >= block, the Boolean block menu appears allowing you to assign a different function to the block.

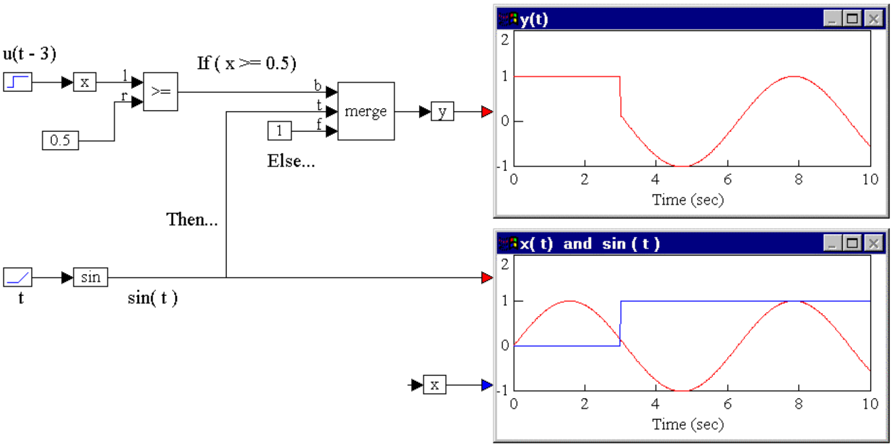
Examples

1. Simple if-then-else construct

Consider a variable *y* such that:

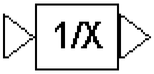
If $x \geq 0.5$ then $y = \sin(t)$; else y = 1

where *t* is simulation time. Let *x* be a unit step delayed by 3 sec. This is usually represented as $u(t - 3)$. This system can be realized as:



Until the onset of the step input at *t* = 3 sec, the Boolean inequality *x* ≥ 0.5 evaluates to false and *y* takes on a value of 1. At *t* = 3 sec, the Boolean inequality evaluates to true and remains true for the duration of the simulation duration. Consequently, from this point onwards, *y* takes on the value of sin(*t*).

9.10 1/X (inverse)



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

Block Category: Arithmetic

The `1/X` block produces the inverse of the input signal. The input can be scalar, vector, or matrix.

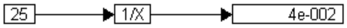
Computing the matrix inverse of a matrix

Use the `invert` block to compute the matrix inverse of a matrix. If a vector or matrix is fed into an `1/X` block, the result will be an element-by-element inversion of the vector or matrix (that is, [one divided by the element] operation). This is not equivalent to a normal vector pseudo-inverse operation or a normal matrix inverse operation.

Examples

1. Computation of 1/X of a scalar

Consider the equation *y*= 1/25, which can be realized as:



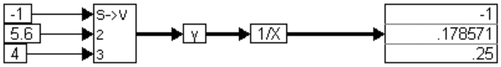
The incoming constant value of 25 results in 1/25 = 0.04.

2. Computation of 1/X of a vector

Consider the equation:

$\mathbf{z} = 1/\mathbf{y}$

where **y** = [-1 5.6 4], and where an element-by-element inversion is implied. This equation can be realized as:



An element-by-element inverse operation is performed on the three elements in the `scalarToVec` block.

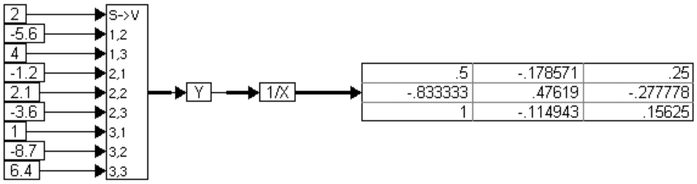
3. Computation of 1/X of a matrix

Consider the equation:

Z = 1/ **Y**

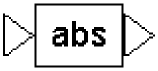
where $$\mathbf{Y} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

This can be realized as:



When the simulation runs, the `1/X` block performs an element-by-element inverse operation on the incoming matrix.

9.11 abs



$$y = |x|$$

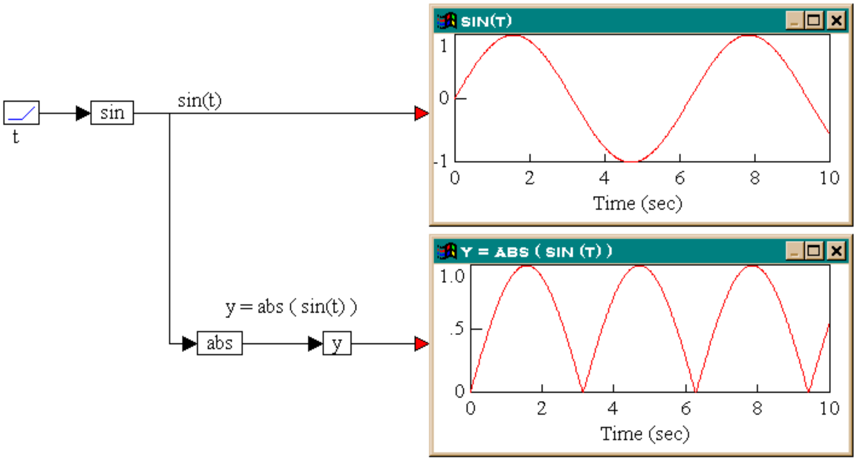
Block Category: Arithmetic

The abs block produces the absolute value of the input signal. The inputs can be scalars, vectors, or matrices.

Examples

1. Absolute value of a scalar

Consider the equation *y* = abs (sin (*t*)), which can be realized as shown below.



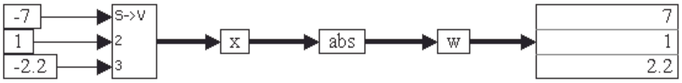
The results in the two plot blocks show that the abs block computes the absolute value of the input signal.

2. Absolute value of a vector

Consider the equation:

w = abs (**x**)

where **x** = [-7 1 -2.2]. This equation can be realized as:



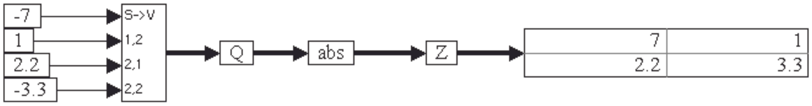
When the simulation runs, the abs block computes and outputs an element-by-element absolute value of the vector **x**.

3. Absolute value of a matrix

Consider the equation:

Z = abs(**Q**)

where $\mathbf{Q} = \begin{bmatrix} -7 & 1 \\ 2.2 & -3.3 \end{bmatrix}$ This equation can be realized as shown below.



Four `const` blocks provide the vector element values of **Q** through a `scalarToVector` block. When the simulation runs, the `abs` block computes the element-by-element absolute value of the incoming matrix.

9.12 acos

$y = \text{arc}\cos x$

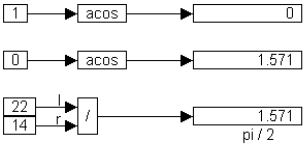
Block Category: Transcendental

The `acos` block produces the inverse cosine of the input signal. The output is an angle in radians.

Examples

1. Computation of cos^{-1}(1) = 0; cos^{-1}(0) = π/2

This equation can be realized as:



Two `acos` blocks are used to compute the inverse cosines. For comparison, the constant value of π/2 is generated by connecting two `const` blocks, set to 22 and 14, to the “l” and “r” inputs of a `/` block. From the results obtained, the `acos` blocks yield correct values for the angles.

9.13 ActiveX read

Block Category: Real Time

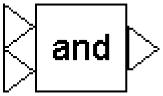
The `ActiveX read` block gets data from an ActiveX container.

9.14 ActiveX write

Block Category: Real Time

The `ActiveX write` block sends data to an ActiveX container.

9.15 and



$y = x_1$ bitwise AND x_2

Block Category: Boolean

The `and` block produces the bitwise AND of two to 256 scalar input signals.

If you click the right mouse button over the `and` block, the Boolean block menu appears allowing you to assign a different function to the block.

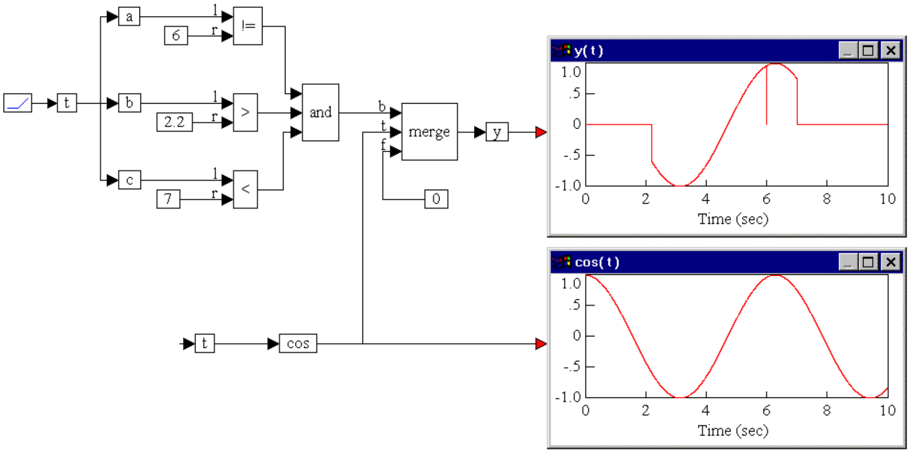
Examples

1. Three variable and

Consider a variable *y* such that:

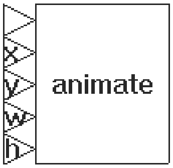
If $a \neq 6$ and $b > 2.2$ and $c < 7$, then $y = \cos(t)$; else $y = 0$

where *t* is simulation time. Furthermore, let *t* be the input to all three parameters *a*, *b*, and *c*. This system can be realized as:



The output of the `and` block is true only when all the three inputs are true. This happens in the range *t* = (2.2, 7), except for the instant *t* = 6. This result is apparent from the top `plot` block. The `variable` *y* is equal to cos(*t*) in the range *t* = (2.2, 7). At the instant *t* = 6, `variable` *a* is momentarily false, and consequently, *y* = 0 at *t* = 6, since the output of the `and` block evaluates to false at that instant.

9.16 animate



Block Category: Animation

The `animate` block lets you animate an image during simulation.

9.17 asin

y = arc sin *x*

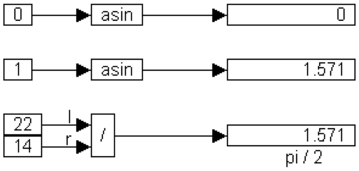
Block Category: Transcendental

The `asin` block produces the inverse sine of the input signal. The output is an angle in radians.

Examples

1. Computation of sin^{-1}(1) = π/2; sin^{-1}(0) = 0

This equation can be realized as:



Two `const` blocks, set to 0 and 1, are fed to the `asin` blocks. For comparison, the constant value of π/2 is generated by feeding two `const` blocks, set to 22 and 14, into the “l” and “r” inputs of a `/` block. From the results obtained, the `asin` blocks yield correct values for the angles.

9.18 atan2

y = 4 quad arctan (x_1,x_2)

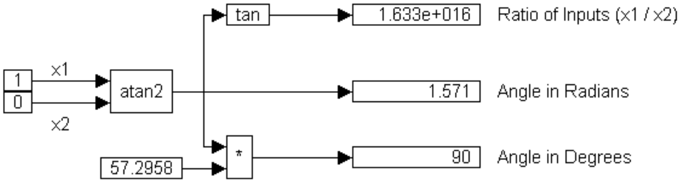
Block Category: Transcendental

The atan2 block computes the four quadrant inverse tangent of the input signals. The atan2 block uses the signs of both input signals to determine the sign of the output signal. The output is an angle in radians.

Examples

1. Computation of tan^{-1}(∝) = π/2

This equation can be realized as:

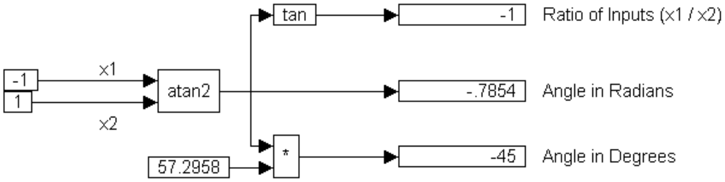


To convert radians to degrees, the angle in radians is multiplied by (180/π) = 57.2958.

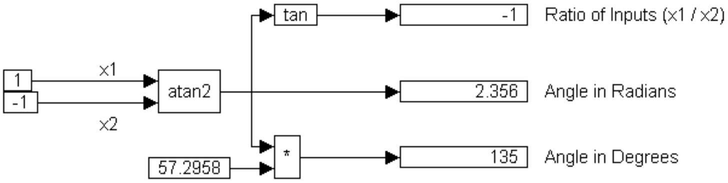
Since the atan2 block uses the value of x_1, the signs of x_1 and x_2, and the ratio x_1/x_2 in computing the inverse tangent, the result depends on all these parameters. In the current case, since the ratio is infinity, atan2 computes the inverse correctly to be π/2 radians, or 90^o. Also, in the current case, x_1 can be any positive value, since its ratio with 0 will be infinity, regardless of its value.

2. Computation of tan^{-1}(-1): quadrant dependency

Using the same configuration in the above example, tan^{-1}(-1) can be realized as:



Here, the angle obtained is -.7854 radians, or -45^o, because the atan2 block determines that the angle lies in the fourth quadrant. However, it is immediately apparent that the same ratio of -1 can be obtained by flipping the signs on x_1 and x_2:



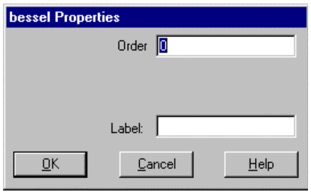
In this case, the atan2 block uses the relative signs of x_1 and x_2 to determine that the angle lies in the second quadrant, and yields an angle of 180 - 45 = 135^o, or 2.356 radians.

9.19 bessel

$$y = \text{bessel}_n x$$

Block Category: Transcendental

The `bessel` block generates the Bessel function of order *n*.



Order: 0

Label:

Order: Sets the order of the Bessel function. Specify the order as an integer. The default is 0.

Label: Indicates a user-defined block label.

Examples

1. Approximation of sin(a sin ϕ)

Bessel functions come up frequently in the analysis and solution of nonlinear differential equations. Consider the following approximation:

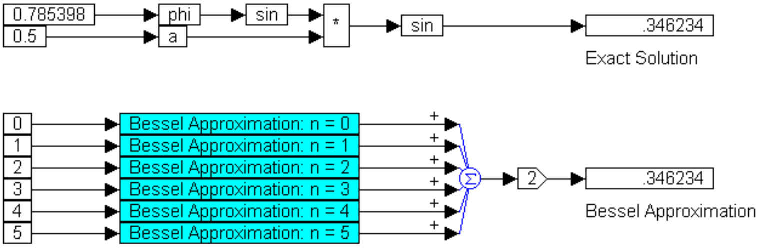
$$\sin(a\sin\phi) = 2\sum_{n=0}^{\infty} J_{2n+1}(a)\sin[(2n+1)\phi]$$

where *a* and ϕ are parameters, and J_m is a Bessel function of order *m*. Such approximations are a part of the standard procedure for obtaining approximate analytical solutions to equations of the type:

$$\ddot{u} + 2\varepsilon\mu\sin\dot{u} + u = K\cos\Omega t$$

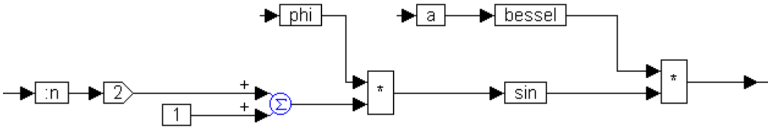
These equations are used in the harmonic analysis of forced oscillations of single degree of freedom systems.

The above approximation can be realized as:


0.785398
0.5
phi
a
sin
*
sin
.346234
Exact Solution
0
1
2
3
4
5
Bessel Approximation: n = 0
Bessel Approximation: n = 1
Bessel Approximation: n = 2
Bessel Approximation: n = 3
Bessel Approximation: n = 4
Bessel Approximation: n = 5
+
Σ
2
.346234
Bessel Approximation

Two `const` blocks produce π/4 and 0.5 as the values for *ϕ* and *a*, respectively. The sine of *phi* is multiplied by *a* and the result is fed through another `sin` block to compute the exact solution.

Six `const` blocks, set to 0 through 5, generate different terms of the infinite series approximation. In this case, only the first six terms of the series are retained. Each of these `const` blocks is connected to a compound block, which has the following internal structure:



The `const` block feeds a value to a local `variable` *:n*. The output of *:n* is connected to a `gain` block set to 2. The output of the `gain` block, and the output of a `const` block set to 1, are fed into two inputs of a `summingJunction` block. The output of the `summingJunction` block and the output of `variable` *phi* are connected to a * block, to compute the term *(2n+1)ϕ*. The `variable` *a* is connected to a `bessel` block whose internal order is set to the correct value (0, 3, 5, 7, 9, or 11, depending on the value of *:n*).

At the top level, the outputs of the six compound blocks are summed using a six input `summingJunction` block, and the output of the `summingJunction` block is connected to a `gain` block set to 2. The output of the `gain` block is connected to a `display` block.

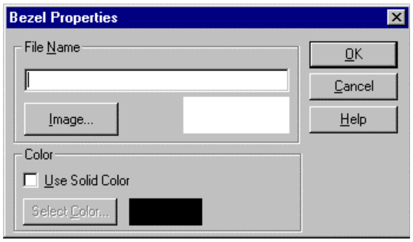
From the results obtained, it is proven that by retaining the first six terms in the approximation, very close agreement can be obtained with the actual value of sin(*a* sin *ϕ*).

9.20 bezel

Block Category: Annotation

The `bezel` block is an effective way to add background characteristics, such as operator control panels, to your screen. Designed to be used in display mode, the `bezel` block accepts bitmaps or background color specifications. When display mode is turned on, `bezel` blocks act as background and appear beneath other blocks.

When display mode is turned off, you can resize a `bezel` block by dragging on its borders. If a bitmap is associated with the `bezel` block, it initially assumes the size of the bitmap. For solid color backgrounds, the chosen color fills in the bezel area and can also be resized. When you turn on display mode, the sizing border goes away.



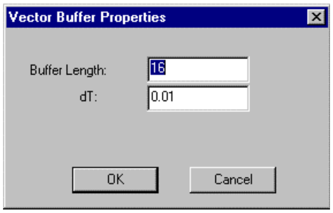
File Name: Indicates the name of the .BMP file used as the background bezel. You can type the file name directly into this box or select one using the Image button.

Color: Lets you use a solid color as the background for the bezel. To select a color, activate Use Solid Color and click on the Select Color button to choose a color. When Use Solid Color is not activated, the `bezel` block defaults to the name of the .BMP file specified in the File Name box.

9.21 buffer

Block Category: Matrix Operations

The `buffer` block places a sequence of values in a buffer based on the buffer length, the time between successive samples, and the duration of the simulation. The `buffer` block accepts a single scalar input and produces a single vector output. It is useful for performing basic digital signal processing operations.



Buffer Length: 16
dT: 0.01

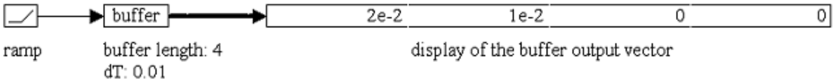
Buffer Length: Determines the number of samples; that is, the size of the buffer.

dT: Determines the time between successive samples; that is, the rate at which samples of the incoming signal are collected and placed in the buffer. The block samples if the current time is greater than or equal to the last sample time plus the dT.

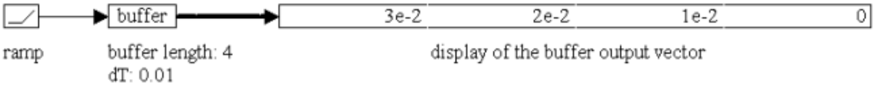
Examples

1. Basic buffer operation

Consider the following `buffer` block, with a buffer length of 4 and time between successive samples of 0.01.



For simplicity, let the simulation step size be equal to 0.01. If the input to the `buffer` is an arbitrary non-zero signal, such as a `ramp` signal, then after two simulation time steps, the output of `buffer` is a vector of length 4, with the first two elements containing non-zero values and the remaining two still at zero. At the very next time step, the simulation appears as:

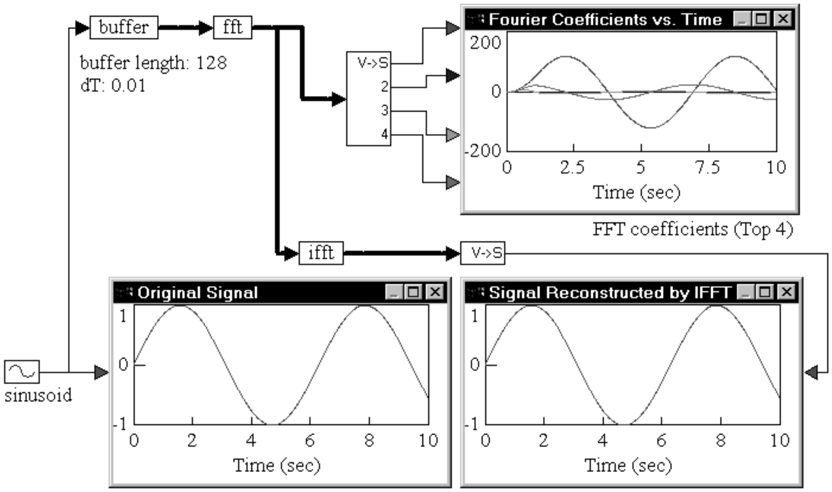


The previous values are pushed down the vector by one cell, and the top cell is occupied by the latest sample. Once the simulation goes beyond four time steps, the output will be a full vector.

Obviously, if the input signal itself is zero for some points, those values will be reflected accurately in the output.

2. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using the IFFT.



A `sinusoid` block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a `buffer` block of length 128 samples and a time between successive samples of 0.01. The output of the `buffer` block is connected to an `fft` block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the `fft` block is Fourier coefficients. The individual coefficients are accessed using a `vecToScalar` block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the `fft` block to an `ifft` block to compute the IFFT. The output of the `ifft` block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the `ifft` output vector does not have any delay because zero time has elapsed between the FFT and IFFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and IFFT algorithms.

9.22 button

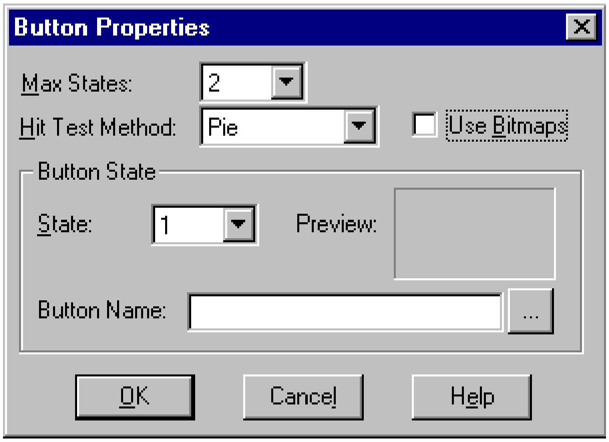
$$y = \text{state} - 1$$

Block Category: Signal Producer

The `button` block lets you dynamically insert signal values during a simulation.

You can set the number of states that a `button` block has, from two to a maximum of 16. You can also associate bitmaps with the states. The `button` block toggles between ON and OFF if it is a 2-state button and there are no bitmaps associated with it.

The `button` block also provides cycle through, pie area, push button horizontal, and vertical hit testing.



Button Properties

Max States: 2

Hit Test Method: Pie

□ Use Bitmaps

Button State

State: 1

Preview:

Button Name:

Max States: Indicates the number of states for the `button` block. The maximum allowable states is 16. The number of listed states in the States box is determined by the value entered in the Max States box.

Hit Test Method: You have the choice of five hit testing methods.

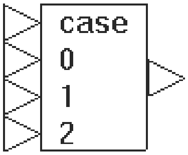
- **Cycle Through:** Causes the state to increase by 1 each time you click the right mouse button over the block. When the maximum state value is reached, the next mouse click changes the state back to 0.
- **Pie:** Divides the block into a number of pie-shaped wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. States advance in a clockwise direction with State 0 at the top.
- **Vertical**: Divides the block into a number of vertical wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. State 0 corresponds to the bottom wedge; state *N* corresponds to the top wedge.
- **Horizontal:** Divides the block into a number of horizontal wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. State 0 corresponds to the leftmost wedge; state *N* corresponds to the rightmost wedge.
- **Push Button:** Activates state 1 while you hold down the mouse button. When you release the mouse button, it activates state 0. Push button hit testing only supports a two-state `button` block.

Use Bitmaps: Lets you associate a bitmap with the selected state. Note that you can use bitmaps for all or none of the states, but not for some of the states. To make an association:

1. Activate the Use Bitmaps checkbox.
2. From the States box, select a state.
3. In the Bmp Path text box, enter the bitmap file name to be associated with the state. If you do not know the name or location of the file, click on the ... button to select one. A picture of the selected bitmap appears in the Preview window.

Button Name: Indicates a name for the `button` block. This text box appears only when you have de-activated the Use Bitmaps checkbox. The button name appears only when there is no bitmap associated with the block.

9.23 case



$$y = x_{n+2}$$

Block Category: Nonlinear

The `case` block lets you specify an unlimited number of execution paths based on the value of a single input, called the case input value. The case input value is the top input to the block and is labeled *case*. The remaining inputs are the possible execution paths. They are labeled 0 through *n*. These inputs operate on alphanumeric text.

The main application of the `case` block is in the construction of large nested if-else decision structures, where regular if-else constructs using Boolean blocks become too cumbersome.

When you want to output a particular element in a matrix, use the `index` block.

Case input value: The following rules apply to values fed into the connector tab labeled *case*:

- The case input value must be scalar. If a non-integer value is fed into it, it is truncated. For example, 0.999 is truncated to 0.
- If the case input value targets an out-of-range input, the `case` block returns an error. For example, an error results if the case input value is 5 for a four-element `case` block.
- If the case input value targets an unconnected input, the `case` block outputs a 0.

Scalar and matrix output: With the exception of the case input, all other inputs to the `case` block can be scalar, vector, or matrix.

Examples

1. Implementation of five scalar branches

Consider the decision tree:

If J = 0, then Y = A;

else

If J = 1, then Y = B;

else

If J = 2, then Y = C;

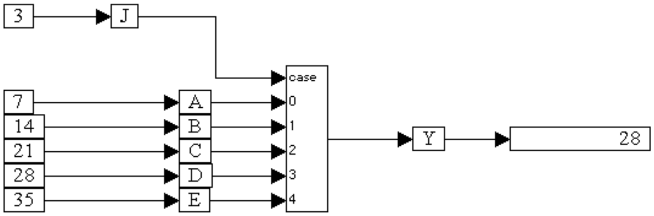
else

If J = 3, then Y = D;

else

If J = 4, then Y = E;

If A, B, C, D, and E are assumed to be constant values equal to 7, 14, 21, 28, and 35 respectively, the decision tree can be realized as:



Six `const` blocks produce values for the `variable` blocks named *J, A*, *B*, *C*, *D*, and *E*. The outputs of these `variables` are connected to a `case` block, with *J* connected to the case input, and `variables` *A*, *B*, *C*, *D*, and *E* connected to inputs 0, 1, 2, 3, and 4 respectively. The output of the `case` block is fed through a `variable` named *Y* and into a `display` block.

Since *J* is set to 3, the `variable` *D* is presented to the output as expected, and consequently, *Y* takes on the value of *D*, namely 28.

2. Implementation of three matrix branches

Consider the following part of the decision tree presented above:

If J = 0, then **Y** = **A**;

else

If J = 1, then **Y** = **B**;

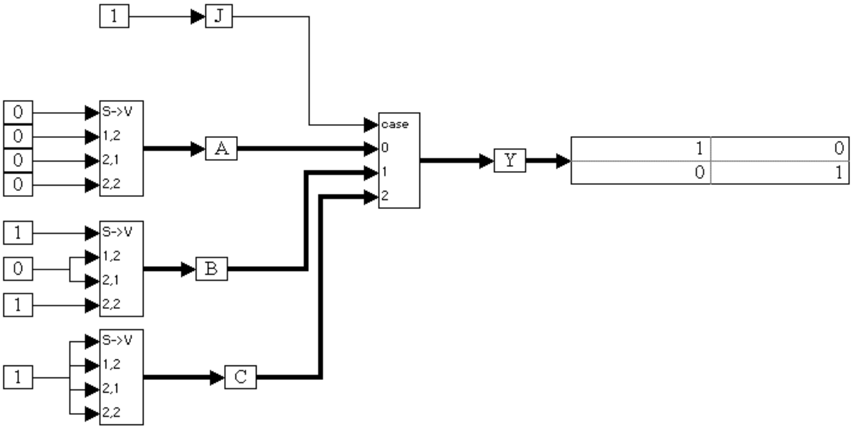
else

If J = 2, then **Y** = **C**;

If you let **A**, **B**, and **C** be:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} ;\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ;\mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

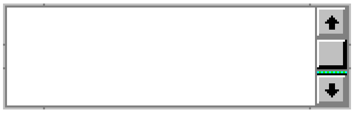
the decision tree can be realized as shown below.



Eight `const` blocks generate the elements of the three matrices, represented as three `scalarToVector` blocks. The `variable` *J* is set to 1 and is fed into the case input of a `case` block. The outputs of variables *A*, *B*, and *C* are wired to inputs 0, 1, and 2 of the `case` block. The output of the `case` block is connected to a `variable` **Y**, which is wired to a `display` block.

Since *J* is set to 1, the contents of variable *B* are presented at the output of the case block such that **Y** = **B**.

9.24 comment



Block Category: Annotation

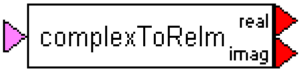
The `comment` block adds a comment to the diagram. When you position the pointer over the block and click the right mouse button, the pointer changes into a vertical I-beam, indicating that you're in text-entry mode. As you insert text, Commsim automatically scrolls the text if it runs out of room in the viewable region of the block. To correct or remove text, use the DEL and BACKSPACE keys. To exit text-entry mode, click the right mouse button on the `comment` block a second time.

You can also copy text from an application file into a `comment` block. For example, to copy text from a WORD document, highlight the text to be copied and press CTRL+V. In the `comment` block, position the I-beam where you want to insert the text and press CTRL+C.

To retain the format of the copied text, activate the Use Rich Text Format under the Preferences tab in the dialog box for the Edit > Preferences command. If Use Rich Text Format is not activated, the text will revert to the text format specified with the View > Fonts command.

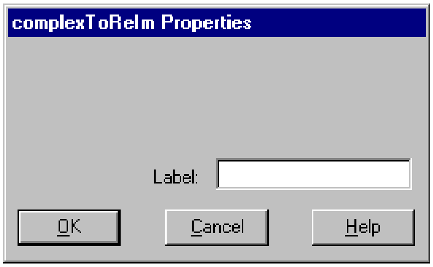
When reading a comment, use the scroll bar to move text in and out of the viewable region. To resize a `comment` block, drag on its edges.

9.25 complexToReIm



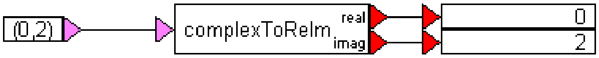
Block category: Arithmetic

The `complexToReIm` block takes a complex number and outputs its real and imaginary parts separately.

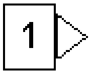


Label: Indicates a user-defined block label.

Examples



9.26 const



$y = \text{constant value}$

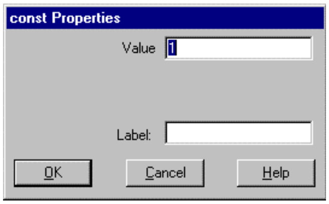
Block Category: Signal Producer

The `const` block generates a constant signal.

The `const` block accepts alphanumeric text strings and matrix data.

Alphanumeric text: You can create alphanumeric text strings that Commsim displays upon the occurrence of a conditional event. The blocks that operate on alphanumeric text strings are `summingJunction`, `case`, `display`, and `merge` blocks.

Matrix data: You can output matrix data from a `const` block. Commsim indicates that the block contains matrix data by using a vector wire when attaching the `const` block to another block.



const Properties

Value: 1

Label:

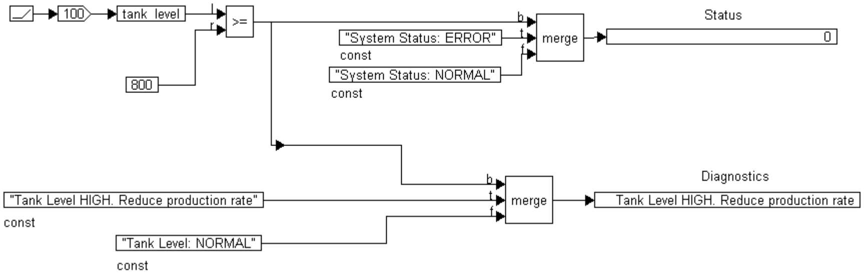
Value: Indicates the value of the output signal. The default is 1.

