Capsim Blocks Documentation

Silicon DSP Corporation

http://www.silicondsp.com

Note: Not all blocks available in text mode kernel.

Some of the blocks documented in this reference were developed at:

Center for Communication and Signal Processing (CCSP)
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7914

and

Dept. of Electrical and Computer Engineering
University of Calfornia, Berkeley

Table of Contents

Sources

addnoise

Description

This block adds white gaussian noise to the input data stream.

Programmer: L. James Faber

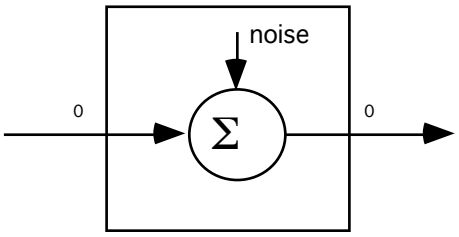
Parameters:
(1) Sets the power of the added noise and should be >= 0.
float var=1.0

(2)Sets a seed for the random number generator. Random sequences can be unique and repeatable for each instance of this block.
int seed = 333

Buffer:

Inputs:
Buffer 0: Input samples (float)
Outputs:
Buffer 0: Output samples (float).

Graphic



arprocess

Description

This block generates a selectable number of samples from an AR process, represented as an IIR filter driven by Gaussian noise. The order of the process, and the weighting values can be selected, via array parameter 2. Maximum order is 10 (since this is max array size). Parameter 3 selects the variance of the gaussian driving noise.

This block supports auto-fanout.

Programmer: L.J. Faber
Date: April, 1988.

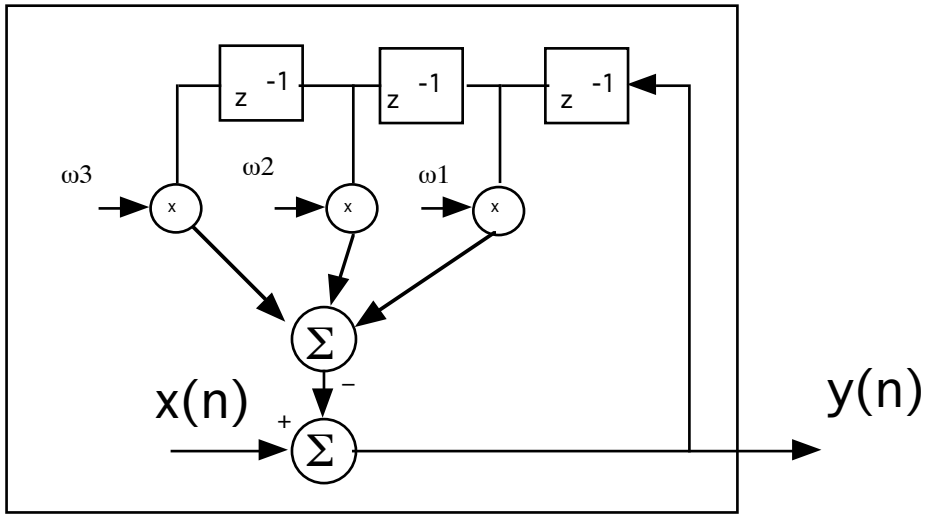
Parameters:

(1) Number of samples to generate:
int samples = 100;
(2) Array of weights representing AR process. (Maximum 10) Ex. 3 0.1 -0.2 0.4
array weights;
(3) The variance of the gaussian driving noise
float variance = 1.0;

Buffer:

Inputs: Auto fanin.
Outputs: Auto fanout

Graphic



bdata

Description

This function generates a random sequence of bits, which can be used to exercise a data transmission system.

The pseudo-random sequence generator uses the polynomial x**10+x**3+1.

CONTROL PARAMETERS:

num_of_samples = total number of samples to output.
pace_rate = multiplies the number of samples received on pace input (if connected) to determine how many samples to output.
samples_first_time = the number of samples to put out on the first call if pace input connected. It can be zero. negative values = 0.

CONTROL DESCRIPTION:

If the pace input is not connected:
The num_of_samples parameter sets the maximum number of samples that the block will output. If num_of_samples < 0, an indefinite number of samples can be output.
The block will output a maximum of NOSAMPLES on each call.
If the pace input is connected:
The num_of_samples parameter sets the maximum number of samples that the block will output. If num_samples_out < 0, an infinite number of samples can be output.
The pace input paces the number of output samples on each call.
At each call of the block all samples are read from the pace input and a running total of how many there have been is kept.
An output_target is computed at each pass = pace_input_total * pace_rate. If pace_rate < 0, the absolute value is used.

On the first call:
output = lesser of (samples_first_time, num_of_samples)
On subsequent calls:
output = lesser of (NOSAMPLES, output_target)
output_target = samples_first_time + pace_rate * pace_input_total - to that point
The total number of samples that will be output:
samples_out_total = lesser of (num_of_samples, samples_first_time + pace_rate * pace_input_total)

Programmer: R. T. Wietelmann/G.H.Brand
Date: Oct 7, 1982
Modified for V2.0 by D.G.Messerschmitt March 11, 1985
Mod: ljfaber 12/87 add 'auto fanout'

Parameters:

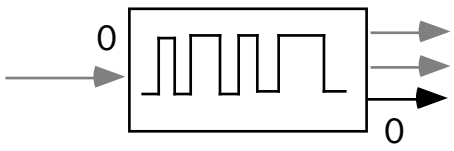
(1)Total length of the sequence to be generated.
int length;
(2) Initialize shift reg for pseudo random sequence.
int initialize = 12;
(3) Pace rate to determine how many samples to output
float pace_rate = 1.0;
(4) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:

Inputs:
Buffer 0: Pace input (optional) (float)

Outputs:
Auto fanout. Binary data.(float)

Graphic



seqgen

Description

This function generates a sequence of bits, which can be used to exercise a data transmission system. Any degree polynomial can be implemented as specified in a parameter array.

For example to specify a polynomial x**0+x**3+1 use,

array poly= 10 0 0 1 0 0 0 0 0 0 1

CONTROL PARAMETERS:

num_of_samples = total number of samples to output.
pace_rate = multiplies the number of samples received on pace input (if connected) to determine how many samples to output.
samples_first_time = the number of samples to put out on the first call if pace input connected. It can be zero. negative values = 0.

CONTROL DESCRIPTION:

If the pace input is not connected:
The num_of_samples parameter sets the maximum number of samples that the block will output. If num_of_samples < 0, an indefinite number of samples can be output.
The block will output a maximum of NOSAMPLES on each call.
If the pace input is connected:
The num_of_samples parameter sets the maximum number of samples that the block will output. If num_samples_out < 0, an infinite number of samples can be output.
The pace input paces the number of output samples on each call.
At each call of the block all samples are read from the pace input and a running total of how many there have been is kept.
An output_target is computed at each pass = pace_input_total * pace_rate. If pace_rate < 0, the absolute value is used.

On the first call:
output = lesser of (samples_first_time, num_of_samples)
On subsequent calls:
output = lesser of (NOSAMPLES, output_target)
output_target = samples_first_time + pace_rate * pace_input_total - to that point
The total number of samples that will be output:
samples_out_total = lesser of (num_of_samples, samples_first_time + pace_rate * pace_input_total)

Programmer: Ray Kassel

Date: Nov. 10, 1990

Parameters:

(1) Total length of the sequence to be generated.
int length*;*

(2) Number of samples per chip.
int num_per_chip = 8;

(3) Length of shift register.
int shift_length = 10;

(4) Initialize shift reg for pseudo random sequence.
int initialize = 12*;*

(5) Array of polynomial (0 or 1). Note length of array should equal parameter 3;
array poly;

(6) Pace rate to determine how many samples to output
float pace_rate = 1.0*;*

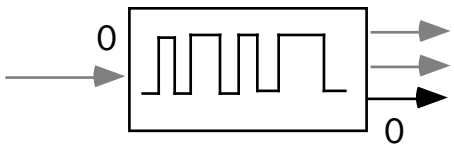
(7) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:

Inputs:
Buffer 0: Pace input (optional) (float)

Outputs:
Auto fanout. Binary data.(float)

Graphic



gauss

Description

This block generates gaussian samples. The first parameter, which defaults to NOSAMPLES, tells how many total samples to send out. The second parameter is the standard deviation which defaults to one. The third parameter is the random number seed.

For control parameters (pacing) see bdata for a description.

Parameters:

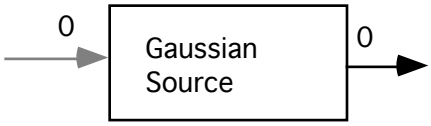
(1) Number of samples to generate.
int length = 128;
(2) Standard deviation.
float sigma = 1.0;
(3) Seed.
int seed;
(4) Pace rate to determine how many samples to output
float pace_rate = 1.0;
(5) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:

Inputs:
Buffer 0: Pace input (optional) (float)

Outputs:
Buffer 0 : Gaussian samples.(float)

Graphic



impulse

Description

This block sends out a unit sample, then a number of zero samples. The only parameter, which defaults to NOSAMPLES, sets the total samples to send out.

Mod: ljfaber 12/87 add 'auto fanout'

Parameters:

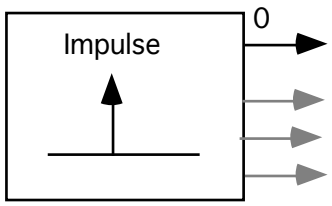
(1) Enter number of samples.
int length = 128;

Buffer:

Inputs:
None

Outputs:
Auto fanout

Graphic



pulse

Description

This block generates a pulse train with variable pulse width, period, DC offset and amplitude.

Programmer: Sasan Ardalan
Date: Nov. 1987

Parameters:

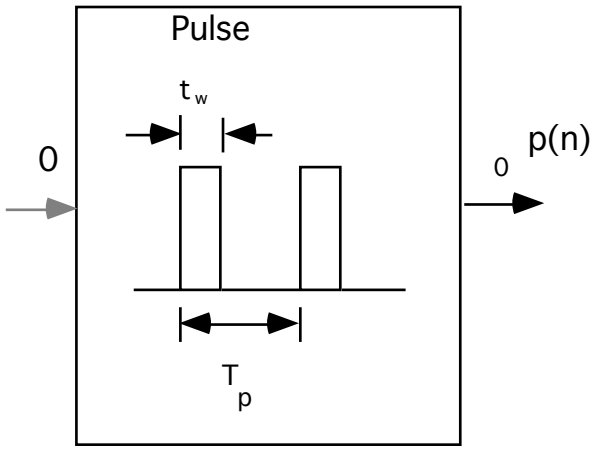
(1) Number of samples to generate.
int length = 128;
(2) Magnitude.
float magnitude = 1.0;
(3) Period in samples.
int np;
(4) Pulse width in samples.
int nw;
(5) Initial delay in samples.
int nd;
(6) DC value.
float dcValue=0.0;
(7) Pace rate to determine how many samples to output
float pace_rate = 1.0;
(8) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:

Inputs:
Buffer 0: Pace input (optional) (float).

Outputs:
Buffer 0 : Samples (float).

Graphic


Pulse
t w
0
p(n)
0
T p

sine

Description

This block generates a sinusoid (cosine for zero phase) . The The first parameter, which defaults to NOSAMPLES (128), tells how many total samples to send out. The second parameter is the magnitude which defaults to one. The third parameter is the sampling frequency. The 4th parameter is the frequency The fifth parameter is the phase is degrees.

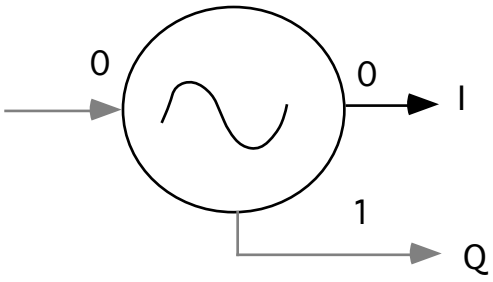
Programmer: Sasan Ardalan
Date: Nov. 1987

Parameters:
(1) Number of samples.
int length = 128;
(2) Magnitude.
float magnitude = 1.0;
(3) Sampling Rate.
float fs = 8000.0;
(4) Frequency.
float freq = 1000.0;
(5) Phase (degrees) .
float phase=0.0;
(6) Pace rate to determine how many samples to output
float pace_rate = 1.0;
(7) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:

Inputs:
Buffer 0: Pace input (optional) (float)
Outputs:
Buffer 0: In phase (float)
Buffer 1: Quadrature (optional) (float)

Graphic


0
0
I
1
Q

rdfile

Description

This function performs the simple task of reading sample values in from a file, and then placing them on it's output buffer. The file may have multiple sample values per line, which can be integer or float.

An example use for this routine is to access a stored waveform as input to a simulation. The parameters are:

file_name = name of file to read from, defaults to "stdin"

Programmer: R. T. Wietelmann / D.G.Messerschmitt
Date: June 5, 1982
Modified for V2.0 by D.G. Messerschmitt March 7, 1985
Modified: July 10, 1985 by D.J.Hait
Modified: April, 1988 L.J.Faber: add "auto-fanout"

Parameters

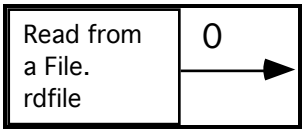
(1) File name to read from.

file file_name = "stdin";

Buffer:

Inputs:
None.
Outputs:
Auto fanout

Graphic



rdmulti

Description

This function performs the task of reading multi column or single column but multiple samples from a file, and then placing them on it's output buffers.The number of output buffers determines the number of columns of the input data. The input data is read regardless of its organization.

The file may have multiple sample values per line, which can be integer or float.

An example use for this routine is to access x,y, and z data from a file or to split odd /even samples in a file to two buffers.

Programmer: Sasan Ardalan
Date: June 5, 1982

Parameters

(1) File name to read from.
file file_name = "stdin";

Buffer:

Inputs:
None.
Outputs:
Each sample read is output to a buffer The next sample is output to the next buffer modulo the number of buffers. Auto fanout

Graphic

rdimage

Description

Read a ASCII image. On each visit a row is read from file and output. The width and height are read from the file and should be on the first line.

Auto fan-out.

*Programmer:*Sasan Ardalan
Date: November 14, 1990

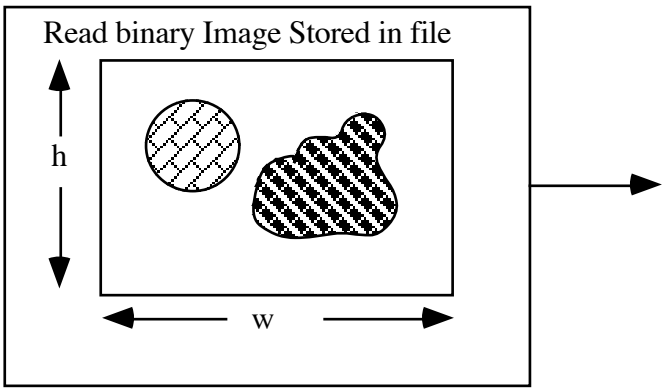
Parameters

(3) File that contains ascii image.
file file_name = "test.img"

Buffer:

Inputs:
None.
Outputs:
Auto fanout

Graphic



rdbinimg

Description

Read a binary image. On each visit a row is read from file and output. The image is assumed to stored a byte to a pixel in binary format with no header information.

Auto fan-out.

*Programmer:*Sasan Ardalan
Date: November 14, 1990

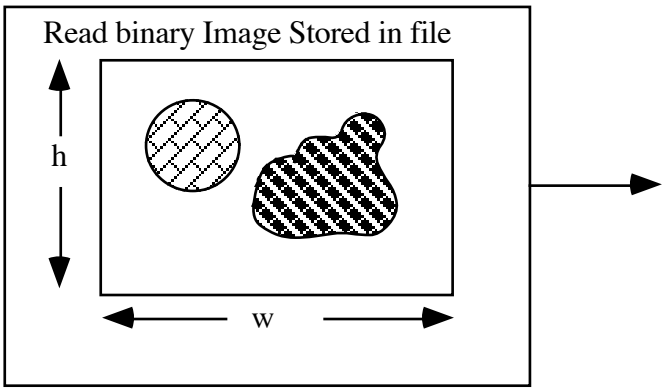
Parameters

(1) Image width.
int width=128;
(2) Image height
int height=128;
(3) File that contains binary image.
file file_name = "test.img
(4) Number of bytes to skip
int skip=0;

Buffer:

Inputs:
None.
Outputs:
Auto fanout

Graphic



time

Description

Function outputs the "time" to all connected output buffers. The time between samples and the time before stopping are both input parameters:

The time between samples defaults to 1.0 ("second")
The time before stopping defaults to infinity

Using the stopping criterion is a convenient way of controlling the length of the simulation to a certain time interval specified in seconds.

Programmer: D.G.Messerschmitt
Date: June 26, 1982
Modification for V2.0: Jan. 10, 1985
Mod: ljfaber 12/87 add 'auto fanout'

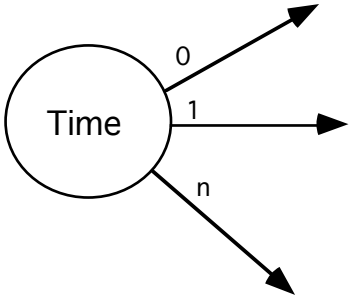
Parameters:

(1) The time between samples defaults to 1.0 ("second")
float time_scale=1.0;
(2) The time before stopping defaults to infinity
float time_stop = -1.0;

Buffer:

Inputs:
None.
Outputs:
Auto fanout.

Graphic



wave

Description

This block simulates a wave generator.

wave_type	wave description
0	sine
1	cosine
2	square
3	triangle
4	sawtooth

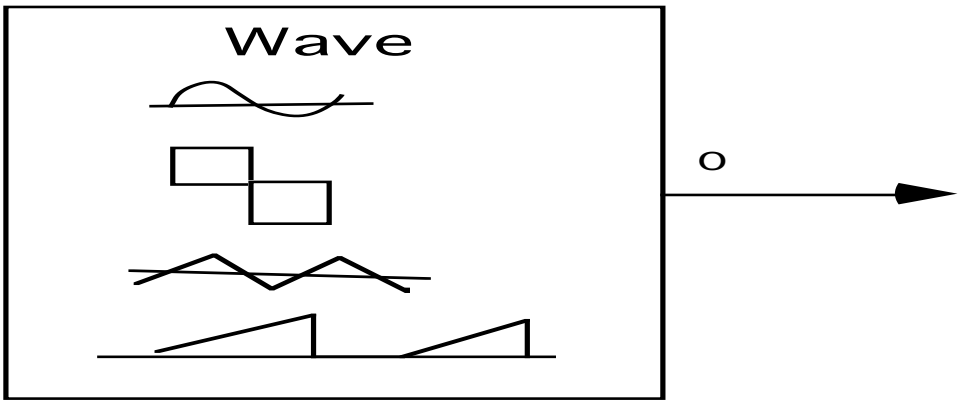
Notes:
Pacer support.
Auto fan-out

Programmer: Prayson W. Pate
Modified by Sasan Ardalan Dec. 1990
Date: August 18, 1987

Parameters:
(1) Number of samples.
int length = 128;
(2) wave_type to be generated
int wave_type = 0;
(3) period of wave
float period = 1.0;
(4) peak value of wave
float peak = 1.0;
(5) Phase (degrees) .
float phase=0.0;
(6) Pace rate to determine how many samples to output
float pace_rate = 1.0;
(7) Number of samples on the first call if paced
int samples_first_time = 128;

Buffer:
Inputs:
Pacer input (optional)
Outputs:
Auto fan-out (float)

Graphic


Wave
o

zero

Description

This block sends out a number of zero samples. The only parameter tells how many zero samples to send out.

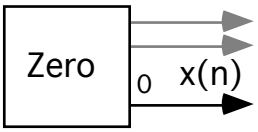
Mod: ljfaber 12/87 add 'auto fanout'

Parameters:
(1) Number of zeroes to output
int length = 0;

Buffer:

Inputs:
None.
Outputs:
Auto fanout.

Graphic



xygen

Description

Total number of samples generated per X and Y is the matrix dimension squared. Think of this block as generating x and y coordinates for a square matrix starting from the top row moving down to the final row with x changing along the columns and y changing along the row.

The results of this block can be used to form functions of x and y. F(x,y)

Generate x and y samples. x changes from minimum X to maximimum x in xsteps as y is kept constant.
Y starts from minimum y to maximum y in y step increments.

Programmer: Sasan Ardalan
Date: April 1991

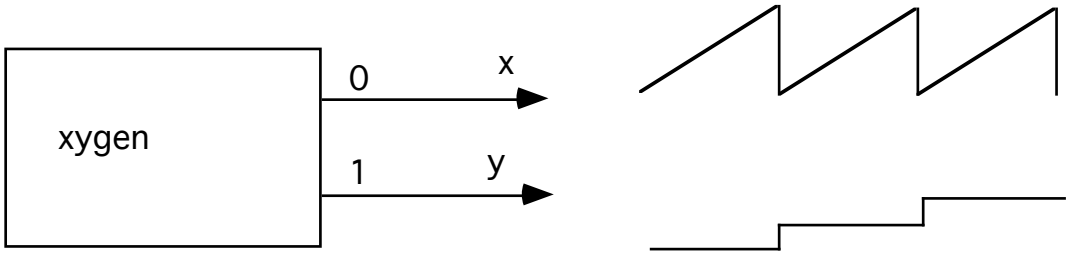
Parameters

(1) Matrix Dimension
int matrixDim = 32;
(2) Minimum x
float xMin=0.0;
(3) x step
float xStep=1.0;
(4) Minimum y
float yMin=0.0;
(5) y step
float yStep=1.0;

Buffers

Outputs:
Buffer 0: x (float)
Buffer 1: y (float)

Graphic



Encoders/Decoders

linecode

Description

This block inputs 0/1 binary data and outputs various line codes.
Line codes are selectable by the first input parameter `code_type':
0 - Binary (NRZ) (1 = +1, 0 = -1) (Default; 1 phase)
1 - Biphase (Manchester) (1 = -1,+1; 0 = +1,-1) (2 phase)
2 - 2B1Q (00 = -3, 01 = -1, 10 = +1, 11 = +3) (1 phase)
3 - RZ-AMI (Alternate mark inversion)
The code output oversampling rate (samples per baud interval) is selected by the second parameter `smplbd'. Note that multi-phase codes require oversampling rates which are integer multiples of the number of phases! I/O buffers are float to be compatible with most blocks.

Programmer: L.J. Faber
Date: 11/25/86

Parameters:

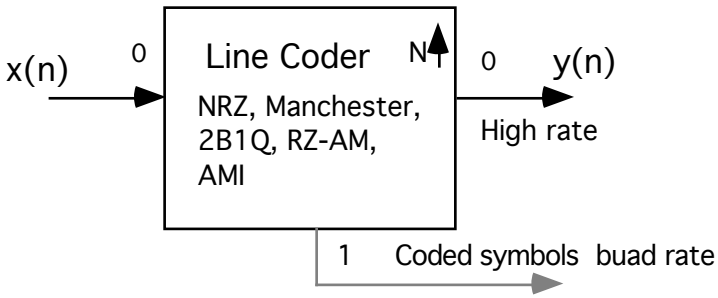
(1) Code type:0-Binary(NRZ),1-Biphase(Manchester),2-2B1Q,3-RZ-AM.
int code_type = 0;
(2) Samples per baud.
int smplbd = 8;

Buffer:

Inputs:
Buffer 0: Binary data (delay_max = 1) (float).

Outputs:
Buffer 0: Coded oversampled samples (float).
Buffer 1:Symbol data (not oversampled) (float).

Graphic



atod

Description

This block simulates a uniform analog to digital converter.

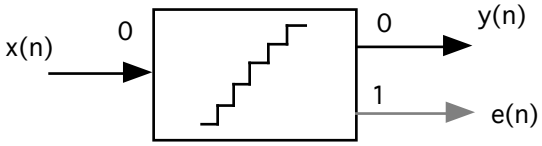
Programmer: Sasan Ardalan
Date: February 17, 1989

Parameters:
The number of bits (e.g. 13 bits)
int bits =13
The Maximum input range (e.g. +- 5 volts)
float maxLevel = 2.0;

Buffer:

Inputs:
Buffer 0->Input samples to be quantized. (float)
Outputs:
Buffer 0 -> Quantized samples (float).
Buffer 1-> Quantization error (float). Optional.

Graphic



dta

Description

This block simulates a digital to analog converter.