When entering alphanumeric text strings, enclose them in quotation marks. When entering matrix data, enclose them in square brackets and separate each element with a space. To start a new row, use a semi-colon.

Label: Indicates a user-defined block label.

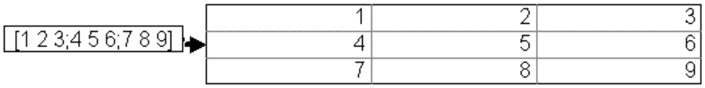
Examples

1. Conditional alarm using const, merge, and display blocks



A tank level is monitored using two `merge` blocks and four `const` blocks. The top `merge` block tracks the tank level. The simulation operates normally until the tank level reaches or exceeds 800, at which point an ERROR is displayed. The lower `merge` block indicates the necessary operator intervention.

2. Creation of a 3 x 3 matrix



1	2	3
4	5	6
7	8	9

9.27 constraint

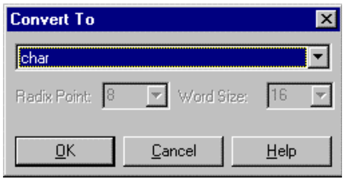
Block Category: Optimization

The `constraint` block is used to solve an implicit equation.

9.28 convert

Block Category: Arithmetic

The `convert` block converts the data type of the input signal to one of the following: char, unsigned char, short, unsigned short, int, long, unsigned long, float, double, matrix double, scaled int, or string. To check for overflow errors, activate Warn Numeric Overflow under the Preferences tab in the dialog box for the Simulate > Simulation Properties command.


Convert To

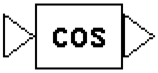
char

Radix Point: 8 Word Size: 16

Radix Point: This parameter lights up only when Scaled Int is selected. It represents the number of bits provided for the integral part of the number. The difference between the word size and the radix point represents the mantissal (or fractional part of the number).

Word Size: Specifies the word size in bits.

9.29 cos



$y = \cos x$

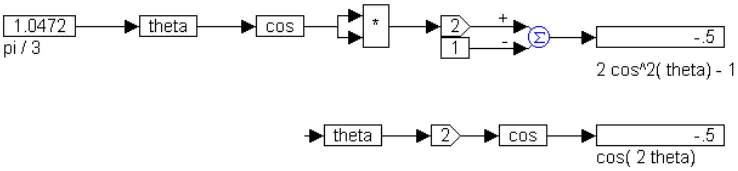
Block Category: Transcendental

The `cos` block produces the cosine of the input signal. The input signal must be represented in radians.

Examples

1. Computation of cos(2θ) = 2cos^2(θ) - 1

With θ chosen to be π/3, the above trigonometric identity can be realized as:



9.30 cosh

$$y = \frac{e^{x} + e^{-x}}{2}$$

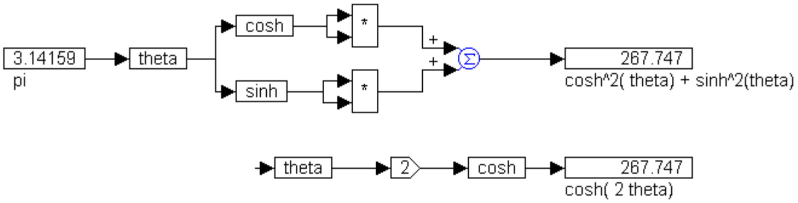
Block Category: Transcendental

The `cosh` block produces the hyperbolic cosine of the input signal. The input signal must be represented in radians.

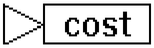
Examples

1. Computation of cosh(2θ) = cosh^2(θ) + sinh^2(θ)

With θ chosen to be π, the above trigonometric identity can be realized as shown below.



9.31 cost



Block Category: Optimization

The `cost` block measures the cost function for parameter optimization.

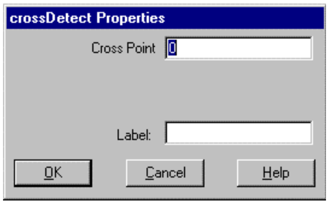
9.32 crossDetect

crossDetect

$$y = \begin{cases} -1\text{if } \; x \text{ crosses crosspoint with neg. slope} \\ 1\text{if } x\text{crossescrosspointwithpos.slope} \\ 0\text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `crossDetect` block monitors its input value and compares it with a user-specified crosspoint. When the input value crosses the crosspoint, the `crossDetect` block outputs either +1 or -1, depending on whether the crossover occurred with a positive slope or negative slope, respectively. If a crossover is not detected, the `crossDetect` block outputs 0.


crossDetect Properties

Cross Point: 0

Label:

Cross Point: Represents the value that, when *x* crosses it, causes the output signal to go to 1, -1, or 0. The default is 0.

Label: Indicates a user-defined block label.

Examples

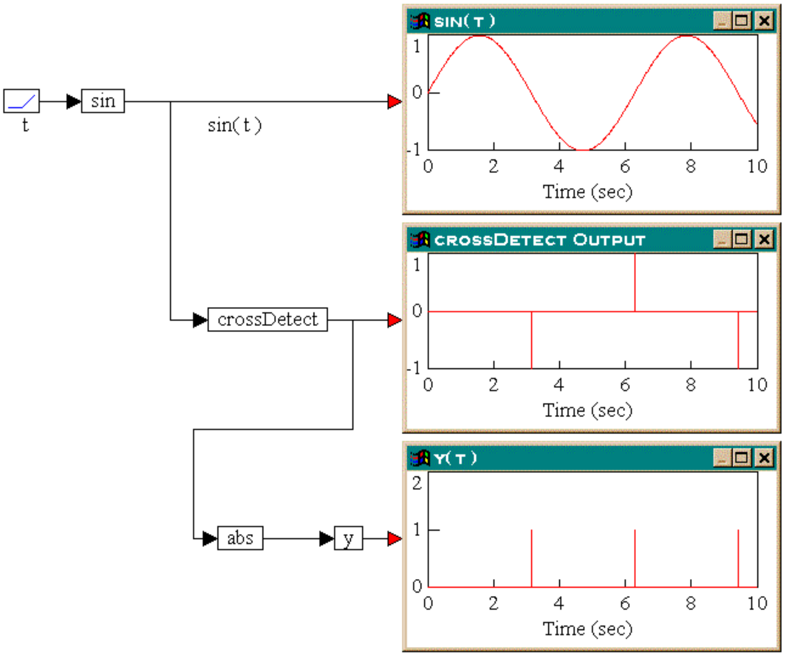
To obtain correct results from the examples described below, increase the point count for the `plot` blocks to at least 1,000.

1. Detection of zero crossover of a sinusoid

Consider the equation:

$y = 1$ if $\sin(t) = 0$, else $y = 0$

This equation can be realized as shown in the diagram below.



As can be seen from the `crossDetect` block output, three 0 crossings are detected in the simulation. The first and third 0 crossings occur with negative slope (that is, the value of sin(*t*) is decreasing, as it approaches zero), while the second 0 crossing occurs with positive slope (that is, the value of sin(*t*) is increasing as it approaches zero.) Consequently, the first and third 0 crossing events are -1, and the second 0 crossing event is + 1.

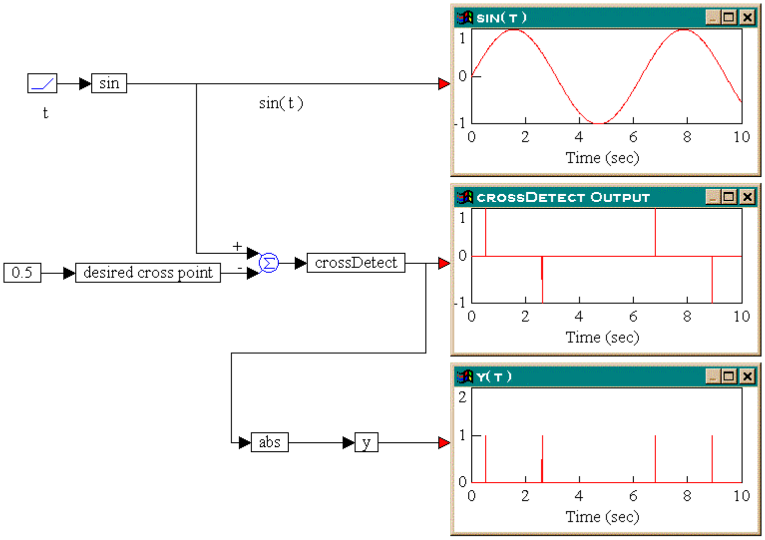
However, since *y* is required to be equal to +1 whenever sin(*t*) = 0, irrespective of the slope, the output of the `crossDetect` block is passed through an `abs` block to extract the absolute value, and this output is defined as the `variable` *y*. The bottom `plot` block shows that *y* = 1 when sin(*t*) = 0; otherwise *y* = 0.

2. Detection of non-zero crossover with externally set crosspoint

Consider the equation:

y = 1 if sin(*t*) = 0.5, else *y* = 0

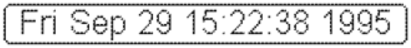
This equation can be realized exactly as above by setting the internal crosspoint on the `crossDetect` block to 0.5. Unfortunately, this may not be acceptable in some cases, particularly when the crosspoint itself is to be computed as a part of the simulation. In such cases, the crosspoint must be set externally, as shown in the diagram below.



The key difference here is that the output of the `sin` block is connected to a `summingJunction` block, which computes the difference between sin(*t*) and a `variable` called *desired cross point*. This difference is connected to the `crossDetect` block, which has an internal crosspoint of 0.

In effect, a non-zero crossover detection problem is converted to a 0 crossover detection problem. That is, the problem of $y = 1$ when sin(*t*) = 0.5 is converted to $y = 1$ when sin(*t*) - 0.5 = 0. The rest of the diagram is identical to the previous one.

9.33 date

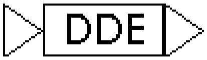


Fri Sep 29 15:22:38 1995

Block Category: Annotation

The `date` block displays the current date and time. The date and time are updated when you move a block, print the diagram, or repaint the screen. If you need to reset the time or date, use the system Control Panel. For more information, see the *Microsoft Windows User's Guide*.

9.34 DDE



Block Category: DDE

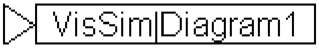
The `DDE` block exchanges information with another Windows application. Use this block when you want to create a link that sends information to and receives information from another application. You can create links between Commsim and other applications that support DDE.

9.35 DDEreceive

Block Category: DDE

The `DDEreceive` block creates a DDE link that passes information from a Windows application (referred to as the source or server) into a block diagram (referred to as the destination or client).

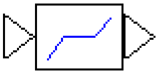
9.36 DDEsend



Block Category: DDE

The `DDEsend` block creates a DDE link that passes information from a block diagram (referred to as the source or server) to another Windows application (referred to as the destination or client).

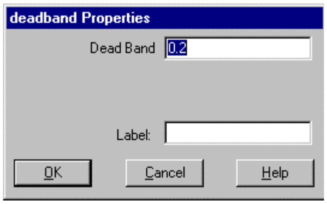
9.37 deadband



$$y = \begin{cases} 0 & \text{if } |x| \le \dfrac{\text{deadband}}{2} \\ x - \left(\text{sign}(x)\dfrac{\text{deadband}}{2}\right) & \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The deadband block produces an output signal, which is the input signal reduced by a zone of lost motion about the signal's 0 value. Use this block to simulate play in mechanical systems, such as gears or chains.



deadband Properties

Dead Band 0.2

Label:

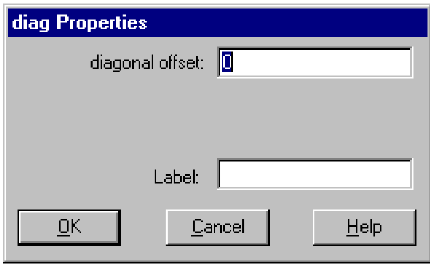
Dead Band: Indicates the width of the zone of lost motion about the input signal's 0 value. The default is 0.2.

Label: Indicates a user-defined block label.

9.38 diag

Block category: Matrix Operations

The diag block creates a square matrix with the vector V on the diagonal and zeros elsewhere. You can specify an offset from the main diagonal.



diag Properties

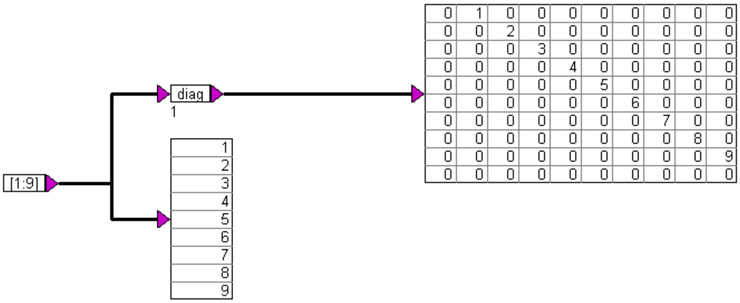
diagonal offset: 0

Label:

Diagonal Offset: Indicates the offset from the main diagonal. The default is 0, which puts vector V on the main diagonal.

Label: Indicates a user-defined block label.

Examples

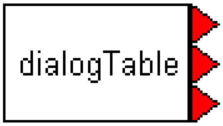


0	1	0	0	0	0	0	0	0	0
0	0	2	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	0
0	0	0	0	4	0	0	0	0	0
0	0	0	0	0	5	0	0	0	0
0	0	0	0	0	0	6	0	0	0
0	0	0	0	0	0	0	7	0	0
0	0	0	0	0	0	0	0	8	0
0	0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	0	0

1
2
3
4
5
6
7
8
9

The `diag` block produces a 10-by-10 matrix. The 9-element vector is placed on the diagonal with an offset of 1. Commsim pads the remaining cells with zeros.

9.39 dialogConstant



Block Category: Signal Producer

The `dialogTable` block lets you access lists of data specific to individual items in a single dialog box.

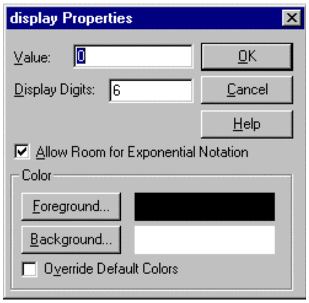
9.40 display

$$\text{display} = x_1$$

Block Category: Signal Consumer

The `display` block displays the current value of the input signal in any number of significant digits. You can select a color for the displayed value, as well as a background color for the block.

The `display` block automatically expands to display vector and matrix elements in individual cells. To display alphanumeric text strings in their entirety, you may have to increase the value in the Display Digits box.


display Properties

Value: 0

Display Digits: 6

☑ Allow Room for Exponential Notation

Color

□ Override Default Colors

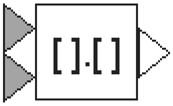
Value: Controls the current value in the display. The default is 1.

Display Digits: Indicates the number of displayed significant digits. The value you enter overrides the setting of the High Precision Display parameter under Preferences in the dialog box for the Edit > Preferences command. The default is 6.

Allow Room For Exponential Notation: Expands the display block so there is room for exponential notation. If you wish to have a very small display block, perhaps for use in display mode, you should turn off this option.

Color: Applies background and foreground color to the display block. Click on the Foreground and Background buttons to select a color. The selected colors are displayed to the right of the buttons. To override the background color selected using the View > Colors command, activate Override Default Color.

9.41 dotProduct



$$y = \sum_{K=1}^{N} x1_K \times x2_K$$

Block Category: Matrix Operations

The dotProduct block produces a single value summation of an element-by-element multiply. The dotProduct block accepts two vector inputs and produces a scalar output. If the input vectors have an uneven number of elements, an error occurs.

Multiplying scalars and matrices

To multiply two or more scalars, use the * block.

To multiply two matrices, use the multiply block.

9.42 embed

The embed block lets you embed a multi-level block diagram in the current block diagram.

9.43 error

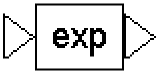
$$\text{error condition} = x_1$$

Block Category: Signal Consumer

The `error` block flags an error in a simulation. When the input signal becomes non-zero, the `error` block and all compound blocks which contain it are highlighted in red and the simulation is stopped.

You can reset the error condition by clicking the right mouse button on the `error` block.

9.44 exp



$$y = \mathrm{e}^x$$

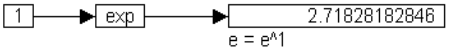
Block Category: Transcendental

The `exp` block performs the inverse operation of the `ln` block and raises the input as a power of *e*. The irrational number *e* is the base of natural logarithms and is approximately equal to 2.7182828.

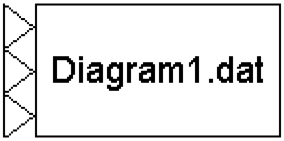
Examples

1. Computation of the value of *e*

The value of *e* can be obtained by providing an input value of 1 to an `exp` block as:



9.45 export

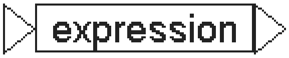


$$\text{data file column}_n = x_n$$

Block Category: Signal Consumer

The `export` block writes signals to a file in .DAT, .M, .MAT, or .WAV file format. The file can subsequently be used as input to Commsim or to a variety of other programs, such as MatLab and Microsoft Excel.

9.46 expression

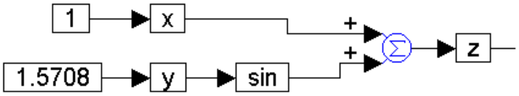


The `expression` block allows you to enter a C expression or matrix data that Commsim parses and acts upon.

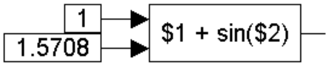
With expressions, you can significantly reduce the number of blocks in your diagrams. For example, consider the simple equation:

$$x + \sin(y) = z$$

Without the `expression` block, the block diagram representation of this equation is:



Instead of using the `variable`, `sin`, and `summingJunction` blocks, you can create a single C expression that performs the same function:



The elements $1 and $2 are Commsim-specific notation that reference the inputs.

What you can do with expression blocks:

- Speed up simulation time

  The more blocks in a diagram, the longer it takes to simulate to the diagram. Consequently, as you replace series of blocks with `expression` blocks, the simulation time decreases.

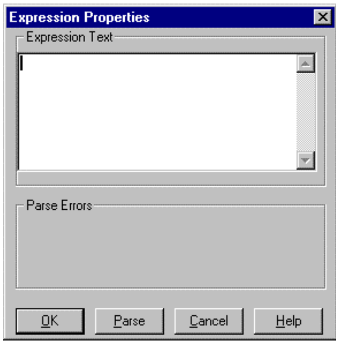
- Reduce development time

  Instead of inserting groups of blocks and wiring them together, you can insert a single `expression` block that performs the same function.

- Simplify troubleshooting

  You can request that Commsim check the logic of the expression before you simulate the diagram. You simply press a key and Commsim does the rest.

Writing an expression: The Expression Properties dialog box lets you set up your expression.



Expression Text: Indicates a C expression. A C expression consists of one or more operands and zeros or more operators linked together to compute a value. You enter expressions according to the syntax rules for the C language. If you’re unfamiliar with the language, refer to *C: A Software Engineering Approach*, (Springer-Verlag, 1990).

The following Commsim-specific rules apply to entering C expressions:

- Inputs are referenced using the notation $*n*, where *n* represents an connector number. For example, $1 is input 1 (the top input connector), $2 is input 2 (the second from the top input connector), and so on.

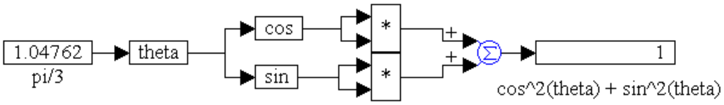
- Only one output value is allowed.

Parse Errors: Lists the errors that occur when Commsim parses the C expression. This is a read-only box.

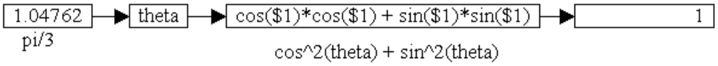
Examples

1. Computation of $\cos^2(\theta) + \sin^2(\theta) = 1$

If θ is chosen to be π/3, the above expression can be realized as:



The same equation can be realized using an `expression` block as:



1.04762
pi/3
theta
cos($1)*cos($1) + sin($1)*sin($1)
1
cos^2(theta) + sin^2(theta)

Here, the expression $\cos^2(\theta) + \sin^2(\theta)$ is entered directly into the `expression` block as cos($1) * cos($1) + sin($1) * sin($1), where $1 corresponds to the only input on the `expression` block. When the simulation runs, Commsim substitutes $1 in the expression with the top input connected to it and then evaluates the expression.

From the results obtained, both methods yield the correct answer.

9.47 fft

Block Category: Matrix Operation

The `fft` block converts data from time domain to frequency domain.

The `fft` block computes an *n*-sample FFT at every simulation time step, where *n* is the length of the input vector.

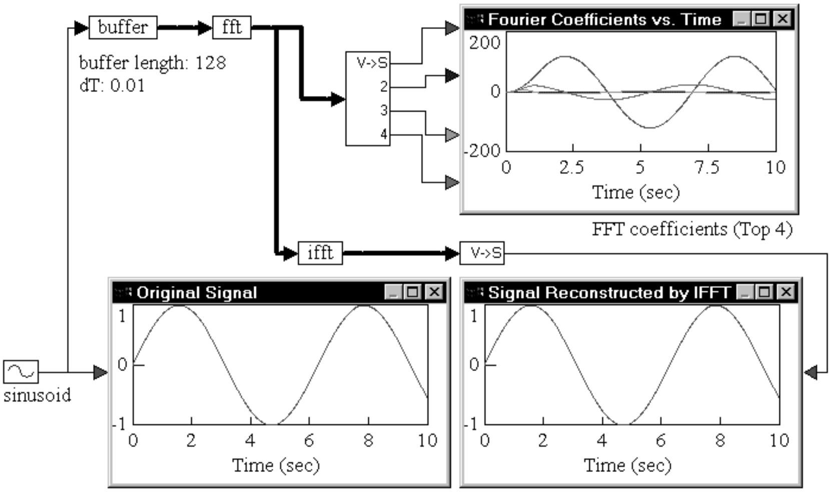
If the input to the `fft` block is not an integral power of 2, automatic zero padding is performed to make the input vector size an integral power of 2. This is a standard procedure in

FFT computation. The output of the fft block is Fourier coefficients. Individual coefficients can be accessed using a vecToScalar block.

Examples

1. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using inverse FFT.



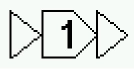
A sinusoid block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a buffer block of length 128 samples and a dT of 0.01. The output of the buffer block is connected to an fft block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the fft block is Fourier coefficients. The individual coefficients are accessed using a vecToScalar block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the fft block to an ifft block to compute the inverse FFT. The output of the ifft block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the ifft output vector does not have any delay because zero time has elapsed between the FFT and inverse FFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and inverse FFT algorithms.

9.48 gain

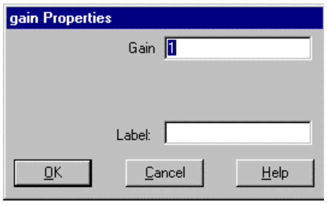


$y = x \cdot \text{gain}$

Block Category: Arithmetic

The `gain` block multiplies the input signal, by the gain amount. The input can be a scalar, vector, or matrix.

You can set initial conditions on `gain` blocks by using variable names. During simulation, the gain value is evaluated only once at the start of the simulation.



gain Properties

Gain: 1

Label:

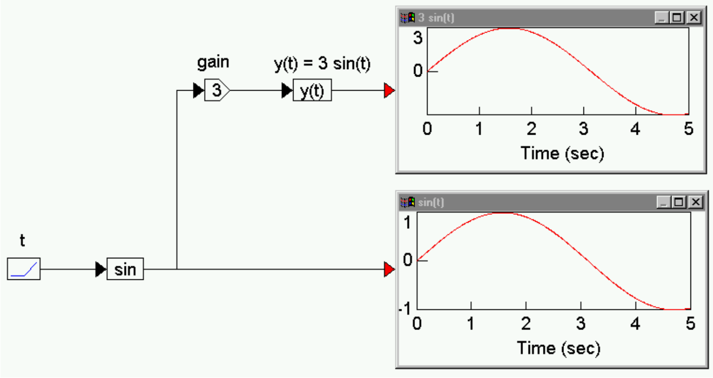
Gain: Indicates the constant multiplier of the input signal. The default is 1.

Label: Indicates a user-defined block label.

Examples

1. Gain of a scalar

Consider the equation $y(t) = 3 \sin(t)$, which can be realized as:



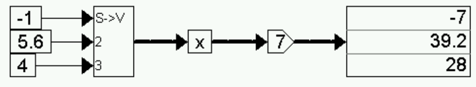
A `ramp` block is used to access simulation time *t*, a `sin` block generates sin(*t*), a `gain` block amplifies sin(*t*) to 3 sin(*t*). Both sin(*t*) and y(*t*) are shown in `plot` blocks for comparison.

2. Gain of a vector

Consider the equation:

$\mathbf{z} = 7\,\mathbf{x}$

where $\mathbf{x} = [-1\ \ 5.6\ \ 4]$. This equation can be realized as:



-7
39.2
28

The `gain` block performs an element-by-element gain operation on the incoming vector.

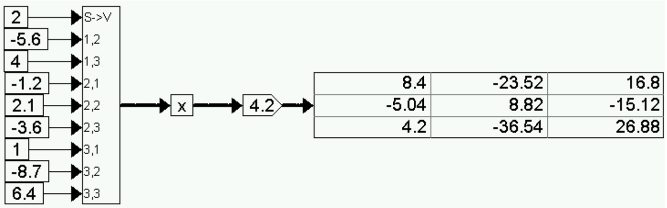
3. Gain of a matrix

Consider the equation:

$\mathbf{Z} = 4.2\,\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

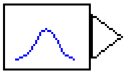
This equation can be realized as:



8.4	-23.52	16.8
-5.04	8.82	-15.12
4.2	-36.54	26.88

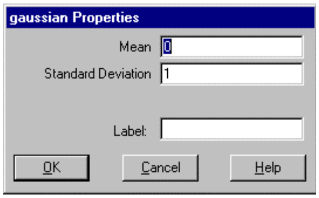
The `gain` block performs an element-by-element gain operation on the incoming matrix.

9.49 gaussian



Block Category: Random Generator

The `gaussian` block creates a normally distributed, random noise signal. You specify a random seed value under the Preferences tab in the dialog box for the Simulate > Simulation Properties command.



gaussian Properties

Mean: 0

Standard Deviation: 1

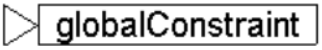
Label:

Mean: Indicates the center of the distribution. The default value is 0.

Standard Deviation: Indicates the distance from the mean, which covers one standard deviation. The default value is 1.

Label: Indicates a user-defined block label.

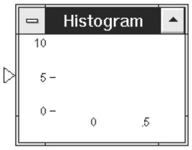
9.50 globalConstraint



Block Category: Optimization

The `globalConstraint` block provides side constraint information when writing your own global optimizer.

9.51 histogram



Block Category: Signal Consumer

The `histogram` block shows how data are distributed over the course of a simulation.

9.52 ifft

Block Category: Matrix Operation

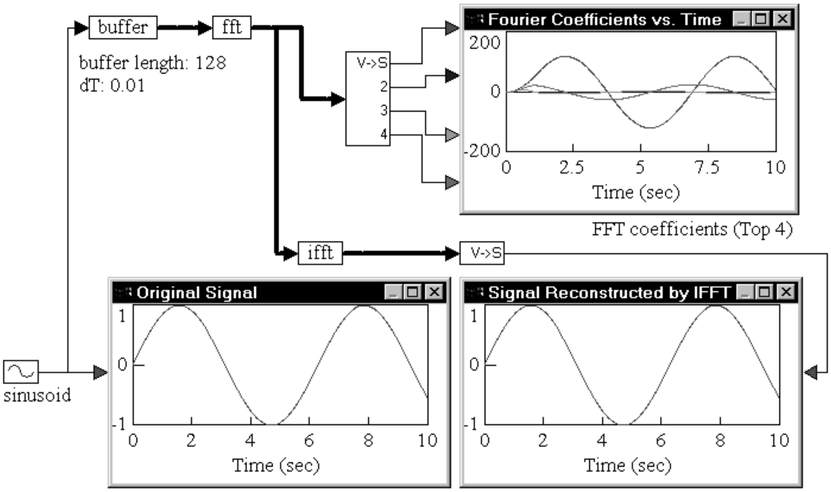
The `ifft` block converts data from frequency domain to time domain. The `ifft` block computes an *n*-sample inverse FFT at every simulation time step, where *n* is the length of the input vector.

If the input to the `ifft` block is not an integral power of 2, automatic zero padding is performed to make the input vector size an integral power of 2. This is a standard procedure in inverse FFT computation. The output of the `ifft` block is Fourier coefficients. Individual coefficients can be accessed using a `vecToScalar` block.

Examples

1. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using inverse FFT.



A `sinusoid` block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a `buffer` block of length 128 samples and a sampling rate of 0.01. The output of the `buffer` block is connected to an `fft` block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the `fft` block is Fourier coefficients. The individual coefficients are accessed using a `vecToScalar` block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the `fft` block to an `ifft` block to compute the inverse FFT. The output of the `ifft` block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the `ifft` output vector does not have any delay because zero time has elapsed between the FFT and inverse FFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and inverse FFT algorithms.

9.53 import

Diagram1.dat

$$y_n = \text{datafile column}_n$$

Block Category: Signal Producer

The `import` block imports data points from a .DAT, .M, .MAT, or .WAV file and translates them into output signals. The data can be either fixed interval or asynchronous.

9.54 index

Block Category: Annotation

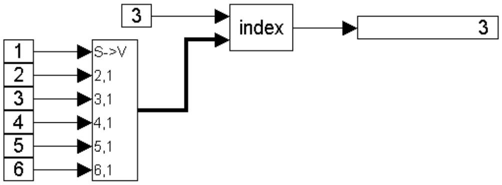
The index block allows you to address a single element within a matrix. The index block can have two or three inputs. When there are two input, the top input specifies the matrix row (the matrix column is assumed to be 1), and the bottom input is the matrix. When there are three inputs, the top input specifies the matrix row, the middle input specifies the matrix column, and the bottom input is the matrix.

If you address an element outside the bounds of the matrix, the index block displays the following warning message:

Index *n* too big.

Index n is either 1 or 2, depending on which index exceeds the bounds first.

For example, a 6 × 1 vector fed into the `index` block yields six possible execution paths, as shown below.



The `index` block outputs 1, 2, 3, 4, 5, or 6 depending on whether the index value is 1, 2, 3, 4, 5, or 6, respectively. In this example, the index value is 3, causing a 3 to be output.

It is important to know how an index value references matrix elements. Index values map to matrix elements in sequential order, starting with the element in *column*1-*row*1, through *column*1-*rowN*; then *column*2-*row*1 through *column*2-*rowN*; and so on. For example, in the following 2 × 3 matrix, an index value of 3 yields 5:

INDEX VALUE 1: 1	INDEX VALUE 4: 2
INDEX VALUE 2: 3	INDEX VALUE 5: 4
INDEX VALUE 3: 5	INDEX VALUE 6: 6

In a 3 × 2 matrix, an index value of 3 yields 2:

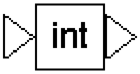
1	2	3
4	5	6

Index value: The following rules apply to the index value:

- Index values that are non-integers are truncated. For example, 0.999 is truncated to 0.
- If the index value targets an unconnected matrix or vector element, the `index` block outputs a 0.

- If the index value targets an out-of-range matrix or vector element, the `index` block outputs spurious results. For example, if the index value is 5 for a four-element matrix, the output, might look something like this: 1.06983e-306.

9.55 int



$$y = \text{integer part } x$$

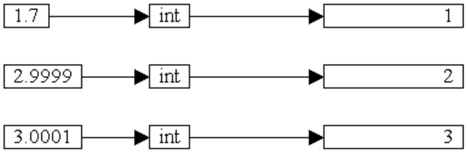
Block Category: Nonlinear

The `int` block accepts a scalar input and outputs only the integer portion of the input. The `int` block does not perform numerical round-off operations. Thus, an input of 2.9999 yields 2. Inputs can be scalar constants or scalar variables.

Examples

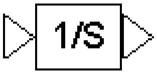
1. Integer portions of scalar inputs

Consider three scalar inputs 1.7, 2.9999, and 3.0001. These inputs are applied to the `int` blocks, as shown below:



The `int` blocks isolate and output the integer portion of the scalar inputs.