Programmer: Sasan Ardalan
Date: February 17, 1989

Parameters:

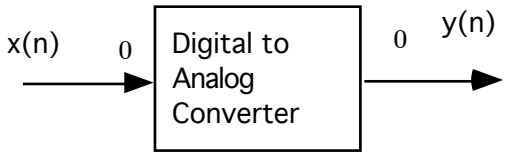
(1) The number of bits (e.g. 13 bits)
int bits = 13;
(2) The Maximum output range (e.g. +- 5 volts)
float maxLevel = 2.0;

Buffer:

Inputs:
Buffer 0: Digital Samples (float).

Outputs:
Buffer 0 : Analog Samples (float).

Graphic



mulaw.

Description

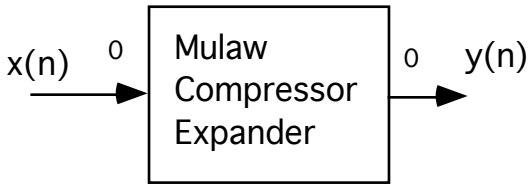
This block implements a mulaw quantizer.
Programmer: Sasan Ardalan

Parameters:
(1) Compress or Expand Flag. Compress=0, Expand=1.
int compExFlag = 0

Buffer:

Inputs: input samples (float).
Outputs: output samples (float).

Graphic



ds2

Description

This block is a self-contained second order delta sigma modulator.
Ouput 0 is the output of the circuit. Output 1 is the input to the comparator.

-Parameter one: the gain of the first integrator
-Parameter two: the gain of the second integrator
-Parameter three: the value for delta

Programmer: John T. Stonick
Date: February 1989

Parameters

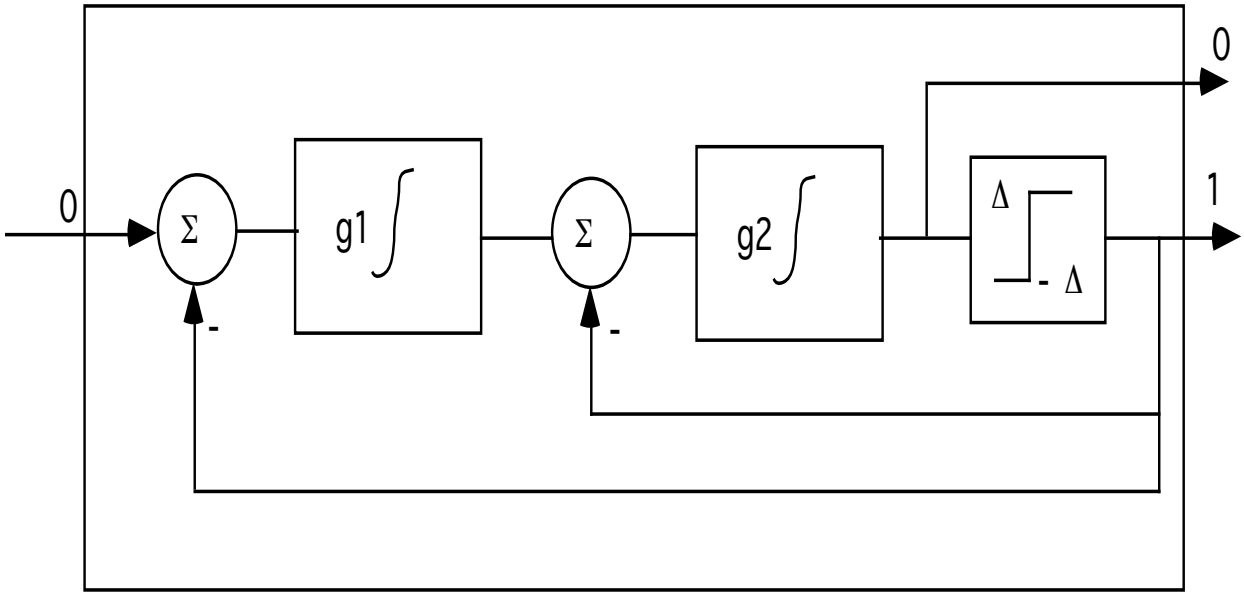
(1) The gain of the first integrator
float g1=1.0;
(2) The gain of the second integrator
float g2=1.0;
(3) Binary Quantizer Level
float delta=1.0;

Buffers

inputs: Buffer 0 in (float)

outputs: Buffer0 error (float)
Buffer 1 delta sigma bit stream (float)

Graphic


0
Σ
-
g1
Σ
-
g2
Δ
- Δ
0
1

ds3

Description

This block is a self-contained third order delta sigma modulator.
Ouput 0 is the output of the circuit. Output 1 is the input to the comparator.

-Parameter one: the gain of the first integrator
-Parameter two: the gain of the second integrator
-Parameter two: the gain of the third integrator
-Parameter three: the value for delta

Programmer: John T. Stonick
Date: February 1989

Parameters

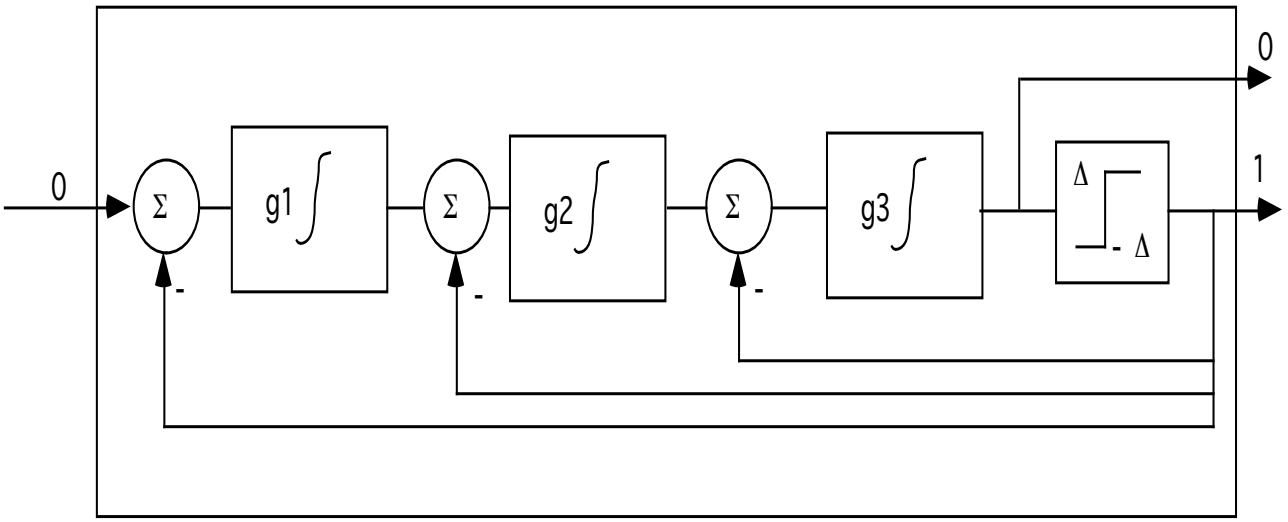
(1) The gain of the first integrator
float g1=1.0;
(2) The gain of the second integrator
float g2=1.0;
(3) The gain of the second integrator
float g3=1.0;
(4) Binary Quantizer Level
float delta=1.0;

Buffers

inputs: Buffer 0 in (float)

outputs: Buffer0 error (float)
Buffer 1 delta sigma bit stream (float)

Graphic


0
Σ
g1
Σ
g2
Σ
g3
Δ
- Δ
0
1
-
-
-

slice

Description

This block simulates a decision element for a data receiver. It compares the incoming signal to a set of thresholds, which are COMPUTED from the user-specified set of output levels. Thresholds are always exactly HALF way between specified output levels.

Output levels are specified via a topology file parameter array. Parameter arrays are float, so there are no restrictions on specified output values. Only 10 levels are allowed, and must be listed in ascending magnitude order.

Examples-- binary decision (threshold = 0.0)
param array 2 -1.0 1.0

quaternary decision (thresholds -2/0/2)
param array 4 -3. -1. 1. 3.

The number of output channels is determined at run-time (auto-fanout).

Programmer: L.J. Faber
Date: April, 1988

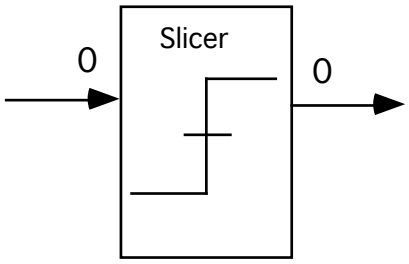
Parameters:

(1) Array of decision levels.
array level;

Buffer:

Inputs:
Buffer 0: Input signal.
Outputs:
Auto fanout.

Graphic



v29encoder

Description

This block inputs data and ouputs the coordinates of the CCITT v.29 encoder constellations. Output0 corresponds to the real value and Output1 corresponds to the coordinates of the imaginary.

Programmer: A. S. Sadri
Date: Aug. 2, 1990

Parameters

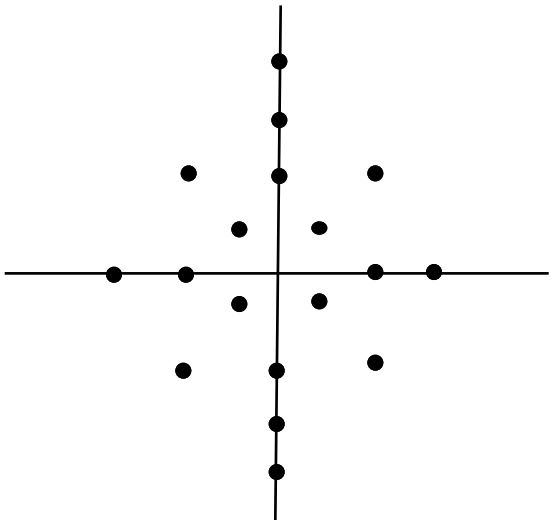
None

Buffer:

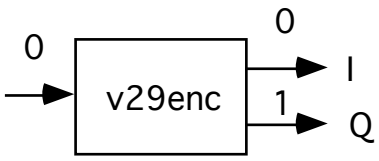
Inputs:
Buffer0 : data (float)

Outputs:

Buffer 0: inPhase (float)
Buffer 1: quadPhase (float)



Graphic


0
v29enc
0
I
1
Q

Measurement Models

ecount

Description

"error counter"

This block compares two data streams for "equality". (Since the input streams are floating point, a guard band is used.) An output stream is created, with 'zero' output for equality, and 'one' if there is a difference. (Note: the output stream is optional--if no block is connected to the output, there is no output.) Param. 1 selects an initial number of samples to be ignored for the final error tally (used during training sequences); default zero. Param 2 sets an index, after which a message is printed to stderr for each error. It defaults to "infinity", i.e. no error messages. This block prints a final message to stderr giving the error rate (errors/smpl), disregarding the initial ignored samples.

Programmer: L.J. Faber
Date: Dec 1987

Parameters:

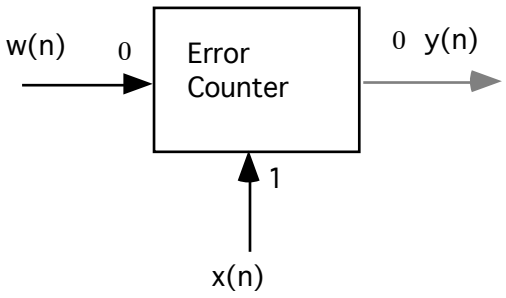
(1) Number of initial samples to ignore.
int ignore = 0;
(2) Number of samples after which errors are reported to stdout.
int err_msg = 30000;

Buffer:

Inputs:
Buffer0 : w (float)
Buffer1: x (float)

Outputs:
If output connected then it outputs results (float)..

Graphic



jitter

Description

This block generates the jitter sequence from a timing wave. A reference square wave clock is used. The jitter is in degrees.

Written by : Sasan Ardalan, October 1989.