9.56 integrator (1/S)



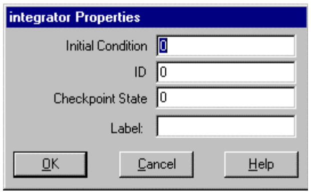
$$y = \int_{t_{\text{start}}}^{t_{\text{end}}} x\,dt$$

Block Category: Integration

The `integrator` block performs numerical integration on the input signal using the integration algorithm (Euler, trapezoidal, Runge Kutta 2d and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff)) established with the Simulate > Simulation Properties command.

The `integrator` block is one of the most fundamental and powerful blocks in Commsim. This block, together with the `limitedIntegrator` and `resetIntegrator` blocks, allow you to solve an unlimited number of simultaneous linear and nonlinear ordinary differential equations.

You can set initial conditions on `integrator` blocks by using variable names. You can also reset the `integrator` block to its initial state using the Simulate > Reset States command.



integrator Properties

Initial Condition: 0

ID: 0

Checkpoint State: 0

Label:

Initial Condition: Indicates the initial value of the integrator. The default value is 0.

ID: Represents an identification number for the block. It keeps track of the state number that Commsim assigns to the integrator. The number of states in any block diagram equals the number of integrators. The default value is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation using the Simulate > Simulation Properties command, the value is 0.

Label: Indicates a user-defined block label.

Examples

1. Solving a first order ODE

Consider the simple first order linear differential equation:

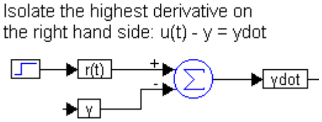
$$\dot{y} + y = r(t)$$

where *r*(*t*) is an external input. In this case, assume that the external input to the system is a step function. In Commsim, such equations are best solved by numerical integration.

The first step is to isolate the highest derivative term on one side. To understand the procedure better, it is easier to think of isolating the highest derivative term on the right-hand side as:

$$r(t) - y = \dot{y}$$

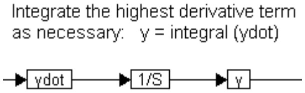
This equation can be constructed as:



Isolate the highest derivative on
the right hand side: u(t) - y = ydot

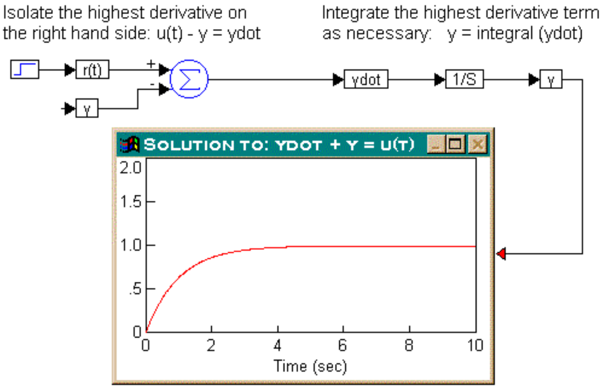
Here, three `variable` blocks are used for *r*(*t*), *y*, and *ydot*.

The second step is to integrate the highest derivative term a sufficient number of times to obtain the solution. Since the highest derivative is of first order, *ydot* must be integrated once to obtain *y*. This can be realized as:



Integrate the highest derivative term
as necessary: y = integral (ydot)

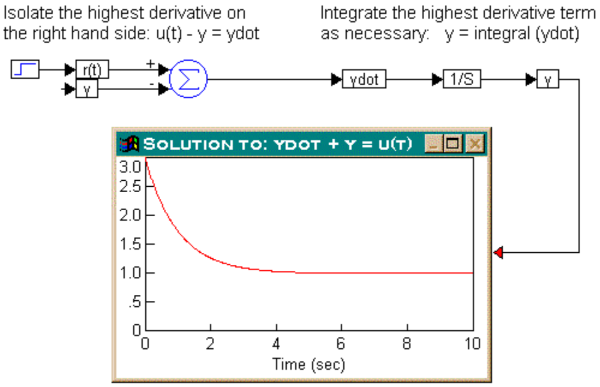
The overall simulation is shown below.


Isolate the highest derivative on
the right hand side: u(t) - y = ydot
Integrate the highest derivative term
as necessary: y = integral (ydot)
r(t)
+
-
y
ydot
1/S
y
SOLUTION TO: YDOT + Y = U(T)
2.0
1.5
1.0
.5
0
0
2
4
6
8
10
Time (sec)

The result shown in the `plot` block indicates the solution of the differential equation subjected to a step external forcing function.

2. Setting the integrator initial condition internally

Consider the same problem above with the assumption that *y*(0) = 3. In this case, in addition to the external input *r*(*t*), the system response also depends on *y*(0). This initial condition can be set directly in the `integrator` block. The result for this case is:


Isolate the highest derivative on
the right hand side: u(t) - y = ydot
Integrate the highest derivative term
as necessary: y = integral (ydot)
r(t)
+
-
y
ydot
1/S
y
SOLUTION TO: YDOT + Y = U(T)
3.0
2.5
2.0
1.5
1.0
.5
0
0
2
4
6
8
10
Time (sec)

The `plot` block shows that the response *y*(*t*) begins at *y*(0) = 3 and settles down to 1 for *t* > 4.5, as expected.

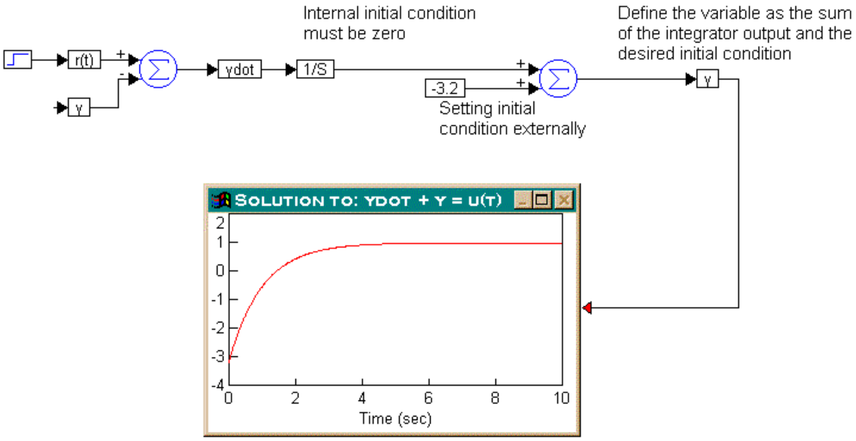
It is important to note that the initial condition on any state (or variable) must be set on the `integrator` block that is generating that state. (This concept becomes clearer in Example 4.)

3. Setting the integrator initial condition externally

Consider once again the following ordinary differential equation:

$$\dot{y} + y = r(t)$$

Let $r(t)$ be a step function and assume that $y(0)$ = -3.2. The initial condition can be set externally, as below.



Internal initial condition
must be zero
Define the variable as the sum
of the integrator output and the
desired initial condition
r(t)
+
-
y
ydot
1/S
+
+
-3.2
Setting initial
condition externally
y
SOLUTION TO: YDOT + Y = U(T)
2
1
0
-1
-2
-3
-4
0
2
4
6
8
10
Time (sec)

In this configuration, make sure that the internal initial condition of the `integrator` is set to zero. By default, all `integrators` have zero initial condition.

The results indicate that the solution of the ordinary differential equation, subject to the external input and the initial conditions, is computed correctly.

4. Second order nonlinear ODE with external initial conditions

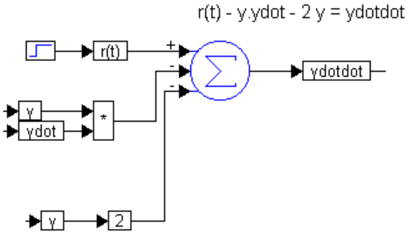
Consider a second order nonlinear system given by:

$$\ddot{y} + y\dot{y} + 2y = r(t)$$

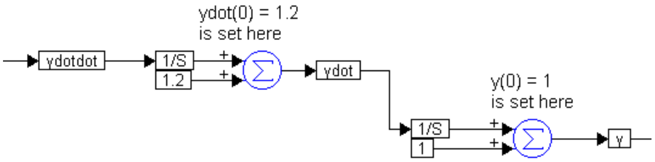
Furthermore, assume that $r(t)$ is a unit step function and that the initial conditions are given by:

$$y(0) = 1.0 \quad \text{and} \quad \dot{y}(0) = 1.2$$

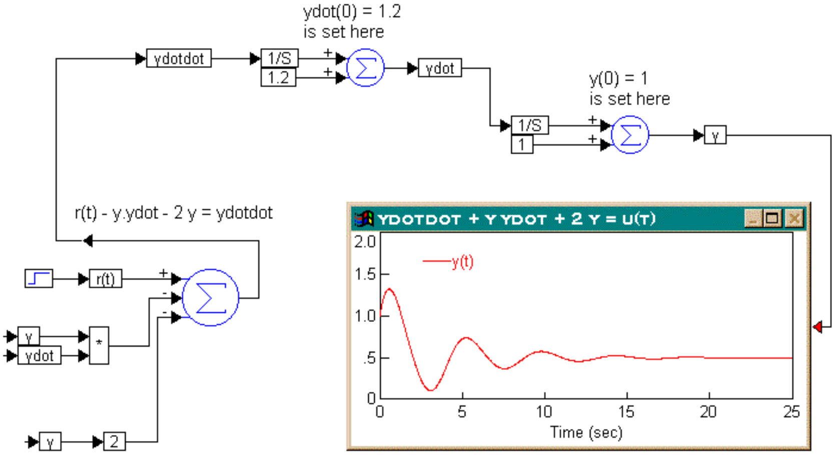
The first step is to isolate the highest derivative term on the right-hand side as $r(t) - y\dot{y} - 2y = \ddot{y}$. This segment can be coded in Commsim as shown below:



The second step is to integrate *ydotdot* twice: once to generate *ydot*, and once more to generate *y*. As can be imagined, it is crucial to maintain consistent variable names throughout. Furthermore, the initial conditions must be added using the same procedure described in Example 3. This segment can be realized as:



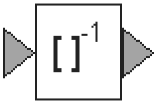
The complete solution for this problem is given by:



The solution of the equation, *y*(*t*) is shown in the `plot` block.

This example illustrates the real power of numerical integration using Commsim. If you want to use the results of a computational segment in a given Commsim diagram as initial conditions for one or more integrators, replace the `const` blocks with appropriate `variable` blocks when setting the external initial conditions.

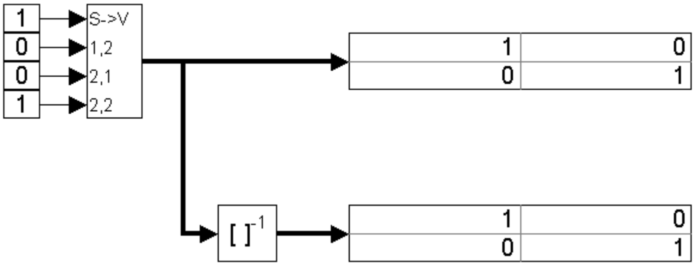
9.57 invert



$$[\mathbf{A}]^{-1} = \frac{adj(\mathbf{A})}{det(\mathbf{A})}$$

Block Category: Matrix Operation

The `invert` block inverts a square matrix using singular value decomposition. The `invert` block accepts one vector input and produces one vector output.

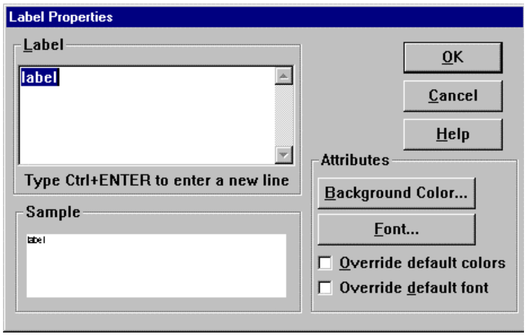


9.58 label

Block Category: Annotation

The `label` block lets you insert floating labels in a block diagram. You can choose the text attributes for the label, as well as a colored background. The `label` block is particularly useful for tagging signals.

Most block dialog boxes let you enter a label for the block. To display these labels, turn on the View > Block Labels command.



Label Properties

Label: label

Type Ctrl+ENTER to enter a new line

Sample: label

□ Override default colors

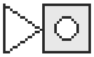
□ Override default font

Label: Specifies a label. To continue a label to a new line, hold down the CTRL key while you simultaneously press the ENTER key.

Attributes: Assigns a background color and text attributes to the label. Click on the Background Color button to select a background color for the label. Click on the Fonts button to select a font, font style, point size, color, and special effects for the text. A sample of the text is displayed in the Sample box.

To override the selections in the View > Colors and View > Fonts dialog boxes, activate Override Default Colors and Override Default Font, respectively.

9.59 light



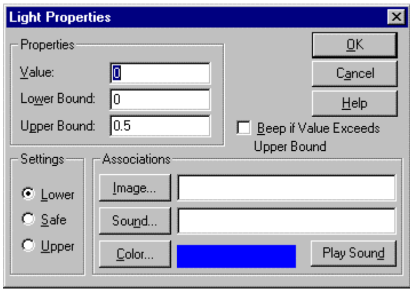
$$y = \begin{cases} \text{red if } x_1 > \text{ub} \\ \text{green if lb} \le x_1 \le \text{ub} \\ \text{blue if } x_1 < \text{lb} \end{cases}$$

Block category: Signal Consumer

The `light` block is a tri-state alarm that glows a color, displays a bitmap image, or plays sound when supplied with a signal. By default, the `light` block glows red when the signal is greater than the upper bound; blue when the signal is less than the lower bound; and green when the signal is less than or equal to the upper bound and greater than or equal to the lower bound.

Associating an action with a state: To associate an action — for example, the display of a bitmap image file — with a given state, select the state from the Settings box; then click on the Bitmap button and choose the .BMP file to be associated with the state.

Setting up a light block: The `light` block's Properties dialog box lets you control its audio and visual alarms.



Light Properties

Properties

Value: 0

Lower Bound: 0

Upper Bound: 0.5

□ Beep if Value Exceeds Upper Bound

Settings

● Lower

○ Safe

○ Upper

Associations

Properties: Establishes the lower and upper bounds for the signal, as well as the initial setting of the signal.

- **Value:** Indicates the initial setting for the signal. The default is 0.
- **Lower Bound:** Indicates the lower bound for the signal. When the signal is less than the specified lower bound, the `light` block performs the action (emits a color, sound, or image) associated with the Lower setting. The default is 0.
- **Upper Bound:** Indicates the upper bound for the signal. When the signal is greater than the specified upper bound, the `light` block performs the action (emits a color, sound, or image) associated with the Upper setting. The default is 0.5.

Settings: Indicates the setting to which color, sound, or an image is to be applied.

- **Lower:** The signal is less than the specified lower bound.
- **Safe:** The signal is less than or equal to the specified upper bound and greater than or equal to the specified lower bound.
- **Upper:** The signal is greater than the specified upper bound.

Associations: Indicates whether an image, sound, or color is to be applied to the specified setting.

- **Image:** Opens the File Select dialog box in which to choose a .BMP file to associate with the selected setting.
- **Sound:** Opens the File Select dialog box in which to choose a .WAV file to associate with

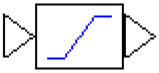
the selected setting.

- **Color:** Opens the Color dialog box in which to choose a color to associate with the selected setting.

Play Sound: Plays the sound for the selected setting.

Beep If Value Exceeds Upper Bound: Forces the `light` block to beep when the signal exceeds the specified upper bound.

9.60 limit

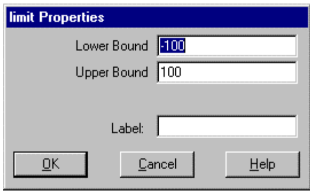


$$y = \begin{cases} x_1 \ \text{if} \ \text{lb} \le x_1 \le \text{ub} \\ \text{lb} \ \text{if} \ x_1 < \text{lb} \\ \text{ub} \ \text{if} \ x_1 > \text{ub} \end{cases}$$

Block Category: Nonlinear

The `limit` block limits the output signal to a specified upper and lower bound. The `limit` block accepts a scalar input. If the input is less than the lower bound, the `limit` block limits the output to the lower bound. Similarly, if the input is greater than the upper bound, the `limit` block limits the output to the upper bound. If the input falls within the specified bounds, the input is transferred to the output unchanged.

The `limit` block is particularly useful for simulating variables or processes that reach saturation.



limit Properties

Lower Bound: -100

Upper Bound: 100

Label:

Lower Bound: Indicates the lowest value that the output signal can attain. The default is -100.

Upper Bound: Indicates the highest value the output signal can attain. The default is 100.

Label: Indicates a user-defined block label.

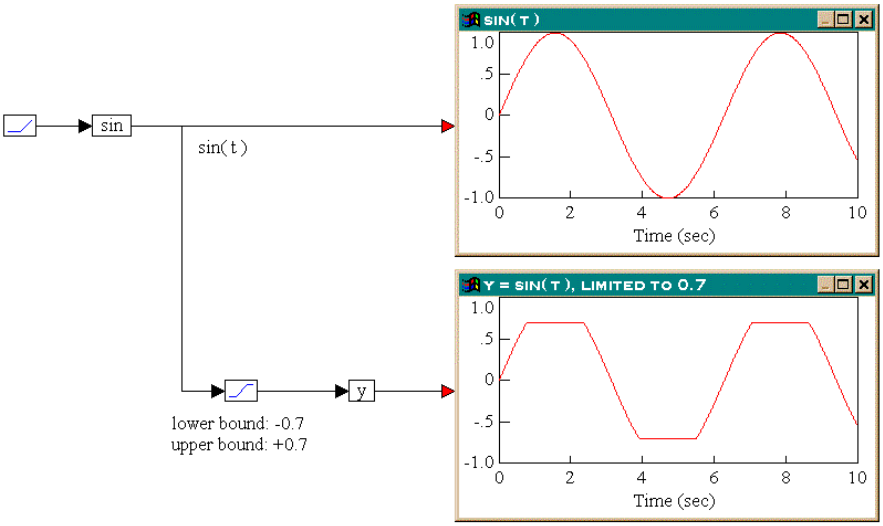
Examples

1. Simulation of saturation

Consider a variable *y* such that:

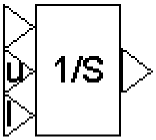
$$y = \sin(t)$$

Furthermore, assume that *y* reaches saturation at +0.7 and -0.7. This equation can be realized as shown below.



From the results in the two `plot` blocks, the output of the `limit` block is identical to the input, when the input is within the bounds (-0.7 to +0.7). When the input is out of these bounds, the output is limited to the upper or lower bound values.

9.61 limitedIntegrator (1/S)



$$y = \begin{cases} \int_{t_{\text{start}}}^{t_{\text{end}}} \mathrm{x}_1 dt \text{ if } x_2 \geq \int_{t_{\text{start}}}^{t_{\text{end}}} x_1 dt \geq x_3 \\ x_2 \quad \text{if } \int_{t_{\text{start}}}^{t_{\text{end}}} x_1 dt > x_2 \\ x_3 \quad \text{if } \int_{t_{\text{start}}}^{t_{\text{end}}} x_1 dt < x_3 \end{cases}$$

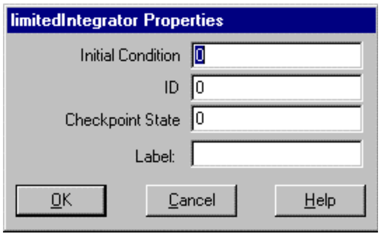
Block Category: Integration

The `limitedIntegrator` block integrates the input value and limits the internal state to specified upper and lower limits. If the integral state reaches its limit, it backs off the limit as soon as the derivative changes sign. You set the integration algorithm with the Simulate > Simulation Properties command. Available algorithms are Euler, trapezoidal, Runge Kutta 2nd and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff).

The inputs to the block are x_1, the derivative; x_2 (U), the upper limit; and x_3 (L), the lower limit.

You can set initial conditions on limited`Integrator` blocks by using variable names. You can also reset the `limitedIntgrator` block to its initial state using the Simulate > Reset States command.

The `limitedIntegrator` block is used in the prevention of wind-up in PI and PID controllers in control applications. It is also used in kinematics, electrical circuits, process control, and fluid dynamics.



limitedIntegrator Properties

Initial Condition: 0

ID: 0

Checkpoint State: 0

Label:

Initial Condition: Indicates the initial value of the integrator. The default is 0.

ID: Represents an identification number for the block, which holds the state number that Commsim assigns to the integrator. The number of states in any block diagram equals the number of integrators. The default is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation using the Simulate > Simulation Properties command, the value is 0.

Label: Indicates a user-defined block label.

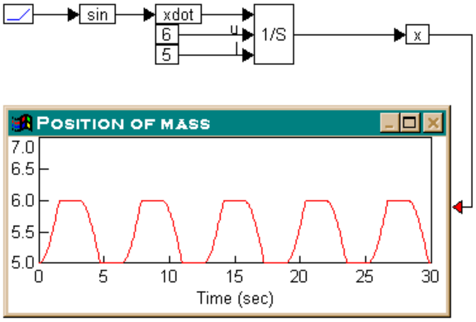
Examples

1. Integration with constant limits

Consider a system whose dynamics are given by the differential equation:

$$\dot{x} = \sin(t)$$

Furthermore, assume that x must lie in the limits $5 \le x \le 6$ and that $x(0) = 5$. This system can be realized as shown below.



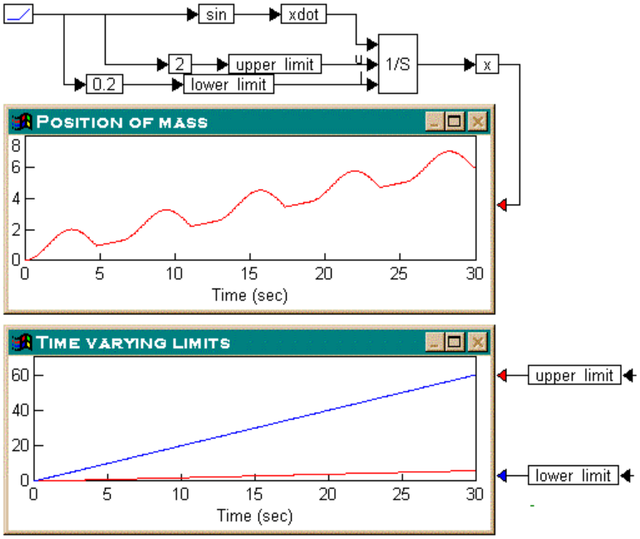
During simulation, the `limitedIntegrator` block limits the output to be within the upper and lower limits, namely 6 and 5, respectively.

2. Integration with time-varying limits

Consider a system whose dynamics are given by the differential equation:

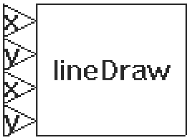
$$\dot{x} = \sin(t)$$

Furthermore, assume that x must lie in the limits $0.2t \le x \le 2t$ and that $x(0) = 0$. This system can be realized as:



A `ramp` block is used to access simulation time, *t*; simulation time is then used to feed the `sin` block, and two `gain` blocks, set to 2 and 0.2, to generate the time-varying upper and lower limits. During simulation, the time-varying limits and the output of the `limitedIntegrator` block are displayed in the `plot` blocks.

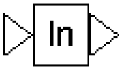
9.62 lineDraw



Block Category: Animation

The `lineDraw` block lets you animate a line during simulation. You define the line by specifying two sets of *x*,*y* coordinate endpoints. You can also set the color, thickness, and style of the line.

9.63 ln



$$y = \log_e x$$

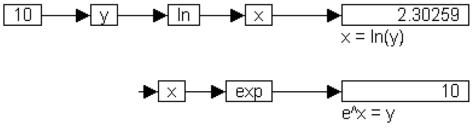
Block Category: Transcendental

The `ln` block generates the natural (Naperian) log of the input signal.

Examples

1. If ln(y) = x, then e^x = y

With *y* chosen to be 10, and *e* as the base of the natural logarithm, this equation can be realized as:



The `exp` block raises its input as a power of *e*, the base of natural logarithm. The quantity *e* is an irrational number, which is approximately equal to 2.718281828. From the results obtained, if ln(*y*) = *x*, then $e^x = y$, where *e* is the base of the natural logarithm.

9.64 log10

$y = \log_{10} x$

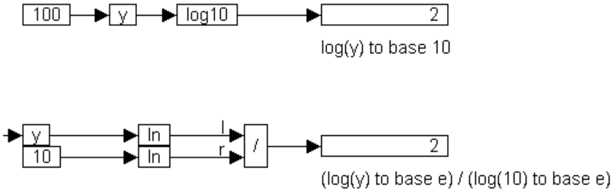
Block Category: Transcendental

The `log10` block generates the log base 10 of the input signal. The logarithm of 0 to any base is undefined. The logarithm of any number, when the base is the same number, is 1.

Examples

1. Computation of $\log_{10}$ y = $\log_e$ y / $\log_e$ 10

With *y* chosen to be 100, and *e* as the base of the natural logarithm, this equation can be realized as:



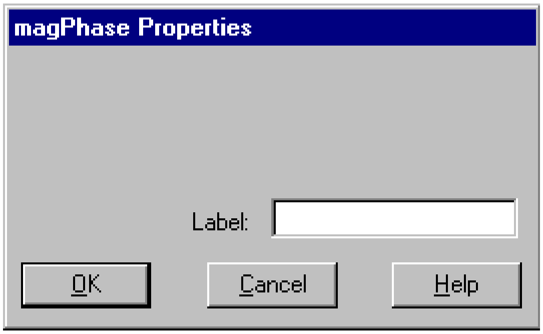
From the results obtained, $\log_{10} y = \log_e y / \log_e 10$ where *e* is the base of the natural logarithm. It can further be proved that $\log_a y = \log_b y / \log_b a$, where *a*, *b*, and *y* are any positive non-zero numbers.

9.65 magPhase

Block category: Arithmetic

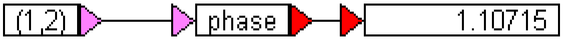
The `magPhase` block returns the magnitude and phase of the complex number. The magnitude is equal to $\sqrt{x^2 + y^2}$. The phase is equal to the value $\tan^{-1}\left(\frac{y}{x}\right)$, which is an angle expressed in radians.

When y is positive and x is 0, the phase is equal to $\frac{\pi}{2}$. When y is negative and x is 0, the phase is equal to $\frac{3\pi}{2}$.



Label: Indicates a user-defined block label.

Examples



9.66 map

Diagram1.map

$$y_1(y_2 ... y_n) = \text{tablelookup} x (1-\text{D})$$
$$y = \text{tablelookup}(x_1, x_2)\ (2-\text{D})$$
$$y = \text{tablelookup}(x_1, x_2, x_3)\ (3-\text{D})$$

Block Category: Nonlinear

The `map` block performs piecewise linear interpolated 1- 2-, and 3-dimensional table look-ups.

9.67 MatLab Expression

Block Category: MatLab Interface

The `MatLab Expression` block evaluates a MatLab expression.

9.68 MatLab Read Variable

Block Category: MatLab Interface

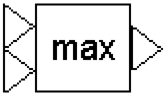
The `MatLab Read Variable` block reads a variable from the MatLab workspace into Commsim.

9.69 MatLab Write Variable

Block Category: MatLab Interface

The `MatLab Read Variable` block reads a variable from the MatLab workspace into Commsim.

9.70 max



$$y = \begin{cases} x_1 \text{ if } x_1 > x_2 \\ x_2 \text{ if } x_1 < x_2 \end{cases}$$

Block Category: Nonlinear

The `max` block compares scalar inputs for a higher value and generates an output signal with the higher value.

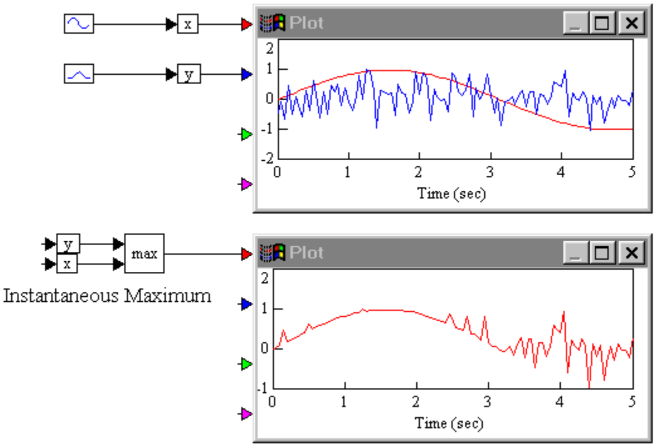
Examples

1. Comparison of two values

Consider the equation:

$z = \max(x,y)$

If x is a sinusoid that varies between -1 and +1, and y is a uniform random variable that varies between -1 and +1, this equation can be realized as shown below.

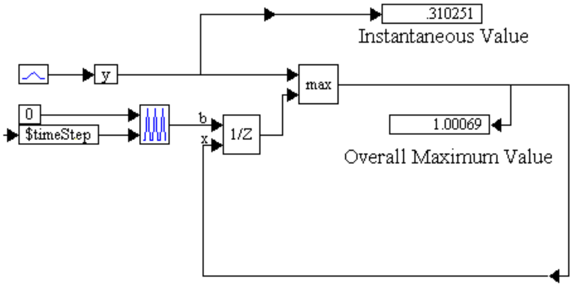


2. Computation of the maximum value of a given time-varying signal

Consider the equation:

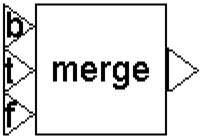
$z = max(y)$

where y is a uniform random variable that varies between -1 and +1. To find the maximum value that z attains, create the following diagram:



A `unitDelay` block stores the previous value of y. The `max` block compares the current and previous values of y. The larger of the current and previous values is fed back into the `unitDelay` block for the next round of comparisons. This way, the output of the `max` block is always the largest encountered value of *y*. The working details of this procedure are best examined by single-stepping through the simulation.

9.71 merge



$$y = \begin{cases} x_2 \text{ if} |x_1| \geq 1 \\ \mathrm{x}_3 \text{ if} |x_1| < 1 \end{cases}$$

Block Category: Nonlinear

The `merge` block examines x_1 (Boolean signal) to determine the output signal. The letters b, t, and f on the input connector tabs stand for Boolean, True, and False. The `merge` block accepts scalar, vector, and matrix input.

The `merge` block is particularly well-suited for performing if-then-else decisions.

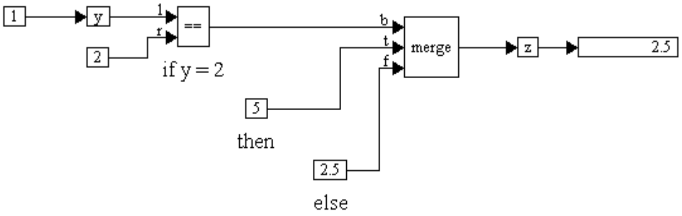
Examples

1. Simple merge

Consider the equation:

If *y* = 2, then *z* = 5, else *z* = 2.5

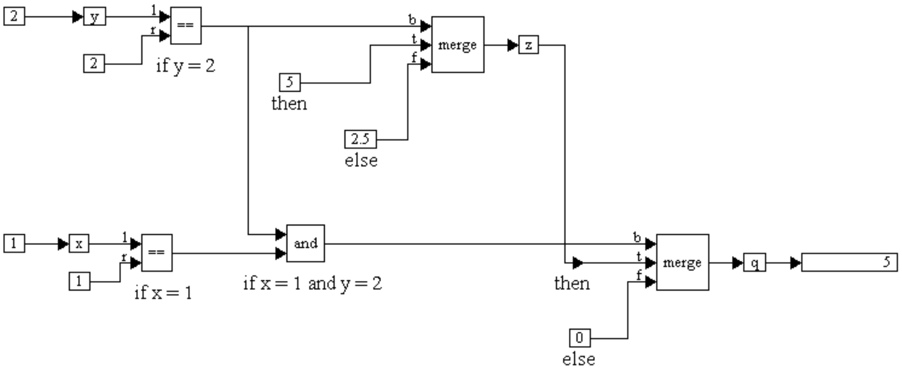
This equation can be realized as:



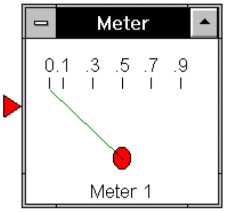
2. Cascade merge

Consider the equation:

If *x* = 1 and *y* = 2, then *q* = *z,* else *q* = 0 where (if y = 2, then z = 5, else z = 2.5). This logical relation can be realized as shown below.