Parameters:

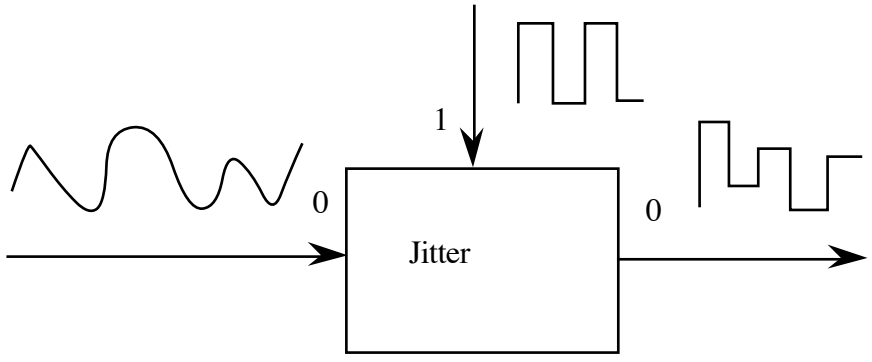
(1) Trigger edge: 1= Rising, 0 = Falling.
int edge = 1;

(2) Output Rate: Synchronous or One per cycle.
int sync=1;

Buffer:

Inputs:
Buffer 0: Timing wave (float).
Buffer 1: Reference Clock (Square wave zero mean)(float).

Outputs:
Buffer 0: Jitter sequence in degrees. (float)



powmeter

Description

This block is an averaging logarithmic power meter, which can be connected either in-line or terminating. If an output is connected, input 0 is passed through unchanged. If no output is connected, the signal is absorbed (like sink).

The block computes 10*log10(square) of the signal at input 0, and (optionally) compares it to another signal at input 1. If no signal is connected to input 1, power is referenced to unity. This block ultimately prints an ASCII file with the power results.

Parameter 1: (file) name of output file; default => powfile
2: (int) number of samples to average; default => 1

Programmer: L.J. Faber
Date: May 1988
Modified: 9/88 output to stdout if desired

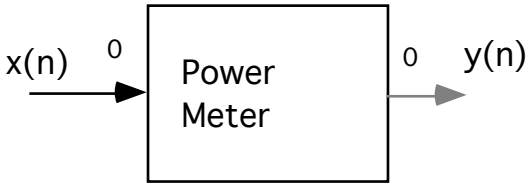
Parameters:

(1) Name of file to store results.
file powfile_name = "powfile.dat";
(2) Number of samples to average.
int N = 1;

Buffer:

Inputs:
Buffer 0: Samples (float).
Outputs:
Auto termination on Buffer 0 if no output connected.

Graphic



sdr

Description

This block computes the signal to total harmonic and noise ratio.

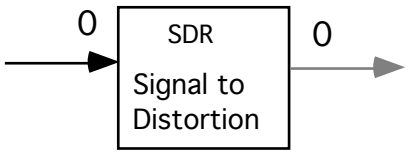
Programmer: Sasan Ardalan
Date: 2/16/89
Modified: L.J. Faber 1/3/89. Add flow through; general cleanup.

Parameters:
(1) Number of buffer points in each plot.
int npts ;
(2) Points to skip before first plot.
int skip ;
(3) File to store sdr results.
file sdrRes;
(4) Window flag (0=Rect 1= Hamming).
int wind=1;

Buffer:

Inputs:
Buffer 0 : Input samples.(float)
Outputs:
Optional feed-through of input channels.

Graphic



stats

Description

This block calculates the statistics of the incoming signal. The signal variance, mean, standard deviation, and minimum and maximum values are computed.The parameter is a file name for storage of the results. The results are also dumped to the screen.

Programmer: Prayson W. Pate
Date: December 8, 1987
Modified: February 22, 1987
April 1988
May 1988 ljfaber: add 'flow-thru' capability
: add signal identifier

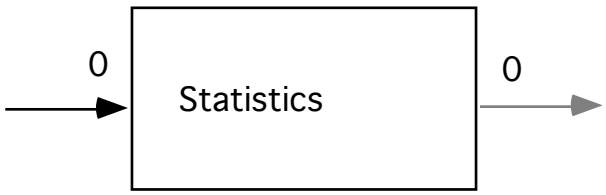
Parameters:

(1) Number of points to skip.
int skip=0;
(2) File to store results.
file stat_file = "dummy_name";

Buffer:

Inputs:
Buffer 0: Single to be analyzed.(float)
Outputs:
Buffer 0: optional: terminate signal or flow through.(float)

Graphic



Filters

iirfil

Description

This block designs and simulates IIR low pass, high pass, band pass, and band stop filters. Butterworth, Chebyshev, and Elliptic filters are supported.

During the design phase, three files are created:
(1) tmp.dat stores the filter design information including specs and results.
(2) tmp.pz stores the poles and zeroes and the normalization constant.
(3) tmp.cas stores the cascade coefficients and the normalization constant.
These files are overwritten when more than one instance of the block is used.

Parameters:

(1) Filter Type:
1=Low Pass,
2=High Pass
3=Band Pass
4=Band Stop
int filterType=1;

(2) Design Type:
1=Butterworth
2=Chebyshev
3=Elliptic
int designType=3;

(3) Sampling Frequency, Hz
float fs=32000.0;

(4) Passband Frequency, Hz (Low pass / High pass filters only).
float fpb=3400.0;

(5) Stopband Freqency, Hz (Low pass/High pass filters only)
float fsb=4400.0;

(6) Lower Passband Frequency (Band pass/Band stop filters only)
float fpl=200.0;

(7) Upper Passband Frequency (Band pass/Band stop filters only)
float fpu=3400.0;

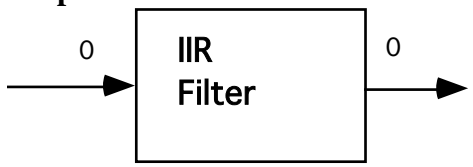
(8) Lower Stopband Frequency (Band pass/Band stop filters only)
float fsl = 10.0;

(9) Upper Stopband Frequency (Band pass/Band stop filters only)
float fsu = 4400;
(10) Filter name (note filter design parameters are stored *name*.*)
file name=tmp;
Note: Default values for low pass and band pass filters.
Buffer:

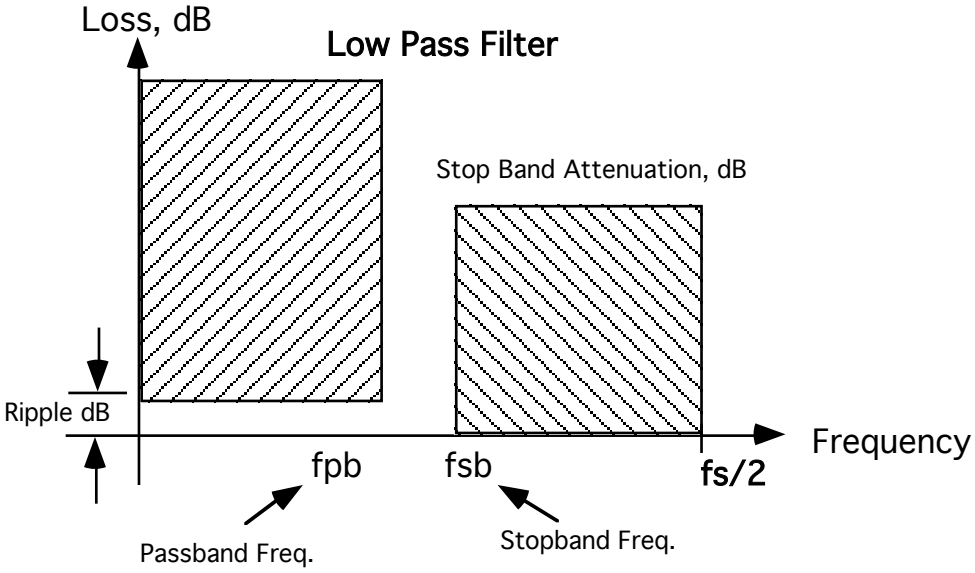
Inputs:
Buffer 0: Samples to be filtered (float).

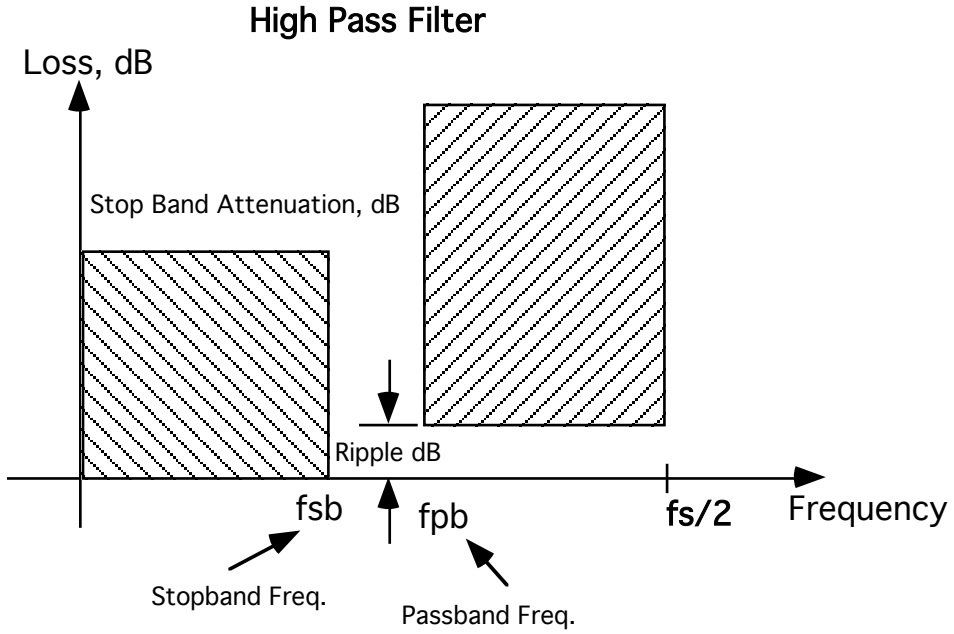
Outputs:
Buffer 0: Filtered samples. (float).