9.72 meter

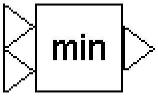


$$\text{display} = x_1$$

Block Category: Signal Consumer

The `meter` block displays signals in either a gauge- or bar-style display. Initially, the `meter` block appears as a gauge-style display with one input connector tab.

9.73 min



$$y = \begin{cases} x_1 \text{ if } x_1 < x_2 \\ x_2 \text{ if } x_1 > x_2 \end{cases}$$

Block Category: Nonlinear

The `min` block compares two scalar inputs for a lower value and generates an output signal with the lower value.

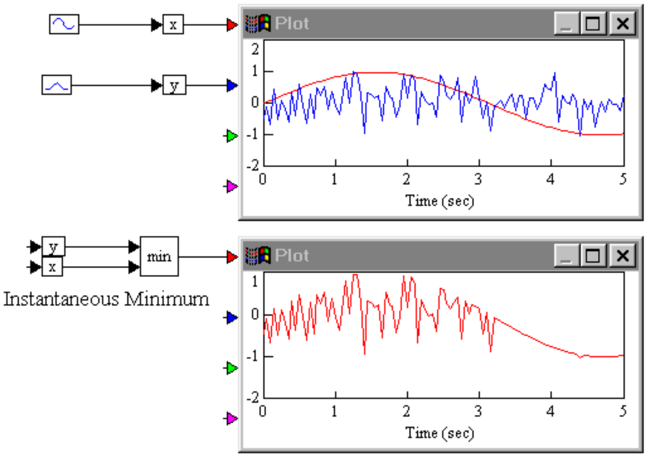
Examples

1. Comparison of two values

Consider the equation:

$z = \min(x,y)$

If *x* is a sinusoid that varies between -1 and +1, and *y* is a uniform random variable that varies between -1 and +1, this equation be realized as shown below.

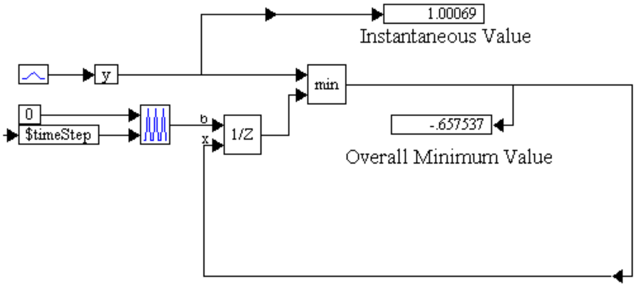


2. Computation of the minimum value of a given time-varying signal

Consider the equation:

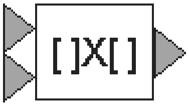
$z = \min(y)$

where *y* is a uniform random variable that varies between -1 and +1. To find the minimum value that *z* attains, create the following diagram:



A `unitDelay` block stores the previous value of *y*. The `min` block compares the current and previous values of *y*. The smaller of the current and previous values is fed back into the `unitDelay` block for the next round of comparisons. This way, the output of the `min` block is always the smallest encountered value of *y*. The working details of this procedure are best examined by single-stepping through the simulation.

9.74 multiply



Block Category: Matrix Operation

The `multiply` block performs a matrix multiplication. The `multiply` block accepts two vector inputs and produces one vector output.

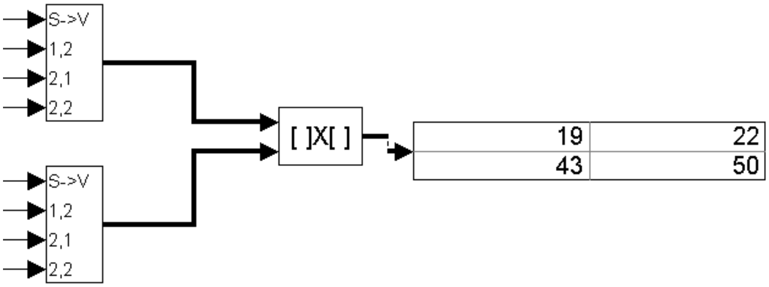
Multiplying scalars and vectors

To multiply two or more scalars, use the * block.

To perform a single value summation of an element-by-element multiply of two vectors, use the `dotProduct` block.

Examples

1. Simple matrix multiply



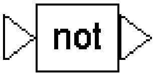
Here

$$\mathbf{A} = \begin{bmatrix} 1\,2 \\ 34 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 56 \\ 78 \end{bmatrix}$$

Then

$$\mathbf{AB} = \begin{bmatrix} 1(5)+2(7)1(6)+2(8) \\ 3(5)+4(7)3(6)+4(8) \end{bmatrix} = \begin{bmatrix} 1922 \\ 4350 \end{bmatrix}$$

9.75 not



$$y = \begin{cases} 1 \text{if}\, x_1 = 0 \\ 0 \,\text{otherwise} \end{cases}$$

Block Category: Boolean

The `not` block produces the Boolean NOT of the input signal. The output is true when the input is false; and the output is false when the input is true.

If you click the right mouse button over the `not` block, the Boolean block menu appears allowing you to assign a different function to the block.

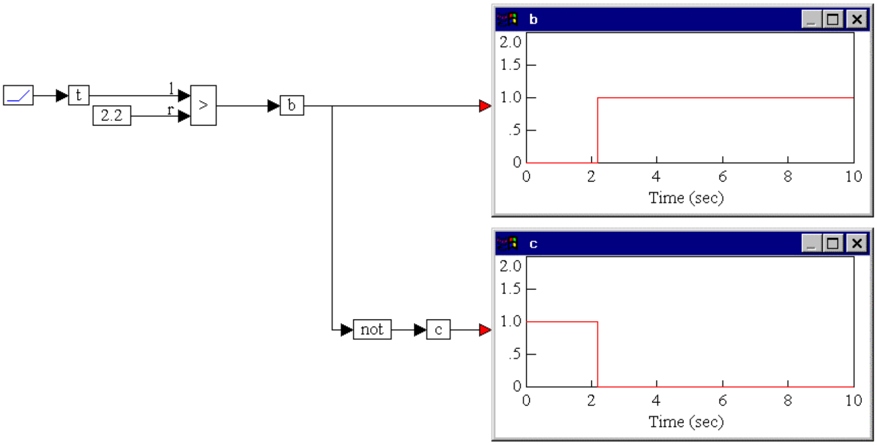
Examples

1. Using a not block

Consider a variable *c* such that:

$$c = \overline{b}$$

or in other words, *c* = not(*b*). Furthermore, assume that *b* is true if *t* > 2.2; else *b* is false, where *t* is the simulation time. This system can be realized as shown below.



From the outputs obtained in the two `plot` blocks, *b*, given by the output of the > block is true only when *t* is > 2.2. This requires that *c*, which is defined to be `not`(*b*), be true only the range *t* < 2.2, as obtained in the bottom `plot` block.

9.76 OLEobject

The `OLEobject` block lets you embed existing objects from files or insert new blank objects and create the information right in your diagram.

By embedding information in a Commsim diagram, you gain fast access to the functionality of another application without having to return to that application each time you want to make a change.

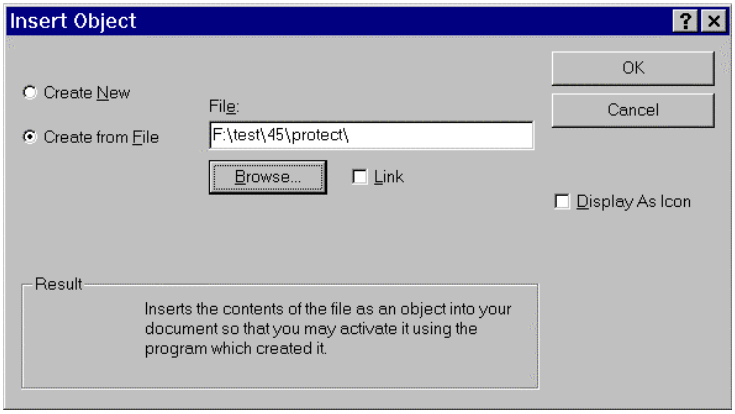
The technique of embedding is simple: just insert a file into your diagram. The inserted file is called an *object*, and appears as an `OLEobject` block in your diagram. When you embed an object, all of the instructions used to create the object are encapsulated and inserted in the

`OLEobject` block along with it. The instructions include the object’s file format, the application used to create the object, and all of the information about how to display it. You don’t see this information, but the application uses it to display and edit the object.

Embedding makes editing and updating the object quick and easy. You can just double-click on the `OLEobject` block in the Commsim diagram to open the object in its source application. Then you make your changes and close the file, and the object you embedded is updated automatically.

➢ To embed existing objects:

1. Choose Blocks > `OLEobject` and click the mouse in the Commsim workarea. The Insert Object dialog box appears.



○ Create New

● Create from File

File: F:\test\45\protect\

☐ Link

☐ Display As Icon

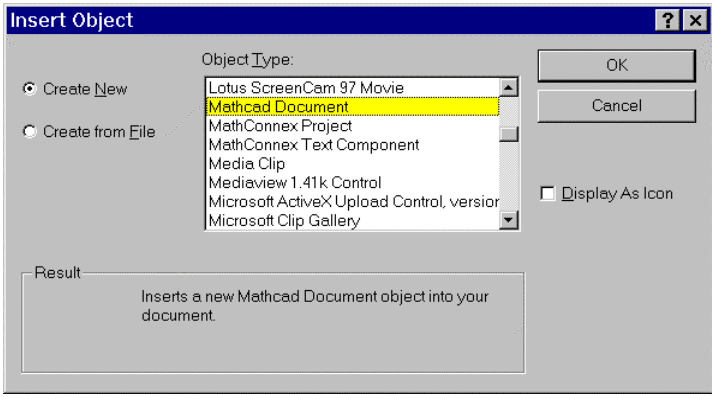
Result

Inserts the contents of the file as an object into your document so that you may activate it using the program which created it.

2. Activate the Create from File option.
3. Click on the Browse button to find the object to be embedded in your Commsim diagram.
4. Click on the OK button, or press ENTER.

- ➢ To embed a new object:

1. Choose Blocks > `OLEobject` and click the mouse in the Commsim workarea. The Insert Object dialog box appears.



● Create New

○ Create from File

Object Type:

- Lotus ScreenCam 97 Movie
- Mathcad Document
- MathConnex Project
- MathConnex Text Component
- Media Clip
- Mediaview 1.41k Control
- Microsoft ActiveX Upload Control, versior
- Microsoft Clip Gallery

□ Display As Icon

Result

Inserts a new Mathcad Document object into your document.

2. Activate the Create New option.
3. In the Object Type list box, select the type of object you want to insert. For Mathcad 2000 documents, click on Mathcad Document
4. Choose the OK button, or press ENTER.

9.76.1 Editing and updating objects

You can edit an embedded objects directly from Commsim, or you can invoke the application as a separate, stand-alone program. To update the embedded object and return to the Commsim environment, simply click outside the `OLEobject` block.

- ➢ To edit an embedded object directly from Commsim:

1. Do one of the following:
   - Double-click on the embedded object.

     -Or-
   - Click the right mouse button on the embedded object.
   - From the drop-down menu, choose Object > Edit.

   Commsim opens a window with the application file.
2. Make the changes you want.
3. Click outside the embedded object to update the object and return to Commsim.

- ➢ To invoke the object as a stand-alone application:

1. Click the right mouse button on the embedded object.
2. From the drop-down menu, choose Object > Open. A stand-alone window is opened with the application file.
3. Make the changes you want.
4. To update the object and return to Commsim, do the following:
    - Choose File > Update.
    - Choose File > Exit and Return to.

9.76.2 Examining the properties of an embedded object

The Properties dialog box lists the size of your embedded object and allows you to control the general appearance of the document in your Commsim diagram. You can also use the Properties dialog box to scale the corresponding embedded object. This feature is described in the next section.

- ➢ To access general information about an embedded object:

You can display the size and location of the embedded object.

1. Click right mouse button over the embedded object.
2. In the drop-down menu, choose Properties.
3. The Properties dialog box appears.
4. Click on the General tab.

- ➢ To control the appearance of an embedded object:

You can display the size and location of the embedded object.

1. Click right mouse button over the embedded object.
2. In the drop-down menu, choose Properties.
3. The Properties dialog box appears.
4. Click on the View tab.

5. Do the following:

To	Do this
Display the contents of the embedded object	Activate the Display As Editable Information option.
Display an icon	Activate the Display As Icon option. Click on the Change Icon button to select the type of icon to be displayed.

6. Click on the OK button.

9.76.3 Scaling and cropping OLEobject blocks

You can resize an `OLEobject` block by scaling or cropping it. When you scale an `OLEobject` block, the text within the block is also scaled. When the dimensions of the `OLEobject` block are much larger than the text, you can adjust the size of the block by cropping its sides. The size of the text within the block is not affected by this action.

➢ To scale an OLEobject block manually:

1. Click on the `OLEobject` block.
2. Position the pointer over one of the handles on the block.
3. Hold down the mouse button and drag until the block is the size you want.
4. Click outside the `OLEobject` block to return to Commsim.

➢ To scale an OLEobject block using the Properties dialog box:

1. Click right mouse button over the `OLEobject` block.
2. In the drop-down menu, choose Properties.
3. The Properties dialog box appears.
4. Click on the View tab.
5. To scale the block with respect to its original size, activate the Relative To Original Size option.
6. In the Scale box, select the scaling factor.
7. Click on the Apply button to preview the new size.
8. Click on the OK button to make the change permanent.

➢ To crop an OLEobject block manually:

1. Double-click on `OLEobject` block.
2. Position the pointer over one of the handles on the block.

3. Hold down the mouse button and drag until the block is the size you want.
4. Click outside the `OLEobject` block to return to Commsim.

➢ To reset the size of an OLEobject block:

You can use this method when you cropped the `OLEobject` block or when you scaled in manually.

1. Click the right mouse button on the `OLEobject` block.
2. From the drop-down menu, choose Reset Size.

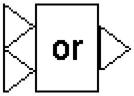
Examples

1. Embedding an existing PowerPoint file

To embed an existing Powerpoint file:

1. In Commsim, insert an `OLEobject` block.
2. In the Insert Object dialog box, do the following:
   - Activate the Create From File option.
   - Click on the Browse button, and select the PPT file to be embedded in your diagram.
   - Click on the OK button, or press ENTER.

9.77 or



$$y = x_1 \text{bitwiseOR} x_2$$

Block Category: Boolean

The `or` block produces the bitwise OR of two to 256 scalar input signals. The output of the `or` block is true when at least one of the inputs is true. When all the inputs are false, the output is false.

If you click the right mouse button over the `or` block, the Boolean block menu appears allowing you to assign a different function to the block.

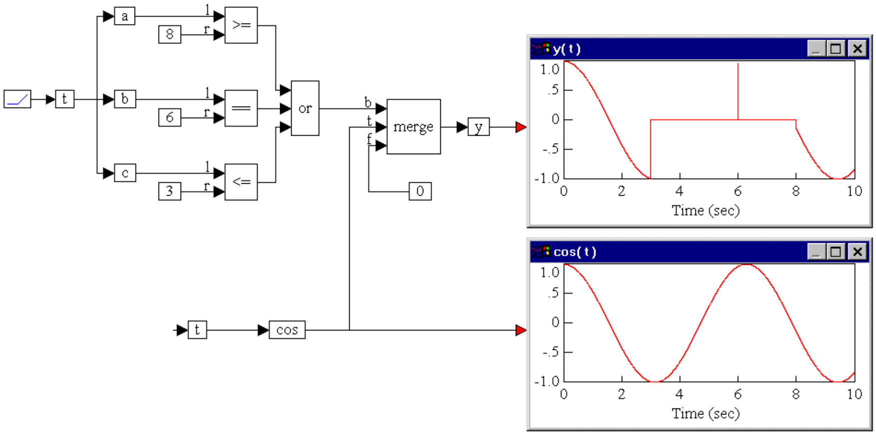
Examples

1. Computation of three inputs

Consider a variable *y* such that:

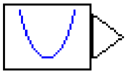
If $a \geq 8$ or $b = 6$ or $c \leq 3$, then $y = \cos(t)$; else $y = 0$

where *t* is simulation time. Furthermore, let *t* be the input to all three parameters *a*, *b*, and *c*. This system can be realized as shown below.



During simulation, the `or` block evaluates to false in the interval *t* = (3,8), except for the instant *t* = 6. In this case, the `variable` *y* takes on the value of 0. The output of `or` evaluates to true in the remaining parts of the simulation, and as a result, *y* takes on the value of cos(*t*) in these periods, including the instant *t* = 6.

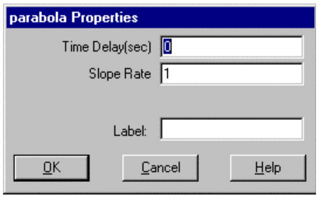
9.78 parabola



$$y = \text{slope}\cdot(t - t_{delay})^2$$

Block Category: Signal Producer

The `parabola` block creates a parabolic signal.



parabola Properties

Time Delay(sec) 0

Slope Rate 1

Label:

Time Delay: Indicates an offset that is used in the calculation of a signal. For a constant-valued delay, wire the block into a `unitDelay` or `timeDelay` block with an initial condition of the desired constant value.

Specify the offset in seconds. The default value is 0.

Slope Rate: Scales the curvature of the parabola. The default value is 1.

Label: Indicates a user-defined block label.

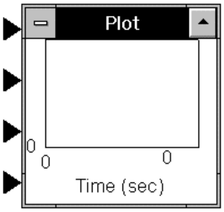
9.79 parameterUnknown

parameterUnknown

Block Category: Optimization

The `parameterUnknown` block works with the `cost` block to find globally optimal values that minimize a scalar cost function.

9.80 plot

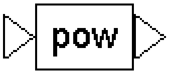


plot = $x_1 ... x_4$

Block Category: Signal Consumer

The `plot` block displays simulation data graphically in a customizable plots.

9.81 pow



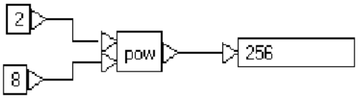
$$y = \begin{cases} x^{\text{exponent}} \\ or \\ x_1^{x_2} \end{cases}$$

Block Category: Arithmetic

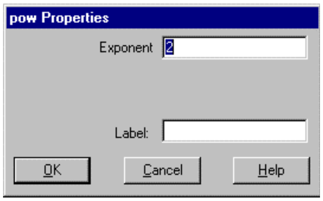
The pow block creates an output signal based on the value of the input signal raised to the power of a specified exponent. Inputs can be scalars, vectors, or matrices. When the input is a vector or matrix, the pow block computes the output on an element-by-element basis.

The pow block is useful for solving equations of the type $y = x^z$. Do not use the pow block to compute matrix dot products, such as $\mathbf{Y} = \mathbf{A}^2$, where the dot product is implied. Instead use the dotProduct block.

By adding an input connector tab to the pow block, you can specify an external exponent parameter to override the block's exponent parameter. For example:



This diagram raises two to the eighth power. The display block verifies the results. The main advantage of setting the exponent externally is that the value of the exponent can be varied dynamically as the simulation progresses.



pow Properties

Exponent: 2

Label:

Exponent: Specifies the power to which the input signal is raised. The default is 2.

Label: Indicates a user-defined block label.

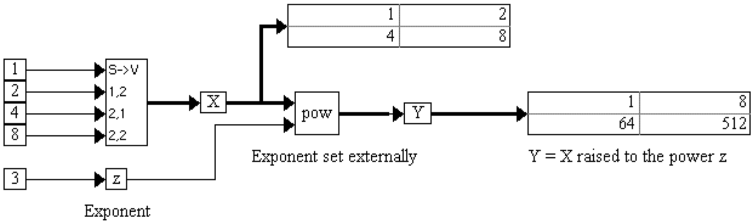
Examples

1. Raising a matrix input to a power

Consider the equation:

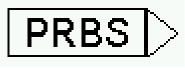
$$\mathbf{Y} = \mathbf{X}^{Z}$$

where $\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 4 & 8 \end{bmatrix}$ and z = 3. This equation can be realized as shown below:



The pow block raises each element of the incoming matrix to power 3.

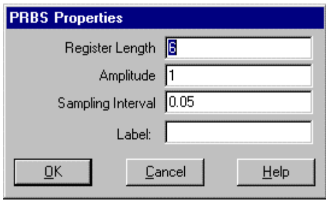
9.82 PRBS



Block Category: Random Generator

The PRBS block produces a pseudo-random sequence of unit amplitude pulses. You can control the frequency of oscillation and the register length.

The PRBS block can be used to see how random perturbations affect a system. System Identification software can use the output of a PRBS block to create a mathematical model of the system.



PRBS Properties

Register Length: 6

Amplitude: 1

Sampling Interval: 0.05

Label:

Register Length: Controls when the sequence of pulses repeats. The default is 6

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Sample Interval: Indicates the frequency of oscillation. The default is 0.05.

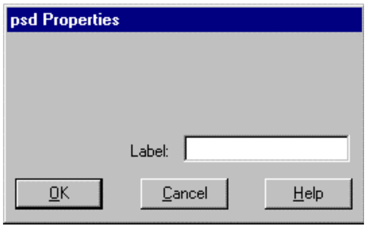
Label: Indicates a user-defined block label.

9.83 psd

Block Category: Matrix Operations

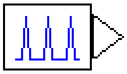
The psd block accepts a vector of signal values and produces a vector of power spectrum values. The psd block performs an FFT on the input vector then transforms the FFT to represent power versus frequency. To capture all the power spectrum information, plot the output on a Log Y plot.

The change in frequency of the psd output vector equals $\frac{\frac{1}{2}\,sim\ \ frequency}{output\ vector}$.



Label: Indicates a user-defined block label.

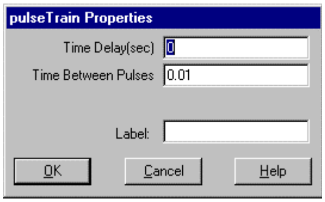
9.84 pulseTrain



$$y = \begin{cases} 1 \text{if}\, t \,\text{mod}_{\text{timepulse}} == 0 \\ 0\, \text{otherwise} \end{cases}$$

The `pulseTrain` block produces a sequence of unit amplitude pulses separated by zeros. You cannot control the duration of the pulse; you can only control the time between pulses.

You can add two input connector tabs to the `pulseTrain` block. The top input connector tab lets you specify an external time delay; the bottom one lets you specify an external time between pulses. These additional inputs override the existing parameters. If you add only one input connector tab, it corresponds to the external delay.



pulseTrain Properties

Time Delay(sec) 0

Time Between Pulses 0.01

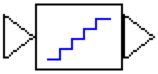
Label:

Time Delay (sec): Specifies, in seconds, how long to delay before calculating the value of the output signal. The default is 0.

Time Between Pulses: Specifies the time between pulses. This is useful for clocking delays and sample holds. The default is 0.01.

Label: Indicates a user-defined block label.

9.85 quantize



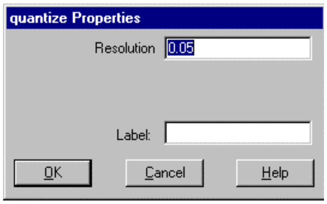
$$y = \left[\text{integer part}\left(\frac{x}{\text{resolution}}\right)\right]\text{resolution}$$

Block Category: Nonlinear

The `quantize` block is useful for simulating approximations of a continuously varying signal that possibly requires the use of an infinite number of values or levels by a discontinuous signal with a finite number of values.

The `quantize` block rounds the precision of input signal based on the signs of the input and the resolution. When the resolution is positive, the signal is rounded down to -∞. For example, 1.9 quantized to a resolution of 1 becomes 1, and -1.9 quantized to a resolution of 1 becomes 2. When the resolution is negative, the signal is rounded to +∞. For example, 1.1 quantized to a resolution of -1 becomes 2, and -1.1 quantized to a resolution of -1 becomes -1.

The `quantize` block is applicable to simulations that involve the conversion of analog signals to digital signals.



quantize Properties

Resolution: 0.05

Label:

Resolution: Specifies the value to which the input signal is rounded or truncated. The default is 0.05.

Label: Indicates a user-defined label.

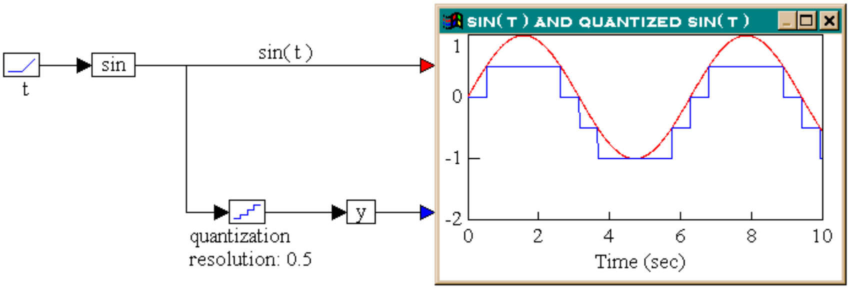
Examples

1. Quantization of a sinusoid: positive resolution

Consider a variable *y* such that:

$$y = \sin(t)$$

quantized with a resolution of +0.5. This equation an be realized as:



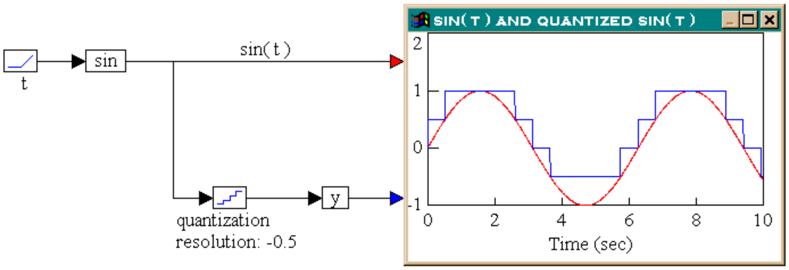
The `quantize` block approximates the sinusoid input using four values (0, +0.5, -0.5, and -1).

2. Quantization of a sinusoid: negative resolution

Consider a variable *y* such that:

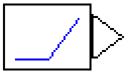
$$y = \sin(t)$$

quantized with a resolution of -0.5. This equation can be realized as:



The `quantize` block approximates the sinusoid input using four values (0, +0.5, +1, and -0.5). By comparing these results with those in Example 1, the effects of using positive and negative resolutions in a `quantize` block becomes clear.

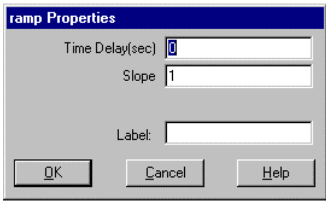
9.86 ramp



$$y = \text{slope} \cdot (t - t_{delay})$$

Block Category: Signal Producer

The `ramp` block creates a unit ramp signal based on simulation time.



Time Delay(sec): Indicates an offset that is used in the calculation of a signal. For a constant-valued delay, wire the `ramp` block into a `unitDelay` or `timeDelay` block with an initial condition of the desired constant value. Specify the offset in seconds. The default is 0.

Slope: Specifies the ramp slope. The default is 1.

Label: Indicates a user-defined block label.

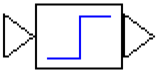
9.87 realTime

$$y = t$$

Block Category: Signal Producer

The `realTime` block provides the current time in milliseconds since the start of your Commsim session. Note that this is not simulation time.

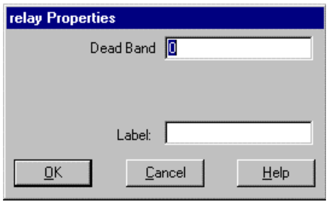
9.88 relay



$$y = \begin{cases} -1 \text{if } x < \dfrac{\text{- deadband}}{2} \\ 1 \text{if } x > \dfrac{\text{deadband}}{2} \\ 0 \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `relay` block simulates a tri-state relay operator. This block is useful for simulation switches or switching operators.



Dead Band: 0

Label:

Dead Band: Indicates the width of the zone of lost motion about the input signal's 0 value, thereby creating a tri-state relay operator (-1, 0, 1). When input is less than half the negative Dead Band value, the `relay` block outputs -1. When input is greater than half the positive Dead Band value, the `relay` block outputs +1. When input lies within the range (-Dead Band/2, +Dead Band/2), the `relay` block outputs 0. You cannot specify a negative value for this parameter. The default is 0.

Label: Indicates a user-defined block label.

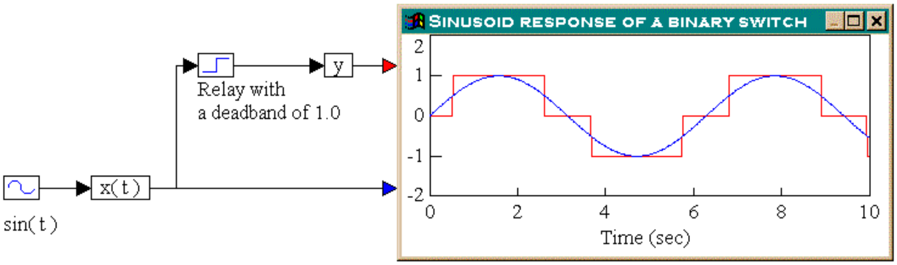
Examples

1. Constructing a tri-state switch

Consider a tri-state variable *y* such that:

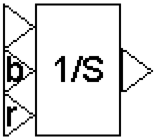
$$y = \begin{cases} +1 if\ x(t) > 0.5 \\ -1 if\ x(t) < -0.5 \\ \quad 0 otherwise \end{cases}$$

Assuming that *x*(*t*) = sin(*t*), this equation can be realized as shown below.



The Dead Band of the `relay` block is set to 1.0. During simulation, the `relay` block changes its output state based on whether the input signal is greater than or less than Dead Band/2.

9.89 resetIntegrator (1/S)



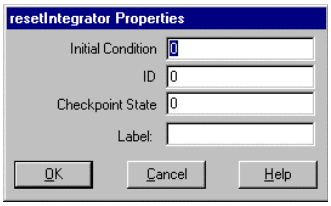
$$y = \begin{cases} \displaystyle\int_{t_{start}}^{t_{end}} x_1 dt \quad \text{if} \left|x_2\right| < 1 \\ x_3 \qquad\qquad \text{if } \left|\text{x}_2\right| \geq 1 \end{cases}$$

Block Category: Integration

The `resetIntegrator` block integrates the input signal with an optional reset capability. When the Boolean input (b) is 0, the `resetIntegrator` behaves like a normal integrator. When the Boolean input goes to 1, the `resetIntegrator` takes the value of the reset input (r) for as long as the Boolean value stays high.

The `resetIntegrator` block integrates the input signal using the integration algorithm established in the dialog box for the Simulate > Simulation Properties command. The available algorithms are Euler, trapezoidal, Runge Kutta 2d and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff). You can reset the `resetIntegrator` block to zeros using the Simulate > Reset States command.

The inputs to the `resetIntegrator` block are x_1, x_2 (b), and x_3 (r).



resetIntegrator Properties

Initial Condition: 0

ID: 0

Checkpoint State: 0

Label:

Initial Condition: Indicates the initial value of the integrator upon simulation start-up. This parameter can be overridden if x_2 is non-zero on the first step of the simulation. The default is 0.

ID: Represents an identification number for the block. This number keeps track of the state number that Commsim assigns to the integrator. The number of states in any Commsim diagram equals the number of integrators. The default is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation via the Simulate > Simulation Setup command, the default is 0.

Label: Indicates a user-defined block label.

Examples

1. Instantaneous momentary reset

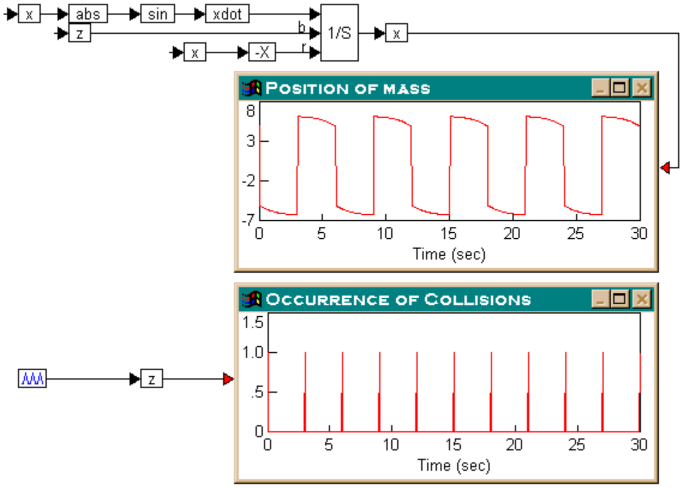
Consider a system whose dynamics are given by the differential equation:

$$\dot{x} = \sin|x|$$

When a particular variable *z* equals 1, *x* must be reset to -*x*. To make matters simple, assume that *z* becomes momentarily equal to 1, every three seconds, and that *x*(0) = 5.

Equations of this type are frequently used in kinematic systems that undergo collisions, electrical circuits that involve switching phenomena, chemical processes, and fluid dynamics.

This system can be realized as shown below.



As with any differential equation, the right-hand side of the equation is realized first by creating a `variable` *x*, and then connecting it successively to an `abs` block, a `sin` block, and another `variable` *xdot*.

At this point, only *xdot* is defined in terms of *x*. To define the relationship between *xdot* and *x*, *xdot* is fed into the top input tab of the `resetIntegrator` block, which is fed into the `variable` *x*.

The Boolean input tab of the `resetIntegrator` is fed by `variable` *z*, which generates pulses that are three seconds apart. The negative value of a given signal can be directly generated using the `-X` block, and then fed into the reset input tab of the `resetIntegrator`.

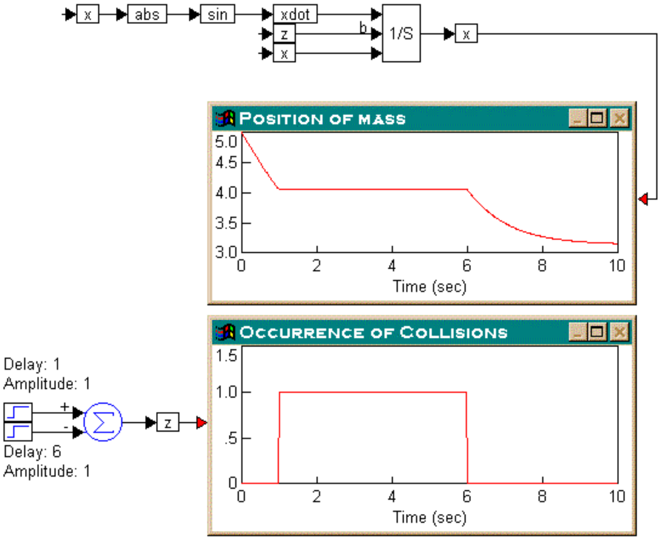
From the results of simulation shown in the two `plot` blocks, *z* becomes high every three seconds, and at each of these instances, the output of the `resetIntegrator` is reset to -*x*.

2. State reset for a duration

As mentioned above, the `resetIntegrator` output is held at the reset value as long as the Boolean input is high. To illustrate this property, consider the differential equation:

$$\dot{x} = \sin|x|$$

When a particular variable *z* is equal to 1, *x* must be held at its current value. Assume that *z* = 1 when 1 ≤ *t* ≤ 6 and that *x*(0) = 5. This case can be realized as shown below.



To construct *z*, two `step` blocks and a `summingJunction` block are used. The delay and amplitude of the top `step` block are both set to 1; for the bottom `step` block, they are set to 6 and 1, respectively. By subtracting the outputs of the two `step` blocks and defining the output of the `summingJunction` block as *z*, *z* = 1 when 1 ≤ *t* ≤ 6.

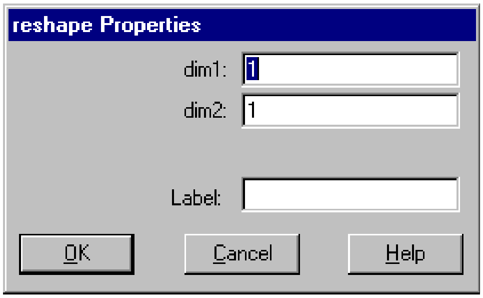
By coding *z* in this manner, *z* is redefined as $z(t) = u(t - 1) - u(t - 6)$ where *u*(*t*) represents a unit step. Consequently, *u*(*t* - 1) is a unit step delayed by 1 sec, and *u*(*t* - 6) is a unit step delayed by 6 sec.

During simulation, the output of the `resetIntegrator` holds constant at *x*(1) for the duration 1 ≤ *t* ≤ 6.

9.90 reshape

Block category: Matrix Operations

The reshape block redimensions matrix input. The data remains the same, but the dimensions change. If the matrix dimensions are too large, zeros are padded in the extra cells. If the matrix dimensions are too small, the matrix data is truncated.



reshape Properties

dim1: 1

dim2: 1

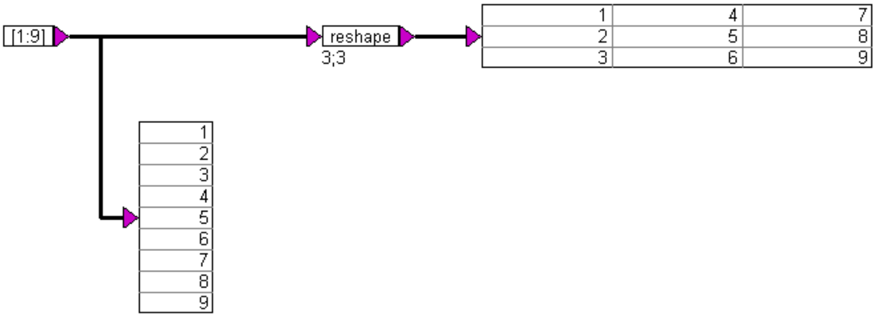
Label:

Dim1: Indicates the x dimension of the matrix.

Dim2: Indicates the y dimension of the matrix.

Label: Indicates a user-defined block label.

Examples

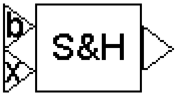


1	4	7
2	5	8
3	6	9

1
2
3
4
5
6
7
8
9

The reshape block produces a 3-by-3 matrix from a 9-element vector.

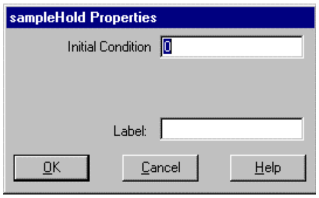
9.91 sampleHold



$$y = \begin{cases} x_2 & \text{if}\ |x_1| \ge 1 \\ y_{\text{previous}} & \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `sampleHold` block latches an input value under the control of a clock signal, x_1, which is represented as Boolean input (b). When b is true, input signal x_2, which is represented as input (x) is sampled and held until b is true again. Boolean inputs can be regularly or irregularly spaced.



sampleHold Properties

Initial Condition: 0

Label:

Initial Condition: Indicates the initial condition for the `sampleHold`. The default is 0.

Label: Indicates a user-defined block label.

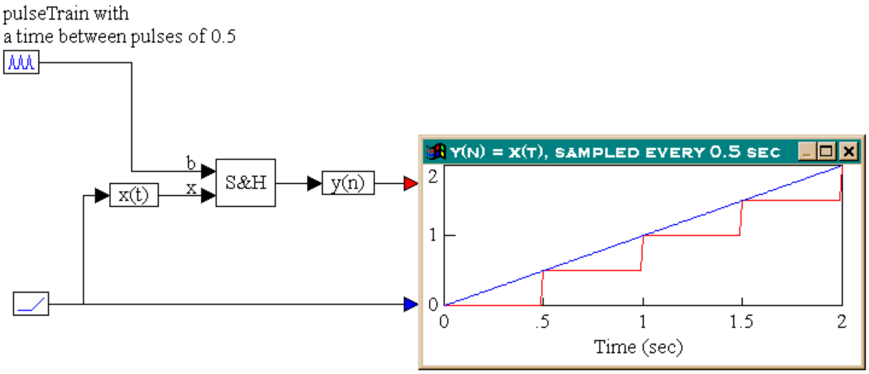
Examples

1. Sample and hold with regularly-spaced clock

Consider the equation:

$y(n) = x(t)$

sampled every 0.5 sec. Furthermore, let *x*(*t*) be a ramp signal. This system can be realized as shown below.


pulseTrain with
a time between pulses of 0.5
b
x
x(t)
S&H
y(n)
Y(N) = X(T), SAMPLED EVERY 0.5 SEC
2
1
0
0
.5
1
1.5
2
Time (sec)

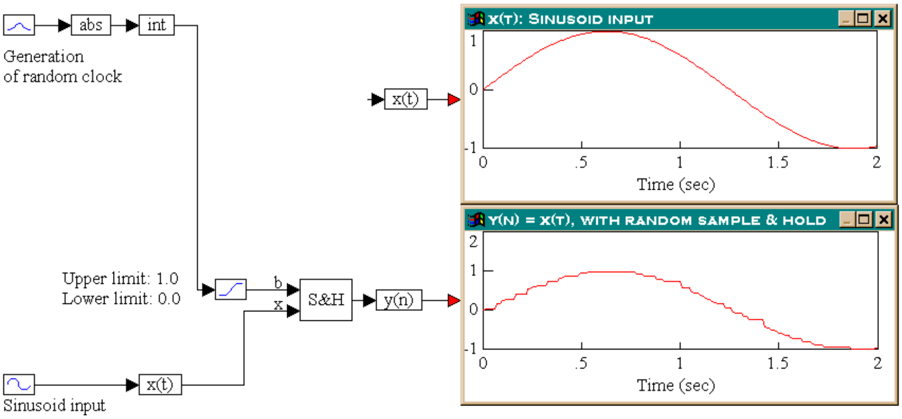
As seen in the `plot` block, the first clock pulse occurs at 0.5 sec. Until this time, the output of the `sampleHold` block is zero. At 0.5 sec, the input signal is sampled and the value is used as output for the `sampleHold` block. The output of the `sampleHold` block is held at this value until the occurrence of the next clock pulse at 1.0 sec. At this time, the input signal is again sampled and the new value is presented to the output of the `sampleHold` block, and the process repeats itself.

2. Sample and hold with irregularly-spaced clock

Consider the equation:

$y(n) = x(t)$

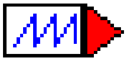
sampled randomly. Furthermore, let *x*(*t*) be a sinusoid signal with a frequency of 2.5 rad/sec. This system can be realized as shown below.



A `sinusoid` block with a frequency of 2.5 rad/sec generates the sinusoid signal and a `gaussian` block produces a randomly varying signal. The randomly varying signal is converted to a random clock by taking the absolute value of the random signal and then using only the integer portion of it. The output of the `int` block is passed through a `limit` block to restrict the signal to the range (0, 1). The output of the `limit` block is connected to the top input of the `sampleHold` block. The output of the `sampleHold` block is connected to the `variable` *y*(*n*),which is connected to a `plot` block. The actual input, *x*(*t*) is monitored separately in another `plot` block.

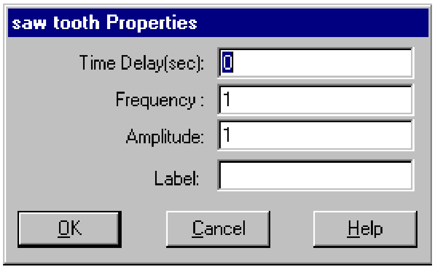
By comparing the outputs in the two `plot` blocks, the output of the `sampleHold` block is a randomly sampled and held version of the input sinusoid.

9.92 sawtooth block



Block category: Signal Producer

The `sawtooth` block creates a unit sawtooth signal.



saw tooth Properties

Time Delay(sec):	0
Frequency :	1
Amplitude:	1
Label:	

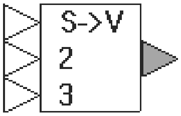
Time Delay (sec): Indicates the time offset that is used in the calculation of the signal. Specify the offset in seconds. The default is 0.

Frequency: Controls the frequency of oscillation of the output signal. Specify the frequency in hertz. The default is 1.

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Label: Indicates a user-specified label.

9.93 scalarToVec



Block Category: Annotation

The `scalarToVec` block reduces wiring clutter by letting you combine input signals into a single vector wire. This is usually a prerequisite for performing vector and matrix algebra. Use the `vecToScalar` block to unbundle vector wires.

Examples

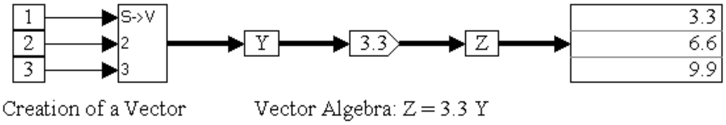
1. Creation of a vector

Consider the equation:

$\mathbf{Z} = 3.3\ \mathbf{Y}$

where **Z** and **Y** are vectors. Further, assume that $\mathbf{Y} = [1\ 2\ 3]^T$.

This equation can be realized as:



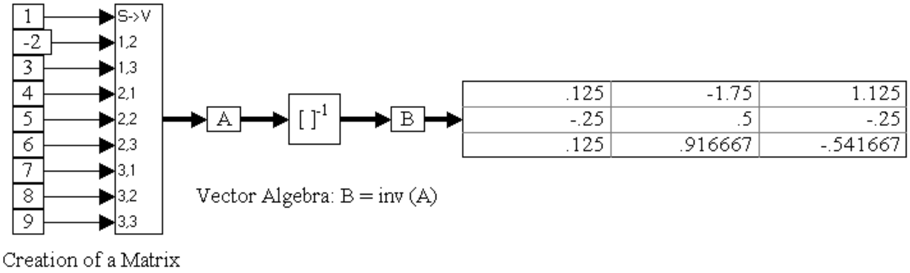
The `display` block displays all the elements of the incoming vector line. The results indicate that the vector operation is performed correctly.

2. Creation of a matrix

Consider the equation $\mathbf{B} = \mathbf{A}^{-1}$, where

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

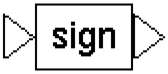
The above equation can be realized as shown below.



.125	-1.75	1.125
-.25	.5	-.25
.125	.916667	-.541667

The `display` block displays all the elements of the incoming matrix line. The results indicate that the matrix operation is performed correctly.

9.94 sign



$$y = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x = 0 \\ -1 \text{ if } x < 0 \end{cases}$$

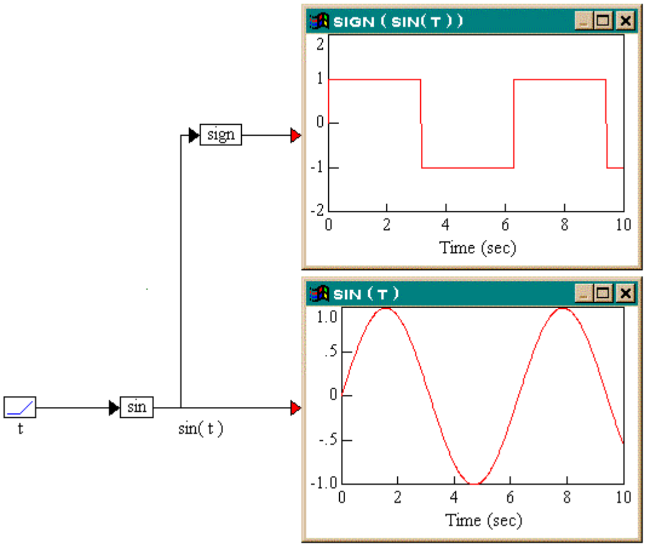
Block Category: Arithmetic

The `sign` block determines the sign of the scalar input signal. The `sign` block outputs +1 when the input is greater than zero; -1 when the input is less than zero; and 0 when the input is zero.

Examples

1. **Computation of the sign of sin(*t*)**

This equation can be realized as shown below.



The results obtained indicate that when the input is greater than zero, the `sign` block outputs +1, when the input is less than zero the output is -1, and when the input is zero, the output is 0.

9.95 sin

$$y = \sin x$$

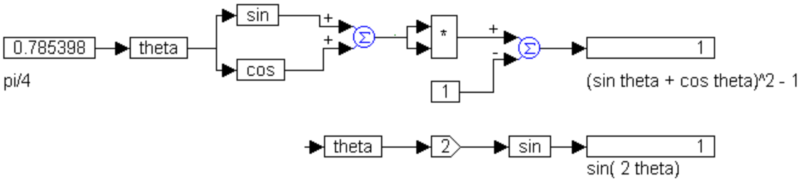
Block Category: Transcendental

The `sin` block produces the sine function of the input signal. The input signal is represented in radians.

Examples

1. Computation of sin(2θ) = (cos(θ) + sin(θ))^2 - 1

With θ chosen to be π/4, the above trigonometric identity can be realized as:



9.96 sinh

$$y = \frac{e^{x} - e^{-x}}{2}$$

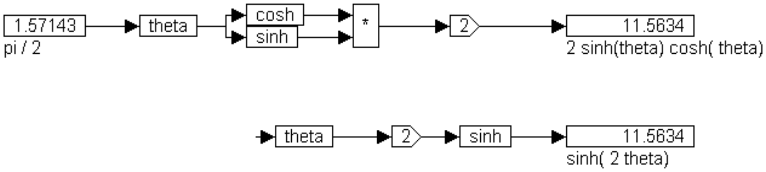
Block Category: Transcendental

The `sinh` block produces the hyperbolic sine function of the input signal. The input signal is represented in radians.

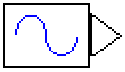
Examples

1. Computation of sinh(2θ) = 2 sinh(θ) cosh(θ)

With θ chosen to be π/2, the above trigonometric identity can be realized as:



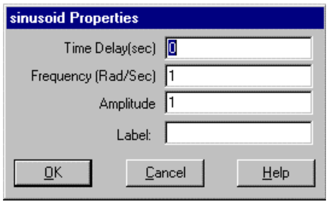
9.97 sinusoid



$$y = \mathrm{A}\cdot\sin\left(\omega\cdot(t - t_{delay})\right)$$

Block Category: Signal Producer

The `sinusoid` block creates a unit sine wave signal.



sinusoid Properties

Time Delay(sec): 0

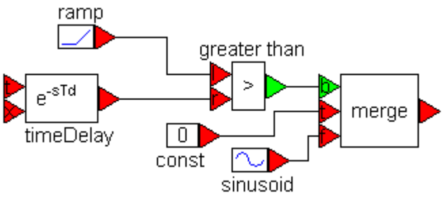
Frequency (Rad/Sec): 1

Amplitude: 1

Label:

Time Delay (sec): Indicates a time or phase offset that is used in the calculation of a signal. Specify the offset in seconds. The default is 0.

For a constant-valued delay, wire the `sinusoid` block into a `merge` block as follows:

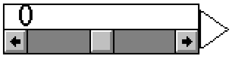


Frequency (rad/sec): Controls the frequency of oscillation of the output signal. Specify the frequency in radians per second. For example, if you specify a frequency of 1, one oscillation completes in 2π seconds. If you specify a frequency of π, one oscillation completes in 0.5 seconds. The default is 1.

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Label: Indicates a user-defined block label.

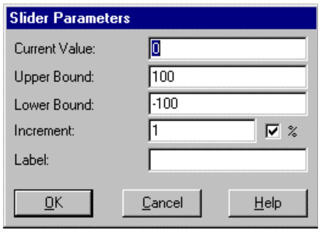
9.98 slider



Block Category: Signal Producer

The `slider` block allows mouse input to dynamically modify a signal value during a simulation, between a lower and upper bound in 1% and 10% increments. The `slider` block displays the current value applied to the signal. Use the scroll bar to adjust the signal value.

Slider precision is affected by the High Precision Display parameter under Preferences in the dialog box for the Edit > Preferences command. When activated, slider precision is shown at up to 15 significant digits; when de-activated, slider precision is shown at up to 6 significant digits.



Slider Parameters

Current Value: 0

Upper Bound: 100

Lower Bound: -100

Increment: 1 ☑ %

Label:

Current Value: Specifies the initial value of the output signal. The default is 0.

Upper Bound: Specifies the largest value the output signal can attain. The default is 100.

Lower Bound: Specifies the smallest value the output signal can attain. The default is -100.

Increment: Indicates the amount by which the slider changes when you click on the slider bar. If you activate the % box, the amount you specify indicates a percentage change. If you do not activate the % box, the amount you specify indicates an absolute value.

Label: Indicates a user-defined block label.

9.99 sqrt

$$y = \sqrt{x}$$

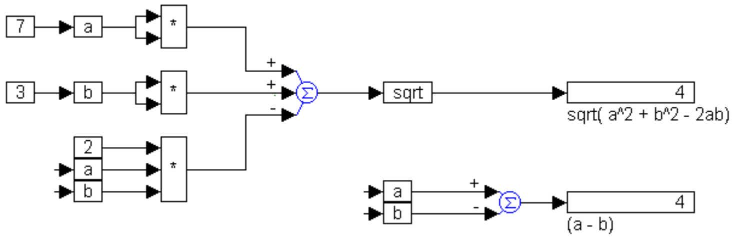
Block Category: Transcendental

The `sqrt` block produces an output signal that is the square root of a positive input signal. The `sqrt` block does not accept negative inputs. And, there is no square root of 0.

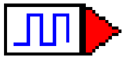
Examples

1. Computation of the sqrt($a^2 + b^2 - 2ab$) = (a - b)

With *a* and *b* chosen to be 7 and 4 respectively, the above equation can be realized as:

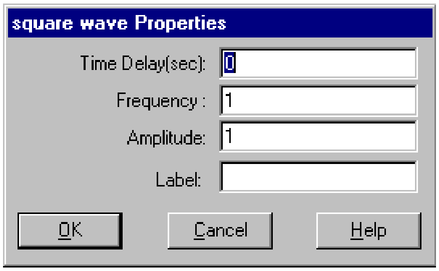


9.100 squareWave block



Block category: Signal Producer

The `squareWave` block creates a unit square wave signal.



square wave Properties

Time Delay(sec): 0

Frequency : 1

Amplitude: 1

Label:

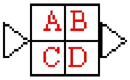
Time Delay (sec): Indicates the time offset that is used in the calculation of the signal. Specify the offset in seconds. The default is 0.

Frequency: Controls the frequency of oscillation of the output signal. Specify the frequency in hertz. The default is 1.

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Label: Indicates a user-specified label.

9.101 stateSpace



Block Category: Linear system

The `stateSpace` block is used to represent a multi-input multi-output linear system in state-space form. The state-space matrices can be specified in the following ways:

- **As an .M file created with Commsim:** The Analyze > Linearize command generates ABCD state-space matrices from a nonlinear system by numerically evaluating the matrix perturbation equations at the time the simulation was halted.
- **As an .M file created with a text editor:** When you create a .M file with a text editor, follow these rules: start each matrix on a new line; enclose matrix elements in square brackets and terminate with a semi-colon; separate matrix elements with spaces; separate matrix rows with semi-colons.

  The following is an example of a user-written .M file:

  function [a,b,c,d] =vabcd

  a = [-.396175 -1.17336 ; 5.39707 .145023];

  b = [-.331182 ; -1.08363];

c = [0 1];

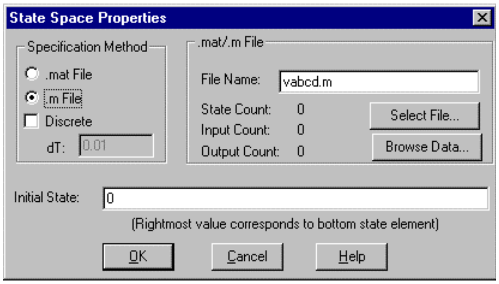
d = [0];

Note that MatLab commands other than array initialization are not allowed.

- **As a .MAT file created with MatLab:** Generating .MAT files is described in the MatLab documentation. Note that when you save the ABCD matrices to a file, the names of the matrices are not important; however, the order in which they appear is.

When you simulate the block diagram, Commsim numerically solves the `stateSpace` block. You can reset `stateSpace` blocks to zeros using the Simulate > Reset States command, as described in the *Commsim 7 Getting Started Guide*.

Commsim supports state-space systems up to the 90th order.



State Space Properties

Specification Method

○ .mat File

● .m File

□ Discrete

dT: 0.01

.mat/.m File

File Name: vabcd.m

State Count: 0

Input Count: 0

Output Count: 0

Initial State: 0

(Rightmost value corresponds to bottom state element)

Specification Method: You have the choice of three specification methods:

- **Discrete:** Indicates a discrete Z-domain system. Enter the time step for the discrete transfer function in the dT box. By default, this parameter is de-activated, which indicates a continuous transfer function.
- **.mat File:** Indicates that the system is to be specified as a MatLab .MAT file. Specify the name of the .MAT file in the .mat/.m File group box.
- **.m File:** Indicates that the system is to be specified as an .M file. Specify the name of the .M file in the .mat/.m File group box.

dT: Specifies the time step for the discrete system. By default, Commsim uses step size parameter from the Simulate menu’s Simulation Setup command.

Initial State: Specifies initial values for the states in the block. The values are right-adjusted. The right-most value corresponds to the lowest order state. Unspecified states are set to 0.

File Name: Indicates the name of the .M or .MAT file to be used as input to the `stateSpace` block. You can type the file name directly into this box or select one using the Select File button. To open the specified file with the default text editor, click on the Browse Data button.

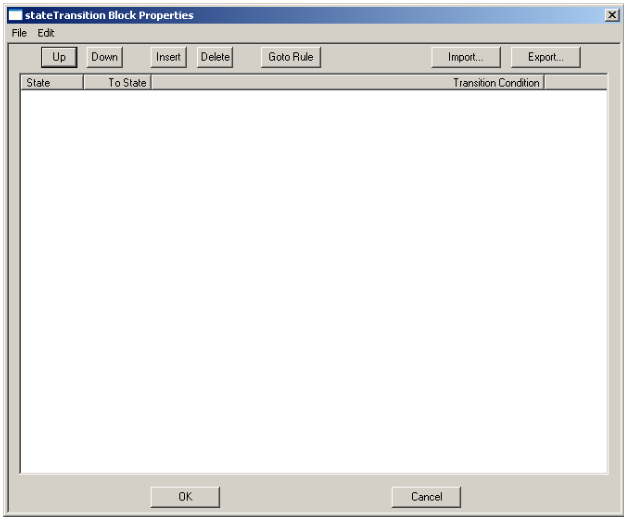
Input Count, Output Count, and State Count: Indicate the number of inputs to the block, the number of outputs from the block, and the number of system states. The number of system states is determined by the size of the A matrix. These options are read-only.

9.102 StateTransition

StateTransition state

Block category: State Transition

The `stateTransition` block models, simulates, and generates code for complex event-driven systems. A system can have a number of states, and for every state, you can have an arbitrary number of rules (or conditions) for transferring to a different state. The Boolean rules determine when to transition from one state to the next. Note that rules for a given state are evaluated in top-down order. The first transition rule to fire will take precedence over all other rules. Commsim maintains the states as members of a C language enumerated type. This means that states must be legal C identifiers: they cannot start with a number and must be composed of alphanumeric characters or underscore `(_)`.



State: Indicates the current state. All state names must be legal C identifiers because they will become enumerated types at C-generation time.

To State: Indicates the state to which to transition. The transition is made once the transition condition is met. All state manes must be legal C identifiers because they will become enumerated types at C-generation time.

Transition Condition: Indicates a Boolean condition or rule written in C. To get information from the diagram into the `stateTransition` block, you reference `variable` blocks in the diagram.

Export: Exports rules to a text file in .STC format. An .STC file is a semi-colon-separated text file that can be examined and edited in Notepad.

Import: Imports rules from the specified .STC file. An .STC file is a semi-colon-separated text file.

9.102.1Adding and deleting conditions

➢ To add a new condition:

1. Open the Properties box and point to the condition before which you want to add a new condition.
2. Press the INSERT button. A copy of the entire line is inserted.
3. To edit the copied line:
   - Point to the text under the State category and press the ENTER key.
   - Enter a new state or select one from the drop-down box.
   - Point to the text under the To State category and press the ENTER key.
   - Enter a new state or select one form the drop-down box.
   - Point to the text under the Transition Condition category and press the ENTER key.
   - Enter a new condition or select one from the drop-down box.

➢ To delete an existing condition:

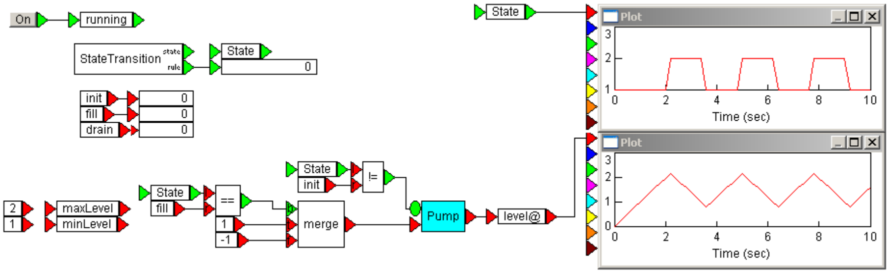
1. Open the Properties dialog box and point to the condition you want to delete.
2. Click the mouse once to highlight the rule.
3. Press the DELETE button.

➢ To move a rule:

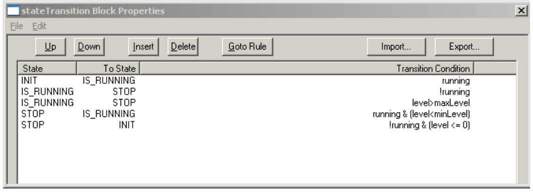
1. Open the Properties dialog box and point to the rule you want to move.
2. Click the mouse once to highlight the rule.
3. Press the UP or DOWN button to move the rule up or down.

Example

The following diagram uses a stateTransition block to simulate filling and draining a tank.



In this model, there are three states, defined in the Properties box for the stateTransition block.

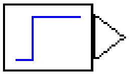


State	To State	Transition Condition
INIT	IS_RUNNING	running
IS_RUNNING	STOP	!running
IS_RUNNING	STOP	level>maxLevel
STOP	IS_RUNNING	running & (level<minLevel)
STOP	INIT	!running & (level <= 0)

In state INIT, the system is idle. In state IS_RUNNING, the tank is filling. In state STOP, the tank is draining. The button block controls whether the system is idle or running. At simulation start and the button block is OFF, the system is idling. When the button block is ON, state INIT transitions to state IS_RUNNING, and the tank begins to fill. When the tank reaches its maximum level, state IS_RUNNING transitions to state STOP, and the tank begins to drain. When the tank reaches its minimum level, state STOP transitions to state IS_RUNNING, and the tank begins to fill again. This seesaw effect can be observed in the plot block above.

You can vary the simulation results by turning the system ON and OFF using the button block. When you turn the system OFF while the tank is filling (state IS_RUNNING), the tank will immediately begin to drain (state IS_RUNNING transitions to state STOP). The system will continue to drain despite the fact the button block is OFF until the tank is empty. At this point, the system is in an idle state (state STOP transitions to state INIT).

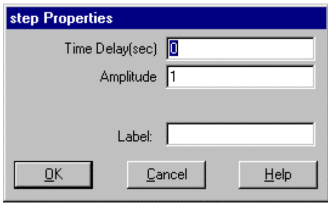
9.103 step



$$y = \begin{cases} 0 & \text{if } t < t_{\text{delay}} \\ \text{A} & \text{otherwise} \end{cases}$$

Block Category: Signal Producer

The `step` block creates a unit step signal.



step Properties

Time Delay(sec) 0

Amplitude 1

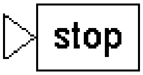
Label:

Time Delay(sec): Specifies, in seconds, how long to delay before calculating the value of the output signal. The delay is relative to the start of the simulation, and not absolute time. The default is 0.

Amplitude: Indicates the maximum strength of the output signal. The default is 1.

Label: Indicates a user-defined block label.

9.104 stop



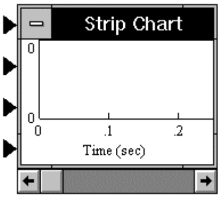
$$\text{If } x\begin{cases} > 2\text{haltsimulationunconditionally} \\ > 1\text{haltcurrentrun;startnextrun} \end{cases}$$

$$\text{Else normal}$$

Block Category: Signal Consumer

The `stop` block conditionally halts a simulation when the input signal is non-zero. For a multi-run simulation, when the input value is 1, Commsim halts the current run, increments $runCount, and starts the next run if the Auto Restart parameter in the dialog box for the Simulate > Simulation Properties command has been activated. When the input value is 2, Commsim stops the multi-run sequence altogether.

9.105 stripChart

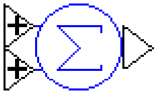


$$\text{stripchart} = x_1...x_4$$

Block Category: Signal Consumers

The `stripChart` block displays up to four signals in a customizable scrolling window.

9.106 summingJunction



$$y = x_1 + x_2 + \ldots x_n$$

Block Category: Arithmetic

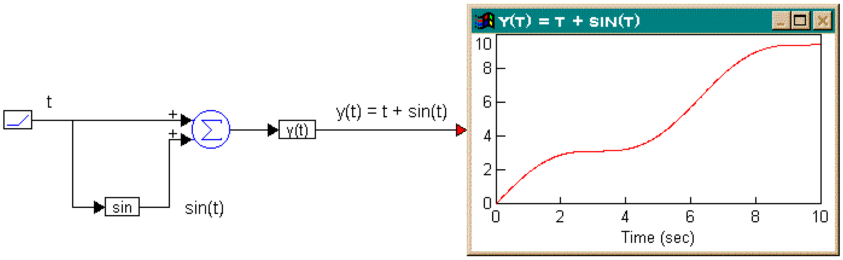
The `summingJunction` block produces the sum of two signed input signals. You can toggle the sign of the input signals (switch from positive to negative and vice versa) by holding down the CTRL key and clicking the right mouse button over the connector tab.