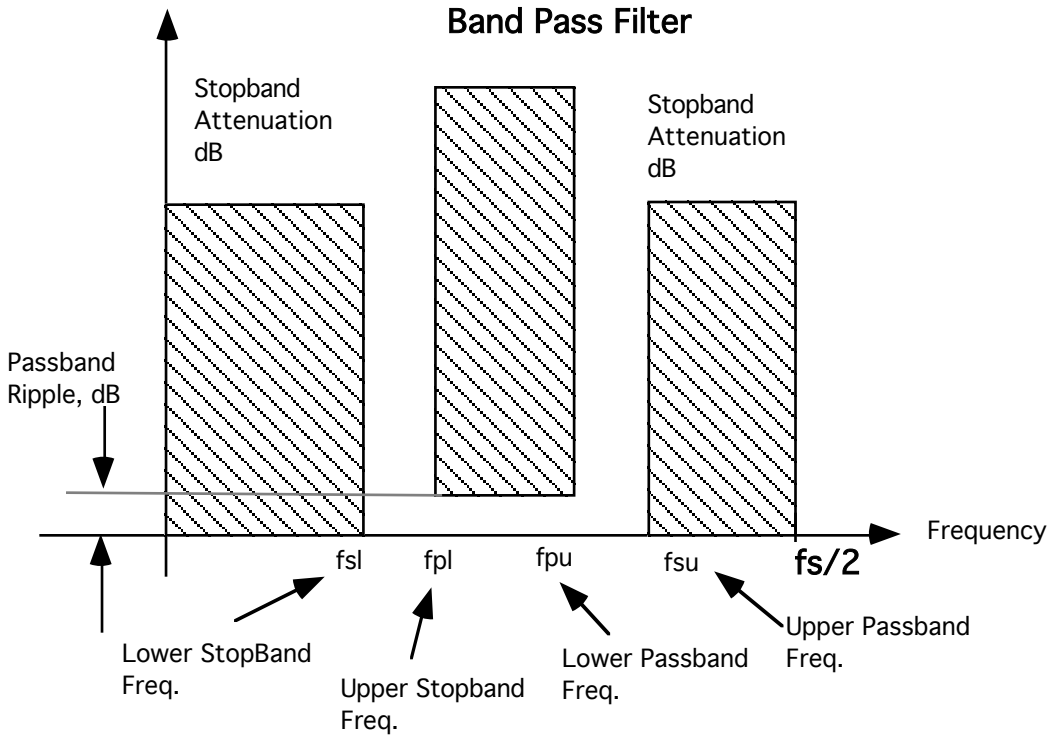
Graphic

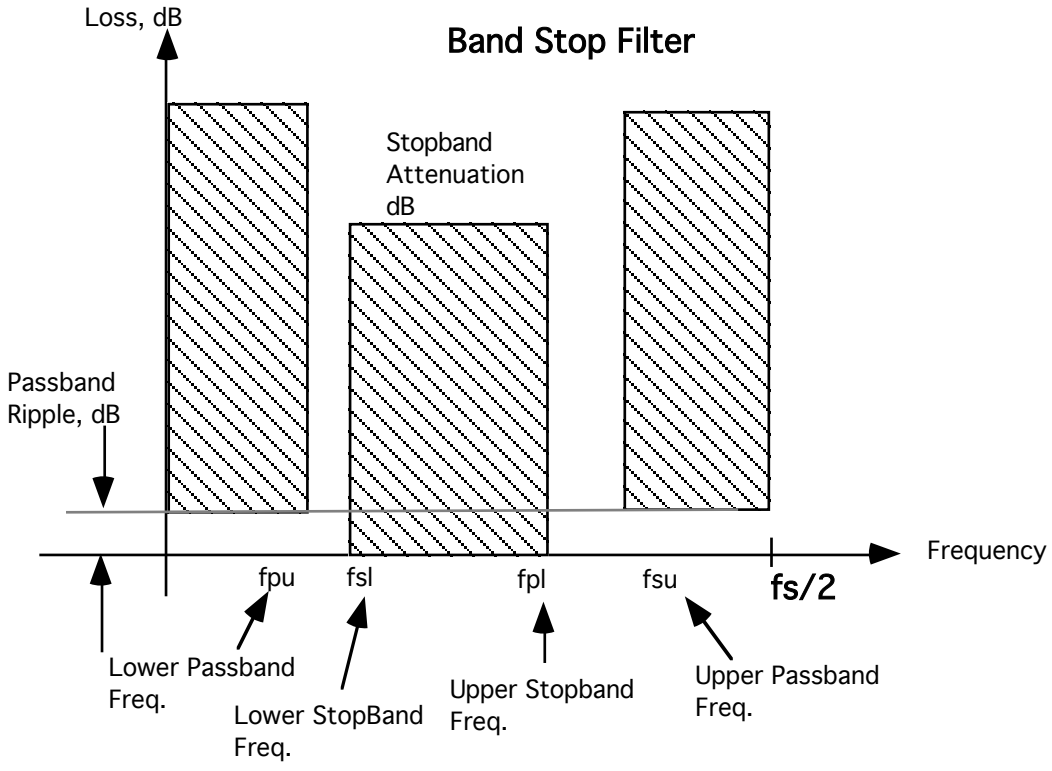


Filter Specifications:






Band Pass Filter
Stopband
Attenuation
dB
Stopband
Attenuation
dB
Passband
Ripple, dB
Frequency
fsl
fpl
fpu
fsu
fs/2
Lower StopBand
Freq.
Upper Stopband
Freq.
Lower Passband
Freq.
Upper Passband
Freq.


Loss, dB
Band Stop Filter
Stopband
Attenuation
dB
Passband
Ripple, dB
Frequency
fpu
fsl
fpl
fsu
fs/2
Lower Passband
Freq.
Lower StopBand
Freq.
Upper Stopband
Freq.
Upper Passband
Freq.

firfil

Description

This block designs and simulates FIR low pass, high pass, band pass, and band stop filters.
The design is based on the windowing method. The following windows are supported:
(1) Rectangular, (2) Triangular, (3) Hamming, (4) Generalized Hamming, (5) Hanning, (6) Kaiser, (7) Chebyshev, (8) Parzen.

During the design phase, two files are created:
(1) tmp.spec stores the filter design information including specs and results.
(2) tmp.tap stores the tap weights for the FIR filter.
These files are overwritten when more than one instance of the block is used.

Parameters:

(1) Filter Type:
1=Low Pass,
2=High Pass
3=Band Pass
4=Band Stop
int filterType=1;

(2) Window Type:
1=Rectangular
2=Triangular
3=Hamming
4=Generalized Hamming
5=Hanning
6=Kaiser
7=Dolph Chebyshev
8=Parzen
int windowType=2;

(3) Number of Taps (set to zero if not specified for Chebyshev window, program will compute based on ripple and transition width).
int ntaps=128;

(4) Cutoff Frequency, normalized. 0=< fc =< 0.5 (Low pass/High pass filters only).
float fc=0.25;

(5) Lower Cutoff Frequency, 0 =< fl =< 0.5 (Band pass/Band Stop filters only)
float fl=0.25;

(6) Upper Cutoff Frequency, 0 =< fh =< 0.5 (Band pass/Band Stop filters only)
float fh=0.35;

(7) Alpha for Generalized Hamming Window 0=< alpha =< 1.0.
float alpha=0.5

(8) Ripple, dB for Chebyshev Window(set to zero if not specified, program will use transition and number of taps).
float dbripple=0.1;

(9) Transition width (normalized) for Chebyshev Window 0<=twidth <= 0.5 (set to zero if not specified, program will use ripple and number of taps).
float twidth=0.05;

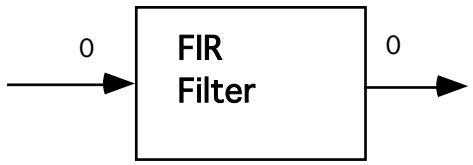
(10) Stop band attenuation, dB for Kaiser Window(b computed from attenuation).
float att=40;

Buffer:

Inputs:
Buffer 0: Samples to be filtered (float).

Outputs:
Buffer 0: Filtered samples. (float).

Graphic



Detailed Description:

The following is adapted from, L.R. Rabiner, C. A. McGonegal, and D. Paul, "FIR Windowed FIlter Design Program - WINDOW," *Programs for Digital Signla Processin*g, IEEE Press, New York, 1979.

Purpose

This block is used to design and implement FIR digital filters using the window method. The program can design lowpass, bandpass, bandstop, and high pass filters for both even and odd values of N, (the impulse response duration in samples), using either a rectangular, a triangular, a Hamming, a Hanning, a Chebyshev, or a Kaiser window.

Method

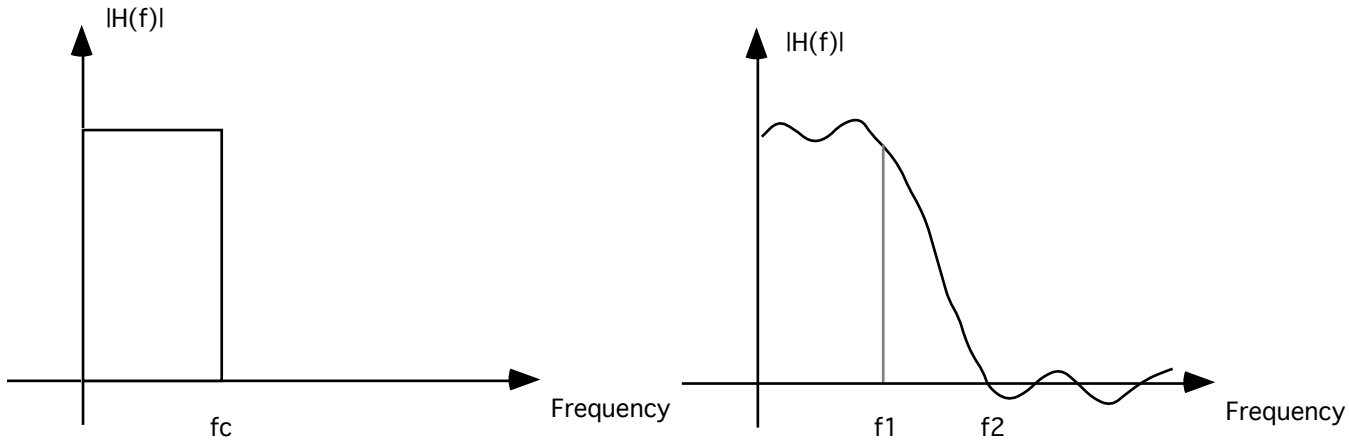
This program uses the well known method of window design for FIR digital filters. If we denote the N-point windows as w(n), for $0 \le n \le N-1$, and we denote the impulse response of the ideal digital filter (obtained as the inverse Fourier transform of the ideal frequency response filter) as h(n), $-\infty < n < \infty$, then the windowed filter is given as

$\hat{h}(n) = w(n)h(n) \quad 0 \le n \le N-1$
$\quad\quad = 0$ otherwise

In the discussion above it is assumed that h(n) incorporates an ideal delay of (N-1)/2 samples, and that w(n) is symmetric around the point (N-1)/2.

From the filter specfications the sequences h(n) and w(n) of Eq. (1) are computed, and the windowed filter is obtained as the final output.

Windows



(a) Filter Specification. (b) Actual smearing due to window

Rectangular Window

$$w_R(n) = \begin{cases} 1.0 & -(\frac{N-1}{2}) \le n \le \frac{N-1}{2} \quad N \text{ odd} \\ 1.0 & -(\frac{N}{2}) \le n \le N/2 - 1 \quad N \text{ even} \end{cases}$$

Triangular Window

$$w_{Triangle}(n) = \begin{cases} 1-|2n|/(N+1) & -(\frac{N-1}{2}) \le n \le \frac{N-1}{2} \quad N \text{ odd} \\ 1-|2n+1|/N & -(\frac{N}{2}) \le n \le (N/2 - 1) \quad N \text{ even} \end{cases}$$

Hanning Window

$$w_{Hann}(n) = \begin{cases} \frac{1}{2}[1 + \cos(\frac{2\pi n}{N+1})] & -(N-1)/2 \le n \le (N-1)/2 \quad N \text{ odd} \\ \frac{1}{2}[1 + \cos(\frac{2\pi(2n+1)}{2(N+1)})] & -(N/2) \le n \le (N/2-1) \quad N \text{ even} \end{cases}$$

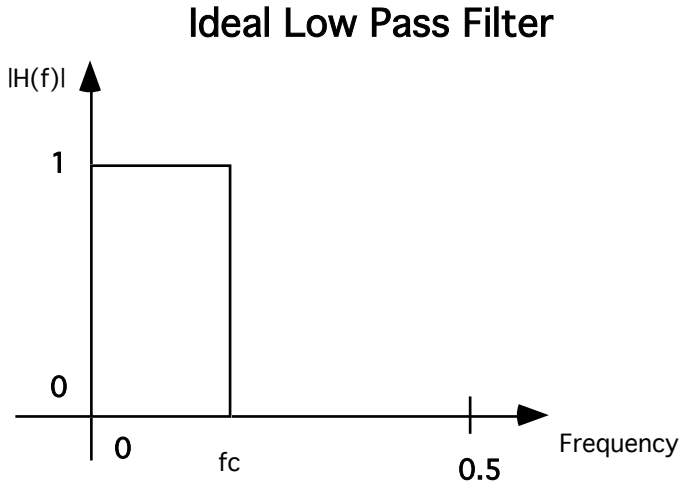
Generalized Hamming Window

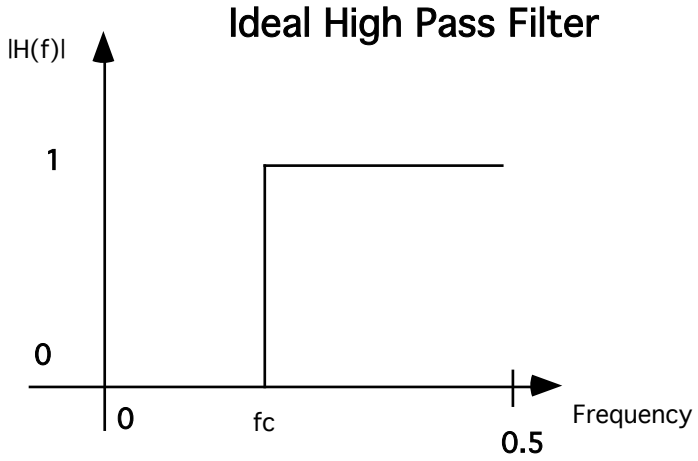
$$w_H(n) = \begin{cases} \alpha + (1-\alpha)\cos(\frac{2\pi n}{N-1}) & -(\frac{N-1}{2}) \le n \le \frac{N-1}{2} \quad N \text{ odd} \\ \alpha + (1-\alpha)\cos(\frac{2\pi(2n+1)}{2(N-1)}) & -(N/2) \le n \le (N/2-1) \quad N \text{ even} \end{cases}$$

Kaiser Window

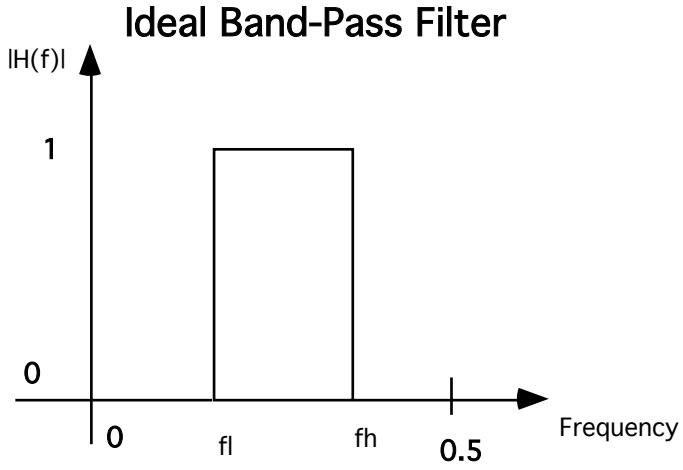
$$w_K(n) = \begin{cases} \dfrac{I_0\left(\beta\sqrt{1-\left[\dfrac{4n^2}{(N-1)^2}\right]}\right)}{I_0(\beta)} & -\left(\dfrac{N-1}{2}\right) \le n \le \dfrac{N-1}{2} \text{ N odd} \\ \dfrac{I_0\left(\beta\sqrt{1-\left[\dfrac{4(n+1/2)^2}{(N-1)^2}\right]}\right)}{I_0(\beta)} & -\left(\dfrac{N}{2}\right) \le n \le \dfrac{N}{2} - 1 \text{ N even} \end{cases}$$

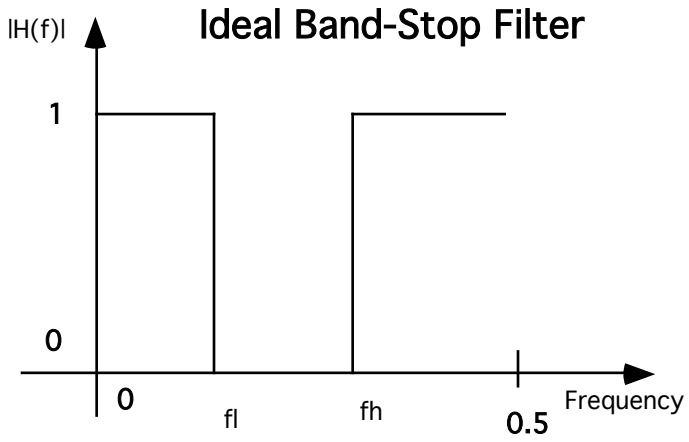
Filter Types





Definition of normalized cutoff frequency for low pass and high pass filters.

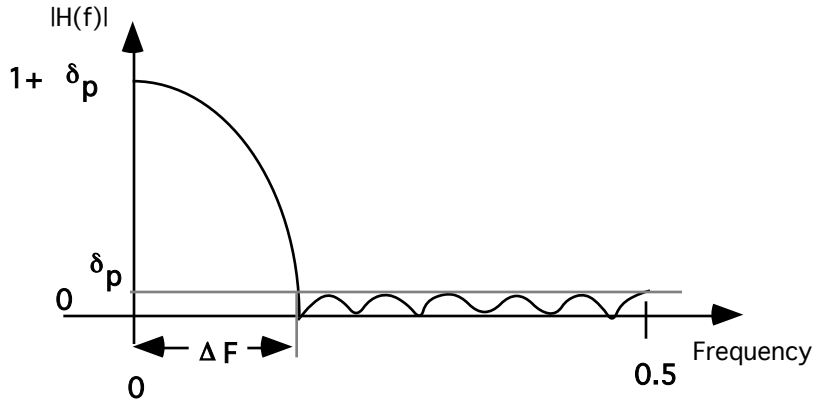




Definition of normalized cutoff frequencies, FL and FH for bandpass and bandstop filters.

Chebyshev Window

w(n) is obtained as the inverse DFT of the Chebyshev polynomial, evaluated at N equally spaced frequencies arouund the unit circle. The parameters of the Chebyshev window are the ripple, d_p, the filter length, N, and the normalized transition width, DF. The Figure below shows a plot of the frequency respomnse of a Chebyshev window illustrating how d_p and DF are measured.



Definition of Chebyshev window parameters in the frequency domain.

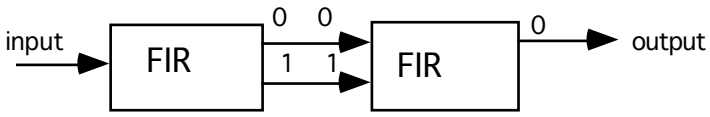
Only 2 of the 3 parameters N, d_p and ΔF can be independently specified. The block computes the 3rd parameter based on the other 2 parameters.

fir

Description

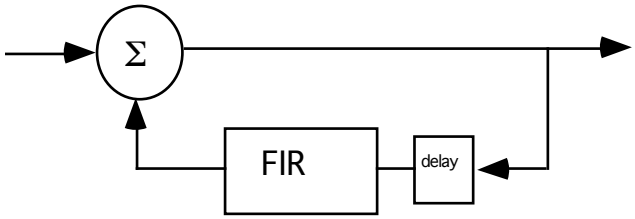
This block outputs a weighted sum of delayed input data.
Parameter one is an array for filter weights, 10 maximum.
Any number of these blocks can be cascaded to implement longer filters.
Connect them like this:

CASCADE



(Input and output buffers are assigned automatically.)
To implement an IIR filter with an fir in a feedback loop,
include a unit delay:

IIR



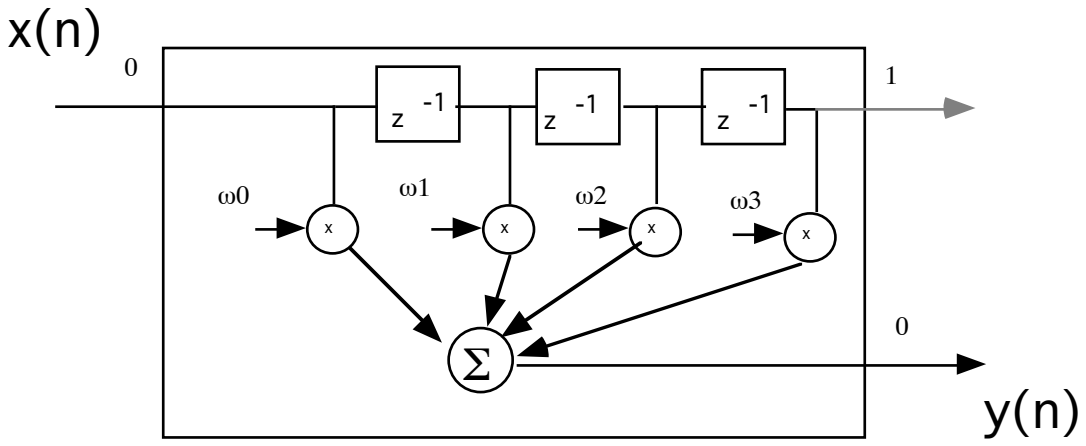
Programmer: L.J. Faber
Date: April 21, 1988.

Parameters:
(1) Array of FIR weights**.**
array weights;

Buffer:

Inputs:
Buffer 0: Samples (float).
Outputs:
Buffer 0: Filtered samples (float).
Buffer 1: Optional, delayed samples for cascading (float).

Graphic


x(n)
0
z -1
z -1
z -1
1
ω0
ω1
ω2
ω3
x
x
x
x
Σ
0
y(n)

convolve

Description

This block convolves the input samples with the impulse response (finite duration, FIR) given in a file.

$$y(n) = \sum_{i=0}^{N-1} x(n-i)*w_i$$

where w_i i=0,...,N-1 are the impulse response samples. N is the length of the impulse response.

Programmer: Adali Tulay
Date: September 23, 1988

Parameters:

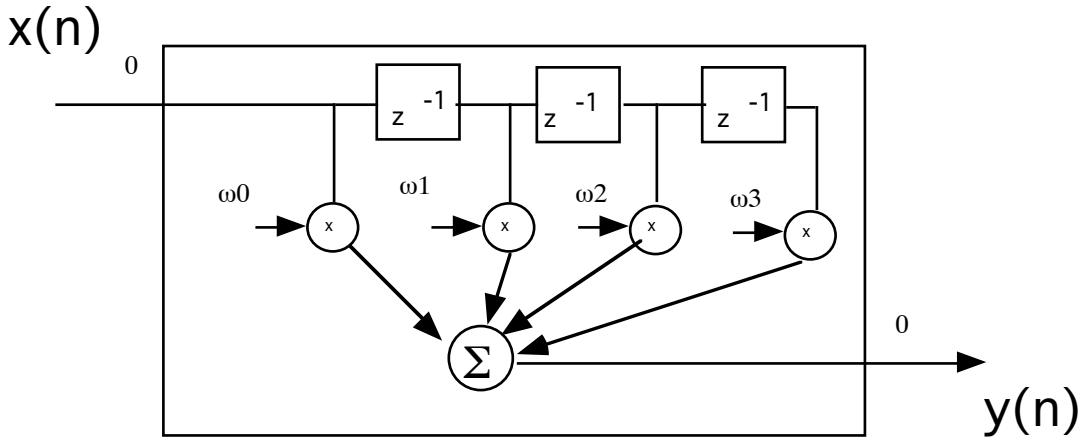
(1) File with impulse response.
file filename="imp.dat";
(2) Number of samples in impulse response.
int N ;

Buffer:

Inputs:
Buffer 0 : Input samples to convolve.(float)

Outputs:
Buffer1: Convolved output samples. (float)

Graphic


x(n)
0
z -1
z -1
z -1
ω0
ω1
ω2
ω3
x
x
x
x
Σ
0
y(n)

lconv/fconv

Description

"Linear Convolution":
This block convolves its input signal with an impulse response to generate the output signal.
Param. 1 - (int) impl: length of impulse response in samples.
2 - (file) impf_name: ASCII file which holds impulse response.
3 - (int) fftexp: log2(fft length).

Convolution is performed by the fft overlap-save method (described in Oppenheim & Schafer, Digital Signal Processing, pp. 113).

The FFT length must be greater than the impulse response length. For efficiency, it should probably be more than twice as long.

Programmer: M. R. Civanlar
Date: November 16, 1986
Modified: ljfaber, Dec86, Feb87
Modified: 6/88 ljfaber update comments, efficiency

Parameters:

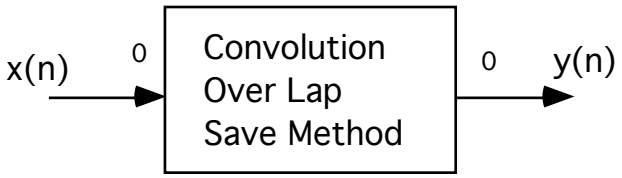
(1) Length of impulse response in samples.
int impl;
(2) ASCII file which holds impulse response.
file impf_name = "imp.dat";
(3) log2(fft length).
int fftexp;

Buffer:

Inputs:
Buffer 0: Samples to be filtered.

Outputs:
Buffer 0: Filtered samples.

Graphic


x(n)
0
Convolution
Over Lap
Save Method
0
y(n)

dfiltfreq

Description

"data filter, frequency points" This block performs general filtering for a data transmission channel, given frequency response data. The frequency data is contained in a file named by the first parameter. The file is assumed to be stored in symmetric, rfft format! The second parameter specifies the sampling frequency, "f sub s", expressed as a multiple of the system baud rate, eg. "8". The third parameter specifies the number of freq. data points. The input data oversampling rate (relative to the baud rate) is specified by the fourth input parameter `smplbd' (integer). The fifth parameter `fxp' specifies the length of the FFT to be used, with 2^fxp the FFT length. The channel impulse response is truncated at 16 baud intervals. This implies that FFT length must be > (16*smplbld) !

Programmer: L.J. Faber
Date: Feb. 1987

Parameters:

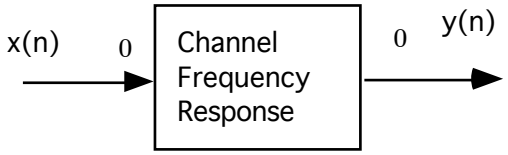
(1) File containing channel frequency response data (complex compact format).
file freq_fname = "freq.dat"
(2) Sampling rate as a multiple of baud rate.
float fmax = 8.;
(3) The number of frequency data points.
int fpts = 1024;
(4) Input data oversampling ratio relative to baud rate.
int smplbd = 8;
(5) Length of FFT to be used. FFT length = 2^fxp.
int fxp = 9;

Buffer:

Inputs:
Buffer 0: Samples into channel. (float)

Outputs:
Buffer 0: Channel response samples. (float)

Graphic



casfil

Description

Block implements a cascade form IIR digital filter.
Parameter: (file) File with the filter coefficients and parameters
The inputs from the file are as follows;

ns: Number of sections
zc1[i] zc2[i] i=1 to ns the numerator coefficients
pc1[i] pc2[i] i=1 to ns the denominator coefficients
in the Z-domain.
Normalization factor.