Inputs can be scalars, vectors, and matrices. When vector and matrix inputs are of unequal lengths, the `summingJunction` block defines the output vector or matrix to be the maximum composite size of all the incoming vectors or matrices and extends all other incoming vectors and matrices to match the length of the longest incoming vector or matrix, by padding each of them with the requisite number of zeros.

Examples

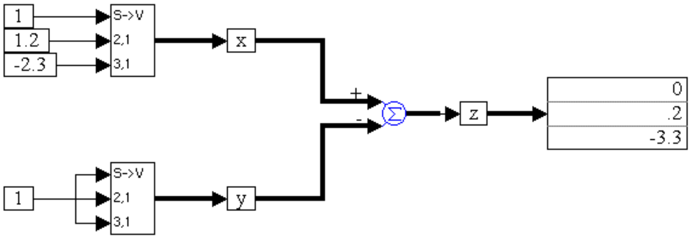
1. Addition of two scalar quantities

Consider the equation $y(t) = t + \sin(t)$. This can be realized as:



2. Subtraction of two vectors

Consider a vector **x** = [1 1.2 -2.3], and another vector **y** = [1 1 1]. The vector difference **z** = **x** - **y** can be computed directly by using a `summingJunction` block as:



The vector display shows that the result of the simulation is a direct element-by-element subtraction of vector **y** from vector **x**.

3. Matrix addition and subtraction

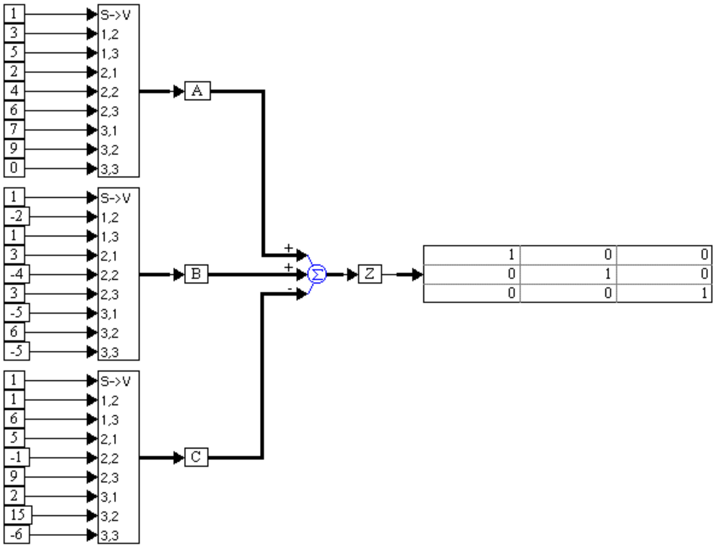
Consider the matrix equation:

Z = ***A*** + ***B*** - ***C***

where:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 9 & 0 \end{bmatrix}; B = \begin{bmatrix} 1 & -2 & 1 \\ 3 & -4 & 3 \\ -5 & 6 & -5 \end{bmatrix}; C = \begin{bmatrix} 1 & 1 & 6 \\ 5 & -1 & 9 \\ 2 & 15 & -6 \end{bmatrix}$$

This equation can be realized as:



1	0	0
0	1	0
0	0	1

The matrix display shows that e result of the simulation is a direct element-by-element matrix operation of ***A*** + ***B*** - ***C***.

9.107 tan

$$y = \tan x$$

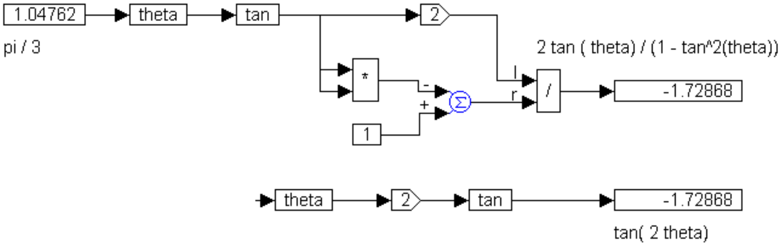
Block Category: Transcendental

The `tan` block produces the tangent of the input signal. The input signal must be represented in radians.

Examples

1. Computation of tan(2θ) = 2 tan(θ) / (1 - tan^2(θ))

With θ chosen to be π/3, the above trigonometric identity can be realized as:



9.108 tanh

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

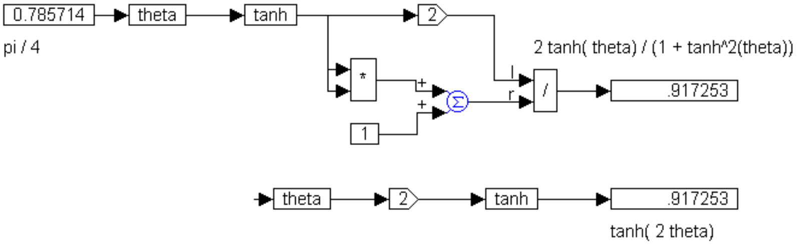
Block Category: Transcendental

The `tanh` block produces the hyperbolic tangent of the input signal. The input signal must be represented in radians.

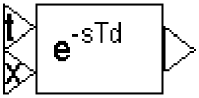
Examples

1. Computation of tanh(2θ) = 2 tanh(θ) / (1 + tanh^2(θ))

With θ chosen to be π/4, the above trigonometric identity can be realized as:



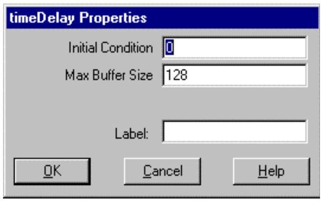
9.109 timeDelay



$$y = x(t - T_d)$$

Block Category: Time Delay

The `timeDelay` block delays the input signal for an absolute time. The input connector tabs are marked t (for the time delay) and x (for the main signal). This block is intended to model a continuous delay in a continuous simulation. Use the `unitDelay` block to model a digital delay.



timeDelay Properties

Initial Condition 0

Max Buffer Size 128

Label:

Initial Condition: Sets an initial condition for the delay. The default is 0.

Max Buffer Size: Controls the granularity of the resulting timeDelay signal. If the signal is too granular, increase the value. The default is 128.

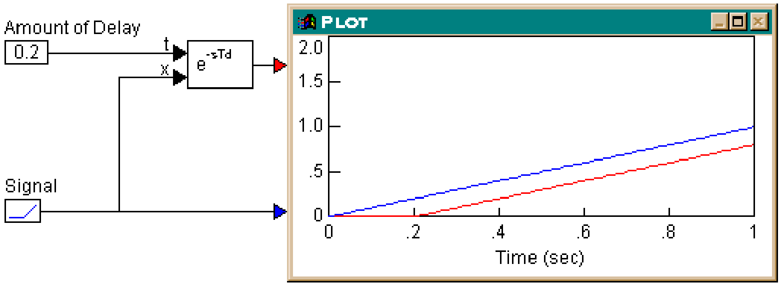
The `timeDelay` block requires a buffer element for each time step in the requested delay amount. The buffer size should be set to the maximum delay time you need divided by the simulation time step.

Label: Indicates a user-defined block label.

Examples

1. Introduction of a constant delay

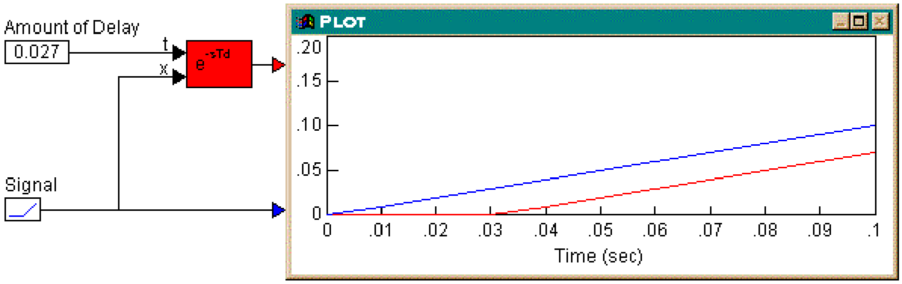
For a given signal, a constant delay can be introduced as shown below.



Here, a `ramp` block is used to produce a test signal and a `const` block is used to produce a time delay of 0.2. For this example, the simulation step size is set to 0.01.

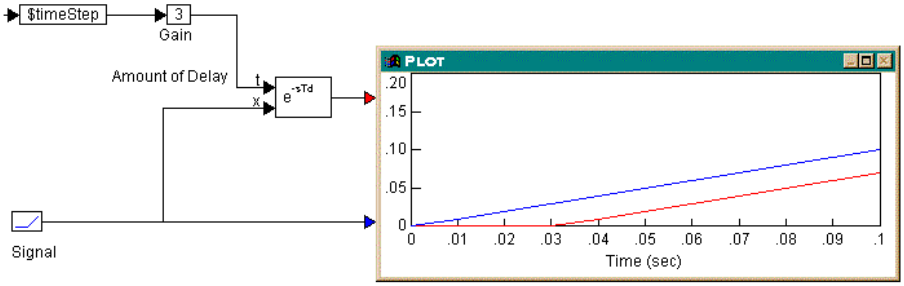
The amount of delay connected to the t input tab must be an integral multiple of the simulation step size. If you had entered 0.027 as the amount of delay and re-run the simulation, Commsim would have issued an error message. This error occurs because Commsim starts the simulation at *t* = 0, and steps through at intervals of 0.01. The time intervals that are hit are 0, 0.01, 0.02, 0.03, and so on. The time delay 0.027 is not honored.

If you choose to ignore the error (by clicking on the Retry or Ignore buttons in the message box), Commsim rounds off the delay to the next nearest time step, which in this case happens to be 0.03, as shown below:



2. Introduction of an integral-multiple delay

One way to achieve multi-step delays is by using the `timeDelay`, `$timeStep`, and `gain` blocks, as shown below.



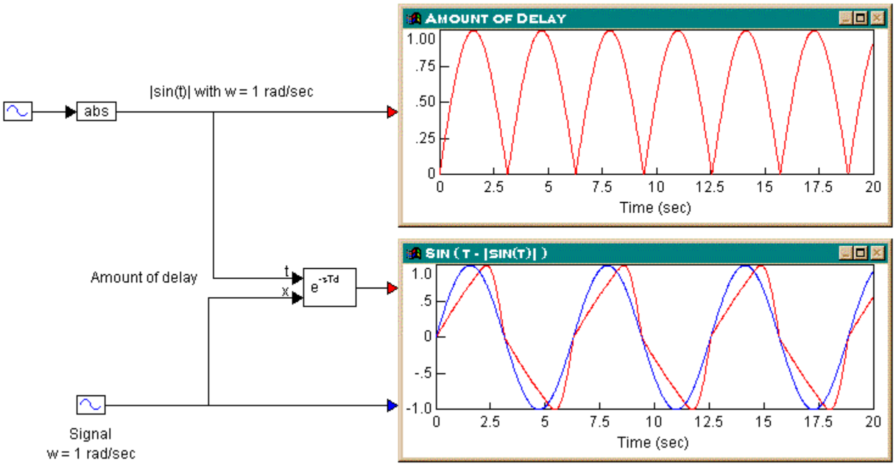
During simulation, the value sent to the t input of the `timeDelay` block is 3 * `$timeStep`, and as a result, the output of the `timeDelay` block is three steps behind the input signal.

3. Introduction of a time-varying delay

The real power of the `timeDelay` block becomes apparent when you implement time delays that are themselves time-varying. As an example, consider the following equation:

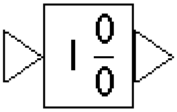
$y = \sin(t - |\sin(t)|)$

Here, the intent is to delay (or shift right) a sinusoid of frequency ω = 1 rad/sec, by a time-varying amount given by the absolute value $|\sin(t)|$. This can be realized as:



An `abs` block computes the absolute value of sin(t), generated by a `sin` block. The output of the `abs` block is fed to the t input of the `timeDelay` block. Another `sin` block generates the actual signal to be delayed. The top `plot` block shows the time-varying delay being implemented. The bottom `plot` block shows the actual and delayed signals.

9.110 transferFunction



$$y = \frac{a_n s^n + a_{n-1} s^{n-1} ... a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} ... b_1 s + b_0} x$$

Block Category: Linear System

The `transferFunction` block executes a single-input single-output linear transfer function specified in the following ways:

- **As an .M file created with Commsim:** The Linearize command in the Analyze menu generates ABCD state-space matrices from the nonlinear system by numerically evaluating the matrix perturbation equations at the time the simulation was halted.
- **As an .M file created with a text editor:** The following is an example of a user-written .M file:

  function [a,b,c,d] =vabcd

  a = [-.396175 -1.17336 ; 5.39707 .145023];

  b = [-.331182 ; -1.08363];

  c = [0 1];

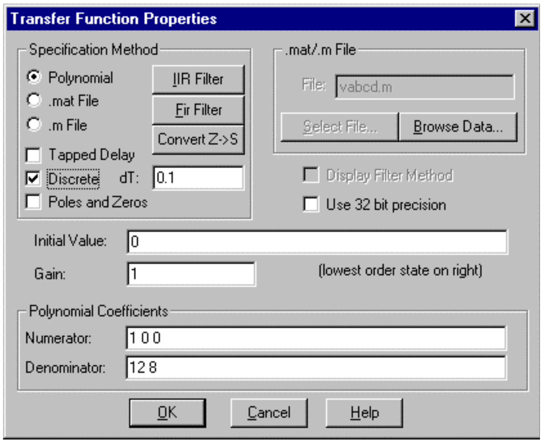
  d = [0];
- **As a .MAT file created with MATLAB:** Generating .MAT files is described in the MatLab documentation. Note that when you save the ABCD matrices to file, the names of the matrices are not important; however, the order in which they appear is.

When you simulate the block diagram, Commsim numerically solves the `transferFunction` block.

You can set initial conditions for `transferFunction` blocks using variables. You can also reset `transferFunction` blocks to zeros using the Simulate > Reset States command.

Digital filter design: The `transferFunction` block supports IIR and FIR digital filter design.

Setting up a transfer function: The `transferFunction` block's Properties dialog box allows you to control how the numerator and denominator polynomials are entered.



Transfer Function Properties

Specification Method

● Polynomial

○ .mat File

○ .m File

□ Tapped Delay

☑ Discrete dT: 0.1

□ Poles and Zeros

.mat/.m File

File: vabcd.m

□ Display Filter Method

□ Use 32 bit precision

Initial Value: 0

Gain: 1

(lowest order state on right)

Polynomial Coefficients

Numerator: 1 0 0

Denominator: 1 2 8

Specification Method: You have three choices of specification method:

- **Polynomial Coefficient:** Indicates that the transfer function is to be specified as numerator and denominator polynomials. Supply the numerator and denominator polynomials and gain under the Polynomial Coefficients group box.
- **.mat File:** Indicates that the transfer function is to be specified as a .MAT file. Specify the name of the .MAT file in the .mat/.m File group box.
- **.m File:** Indicates that the transfer function is to be specified as an .M file. Specify the name of the .M file in the .mat/.m File group box.

Discrete: Indicates a discrete Z-Domain transfer function. Enter the time step for the discrete transfer function in the dT box. By default, Commsim uses the step size established with the Simulate > Simulation Properties command.

When Discrete is de-activated, a continuous transfer function is created.

Tapped Delay: Provides tapped delay implementation for high order FIR filters.

dT: Specifies the time step for the discrete transfer function. By default, Commsim uses step size parameter from the Simulate > Simulation Properties command.

Poles and Zeros: Lets you enter a transfer function via zeros and poles in the Numerator and Denominator boxes, respectively. Enter each root in the following format:

(real-part, imaginary-part)

For large order systems, poles and zeros is more numerically accurate.

Display Filter Method: Displays the filter specification on the block. When Display Filter Method is not activated, Commsim displays the polynomial coefficients.

32 Bit Precision: Simulates the behavior of the transfer function at 32-bit precision. Normally, all Commsim blocks are 64-bit precision. This parameter allows you to simulate the effect of code running on a floating point 32-bit target.

IIR Filter: Opens the IIR Filters Setup dialog box to design a suitable filter using analog prototypes.

FIR Filter: Opens the FIR Filter Setup dialog box to construct Regular Finite Impulse Response filters, differentiators, and Hilbert Transformers.

Convert S ->Z: Uses bilinear transformation to convert a continuous transfer function to an equivalent discrete transfer function with a sampling interval of *dT*. Commsim requests a discrete sampling rate prior to performing the conversion.

An example of the conversion is shown below.

$$H(s) = \frac{a}{s+a}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2}{dT}\frac{z-1}{z+1} \rightarrow s$$

The above transfer function becomes:

$$H_{dT}(z) = \frac{a}{\left(\frac{2}{T}\right)\left\{\frac{z-1}{z+1}\right\} + a}$$

Commsim automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

Convert Z ->S: Uses bilinear transformation to convert a discrete transfer function to an equivalent continuous transfer function. For example, consider:

$$H_{dT}(z) = \frac{z}{z+b}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2+dT.s}{2-dT.s} \rightarrow z$$

The above discrete transfer function becomes:

$$H_{dT}(s) = \frac{2 + dT.s}{(2 + 2b) + (dT - b.dT)s}$$

Commsim automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

It is important to note that in both transformations, the results obtained are dependent on the sampling interval *dT*. In other words, for a given continuous or discrete transfer function, an infinite number of equivalent discrete or continuous transfer functions may be obtained by varying the sampling interval *dT*.

File: Indicates the name of the .M or .MAT file to be used as input to the `transferFunction` block. You can type the file name directly into this box or select one using the Select File button.

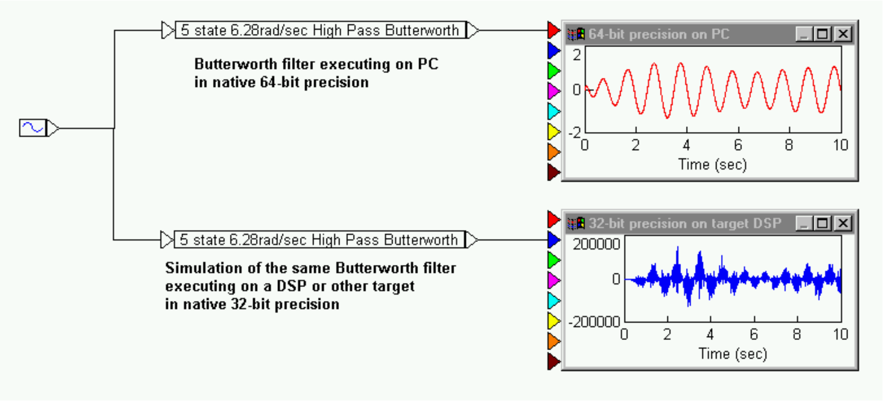
Initial Value: Specifies initial values for the states in the block. The values are right-adjusted. The right-most value corresponds to the lowest order state. Unspecified states are set to 0.

Gain: Indicates the transfer function gain. If the leading terms of the numerator and denominator coefficients are not unity, Commsim will adjust the gain to make it so. The default value is 1.

Denominator: Indicates the denominator polynomial for the `transferFunction` block. Commsim determines the order of the transfer function by the number of denominator coefficients you enter. For example, an nth order transfer function will have n + 1 coefficients. Separate coefficients with spaces.

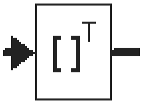
Numerator: Indicates the numerator polynomial for the `transferFunction` block. Separate coefficients with spaces.

Examples


5 state 6.28rad/sec High Pass Butterworth
Butterworth filter executing on PC
in native 64-bit precision
64-bit precision on PC
2
0
-2
0
2
4
6
8
10
Time (sec)
5 state 6.28rad/sec High Pass Butterworth
Simulation of the same Butterworth filter
executing on a DSP or other target
in native 32-bit precision
32-bit precision on target DSP
200000
0
-200000
0
2
4
6
8
10
Time (sec)

This example demonstrates how Commsim can simulate a Butterworth filter in reduced precision, to simulate the effects of implementing the system on a reduced-precision 32-bit floating point target. To turn on reduced precision, activate the 32-Bit Precision option in the dialog box for the `transferFunction` block.

9.111 transpose



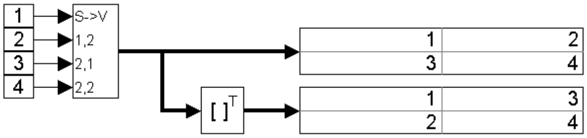
$$[a_{ij}]^T = [a_{ji}]$$

Block Category: Matrix Operation

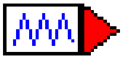
The `transpose` block interchanges each row with the column of the same index number. Thus, if $\mathbf{A} = [a_{ij}]$, then the transpose of **A** is $\mathbf{A}^T = [a_{ji}]$.

The `transpose` block accepts one vector input and produces one vector output.

Examples

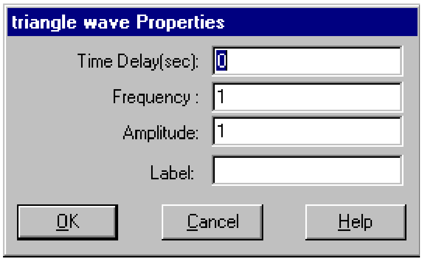


9.112 triangleWave block



Block category: Signal Producer

The `triangleWave` block creates a unit triangle wave signal.



triangle wave Properties

Time Delay(sec): 0

Frequency : 1

Amplitude: 1

Label:

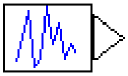
Time Delay (sec): Indicates the time offset that is used in the calculation of the signal. Specify the offset in seconds. The default is 0.

Frequency: Controls the frequency of oscillation of the output signal. Specify the frequency in hertz. The default is 1.

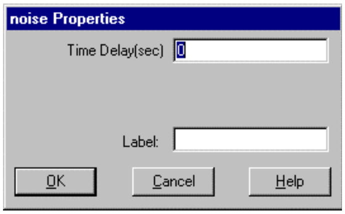
Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Label: Indicates a user-specified label.

9.113 uniform



The `uniform` block creates a uniformly distributed random noise signal with values between zero and one. The random seed is set under Preferences in the dialog box for the Simulate > Simulation Properties command.



noise Properties

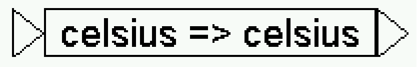
Time Delay(sec) 0

Label:

Time Delay(sec): Indicates, in seconds, how long to delay before calculating the value of the noise signal. The default is 0.

Label: Indicates a user-defined block label.

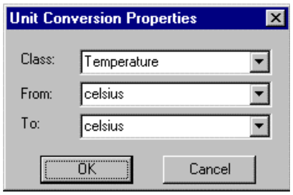
9.114 unitConversion



Block Category: Arithmetic

The `unitConversion` block changes the unit of measurement of the data. You can convert the unit of measurement within numerous categories, including: acceleration, area, capacitance, charge, conductivity, current, energy, flow rate, force, inductance, magnetic flux, mass, position, power, pressure, speed, temperature, volume, and more. For example, you can convert from Fahrenheit to Celsius, watts to kilowatts, or joules to BTUs.

Conversions are always displayed on the block.



Unit Conversion Properties

Class: Temperature

From: celsius

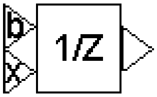
To: celsius

Class: Indicates the category of measurement.

From: Indicates the unit of measurement for the data exiting the block. Click on the DOWN ARROW to select a unit of measurement.

To: Indicates the unit of measurement for the data entering the block. Click on the DOWN ARROW to select a unit of measurement.

9.115 unitDelay



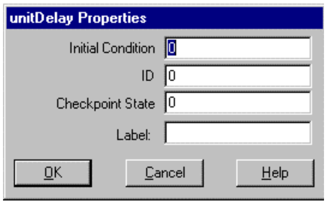
$$y = \begin{cases} y_{\text{buffer}},\, y_{\text{buffer}} = x_2 \text{ if} \left| x_1 \right| \geq 1 \\ y_{\text{previous}} \quad \text{otherwise} \end{cases}$$

Block Category: Time Delay

The `unitDelay` block specifies a clocked unit delay. The input connector tabs are marked b (for Boolean clock) and x (for main signal). When the Boolean clock does not equal zero, the value contained in the single element buffer is copied to the block output (where it holds this value until the next non-zero Boolean clock). The current value of the main signal is stored in the unit buffer.

The `unitDelay` block is intended for modeling a digital delay in a continuous simulation. A typical digital delay is modeled by wiring a `pulseTrain` block to the Boolean input connector tab of the `unitDelay` block. Use the `timeDelay` block to model a continuous delay.

You can set the initial conditions for `unitDelay` blocks with variables. You can also reset `unitDelay` blocks to zeros using the Simulate > Reset States command.



unitDelay Properties

Initial Condition: 0

ID: 0

Checkpoint State: 0

Label:

Initial Condition: Sets an initial value for the output signal. You can enter the initial condition in scalar or vector notation. The initial condition matrix will set the matrix dimensionality for the block. The default is 0.

ID: Reserved for future use.

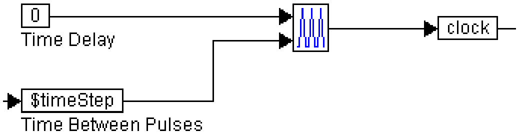
Checkpoint State: Contains the value of the unit delay at the checkpoint. If you have not checkpointed your simulation via the Simulate > Simulation Properties command, the value is 0.

Label: Indicates a user-defined block label.

Examples

1. Clocking the `unitDelay` block

If you are working with `unitDelay` blocks, it is good programming practice to create a clock signal that you can use in every simulation. A typical clock signal can be generated as:



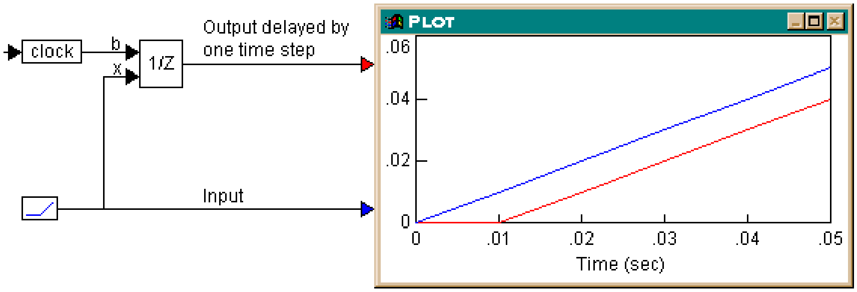
Here, a `pulseTrain` block is assigned two external inputs:

- The top input is the time delay for the `pulseTrain` block. The time delay value for the `pulseTrain` block is the amount of time the `pulseTrain` block waits before producing pulses. This time delay value must not be confused with the amount of time delay generated by the `unitDelay` block.
- The bottom input is the time between pulses.

The output of the `pulseTrain` block is fed to the `variable` *clock*. This variable can be used anywhere in the simulation to clock `unitDelay` blocks.

2. Introduction of a one-step delay

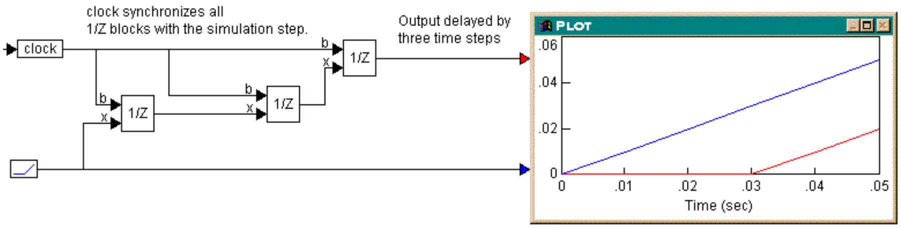
For a given signal, a one-step delay can be introduced as:



During simulation, the actual and delayed signals are plotted in the `plot` block. The output of the `unitDelay` block is delayed by one step (equal to 0.01 in this case) as compared to the input.

3. Using a multi-step delay with cascaded `unitDelay` blocks

To achieve multi-step delays, `unitDelay` blocks that implement one-step delays, can be cascaded. Consider the example where a three-step delay is introduced:



Output of three cascaded unitDelays

Three `unitDelay` blocks, all clocked at the simulation step, are cascaded. Since each `unitDelay` introduces a one-step delay between its input and output, the output of the third `unitDelay` block is delayed by three steps compared to the input. The `plot` block shows this behavior, with a simulation step size of 0.01.

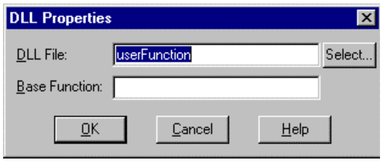
9.116 unknown

Block Category: Optimization

The unknown block works in conjunction with constraint blocks to solve equations for unknowns using Newton-Raphson iteration. For each unknown, there should be a constraint block that is fed directly or indirectly by the unknown. The maximum iteration count, error tolerance, and perturbation are established under the Implicit Solver tab in the dialog box for the Simulate > Simulation Properties command.

9.117 userFunction

The userFunction block lets you create blocks bound to Dynamic Link Library (DLL) functions.



DLL Properties

DLL File: userFunction

Base Function:

DLL File: Indicates the name of the DLL file containing the user function.

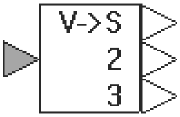
Base Function: Indicates the base name of the function.

9.118 variable

Block Category: Annotation

The `variable` block lets you name a signal and transmit it throughout your diagram without the use of wires.

9.119 vecToScalar



Block Category: Annotation

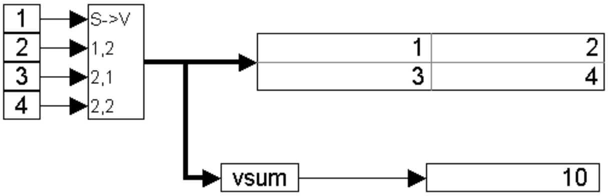
The `vecToScalar` block separates a single vector wire into individual output signals. Use the `scalarToVec` block to bundle signals into a single vector wire.

9.120 vsum

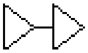
Block Category: Matrix Operation

The `vsum` block produces a single value summation of all the elements in the matrix. The `vsum` block accepts one vector input and produces one scalar output.

Examples



9.121 wirePositioner

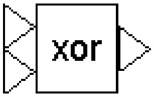


Block Category: Annotation

The `wirePositioner` block lets you create a specific wiring path. A `wirePositioner` block is essentially an input connector tab and an output connector tab that are attached by a flexWire. Since `wirePositioner` blocks don't take any additional computation time, you won't see a decrease in performance during a simulation.

Input to the `wirePositioner` block can be scalar or vector.

9.122 xor



$$y = x_1 \text{bitwiseXOR} x_2$$

Block Category: Boolean

The `xor` block produces the bitwise exclusive OR of two to 256 scalar input signals.

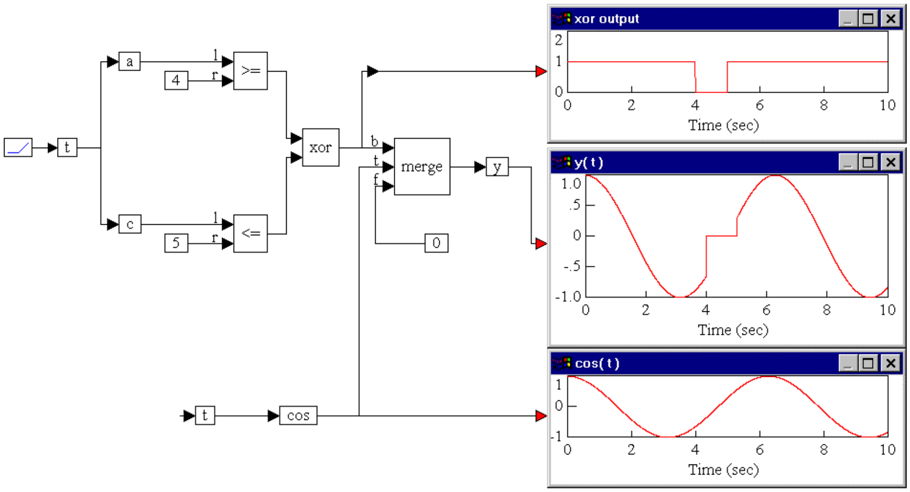
If you click the right mouse button over the `xor` block, the Boolean block menu appears allowing you to assign a different function to the block.