For each section:
y(n) = x(n) + zc1*x(n-1) + zc2*x(n-2) - pc1*y(n-1) -pc2*y(n-2)

Programmer: Tulay Adali
Modified: Sasan Ardalan
Date: October 15, 1988

Parameters:

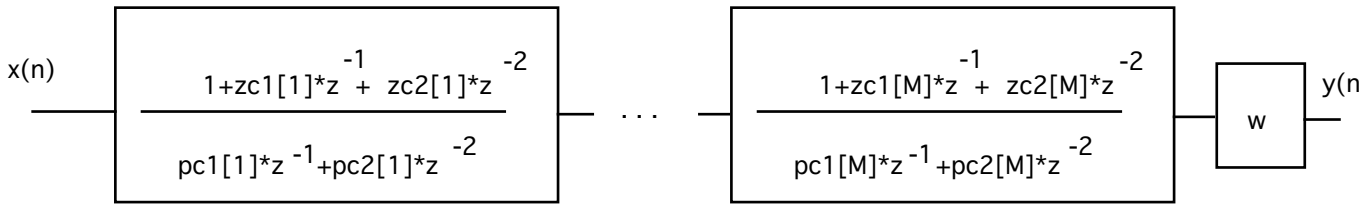
(1) File containing coefficients and normalization factor:

file filename="tmp.cas";

Buffer:

Inputs: Samples to be filtered (float).
Outputs: Filtered samples (float).

Graphic



bpf

Description

Block implements a simple band pass IIR digital filter. Zero response at DC and fs/2.

Programmer: Tulay ADALI
Date: November 23, 1988

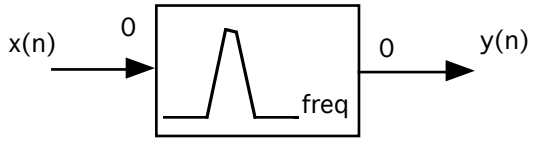
Parameters:

(1) fs: sampling frequency
float fs
(2) freq: resonant frequency
float freq
(3) ro: magnitude of the pole (less than one, for stability) (Q increases when magnitude of the pole approaches to the unit circle, i.e. when ro is close to one.)
float ro

Buffer:

Inputs:
Buffer0-> input samples (float)
Outputs:
Buffer0-> output samples (float)

Graphic



lpf

Description

This block implements an IIR filter as a recursive equation:
y(n)=pole*y(n-1)+(1-pole)*x(n)
This implementation produces unity gain at DC.

Programmer: John T. Stonick

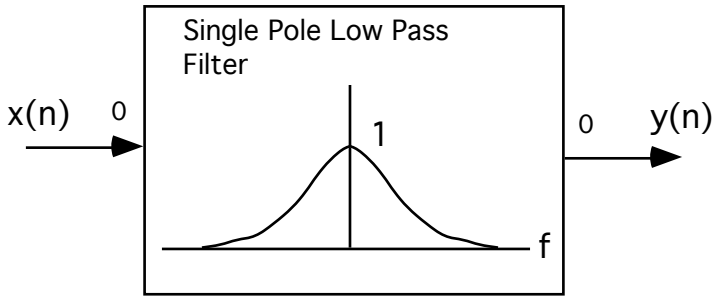
Parameters:

(1) Filter pole <= 1
float pole=.9;

Buffer:

Inputs:
Buffer 0: Samples to be filtered.
Outputs:
Buffer 0: Filtered samples.
delay_max=2;

Graphic



intdmp

Description

This block performs an integrate and dump. Therefore, the sampling rate is reduced by the number of samples per symbol.

parameters

(1) Integration time (symbol time in samples)

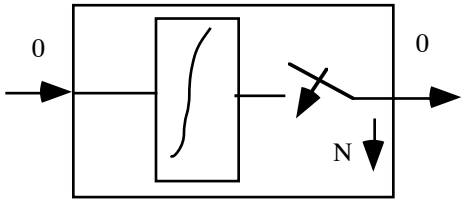
int dmpTime=8

Buffer:

Inputs:
Buffer 0 : Samples.(float)

Outputs:
Buffer 0: Filtered samples.(float) Decimated by integration time.

Graphic



integrate

This block implements a leaky integrator
recursive equation:
y(n)=factor*y(n-1)+x(n)
set factor =1 to integrate.

parameters

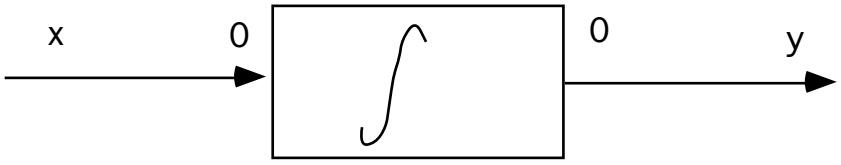
(1) leakage factor <= 1.0
float factor= 1.0;

Buffers:

Inputs:
Buffer 0 : Samples.(float)

Outputs:
delay_max=2;
Buffer 0: Integrated samples.(float)

Graphic



filtnyq/sqrnyq

Description

This block performs Nyquist pulse shaping for a baseband transmitter. See Carlson, Communications Systems, page 381, equation 17b. The Nyquist criterion in the frequency domain is to have an amplitude rolloff which is symmetric about Fb/2 (half baud frequency). First, a frequency-domain amplitude response is created using a raised cosine shape. This computation is affected by:
Param: 1 - (int) smplbd: samples per baud interval. default=>8
2 - (int) expfft: 2^expfft = fft length to use. default=>8
3 - (float) beta: filter rolloff factor, 0<beta<=.5 default=>.5
The amplitude response is changed to impulse response via inverse fft. The impulse response is made causal by right shifting (filter delay), and is time limited to "IMPBAUD" baud intervals (set by definition). (This filter will cause a delay of IMPBAUD/2 baud intervals.) Finally, the impulse response is transformed back to a frequency response, which is used in subsequent linear convolution with the input, which is implemented by the Fast Fourier Transform overlap-save method.

The fft length must be greater than the impulse response length; for efficiency, a factor of two or more in length is desirable. This implies that 2^expfft > smplbd * IMPBAUD. Nyquist shaping has no meaning if smplbd = 1; this implies that each sample would go through the filter unchanged!

Programmer: L.J. Faber
Date: Jan. 14, 1987

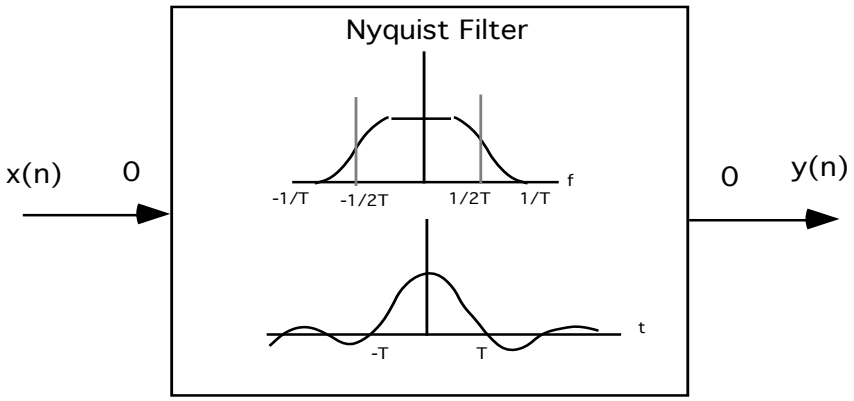
Parameters:

(1) Samples per baud interval.
int smplbd = 8;
(2) 2^expfft = fft length to use.
int expfft = 8;
(3) Filter rolloff factor, 0<beta<=.5.
float beta = .5;

Buffer:

Inputs:
Buffer 0 : Samples.(float)
Outputs:
Buffer 0: Filtered samples.(float)

Graphic



nl

Description

Normalized Lattice Filter

Programmer: Sasan H Ardalan
Date: August 22,1987

Parameters:
(1) File with normalized lattice parameters.
file file_name = "stdin";

Buffer:

Inputs:
Buffer 0: Input samples to be filtered.(float)
Outputs:
Buffer 0: Filtered samples.(float)

Errors

Graphic

hilbert

Description

"Discrete Hilbert Transform":
Reference:
A.V. Oppenheim amd R. W. Schafer,"Digital Signal Processing",pp 360-363,Prentice-Hall,1974 A Blackman window is used in the design. It then uses the overlap save method for fast convolution to model the hilbert transform impulse response.

Notes:
(1) The larger the length, the better the approximation. However, the simulation efficiency suffers considerably.
(2) This block introduces a delay of half the transform length.

Convolution is performed by the fft overlap-save method (described in Oppenheim & Schafer, Digital Signal Processing, pp. 113).

The FFT length must be greater than the impulse response length.
For efficiency, it should probably be more than twice as long.

Programmer: Sasan H. Ardalan, Overlap-save method by M. R. Civanlar
Date: July 26, 1990
Modified by Ray Kassel.

Parameters

(1) length of Hilbert transform impulse response
int impl =17;

(2) bandwidth (0 < BW <= 0.5)
float fbw = 0.5;

(3) log2(fft length)
int fftexp = 8;

Buffer:

Inputs:
Buffer 0: Input samples to be filtered.(float)
Outputs:
Buffer 0: Filtered samples.(float)

Probes

eye

Description

This routine generates the plot of an eye diagram, which is used in the analysis of data communications channels. The input data should be a symmetric, sampled analog waveform. The program overlays 200 baud intervals, starting from baud 10. The user must enter the (integer) oversampling rate (samples per baud); any integer rate up to 32 is supported. The user also enters the baud sampling phase. This phase (float) defines the sampling time (center of the eye) relative to the input data index 0. The eye diagram is always displayed from -.75 to .75 baud intervals. (This blocks accepts only one input buffer).

Note: If capsim is run in non graphic mode, this block will produce a file called *title.eye* where *title* is set by the title parameter.

Programmer: Jim Faber,R.A. Nobakht
Date: 9/16/87
Modified: ljfaber 1/3/89. add optional data flow-through.parameters

Parameters:

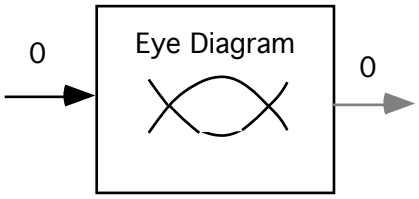
(1) Number of samples to collect for eye diagram (dynamic mode).
int numSamples = 200;
(2) Samples per baud interval.
int smplbd = 8;
(3) Baud sampling phase
float phase = 0.0;
(4) Samples to skip.
int skip = 0;
(5) Control: 1=On, 0 = Off..
int control= 1;
(4) Title.
file title= "eyeDiagram";
(5) 0=Static,1=Dynamic
int mode = 0;

Buffer:

Inputs:
Buffer 0: Samples. (float)

Outputs:
Buffer 0: Feed Through.

Graphic



more

Description

Function prints samples from an arbitrary number of input buffers to the terminal output using the "more" command (one sample from each input is printed on each line)

This routine does not print the time axis; to obtain the time axis, the user should connect the block time() to input #0

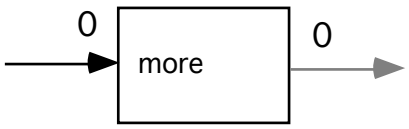
Programmer: D.G.Messerschmitt
Date: May 6, 1986

Parameters:

Buffer:

Inputs:
Auto fanin.
Outputs:
None.

Graphic



hist

Description

This program computes a histogram of the received data. For a large no. of data points this distribution should approach the probability dist. of the signal . Any samples outside the range are put in the appropriate outer-most bin.

Note: If capsim is run in non graphic mode, this block will produce a file called *title.his* where *title* is set by the title parameter.