Examples

1. Using the `xor` block

Consider a variable *y* such that:

If *a* ≥ 4 or *c* ≤ 5, then *y* = cos(*t*); else *y* = 0. Also, if *a* ≥ 4 and *c* ≤ 5, y = 0 where *t* is simulation time. Furthermore, let *t* be the input to parameters *a* and *c*. This system can be realized as shown below.



As shown in the two `plot` blocks, the output of the `xor` block evaluates to false in the interval *t* = (4, 5), since both the inputs to the `xor` block are true in this interval. Consequently, *y* takes on the value of 0. The output of `xor` evaluates to true in the remaining parts of the simulation, and as a result, *y* takes on the value of cos(*t*) in these periods.

Chapter 10
Customizing Commsim

The following are described in this chapter.

10.1 Customizing Commsim start-up

By adding arguments to the Commsim start-up command, you can control such things as how Commsim starts up, the block diagram file opened at start-up, and whether Commsim immediately simulates the opened diagram.

➢ To customize Commsim start-up:

1. Refer to your Windows documentation for instructions on displaying the properties of a program.

2. Click on the Shortcuts tab and enter one or more of the following arguments in the Target box.

   If you enter more than one argument, separate them with spaces. If a block diagram name is included in the argument list, it must be specified last.

Use this argument	To
block-diagram-name	Start Commsim and open the specified block diagram.
-i [*block-diagram-name*]	Start Commsim as an icon and optionally open a block diagram.
-nb [*block-diagram-name*]	Start Commsim without the start-up banner and optionally open a block diagram.
-ne	Suppress the simulation completion dialog box.
-r *block-diagram-name*	Run a simulation read in from the specified block diagram upon start up.
-re *block-diagram-name*	Run a simulation read in from the specified block diagram and exit Commsim upon completion.

3. Click on the OK button.

10.2 Customizing the Commsim window

The Commsim window contains a menu bar, toolbars, scroll bars, a status bar, and a diagram tree. You can display or hide these items at any time during a Commsim session. For example, if you're working on a large a block diagram, you may want to hide the status bar and diagram tree so you can see as much of the diagram as possible. When you simulate the diagram, you may want to display the status bar to keep track of progress of the simulation.

➢ To hide or display Commsim window elements, do one or more of the following:

To display or hide	Do this
Scroll bars	Choose Edit > Preferences. Select the Preferences tab, and select or clear the Show Horizontal Scroll Bar or Show Vertical Scroll Bar check box.
Status bar	Choose View > Status Bar.
Toolbar	Choose View > Toolbar. Select or clear each toolbar check box.
Diagram tree	Drag the right edge of the diagram tree.

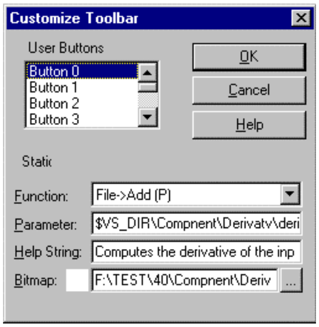
10.2.1 Customizing the toolbar

You can create a custom toolbar that contains buttons for commands and blocks you use most frequently. You can also create your own images for the buttons. The custom toolbar is named User.

➢ To create a custom toolbar button:

1. Choose View > Toolbar.
2. Activate the User option, if it is not already activated, then click on the OK button, or press ENTER.
3. Choose Edit > Toolbar.

The Customize Toolbar dialog box appears.



Function: File->Add (P)

Parameter: $VS_DIR\Compnent\Derivatv\deri

Help String: Computes the derivative of the inp

Bitmap: F:\TEST\40\Compnent\Deriv

4. Under User Buttons, select a button number.

5. In the Function box, click on the DOWN ARROW and select the command to be assigned a toolbar button. If you select:
   - Blocks (P), click on the DOWN ARROW in the Parameter box and select a block to be associated with the toolbar button.
   - Edit Find (P), File Add (P), or File Open (P), you can optionally enter a variable block name or file name in the Parameter box. If you do not enter anything, Commsim opens the Find, File Add, or File Open dialog box when you click on the toolbar button.
6. In the Help String box, you can optionally enter text that will appear in the status bar when you point to the toolbar button. If you do not enter anything, Commsim displays the default help string for the toolbar button.
7. Click on the Find button to choose a bitmap for the toolbar button. Pre-made button bitmap images can be found in \COMMSIM7\BITMAPS\TOOLBAR.

   Custom bitmap should be 16-pixels wide by 15-pixels high for the best display.
8. Click on the OK button, or press ENTER.

➢ To remove a custom toolbar button:

1. Choose Edit > Toolbar.
2. From the toolbar button list, select the toolbar number that corresponds to the toolbar button to be removed from the toolbar.
3. In the Functions box, click on the DOWN ARROW and select <none>.
4. Click on the OK button, or press ENTER.

10.2.2 Customizing other screen settings

In addition to changing general Commsim settings, you can also change many other settings to customize how Commsim looks, such as the use of colors and text fonts in your diagrams, the shape and color of connector tabs, and the amount of information displayed with each block in a diagram.

These settings are controlled with the Edit > Preferences command and the commands under the View menu. When you select a setting, it takes effect immediately and remains in effect until you change it.

➢ To change the display settings in Commsim, do one or more of the following:

To	Do this
Hide input connector tabs and shrink the size of output connector tabs	Choose View > Presentation Mode.
Hide wires and connector tabs, freeze blocks in place, and with the exception of interactive elements on `button` and `slider` blocks, lock block parameter values	Choose View > Display Mode. Typically this mode is used when there is animation in your simulation or you've constructed an instrumentation panel to monitor and control your simulation. Display mode can be turned on or off for individual block diagram levels.
Magnify the blocks on the screen.	Choose View > Zoom In. You can alternatively click on the ⇑ button on the toolbar.
Shrink the blocks on the screen.	Choose View > Zoom Out. You can alternatively click on the ⇓ button on the toolbar.
Color connector tabs according to the type of data entering or exiting the block.	Choose View > Data Types. The four types of data and their corresponding connector tab colors are double-floating point (red), signed integer (green), unsigned integer (blue), and vector (magenta).
Display connector labels on compound blocks	Choose View > Connector Labels.
Change how text is displayed on blocks	Choose View > Fonts. Select the text attributes. Character format can be selectively applied to `label` blocks. Rich text format can be retained in `comment` blocks, as described below.
Retain character format in `comment` blocks	Choose Edit > Preferences. Select the Preferences tab. Activate Use Rich Text Format.

Change the color of the Commsim screen; plotting background on `plot`, `stripChart`, and `histogram` blocks; wires; and diagram text	Choose View > Colors. Select the color for the corresponding screen element. When you choose a default color for the plotting background, Commsim uses the specified color on all `meter`, `plot`, and `stripChart` blocks except those whose background colors were explicitly set in their Properties dialog boxes.
Display block names beneath each block	Choose Edit > Preferences. Select the Preferences tab. Activate Training Mode Labels. Then choose View > Block Labels.
Display parameter values, file names, and block names beneath each block	Choose Edit > Preferences. Select the Preferences tab. Clear Training Mode Labels. Then choose View > Block Labels.
Color compound blocks light blue	Choose Edit > Preferences. Select the Preferences tab. Activate Color Compound Blocks.
Display block diagrams in black and white	Choose Edit > Preferences. Select the Preferences tab. Clear Color Compound Blocks and Color Display.

10.2.3 Adding Commsim addons

The Edit > Preferences command also lets you add and remove Commsim addons by inserting the corresponding .DLL file in the Addons window. Note that if a .DLL uses the set user menu function (which allows the blocks contained in the DLL to be inserted into Commsim menus), you must insert the DLL in the Addons window. Also, all DLLs generated using Commsim/C-Code automatically include the set user menu function. Hence, you must include the DLL in the Addons window in order to access the Commsim/C-Code-generated blocks in the Commsim menus.

➢ To add a DLL:

After you add the DLL, the blocks contained in the DLL are automatically added to the Commsim menus.

1. Choose Edit > Preferences.

   The Preferences dialog box appears.

2. Select the Addons tab.
3. In the Commsim addon list, double-click on the ....

   The Open dialog box appears.

4. Select the DLL to be added into the Addons window.
5. Click on the Open button.

   In the Addons window, the complete path of the DLL is displayed.
6. Click on the OK button, or press ENTER.

➢ To remove a DLL:

After you remove a DLL, you must restart Commsim to see the change.

1. Choose Edit > Preferences.

   The Preferences dialog box appears.
2. Select the Addons tab.
3. In the Commsim addon list, select the DLL to be removed.
4. Click on the Delete button.
5. Restart Commsim.

10.3 Creating custom implicit solvers

You can write an implicit static solver as a .DLL file. Commsim recognizes and uses a user-written solver only if:

- It is named VSOLVER.DLL
- It resides in your current directory when you initiate implicit static solving
- It contains an exported function called userSolver()
- The User Solver parameter in the dialog box for the Simulate > Optimization Properties command is activated

10.3.1 Source files for building a custom implicit solver

The following table lists the source files for building an implicit static solver. These files are installed in \COMMSIM7\VSOLVER and \COMMSIM7\VSDK. They contain code for building a simplified Gauss-Seidel static solver. You may find it easier to edit the files to create your own static solver. To use these files, they must remain in the directories in which they currently reside.

Source file	Description
VSOLVER.FOR or VSOLVER.C	A Fortran or C source file for the implicit static solver. The heart of the solver is the vissimRequest() function that you call to obtain the inputs to the constraint blocks and to supply values to the outputs of the unknown blocks. Using vissimRequest(), you can write a wide variety of solvers. For more information, see the description below.
VSOLVER.DEF	A definition file with linker commands to build a .DLL file from object code. Windows requires that you use a definition file to link the object code.
VSOLVER.MAK	A make file with rules for automatically building a .DLL file.
VSUSER.H	A C language header file with function prototypes and command definitions for the vissimRequest() function.

10.3.2 Using vissimRequest() in a custom implicit solver

The vissimRequest() function is a general-purpose function for making requests to Commsim. A user-written solver uses vissimRequest() to read and write optimization information in a block diagram. The general format of vissimRequest() is:

```
long FAR vissimRequest(long req, long arg2, long arg3 )
```

The first argument (long req) is a message code describing the action for Commsim to take. The list of message codes is defined in the file named VSUSER.H, which is installed in \COMMSIM7\VSOLVER. The message codes that pertain to writing a local static solver are listed in the table below.

Message code	Description
VR_EXECUTE	Executes the diagram on iteration without moving time.
VR_GET_BLOCK_PARAMS	Returns a pointer to a block's parameters.
VR_GET_CONSTRAINTS	Arg2 returns a vector of local constraint values. Ordering of the elements vector can be determined by the value of the ID parameter for the constraint block. Commsim sorts in sequential order, from low to high.
VR_GET_SOLVER_INFO	Arg2 returns information related to the diagram and the implicit solver dialog settings in the following manner: arg2[0] = number of constraints arg2[1] = number of unknowns arg2[2] = relaxation value arg2[3] = maximum iteration value arg2[4] = error tolerance value
VR_GET_UNKNOWNS	Arg2 returns a vector of current local unknown output values. Ordering of the elements vector can be determined by the value of the ID parameter for the unknown block. Commsim sorts in sequential order, from low to high.
VR_GET_UNKNOWNS_INPUT	Arg2 returns a vector of current inputs to the unknown blocks. Ordering of the elements vector can be determined by the value of the ID parameter for the unknown block. Commsim sorts in sequential order, from low to high. (This is useful for initial condition setting.)
VR_GET_VERSION	Returns the current version of Commsim.
VR_GET_VISSIM_STATE	Gets information related to the global state of Commsim. The information provided is a copy of the current internal state; modifying it will not change Commsim's state. Arg2 should contain a pointer to a SIM_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() function. Arg3 should contain the size of this structure (sizeof(SIM_INFO)) to allow for version compatibility checking.
VR_SET_UNKNOWNS	Sets diagram unknowns based on the vector passed as arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the unknown block. Commsim sorts in sequential order, from low to high.

10.3.3 Building a custom implicit solver

Most languages have a Project Build facility that automates the process of building a .DLL file. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you're using for specific instructions.

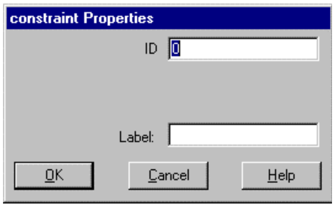
➢ To build a custom implicit solver:

1. Invoke the Compiler environment.
2. Add all the source files listed in "10.3.1 Source files for building a custom implicit solver" on page 10-8 to the project or make file.
3. Under project options, specify the project type as a Windows Dynamic Link Library (.DLL).
4. Under compiler preprocessor options, specify \COMMSIM7\VSDK as the include directory.
5. Build the project.

10.3.4 Using the constraint block with a custom implicit solver

To indicate the number that Commsim uses to sort the block when presented as a vector in a user-written solver, enter it in the ID box of the Constraint Properties dialog box. Commsim does not require the ID to be unique or contiguous; it sorts them in sequential order.

The default is 0.



ID 0

Label:

10.4 Creating custom global optimizers

You can write a global optimizer as a .DLL file. Commsim recognizes a user-written global optimizer when it is named VOPT.DLL and resides in your current directory. VOPT.DLL should also contain an exported function in the following format:

```
int FAR EXPORT USER_OPT_FUNC(DOUBLE *unknownVec, int
unknownCount, int costCount, int globalConstraintCount);
```

Optimize has a prototype declared in VSUSER.H.

Before you initiate global optimization, make sure VOPT.DLL is in your current directory and the User Solver parameter in the dialog box for the Simulate menu's Optimization Setup command is activated.

10.4.1 Source files for building a custom global optimizer

The following table lists the source files for building a global optimizer. These files are installed in \COMMSIM7\VSOLVER and \COMMSIM7\VSDK.

Source file	Description
VOPT.C	A C source file for a sample global optimizer. The heart of the optimizer is the vissimRequest() function that you call to obtain the inputs to the cost blocks and to supply values to the outputs of the parameterUnknown blocks. Using vissimRequest(), you can write a wide variety of optimization algorithms. For more information, see the description below.
VOPT.DEF	A definition file that contains linker commands to build a .DLL file from object code.
VOPT.MAK	A make file that contains rules for automatically building a .DLL file.
VSUSER.H	A C language header file that contains function prototypes and command definitions for the vissimRequest() call.
IMPSIM.LIB	Commsim import library that describes the address of vissimRequest().

10.4.2 Using vissimRequest() in a custom global optimizer

The vissimRequest() function is a general-purpose function for making requests to Commsim. A user-written global optimizer uses vissimRequest() to read and write global optimization information in a block diagram. The general format of vissimRequest() is:

```
long FAR vissimRequest(long req, long arg2, long arg3 )
```

The first argument (long req) is a message code describing the action for Commsim to take. The list of message codes is defined in the file named VSUSER.H, which is installed in \COMMSIM7\VSOLVER. The message codes that pertain to writing a global optimizer are as follows:

Message code	Description
VR_GET_GLOBAL_COST	Writes a vector of current cost block input values into a vector pointed at by arg2.
VR_GET_GLOBAL_CONSTRAINTS	Writes a vector of current globalConstraint block input values into a vector pointed at by arg2.
VR_GET_GLOBAL_CONSTRAINT_BOUNDS	Writes a vector of globalConstraint block low bounds into a vector pointed at by arg2, and a vector of globalConstraint block high bounds into a vector pointed at by arg3.
VR_GET_GLOBAL_OPT_INFO	Gets information related to the global optimization settings in the dialog box for the Optimization Setup command. The information provided is a copy of the current optimization state; modifying it will not change Commsim's state. Arg2 should contain a pointer to an OPT_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() call. Arg3 should contain the size of this structure (sizeof(OPT_INFO)) to allow for version compatibility checking.
VR_GET_GLOBAL_UNKNOWNS	Writes a vector of current parameterUnknown block output values into the vector pointed at by arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the parameterUnknown block. Commsim sorts in sequential order, from low to high.

VR_GET_GLOBAL_UNKNOWNS_INPUT	Writes a vector of current parameterUnknown block input values into the vector pointed at by arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the parameterUnknown block. Commsim sorts in sequential order, from low to high.
VR_GET_GLOBAL_UNKNOWN_BOUNDS	Writes a vector of parameterUnknown block low bounds into a vector pointed at by arg2, and a vector of parameterUnknown block high bounds into a vector pointed at by arg3.
VR_GET_VERSION	Returns the current version of Commsim.
VR_GET_VISSIM_STATE	Gets information related to the global state of Commsim. The information provided is a copy of the current internal state; modifying it will not change Commsim's state. Arg2 should contain a pointer to a SIM_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() function. Arg3 should contain the size of this structure (sizeof(SIM_INFO)) to allow for version compatibility checking.
VR_RESET_XFERS	For internal use only.
VR_RUN_SIMULATION	Starts a simulation run.
VR_SET_GLOBAL_UNKNOWNS	Sets current parameterUnknown block output values from arg2.

10.4.3 Building a custom global optimizer

Most languages have a Project Build facility that automates the process of building a .DLL file. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you're using for specific instructions.

➢ To build a custom global optimizer:

1. Invoke the Compiler environment.
2. Add all the source files listed in “10.4.1 Source files for building a custom global optimizer” on page 10-11 to the project or make file.
3. Under project options, specify the project type as a Windows DLL.
4. Under compiler options, specify the following:
    - Memory Model to be Large.
    - Windows Prolog/Epilog to be Real Mode_far Functions.
5. Build the project.

Chapter 11
Extending the Block Set

The following are described in this chapter.

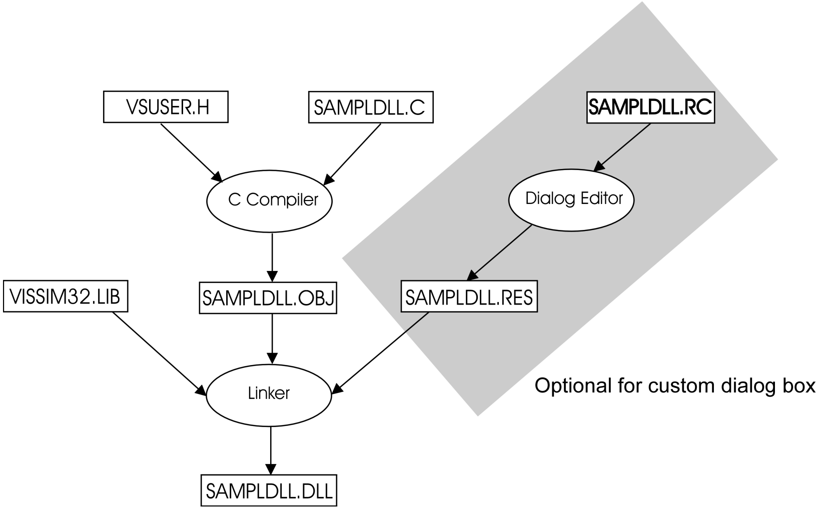
11.1 Before you begin…

Before you begin writing DLLs, make sure that Commsim's …\INCLUDE directory is part of the list of include directories that your compiler searches. The file specification for Commsim's \INCLUDE directory is C:\COMMSIM7\VSDK\INCLUDE.

11.2 The big picture

Commsim provides an Application Programming Interface (API) that allows you to extend the standard block set by creating Dynamic Link Library (DLL) files and binding them to `userFunction` blocks. A DLL file is like a regular executable file with the exception that it cannot start execution on its own. A DLL function can be called just like functions that are part of a normal executable file.

The following diagram shows how files are processed to create Commsim DLLs. This diagram steps you through the process of creating a DLL from a C source file; however, you can also create DLLs in Fortran, Pascal, and C++.



The main steps in the creation of Commsim DLLs are:

1. Create or edit an existing C, Fortran, or Pascal source file.
2. Create a project DLL for your compiler.
3. Execute a build operation, which compiles your source code into an object file.
4. Link the object file with VISSIM32.LIB to produce a DLL.

Commsim also provides a DLL Wizard that automatically performs steps 1, 2, and 4.

11.2.1 Criteria for writing DLLs

You can write DLLs in any language, provided the language has the following capabilities:

- 64-bit floating point array parameters
- Pointers to 16-bit integers
- `_stdcall` calling conventions (default for Microsoft Fortran and Delphi Pascal)

Example DLLs written in C, Fortran, and Pascal are distributed with Commsim and reside in subdirectories under the \COMMSIM7\VSDK directory.

11.2.2 Building a DLL

There are two ways to build a DLL:

- Using the Commsim DLL Wizard.
- Using a Project Build facility.

Using a Project Build facility: Most languages have a Project Build facility that facilitates the process of building a DLL. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you're using for specific instructions.

1. Invoke the Compiler environment.
2. Add the following files to the project file:
   - All the source files
   - All the resource files
   - VISSIM32.LIB
3. Under project options, specify the project type as a Windows Dynamic Link Library (.DLL).
4. Under compiler preprocessor options, specify \COMMSIM7\VSDK as the include directory.
5. Build the project.

11.3 How Commsim talks to a DLL

There are three types of functions used for communication between Commsim and your DLL:

- Simulation level functions
- Block level functions
- Commsim exported functions

Simulation level functions and block level functions are DLL functions Commsim can call via a `userFunction` block. Simulation level functions are called once per simulation-wide event. Block level functions, on the other hand, are called once for each `userFunction` block. If you have no `userFunction` blocks, no calls are made; if you have 100 `userFunction` blocks, they are called 100 times.

Exported functions are Commsim functions that you can call from a DLL. These functions allow a DLL to request information from Commsim, as well as instruct Commsim to perform specific actions.

11.3.1 Calling conventions

The following information pertains to the DLL functions described in the next several sections:

- Functions are shown in C syntax. For Fortran and Pascal syntax, look in the sample source files in subdirectories under \COMMSIM7\VSDK.
- All arguments are pointers to data types. Since Fortran passes variables by reference, a normally declared Fortran variable can be passed as an argument.

11.3.2 Simulation level functions

There are two simulation level functions:

- **vsmInit().** Called at Commsim start-up. You use this function to insert one or more blocks in the Blocks menu.
- **vsmEvent().** Called at simulation-wide events, such as simulation start, end of time step, and simulation end.

11.3.2.1 Initialization function - vsmInit()

This DLL function is called by Commsim to allow the DLL to perform initialization, particularly to insert `userFunction` blocks into the Blocks menu.

EXPORT32 int EXPORT PASCAL vsmInit()

To have vsmInit() be called when Commsim starts up, you must tell Commsim the path to your DLL, as described on page 11-25.

11.3.2.2 System-wide event function - vsmEvent()

This function is called by Commsim to notify the DLL of interesting simulation-wide events.

```
LPSTR PASCAL vsmEvent(int msg, int wParam, long *arg);
```

Message	Function
VSE_GET_CODEGEN_DECL_STR	Sent once per DLL at the start of code generation. Returns a char * string of text to be inserted into the generated code at file scope before block code generation messages are processed.
VSE_GET_PARAM_DESC	Returns descriptor for DLL configuration dialog that has no block associated with it. Syntax is identical to WM_VSM_GET_PARAM_DESC.
VSE_POST_SIM_END	Commsim has stopped, either from user stop or time expiration.
VSE_POST_SIM_START	Commsim has initialized all blocks and is about to start.
VSE_PRE_SIM_START	Commsim is about to start a simulation but has not initialized any blocks.
VSE_SIM_RESTART	Commsim is restarting due to auto-restart.
VSE_TIME_STEP	Simulation has completed a time step.
WM_COMMNOTIFY	Comm port data is ready.
WM_DESTROY	Commsim is exiting.
WM_VSM_WINDOW_HANDLE	Handle to Commsim main window. **arg:** Window handle

11.3.3 Block level functions

In order to interface smoothly with Commsim, Commsim can call seven Pascal-style functions that share the base DLL function name and have an event code suffix corresponding to a Commsim event. You should supply a function for each event that you want your DLL code to handle. The additional functions are described below:

Function name	Purpose	When it’s called
userBlock()	Block time step	Each simulation time step
*userBlock*Event()	Block event handler	On occurrence of a block related event
userBlock PA()	Block parameter allocation	Block creation
userBlock PC()	Block parameter change	Right button click
userBlock PI()	Block parameter initialization	Immediately after *userBlock*PA()
userBlock SE()	Block simulation end	Simulation end time
userBlock SS()	Block simulation start	Simulation start time

The term *userBlock* is a placeholder for your DLL base function name. You specify the DLL base function name when you bind the DLL to a `userFunction` block.

You can have any number of user-written blocks in a single DLL file.

All definitions are kept in the VSUSER.H file. This file should be included in every user DLL.

11.3.3.1 Time step function - *userBlock*()

The *userBlock*() function is called at each time step to calculate simulation values. It is the only function a DLL is required to export.

The *userBlock*() function may be called an arbitrary number of times during a time step interval. The number of calls depends on the integration method and whether the VBF_HAS_STATE flag is active for a block. Note that the outputs are not preserved from call to call. Thus, this function must write to its outputs at each call.

void PASCAL *userBlock*(double param[], double inSig[], double outSig[])

The inSig array is filled by Commsim with the values presented to the input connector tabs on the `userFunction` block. Store the result values in the outSig array. Commsim presents the outSig values to the corresponding output connector tabs on the `userFunction` block.

11.3.3.2 Event handler function - *userBlock*Event()

The *userBlock*Event() function is called at block events, such as block repaint, end of time step, and simulation end.

```
LPSTR PASCAL userBlockEvent(HWND h, int msg, WPARAM wp,LPARAM lp)
```

This function lets you save and restore mixed data types. Note that the arguments are the same as Windows or Windows NT event functions.

Commsim calls your function with the following messages:

Message	**Description**
WM_VSM_ADD_CONNECTOR	Signals a user-request for a connector to be added to a block. **wp:** Current count **lp:** 1 if input 0 if output
WM_VSM_ALLOC_VEC_OUTPUT:	This event is generated after inputs are allocated. This event allows user DLL to allocate vector outputs. **wp:** 0 **lp:** Block handle
WM_VSM_BLOCK_INFO	Used for writing a custom optimizer.
WM_VSM_BLOCK_LABEL	Provides a block label. Return a null terminated string for label under block. **wp:** 0 **lp:** Block handle
WM_VSM_BLOCK_PLACED	Signals that a block has been placed in the diagram. lp: Block handle
WM_VSM_BLOCK_SETUP	This event is generated only if there is no *userBlock*PC() function defined for the block. This event is generated when you click the right mouse button on the block. **wp:** Internal ID **lp:** Block handle
WM_VSM_CAN_ACCEPT_ ALTERNATE_TYPE:	Checks if block inputs can accept alternate data type. Returns 0 if NO, 1 if YES. **wp:** Data type **lp:** Block handle
WM_VSM_CHECKPOINT_STATE	Signals a user-request to checkpoint system states; that is, set the checkpoint buffer to the current state value.

Message	Description
WM_VSM_CONNECTOR_NAME	Returns string of connector label name. Null if no label is desired. **wp:** Port number (negative if output) **lp:** Block handle
WM_VSM_CREATE	Is called when a user block is created. You can return flags to customize block treatment. **wp:** 0 if create from file 1 if create from menu 2 if create from clipboard **lp:** Block handle
WM_VSM_DEL_CONNECTOR	Signals a user-request for a connector to be deleted from a block. **wp:** Current count **lp:** 1 if input 0 if output
WM_VSM_DESTROY	Is called when user block is destroyed. **lp:** Block handle
WM_VSM_FILE_CLOSE	Reserved.
WM_VSM_FILE_READ	Signals that block and all its parameters have been read in from file. **lp:** Block handle
WM_VSM_GET_BANNER	Reserved.
WM_VSM_GET_BLOCK_BITMAP	Reserved.
WM_VSM_GET_BLOCK_NAME	Provides a block with a custom name. Return a null terminated string that contains the new name. The name may contain newline characters.

Message	Description
WM_VSM_GET_CODEGEN_ALLOC_STRING:	Returns C code for file scope declarations used in WM_VSM_GET_CODEGEN_STRING and WM_VSM_GET_CODEGEN_START_STRING. **wp:** 0 **lp:** Block handle
WM_VSM_GET_CODEGEN_END_ STRING	Returns C code string to execute after simulation terminates. **wp:** 0 **lp:** Block handle
WM_VSM_GET_CODEGEN_ INCLUDE_STRING	Sent during code generation. If used, returns text of include file line; otherwise, null.
WM_VSM_GET_CODEGEN_PORT_STORAGE	Returns C code to supply result value on output pin wp. **wp:** Output pin **lp:** Block handle
WM_VSM_GET_CODEGEN_POST_ END_STRING	Sent after all WM_VSM_GET_CODEGEN_END_STRING
WM_VSM_GET_CODEGEN_POST_ START_STRING	Sent after all WM_VSM_GET_CODEGEN_START_STRING.
WM_VSM_GET_CODEGEN_START_STRING:	Returns C code for block initialization. **wp:** 0 **lp:** Block handle
WM_VSM_GET_CODEGEN_ STRING	Returns C code string for main time step function. %1 expands to code for input pin 1; %2 expands to code for input pin 2, and so on. **wp:** 0 **lp:** Block handle
WM_VSM_GET_CODEGEN_TARGET:	Reserved.
WM_VSM_GET_CODEGEN_TIME_ STEP_STRING	Reserved.

<table>
<tr><th>Message</th><th>Description</th></tr>
<tr><td>WM_VSM_GET_CONNECTOR_TYPE:</td><td>Puts a data type on a connector. If you do not handle the message, the connector type is T_DOUBLE. The connector types that can be returned are: T_DOUBLE, T_FLOAT, T_INT, T_LONG, T_ULONG,   ,T_CHAR, T_UCHAR, T_SHORT, T_USHORT, T_STRING, T_MAT_DOUBLE, T_MAT_COMPLEX, T_POINTER, T_COMBO_ITEM, T_SCALED_INT, T_COMPLEX, and T_LAST.
<b>wp:</b> Connector index. If negative, it is an input; if positive, it is an output.
<b>lp:</b> Block handle</td></tr>
<tr><td>WM_VSM_GET_DEFAULT_IN_VALUE</td><td>Reserved.</td></tr>
<tr><td>WM_VSM_GET_IGNORE_MENU_ GRAYING</td><td>Returns TRUE if block should not be grayed in Commsim Viewer.
<b>wp:</b> 0
<b>lp:</b> Block handle</td></tr>
<tr><td>WM_VSM_GET_OBJ_CLASS</td><td>Reserved.</td></tr>
<tr><td>WM_VSM_GET_OUT_SIGN_VEC</td><td>Reserved.</td></tr>
<tr><td>WM_VSM_GET_PARAM</td><td>Reserved.</td></tr>
<tr><td>WM_VSM_GET_PARAM_DESC</td><td>Provides a data descriptor for saving and restoring data. Return a text string that describes your data by following these guidelines:
<table>
<tr><th><u>Format character</u></th><th><u>Data Type</u></th></tr>
<tr><td>i</td><td>2-byte integer</td></tr>
<tr><td>I</td><td>4-byte integer</td></tr>
<tr><td>f</td><td>4-byte float</td></tr>
<tr><td>F</td><td>8-byte float</td></tr>
<tr><td>c</td><td>Single-byte character</td></tr>
</table>
All of the above format characters can take an optional count suffix, which is enclosed in square brackets. For example to save two 8-byte floats and a 32-character string in C, use the following string notation:
“F [2] c[32]”
In Fortran, the string notation is:
‘F [2] c[32]’ c</td></tr>
</table>

Message	Description
WM_VSM_GET_PARAM_STR:	Returns pointer to parameter strings with a semi-colon separator. **wp:** 0 **lp:** Block handle
WM_VSM_INFO	Reserved.
WM_VSM_IS_VEC_CONNECT:	This event is generated when user attempts to connect a block. User returns a value. If the value is 1, the vector or matrix input or output is allowed. If the value is 0, scalar input or output is allowed. **wp:** If positive, input; if negative, output **lp:** Block handle
WM_VSM_PRE_SIM_START:	This event is generated after simulation-wide VSE_PRE_SIM_START and before any other block receives WM_VSM_SIM_START. **wp:** Run count **lp:** Block handle
WM_VSM_PUT_PARAM	Reserved.
WM_VSM_RETAIN_STATE_ RESTART	Restarts with retained states. **lp:** Block handle
WM_VSM_SIM_END	Reserved.
WM_VSM_SIM_RESET:	This message is sent on simulation reset when the user presses the RESET button. **wp:** 0 **lp:** Block handle
WM_VSM_SIM_RESTART	Signals restart due to continue or single step. **lp:** Block handle
WM_VSM_SIM_START	This event is sent on simulation start-up or restart. **wp:** Run count **lp:** Block handle
WM_VSM_SNAP_STATE	Signals a user-request to snap system states; that is, set the initial condition to the current state value.

Message	Description
WM_VSM_STOP_SIM	Signals that Commsim has stopped, either by user or time expiration. **wp:** 1 if single step **lp:** Pointer to parameter vector
WM_VSM_SUPPRESS_WARN_ UNCONNECT	Checks to suppress the unconnected input warning message. Return 1 if suppression is desired. **wp:** Port number (negative if output) **lp:** Block handle
WM_VSM_WINDOW_HANDLE	This event is generated on Commsim start-up. **wp:** 0 **lp:** Main window handle

Return flags for WM_VSM_CREATE

Flag	Description
VBF_ALLOC_VEC_OUTPUT	Causes Commsim to automatically allocate output matrix for matrix input blocks.
VBF_ALLOW_VEC_CHAMELIONS	Causes connectors to accept scalar or vector connections.
VBF_CG_BLK_HAS_NO_RETURN_ VALUE	If your DLL has one output, set this if you do not want to use a function return value.
VBF_CG_BLK_HAS_RETURN_VALUE	Reserved.
VBF_CG_EVENT_FUNC	Allows code generation for your DLL event function.
VBF_EXECUTE_ALWAYS	Causes Commsim to execute the block regardless of graph connectivity.
VBF_HAS_STATE	Tells Commsim that block has state and can break an algebraic loop. Commsim calls this block once, before all other blocks, to present an initial condition; then the block is called during normal diagram execution.

Flag	Description
VBF_MENU_ONLY	This is a menu item only and no block is created; however, the *userBlock*Event() function is called. This flag is useful for menu select to dialog.
VBF_STRAIGHT_WIRES	Causes wires to be drawn as straight lines rather than auto-routing.
VBF_USE_SIGNAL_DESCRIPTORS	Block input and output vector are vector of type SIGNAL (not double).

11.3.3.3 Parameter allocation function - *userBlock*PA()

The *userBlock*PA() function is called when you first enter a DLL file/function pair in the dialog box for the `userFunction` block, or when a diagram is first read into Commsim.

Note: This call is no longer required if block menu insertion is used.

```
long PASCAL userBlockPA(short *ppCount)
```

This function returns the parameter storage requirements, in bytes, for the `userFunction` block, and the number of prompted parameters. If you want Commsim to prompt for parameter values, set ppCount to the desired number. The maximum value for ppCount is 12. You need to allocate eight bytes for each prompted parameter. You can request additional storage for a function's private use. This additional storage can be accessed as array elements of the parameter vector after the first ppCount elements.

When this function is not supplied, no parameter storage is allocated.

11.3.3.4 Parameter change function - *userBlock*PC()

The *userBlock*PC() function is called when you click the right mouse over the `userFunction` block.

```
char * PASCAL userBlockPC(double * param)
```

This function lets you change block parameters for the `userFunction` block. If you want to create a dialog box to browse and set parameter values, you can do so and return a NULL pointer. If you want Commsim to browse and set parameter values for you, you should return a pointer to a NULL terminated string. The string should contain semicolons to separate each parameter prompt. You may have up to 12 parameters using this default method of parameter setting.

11.3.3.5 Parameter initialization function - *userBlock*PI()

The *userBlock*PI() function is called at block creation time, either from the menu or a file, for parameter initialization. It lets you provide initial values for parameters.

```
void PASCAL userBlockPI(double * param)
```

This function is called immediately after the parameter allocation function *userBlock*PA().

11.3.3.6 Simulation end function - *userBlock*SE()()

The *userBlock*SE() function is called just after a simulation ends to perform post simulation processing.

```
void PASCAL userBlockSE(double param[], long * runCount)
```

11.3.3.7 Simulation start function - *userBlock*SS()

The *userBlock*SS() function is called just prior to the start of a simulation to perform initialization processing necessary for a simulation run.

```
void PASCAL userBlockSS(double param[], long * runCount)
```

11.3.4 Exported functions

The seven exported functions are described below.

11.3.4.1 General information request - vissimRequest()

The vissimRequest() function provides a general, extensible request mechanism for obtaining information from Commsim.

```
EXPORT32 long vissimRequest(long req, arg2, arg3);
```

The first argument (long req) is a message code describing the action for Commsim to take. The message codes that pertain to writing a custom block are as follows:

Message	**Description**
VR_CHECK_PORT_NAME_SYNTAX	Reserved
VR_CLEAR_BLOCK	Clears red error state on block. Uses currently executing block if block handle in arg1 is null. **arg1:** Block handle (opt.)
VR_CLEAR_BLOCK_ERR	Clears red error state on block. Uses currently executing block if block handle in arg1 is null. **arg1:** Block handle (opt.)
VR_CODE_GEN	Activates code generation on the current diagram. **arg2:** Pointer to C file name
VR_DISABLE_BLOCK_TYPE	Removes block matching name string from Blocks menu. **arg1:** LPSTR name string
VR_EXECUTE	Runs diagram but doesn’t change time (no integration).
VR_GET_BLOCK_ID	Returns an integer that uniquely identifies the block.
VR_GET_BLOCK_INPUT	Returns the pointer to the input SIGNAL vector. **arg1:** Block handle
VR_GET_BLOCK_OUTPUT	Returns the pointer to the output SIGNAL vector. **arg1:** Block handle
VR_GET_BLOCK_PARAMS	Returns block parameter pointer. **arg1:** Block handle
VR_GET_BLOCK_POS	Returns block position.
VR_GET_CODEGEN_TARGET	Returns pointer (LPCSTR) to the name of the current code generation target.
VR_GET_CONNECT_PORT	Indicates if port is input or output. **arg1:** Block handle **arg2:** Port number. If positive number, the port is an input; if negative number, the port is an output.

Message	Description
VR_GET_CONNECTOR_COUNT	Gets either the input or the output connector count for the block. **arg1:** Block handle **arg2:** If arg2 is 1, it returns the number of inputs; if arg2 is 0, it returns the number of outputs.
VR_GET_CONSTRAINTS	Gets constraint values **arg1:** Double * constraint
VR_GET_CUR_BLOCK_HANDLE	Returns the handle to the current block.
VR_GET_GLOBAL_CONSTRAINT_ BOUNDS	Gets global constraint bounds. **arg1:** Double * upper bound **arg2:** Double * lower bound
VR_GET_GLOBAL_CONSTRAINTS	Gets global constraints. **arg1:** Double *
VR_GET_GLOBAL COST	Gets global cost. **arg1:** Double * global cost
VR_GET_GLOBAL_OPT_INFO	Gets global optimization settings. **arg1:** OPT_INFO
VR_GET_GLOBAL_UNKNOWN_ BOUNDS	Gets global unknown bounds. **arg1:** Double * upper bound **arg2:** Double * lower bound
VR_GET_GLOBAL_UNKNOWNS	Gets global unknowns. **arg1:** Double *
VR_GET_GLOBAL_UNKNOWNS_ INPUT	Gets global unknown input. **arg1:** Double * global unknown initial condition
VR_GET_INTEGRATION_SUBSTEP	Returns integration substep for multistep methods. Note that 0 means start of new step.
VR_GET_OEM_NAME	Pointer (LPCSTR) to OEM name.
VR_GET_OPEN_FILE_PATH	Returns pointer (LPCSTR) to the file name or file path of the active diagram. **arg1:** If 0, then file name; if 1, then file path.

Message	Description
VR_GET_PARAM_STR	Returns pointer to parameter string for block handle. **arg1:** Block handle
VR_GET RANDOM_SEED	Supplies the random seed.
VR_GET_RUN_STATE	Indicates if the simulation is running (TRUE) or not running (FALSE).
VR_GET_SOLVER_INFO	Gets settings from Implicit Solver tab in the Simulation Properties dialog box. **arg1:** Pointer to Implicit Solver
VR_GET_STARTUP_DIR	Returns Commsim directory string.
VR_GET_STARTUP_SCRIPT	Returns pointer (LPCSTR) to current script file path.
VR_GET_SUB_VERSION	Returns version letter suffix.
VR_GET_UNKNOWNS	Gets unknown values. **arg1:** Double * unknowns
VR_GET_UNKNOWNS_INPUT	Get unknown inputs. **arg1:** Double *
VR_GET_VERSION	Returns major version in high byte and minor version in low byte. **arg1:** SIM_INFO pointer
VR_GET_VISSIM_STATE	Gets current simulation state and copy to pointer in arg1.
VR_GET_WINDOW_HANDLE	Returns Commsim main window handle. Useful for model dialog creation.
VR_GRAY_MENU_ITEM	Grays out menu item. **arg1:** Pointer to the menu item name (LPSTR).
VR_ENABLE_MENU_ITEM	Enables menu item. **arg1:** char* name of menu item to be enabled
VR_EXECUTE_CONST_BLOCK	Reserved
VR_EXPAND_PATH_ALIAS	Returns pointer (LPCSTR) to expanded path alias. **arg1:** Pointer (LPCSTR) to input alias

Message	Description
VR_MODIFY_BLOCK	Modifies block settings (such as parameters, shape, rotation). If return value is 0, modification fails; if return value is not 0, modification succeeds. **arg1:** Block handle **arg2:** Pointer (LPCSTR) to string in .VSM block description format
VR_REALLOC_USER_PARAM	Reallocates parameter vector and returns newly reallocated pointer. **arg1:** Block handle **arg2:** New parameter size
VR_REQUEST_PERMISSION	Checks if Commsim add-on is licensed.
VR_RESET_XFERS	For internal use only.
VR_RESYNC_REALTIME_CLOCK	Resynchronizes Commsim's real-time clock to the current time.
VR_ROTATE_BLOCK	Rotates block 180^{o}. **arg1:** Block handle **arg2:** 0
VR_RUN_SIMULATE	Executes the current model.
VR_SET_BLOCK_CONNECTOR_ COUNT	Sets the connector count on the block. **arg1:** Block handle **arg2:** input # = upper word output # = lower word
VR_SET_BLOCK_MENU	Adds user-defined block to Blocks menu. **arg1:** Pointer to initialized USER_MENU_ITEM vector
VR_SET_BLOCK_POS	Sets position of the block. arg1: Block handle arg2 (lower 16 bits): xPos arg2 (upper 16 bits): yPos
VR_SET_CONNECTOR_CHAR	Sets indicator character on block connector. **arg1:** Character to set
VR_SET_CONNECTOR_LABEL	For internal use only.
VR_SET_GLOBAL_UNKNOWNS	Sets global unknowns from supplied vector. **arg1:** Double * global unknown

Message	Description
VR_SET_PARAM_STR	Sets new parameter string for block handle. Note that the block will always copy arg2, so arg2 can be freed after the call. **arg1:** Block handle **arg2:** Pointer to parameter string (LPSTR)
VR_SET_RANDOM_SEED	Sets the random seed as an (unsigned int*). **arg1:** (unsigned int*)
VR_SET_STARTUP_SCRIPT	Sets script file. **arg1:** Pointer (LPCSTR) to script file path
VR_SET_UNKNOWNS	Sets unknown values from user-supplied vector. **arg1:** Double * unknown
VR_SNAP_STATES	Causes Commsim to use current integrator/ delay state as initial condition.

11.3.4.2 Get current simulation time - getSimTime()

This function stores the current simulation time in the double precision float variable pointed to by simTime.

```
EXPORT32 void PASCAL getSimTime( DOUBLE *simTime);
```

11.3.4.3 Get current simulation time step - getSimTimeStep()

This function stores the current simulation time step in the double precision float variable pointed to by simTimeStep.

```
EXPORT32 void PASCAL EXPORT getSimTimeStep( DOUBLE *simTimeStep);
```

11.3.4.4 Print debug message - debMsg ()

This function prints a dialog box containing a debugging message. Because you can't perform normal screen I/O under Windows or Windows NT, Commsim provides debMsg to display information pertaining to the variables for your userFunction block's function. The format is identical to the C printf() function. Since this function allows an arbitrary number of arguments, it must be called using the C language convention. To call it from Fortran or Turbo Pascal, for example, you must declare it as C language code. Commsim displays the output string in a standard dialog box that contains a Retry, Ignore, and Abort button. Press Retry or Ignore to continue the simulation. Press Abort to cancel the simulation.

```
EXPORT32 int CDECL EXPORT debMsg P((char LPSTR fmt , ... ));
```

11.3.4.5 Request simulation end - stopSimulation()

This function requests that Commsim stop a simulation.

```
EXPORT32 void PASCAL EXPORT stopSimulation( int stopVal);
```

If stopVal is 1, the current simulation run is stopped. If you have activated the Auto Restart parameter under the Range tab in dialog box for the Simulate > Simulation Properties command, Commsim starts the next simulation run. If stopVal is 2, all simulation runs are stopped.

11.3.4.6 Flag error - setBlockErr()

This function requests that Commsim flag the currently executing block in red. All nested blocks will be flagged in red as well. To clear a flagged block, click the right mouse button on the block.

```
EXPORT32 void PASCAL EXPORT setBlockErr();
```

11.3.4.7 Add menu item - setUserBlockMenu()

This function adds a block (or menu item) to the Commsim menus.

```
EXPORT32 void EXPORT setUserBlockMenu P((USER_MENU_ITEM * ));
```

This functions recognizes one argument, which is a pointer to an array of structures. The structures define the menu name, DLL name, number of inputs, number of outputs, number of parameters, and help string. The format of the structure is as follows:

```
typedef struct {
  char * menuName;
  char * funcName;
  int inputCount;
  int outputCount;
  int paramCount;
  char * helpText;
} USER_MENU_ITEM;
```

You need one structure for every block (or menu item).

In addition, the last element in the array of structures that is passed back in must be {0}.

11.4 Debugging hints

The following guidelines will make it easier to debug your DLLs:

- MSVC lets you set a breakpoint in your DLL before running Commsim.
- Set VISSIM32.EXE as the calling program by choosing Build > Settings > Debug. This starts Commsim automatically when you press F5 (or choose Debug > Go).
- Use conditional breakpoints to get control near the problem area.
- Single-step with values in watch window to find problems.
- If your program hangs, press CTRL+ALT+PRTSCRN (or choose Debug > Break) to return control to the debugger. This works best under Windows 95 and Windows NT.
- On Debug > Break or a General Protection Fault, use View > Stack Trace to find the location of the offending instruction.

Floating point errors, if continued, often result in General Protection Fault that point to source line of floating point error.

11.5 Using the DLL wizard

If you want a quick way to generate the skeletal code for a custom block, you can use the DLL Wizard. The DLL Wizard takes you step-by-step through the process of creating a custom block in Microsoft C++ version 5+.

The DLL Wizard performs the following functions:

- Creates all the source files, including .CPP, .H, and .RC file
- Adds the source, resource, and VISSIM32.LIB files to the project file
- Specifies the number of inputs, outputs, connector label names for the custom block
- Adds sample code to generate a customizable MFC dialog box

➢ To run the DLL Wizard:

1. Start up the Microsoft Developer's Studio (version 5+).
2. Choose File > New.
3. Click on the Projects tab and do the following:
   - Select Commsim DLL Wizard.
   - In the Project Name box, enter a project name.
   - In the Location box, specify the directory in which the project is to be stored.
   - Click on the OK button, or press ENTER.

   The Commsim DLL Wizard starts up and a dialog box called **Commsim DLL Wizard - Step 1 of 7** appears.

4. Click on the Next button, or press ENTER. A dialog box called **Commsim DLL Wizard - Step 2 of 7** appears.
5. Do the following:
   - In the Base Function Name box, enter the base name of the function. The name you enter here appears in the title bar of the dialog box that corresponds to the custom block you are creating.
   - In the Block Name box, enter the name of the block. This name appears on the block. For long block names on blocks, you can display the name on two lines by separating the first line of text from the second line of text using the notation \n.

     For consistency, the block name you specify here should also be used in step 7.
   - Click on the Next button, or press ENTER.

   A dialog box called **Commsim DLL Wizard - Step 3 of 7** appears.
6. Do the following:
   - In the Block Input Count box, enter the number of input connector tabs for the block.
   - In the Block Output Count box, enter the number of output connector tabs for the block.
   - Click on the Next button, or press ENTER.

   A dialog box called **Commsim DLL Wizard - Step 4 of 7** appears.
7. If you want custom data types for your input and output pins, activate the Want Custom Data Types option and do the following:
   - For input pins, in the Input box, click on the down arrow and select the pin number.
   - In the Type box, click on the down arrow and select the data type. If you choose matrix or scaled integer, two additional boxes appear in which to enter dimension or radix and word size.
   - For output pins, in the Output box, click on the down arrow and select the pin number.
   - In the Type box, click on the down arrow and select the data type. If you choose matrix or scaled integer, two additional boxes appear in which to enter dimension or radix and word size.
   - After you finish entering the data type for each pin, click on the Next button, or press ENTER.

   A dialog box called **Commsim DLL Wizard - Step 5 of 7** appears.
8. To generate C code for your custom block, activate the I want my block to generate C code option and do the following:
   - In the list box, click on the down arrow to see code generation events to which your block can respond. The window below the list box shows the particular string to be included in your code. To edit the string, simply position the pointer in the window and click the mouse. If you do not want a particular string to be included in your code, just

delete it from the window.
- After you have edited the data strings, click on the Next button, or press ENTER.

A dialog box called **Commsim DLL Wizard - Step 6 of 7** appears. This dialog box lets you choose where your block will appear in Commsim's menus. Note that the block is not accessible from Commsim until after you add the corresponding DLL to the Commsim initialization file, as described in "11.6.1 Adding a custom block to a Commsim menu" on page 11-24.

9. Do the following:
   - In the Top Level Menu Name box, specify the name of the menu in Commsim's menu bar under which the category containing your block will appear. If you want to use the standard Blocks menu, enter &Blocks.
   - In the Block Category Name box, specify the name of the category under which your block will appear. You can use an existing category name (for example, Linear Systems or Arithmetics) or you can enter a new name.
   - In the Block Name box, specify the name of your block. For consistency, you should use the block name specified in step 5.
   - To generate a custom MFC dialog box for your block, activate the Include Custom Dialog Box option. For information on editing the resource file to customize the dialog box, see the appropriate Microsoft documentation.
   - Click on the Next button, or press ENTER.

   A dialog box called **Commsim DLL Wizard - Step 7 of 7** appears.
10. Click on the Finish button, or press ENTER.

    The **New Project Information** dialog box appears with information about the newly-created Commsim-compatible DLL.
11. Click on the OK button, or press ENTER.
12. To build a DLL, do the following:
13. In the Microsoft Developer Studio window, choose Build > Build *projectname*.DLL, where *projectname* is the name specified in step 3.

11.5.1 Editing the VSI.CPP file

The DLL Wizard creates a VSI.CPP file which contains the source C++ code for the DLL, as well as placeholders for simulation level functions and block level functions that you may use in your DLL. To edit VSI.CPP, start up the Microsoft Developer Studio and open VSI.CPP. For syntax rules for simulation and block level functions, see "11.3 How Commsim talks to a DLL" on page 11-3.

11.5.2 Editing Connector Tab Labels

The DLL Wizard lets you add, remove, and rearrange connector tab labels.

➢ To assign labels to connector tabs:

1. In the Labels box, enter a label name.
2. Click on the + button.

➢ To remove labels from connector tabs:

1. Highlight the label to be removed.
2. Click on the - button.

➢ To change the order of the labels:

1. Highlight the label to be moved.
2. Click on the up or dn button.

11.6 Accessing a custom block from Commsim

There are two ways to access a block from Commsim:

- Add the block to a Commsim menu
- Bind the DLL to a `userFunction` block

By adding a block to a Commsim menu, you make it easier to access and use the block. When you bind the DLL to a userFunction block.

11.6.1 Adding a custom block to a Commsim menu

Whether you created your custom block using the DLL Wizard or by writing it entirely by hand in C++, Pascal, or Fortran, you can easily add the block to a Commsim menu by following the procedure below.

If you wrote your DLL by hand (that is, you didn’t use the DLL Wizard), make sure the DLL function:

- Exports the vsmInit() function
- Calls setUserBlockMenu() from within vsmInit()

➢ To add a custom block to a Commsim menu:

1. In Commsim, choose Edit > Preferences.
2. Click on the Addons tab.

3. Double-click on the ellipsis (…) and type in the path to the DLL function, or click on the … button to locate the DLL function. If you created the DLL function using the DLL Wizard, the DLL function will reside in the \DEBUG directory directly below the project directory.
4. Click on the OK button, or press ENTER.

11.6.2 Binding a DLL to a userFunction block

If you do not want your custom block to appear in a menu, you can bind the corresponding DLL to a `userFunction` block. Commsim calls the DLL each time the block is executed.

➢ To bind a DLL to a userFunction:

1. Insert a `userFunction` block in your diagram.
2. Choose Edit > Block Properties.
3. Point to the `userFunction` block and click the mouse.
4. Do the following:
   - In the DLL File Name box, enter the name of the DLL file containing the user function.
   - In the Base Function Name box, enter the base name of the function.
5. Click on the OK button, or press ENTER.

Chapter 12
Commsim Library

This chapter describes the compound blocks and data files provided in the COMMLIB folder.

12.1 Compound Blocks

12.1.1 COSTAS_C.VSM

This compound block implements a complex Costas loop, which is used to track the phase of PSK modulated signals. This compound block accepts a modulated complex signal and outputs I and Q channel baseband signals and a complex VCO output. This block is based on a second order PLL design. After additional filtering, the multiplier’s I and Q outputs are both used to compute the phase error term. The phase error detector provided with this compound block is tailored to BPSK. For other modulation schemes, a more suited detector should be substituted.

It is important to ensure that the phase detector gain and VCO gain are properly set in the `Loop Filter` block’s parameters. The amplitude of the input signal is assumed to be 1. If this is not the case, its value must be taken into account in determining the phase detector gain. A unity phase detector gain indicates that an error signal of 0.1 V at the `Loop Filter` input represents a 0.1 rad phase error. In order for the PLL to operate properly, the simulation sampling frequency should be much larger than the PLL natural frequency (`Loop Filter` block parameter). Also, the IIR filter cutoff frequencies should be appropriately matched to the data rate.

x = Input signal (complex)

y_1 = Output signal (complex)

y_2 = VCO output signal (complex)

12.1.2 COSTAS_R.VSM

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

x = Input signal (real)

y_1 = Output signal (complex)

y_2 = VCO output signal (complex)

12.1.3 GFSK.VSM

This compound block implements a GFSK modulator. It employs a Gaussian FIR Filter block and an FM Modulator block as its internal components. The internal parameters of each of these blocks need to be specified for proper operation. The default settings reflect a Bluetooth implementation.

x = Input data signal (binary)

y = Complex output signal [Re, Im]

12.1.4 GMSK.VSM

This compound block implements a GMSK modulator. It employs a `Gaussian FIR` block and an `FM` modulator block as its internal components. The internal parameters of each of these blocks need to be specified for proper operation.

x = Input data signal (binary)

y = Output signal (complex)

12.1.5 PLL1CPLX.VSM

This compound block implements a complex first order PLL. The VCO output attempts to follow the input signal. The internal phase error signal is obtained by multiplying the input signal by the VCO output. This error signal is then passed through a `gain` block and used as the VCO drive signal. The loop gain is computed by multiplying the phase detector gain by the VCO gain. A phase detector gain value of 1 indicates that an error signal of 0.1 V represents a 0.1 rad phase error. A first order loop cannot remove a frequency offset and also achieve 0 phase error. If a frequency offset is present, use a second order PLL.

x = Input signal (complex)

y_1 = VCO output signal (complex)

y_2 = VCO phase (radians)

12.1.6 PLL1REAL.VSM

This compound block is identical to the PLL1CPLX.VSM compound block, except that it accepts a real signal input. The gain factor of -2 prior to the `multiply` block is used to compensate for the factor of two loss through the phase detector due to the formation of a double frequency term in this implementation.

x = Input signal (real)

y_1 = VCO output signal (real)

y_2 = VCO phase (radians)

12.1.7 PLL2CPLX.VSM

This compound block implements a second order PLL. The VCO output attempts to follow the input signal. The internal phase error signal is obtained by complex multiplying the input signal by the VCO output. This error signal is then passed through the `Loop Filter` block and its output is used as the VCO drive signal. A second order PLL can remove both a phase offset and a frequency offset (assuming a high gain loop). If Doppler rate is present, then a third order PLL is recommended. For descriptions of the `VCO` and `Loop Filter` blocks, look under their respective names earlier in this section.

It is important to ensure that the phase detector gain and VCO gain are properly set in the `Loop Filter` block's parameters. The amplitude of the input signal is assumed to be 1. If this is not the case, its value must be taken into account in determining the phase detector gain. A unity phase detector gain indicates that an error signal of 0.1 V at the `Loop Filter` input represents a 0.1 rad phase error. In order for the PLL to operate properly, the simulation sampling frequency should be much larger than the PLL natural frequency (`Loop Filter` block parameter).

x = Input signal (complex)

y_1 = VCO output signal (complex)

y_2 = VCO phase (radians)

12.1.8 PLL2REAL.VSM

This compound block is identical to the PLL2CPLX.VSM compound block, except that it accepts a real signal input. The gain factor of -2 prior to the `multiply` block is used to compensate for the factor of two loss through the phase detector due to the formation of a double frequency term in this implementation.

x = Input signal (real)

y_1 = VCO output signal (real)

y_2 = VCO phase (radians)

12.1.9 TWTA_TBL.VSM

This compound block provides a TWTA channel based on AM/AM and AM/PM conversion look-up tables. You can modify the files AMAM.MAP and AMPM.MAP to simulate the desired tube characteristics. The AMAM.MAP file maps tube Input Power (decibels) to Output Power (decibels). The AMPM.MAP file maps tube Input Power (decibels) to Output Phase Rotation (degrees). An externally provided Reference Power Level (watts) is used to specify the saturation operating point, also referred to as the 0 dB backoff point.

x_1 = Input signal (complex)

x_2 = Reference power level

y = Output signal (complex)

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

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

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

12.1.10V32DIFDE.VSM

This compound block implements a V.32 differential decoder for use in simulating V.32 trellis decoding. The input symbol is divided into four bit streams, and the two LSBs are differentially decoded (modulo 4). The bit streams are then recombined to form the output symbol.

x_1 = Input symbol

x_2 = Clock

y = Output symbol

12.1.11 V32DIFEN.VSM

This compound block implements a V.32 differential encoder for use in simulating V.32 trellis decoding. The input symbol is divided into four bit streams, and the two LSBs are differentially encoded (modulo 4). The bit streams are then recombined to form the output symbol.

x_1 = Input symbol

x_2 = Clock

y = Output symbol

12.1.12 VCPG.VSM

This compound block implements a *voltage controlled pulse generator* (VCPG). The output represents a pulse train where the pulse frequency is controlled by the input signal level. The block comprises a `VCO` block, a `crossDetect` block, and a `limit` block to eliminate the negative pulse at the half cycle point. When the input drive level is 0, the `VCPG` outputs a pulse train at the specified VCO center frequency. With a non-zero input, the pulse frequency will vary depending on the magnitude of the drive signal and the specified VCO gain.

x = Input signal

y = Pulse train output (0, 1)

12.2 Data Files

12.2.1 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.

12.2.2 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.

12.2.3 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.

12.2.4 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 “8.9.11 PSK Modulator” on page 8-101.

12.2.5 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 “8.9.12 QAM/PAM Modulator” on page 8-106.

12.2.6 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.

12.2.7 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 “8.9.12 QAM/PAM Modulator” on page 8-106.

12.2.8 V32TRELS.DAT

This data file contains the trellis mapping for V.32 trellis coded modulation. For a description of the file format, refer to “8.5.13 Trellis Encoder” on page 8-55.

12.2.9 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 “8.5.15 Viterbi Decoder (Soft)” on page 8-58.

Chapter 13
Sample Block Diagrams

The following are described in this chapter:

13.1 Standard Block Diagrams

Block Diagram	Description
AM_MOD.VSM	AM example
AUTOCORR_EX.VSM	Computation of autocorrelation of a signal
BERCURVE.VSM	BER curve generation example
BPSKTRAC.VSM	BPSK modulated signal is tracked using a Costas loop
COMPAND.VSM	Compander use example
CONV_ENC.VSM	Convolutional encoder and Viterbi hard decision decoding
DELAYEST.VSM	Use of the Delay Estimator block
DSSS_BER_EX.VSM	Computation of BER curve for spread spectrum QPSK link example
DSSS_TRACK_PERF.VSM	Tracking loop performance for spread spectrum receiver
ECHO CANCEL.VSM	Echo cancellation using LMS adaptive equalization
EYE_PLOT.VSM	Eye diagram

Block Diagram	Description
FFT_TEST.VSM	Use of forward and inverse FFTs
FILTER_EX.VSM	Filtering within Commsim and use of filter viewer
FIR_IMP.VSM	Use of FFT block to compute the freq. response of an FIR filter
GMSK_EX.VSM	Comparison of GMSK and MSK modulation formats
MFSK_BER.VSM	BER curve for MFSK simulation
MFSK_DETECTOR.VSM	Detection of MFSK signal
OSCILLOSCOPE.VSM	Use of Oscilloscope block
PAM8EQ.VSM	Adaptive equalization example using 8-PAM
QAM16_EQ.VSM	Adaptive equalization example using 16-QAM
RAYLEIGH.VSM	Rayleigh channel fading example
REEDSOLOMON.VSM	Reed-Solomon coding and decoding example
SPECTRUM ANALYZER.VSM	Use of Spectrum Analyzer block
SPECTRAL MASK. VSM	Overlay of a spectral mask over a spectrum plot
TONE REJECTION.VSM	Filtering of audio .WAV files
TWO_TONE.VSM	Two tone intermodulation example with a saturating amplifier
TWTA_EYE.VSM	Lookup table TWTA example
V32TRELS.VSM	V.32 trellis encoding and decoding example

13.2 WLAN Block Diagrams

The following sample application diagrams are included with the Commsim WLAN module, and are located in the “Wireless” folder in the Comm Examples directory.

Block Diagram	Description
80211a_OFDM.vsm	Example of OFDM modulation w/o the use of a guard interval
80211a_OFDM_GI.vsm	Example of OFDM modulation using a guard interval (1/4 FFT size)
80211a_interleaver.vsm	Illustrates the behavior of the 802.11a interleaver block
80211b_CCK.vsm	Illustrates the use of CCK modulation (5.5 Mbps mode)
80211b_DBPSK.vsm	Differential BPSK modulation example including Barker code usage
Bluetooth_GFSK.vsm	GFSK modulation example as used in the Bluetooth specification
Bluetooth_Spectrum.vsm	Bluetooth hopping spectrum simulation
BluetoothHopGen.vsm	Illustrates the hopping sequences associated with Bluetooth
BPSK_OFDM_BER.vsm	Shows OFDM performance using BPSK mapping and no encoding
CCITT_CRC16.vsm	CRC generation example
OFDM_GI_BER.vsm	Shows OFDM performance using BPSK mapping and no encoding, but including a guard interval
Scrambler_errors.vsm	Illustrates error propagation associated with the use of the 802.11a scrambler
Short_Hamming.vsm	Illustrates the error correction capability of the (15,10) Hamming code

Chapter 14
Acronyms and Abbreviations

Term	Definition
16-PSK	16-phase shift keying
16-QAM	16-level quadrature amplitude modulation
256-QAM	256-level quadrature amplitude modulation
32-QAM	32-cross quadrature amplitude modulation
64-QAM	64-level quadrature amplitude modulation
8-PSK	eight-phase shift keying
ADC	analog-to-digital converter
AM	amplitude modulation
ASK	amplitude shift keying
AWGN	additive white Gaussian noise
BER	bit error rate
BPSK	binary phase shift keying
BSC	binary symmetric channel
CCK	complementary code keying
DOS	disk operating system
DPSK	differential phase shift keying
DQPSK	differential quadrature phase shift keying
DSB-AM	double-sideband amplitude modulation
FIFO	first in first out

FIR	finite impulse response
FM	frequency modulation
FSK	frequency shift keying
GFSK	gaussian frequency shift keying
IIR	infinite impulse response
IM	intermodulation
ISI	inter-symbol inference
LIFO	last in first out
LSB	least significant bit
MSB	most significant bit
MSK	minimum shift keying
OFDM	orthogonal frequency division multiplexing
OQPSK	offset quadrature phase shift keying
PAM	pulse amplitude modulation
PBCC	packet binary convolutional coding
PLL	phase-locked loop
PM	phase modulation
PN	pseudo noise
PPM	pulse position modulation
PRBS	pseudo random binary sequence
QAM	quadrature amplitude modulation
QPSK	quadrature phase shift keying
SER	symbol error rate
SNR	signal to noise ratio
SQPSK	staggered quadrature phase shift keying
TWTA	traveling wave tube amplifier
VCO	voltage controlled oscillator
VCPG	voltage controlled pulse generator

Index