Programmer: John T. Stonick
Date: January 1988
Modified: March 29, 1988

Parameters:
(1) Starting point of left most bin.
float start = 0.0;
(2) Ending point of right most bin.
float stop = 1.0;
(3) Number of bins.
int no_of_bins = 10;
(4) File name for output.
file file_spec = "none";
(5) Number of points to collect.
int npts=1000;
(6) X axis label.
file x_axis = "Bins ";
(7) Y axis label.
file x_axis = "Histogram";
(8) Plot Style: 1=Line,2=Points,5=Bar Chart.
int plotStyleParam=5;
(9) Control: 1=On, 0=Off
int control=1;
(10) Buffer Type: 0= float, 1=image
int bufferType=0;

Buffer:

Inputs:
Buffer 0: Samples. (float)

Outputs:
Feed Through

Graphic


0
Histogram
0

plot

Description

This routine will produce a set of plots from an arbitrary number of input channels. Optionally, the input channel data can 'flow through' to the correspondingly numbered output channel. This is useful if this block is to be placed in line in a simulation (e.g. probe).

Note: If capsim is run in non graphic mode, this block will produce a file called *title.tim* where *title* is set by the title parameter.

Programmer: Sasan Ardalan
Date: 8/16/87
Modified: L.J. Faber 1/3/89. Add flow through; general cleanup.

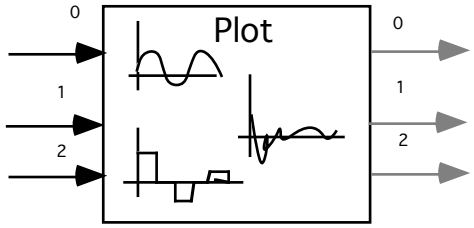
Parameters:
(1) Number of points in each plot (dynamic window size).
int npts = 100;
(2) Points to skip before first plot.
int skip = 0;
(3) Plot title.
file title = "PLOT";
(4) X-Axis label.
file x_axis = "X";
(5) Y-Axis label.
file y_axis = "Y";
(6) Plot Style: 1=Line,2=Points,5=Bar Chart.
int plotStyleParam=1;
(7) Control : 1= On, 0 = Off.
int control= 1;

(8) 0=Static,1=Dynamic
int mode = 0;
(9) Sampling Rate
int samplingRate = 1;

Buffer:

Inputs:
Auto fanin.
Outputs:
Feed Through

Graphic



cxplot

Description

inputs: complex

outputs: (optional feed-through of input channels)

This routine will produce a plot with the real and imaginary multiple with legends
Or, it will produce two plots, one for the magnitude and one for the phase.
Optionally, the input channel data can 'flow through' to the
correspondingly numbered output channel. This is useful if this block is to be placed in line in a simulation (e.g. probe).

Programmer: Sasan H. Ardalan
Date: August 6, 1991

Parameters

(1) Number of points in each plot
int npts = 128;
(2) Points to skip before first plot
int skip = 0;
(3) Plot title
file title = "ComplexPlot";
(4) X-Axis label
file x_axis = "X";
(5) Y-Axis label
file y_axis = "Y";
(6) Plot Style: 1=Color,2=Dashed,1=Same
int plotStyleParam=1;
(7) 0=Real/Imaginary 1=Mag/Phase Plot
int plotType=0;
(8) Control: 1=On, 0=Off
int control=1;

Buffers

Input: Buffer 0 x (complex)

Output: Buffer 0 (Optional Feed Thru) (complex)

multiplot

Description

Create a plot with overlayed curves where each curve is an input buffer. Also plot legends.

inputs: (arbitrary number)

outputs: (optional feed-through of input channels)

This routine will produce a plot with multiple curves with legends from an arbitrary number of input channels. Each curve is a channel.
Optionally, the input channel data can 'flow through' to the correspondingly numbered output channel. This is useful if this block is to be placed in line in a simulation (e.g. probe).

Programmer: Sasan H. Ardalan
Date: August 6, 1991

Parameters

(1) Number of points in each plot
int npts = 128;
(2) Points to skip before first plot
int skip = 0;
(3) Plot title
file title = "PLOT";
(4) X-Axis label
file x_axis = "X";
(5) Y-Axis label
file y_axis = "Y";
(6) Plot Style: 1=Color,2=Dashed,1=Same
int plotStyleParam=1;
(7) Control: 1=On, 0=Off
int control=1;

Buffers:

Inputs:
Auto fan-in (float)

Outputs:
Feed Through (Probe)

plt3d

Description

This is a probe block for displaying three dimensional data. It also supports contour plots. Input data is received one row at a time. The number of rows is height and the number of columns is width.

inputs: One channel

outputs: (optional feed-through of input channels)

Programmer: Sasan H. Ardalan
Date: June 18, 1991

Parameters

(1) Image width
int pwidth = 1;
(2) Image height
int pheight = 1;
(3) Title
file imageTitle = "3-D Plot";
(4) 0=Three Dim, 1= Contour Plot
int contourFlag=0;
(5) X-Axis Label
file x_axis = "X";
(6) Y-Axis Label
file y_axis = "Y";
(7) Z-Axis Label
file z_axis = "Z";
(8) 0=Auto Scale, 1=Fixed
int scaleFlag=0;
(9) zmin
float zmin1=0;
(10) zmax
float zmax1=256.0;
(11) zstep
float zstep1=50;
(12) x view
float xvu1=8.8;
(13) y view
float yvu1=3.66;
(14) z view

```
        float    zvu1=3.0;
(15) Contour label interval
        int      labelInt=1;
(16) Control: 1=On, 0=Off
        int      control=1;
(17) Visible Surface: 0=both,1=top,2=bottom
        int      which=0;
```

Buffers:

Input:
Buffer 0: each row starting from top row hieght number of rows, with each row containing width samples.(float)

Output:
Feed Thru

pltxyz

Description

This is a probe block for displaying irregular three dimensional data. It also supports contour plots and three dimensional curves.

x , y and z are input as three buffers.

inputs: 3 input buffers: x,y, and z channels

outputs: (optional feed-through of input channels)

Programmer: Sasan H. Ardalan
Date: June 18, 1991

Parameters

(1) Number of points
int npts = 128;
(2) Title
file imageTitle = "3-D Irregular Plot";
(3) X-Axis Label
file x_axis = "X";
(4) Y-Axis Label
file y_axis = "Y";
(5) Z-Axis Label
file z_axis = "Z";
(6) 0=Three Dim, 1= Contour Plot, 2= Curve
int contourFlag=0;
(7) 0=Auto Scale, 1=Fixed
int scaleFlag=0;
(8) x Grids
int numxGrids=50;
(9) y Grids
int numyGrids=50;
(10) zmin
float zmin1=0;
(11) zmax
float zmax1=256.0;
(12) zstep
float zstep1=50;

(13) x view
float xvu1=8.8;
(14) y view
float yvu1=3.66;
(15) z view
float zvu1=3.0;

(16) Contour label interval
int labelInt=1;
(17) Control: 1=On, 0=Off
int control=1;
(18) Visible Surface: 0=both,1=top,2=bottom
int which=0;

Buffers:

Input:
Buffer 0: x (float)
Buffer 1: y (float)
Buffer 2: z (float)

Output:
Feed Thru

prfile

Description

Prints samples from an arbitrary number of input buffers to a file, which is named as a parameter. If the file name is set to "stdout", or "stderr" the output goes to the terminal.

- A sample from each input is printed in columns on a single line.
  If printing to stdout, these are labeled with signal names.
- The printing function can be disabled without removing the block,
  via a control parameter.
- Data "flow-through" is implemented: if any outputs are connected,
  their values come from the correspondingly numbered input.
  (This feature is not affected by the control parameter.)
  (There cannot be more outputs than inputs.)

Programmer: L.J.Faber

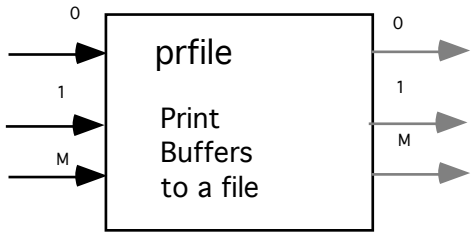
Parameters:

(1) Name of output file.
file file_name = "stdout";
(2) Print output control (0/Off, 1/On).
int control = 1;
(3) Buffer type:0= Float,1= Complex, 2=Integer
int bufferType=0;

Buffer:

Inputs:
Auto fanin.(float, complex , or integer)
Outputs:
Feed Through.(float, complex , or integer)

Graphic



spectrum

Description

This routine will produces the time domain and frequency domain spectrum of the input buffer. The spectrum is plotted until N/2 points where N is the number of points specified but reduced to the closest power of 2. To relate the sample numbers in the spectrum plot to frequency use the following expressions:

freq = fs * sampleNumber/N

and

sampleNumber = freq * N/fs

where fs is the sampling rate. SampleNumber is the sample number in the spectrum plot.

Note: If capsim is run in non graphic mode, this block will produce a file called *title.tim* and *title.fre* where *title* is set by the title parameter.

Programmer: Sasan Ardalan, Ramin Nobakht
Date: 2/16/89

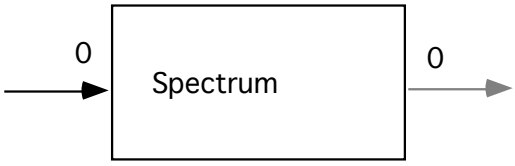
Parameters:

(1)) Number of points in each plot (dynamic window size).
int npts=128 ;
(2) Number of points to skip.
int skip=0 ;
(3) Plot title.
file title = "PLOT";
(4) Linear = 0, dB = 1.
int dbFlag = 0;
(5) Window: 0 = Rec., 1=Hamming.
int windFlag = 0;
(6) Plot Style: 1=Line,2=Points,5=Bar Chart.
int plotStyleParam=1;
(7) Time domain On/Off (1/0).
int timeFlag=1;
(8) Sampling rate (if 0 then x axis is in bins, if negaitive then x axis is normalized).
int sampFreq=0;
(9) Control : 1= On, 0 = Off.
int control= 1;
(10) Buffer type:0= Float,1= Complex, 2=Integer
int bufferType=0;
(11) 0=Static,1=Dynamic
int mode = 0;

Buffer:

Inputs:
Buffer 0: Samples to be analyzed.(float, complex, or integer)
Outputs:
Feed Through.

Graphic



scatter

Description

This routine will produce a scatter plot of the two input channels. If only one channel is connected, the block will set the Q channel to zero. Optionally, the input channel data can 'flow through' to the correspondingly numbered output channel. This is useful if this block is to be placed in line in a simulation (e.g. probe).

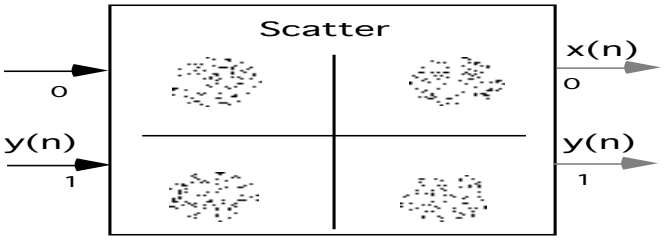
Note: If capsim is run in non-graphic mode, this block will produce a file called *title.sct* where *title* is set by the title parameter.

Programmer: Sasan Ardalan
Date: 8/16/87

Parameters:
(1)) Number of points in each plot (dynamic window size).
int npts = 100; / number of buffer points in each plot */*
(2) Number of points to skip.
int skip = 0; / points to skip before first plot */*
(3) Title.
file title = "PLOT";
(4) x Axis.
file x_axis = "X";
(5) y Axis.
file y_axis = "Y";
(6) Plot Style: 1=Line,2=Points,5=Bar Chart.
int plotStyleParam=2;
(7) Fixed Bounds (0=none, 1=fixed)
int fixed=0;
(8) Minimum x
float minx= -1.2;
(9) Maximum x
float maxx= 1.2;
(10) Minimum y
float miny= -1.2;
(11) Maximum y
float maxy= 1.2;
(12) Marker type:0=dot,1=O,2=+,3=X,4=*,5=square,6=diamond,7=triangle
int markerType=0;
(13) Control : 1= On, 0 = Off.
int control= 1;
(14) Buffer type:0= Float,1= Complex, 2=Integer
int bufferType=0;
(15) 0=Static,1=Dynamic
int mode = 0;

Buffer:
Inputs:
Buffer 0: I Channel (x) (float, complex, or integer see parameter 14)
Buffer 1: Q Channel (y)(float or integer) (may be omitted)
Outputs:
(optional feed-through of input channels)

Graphic

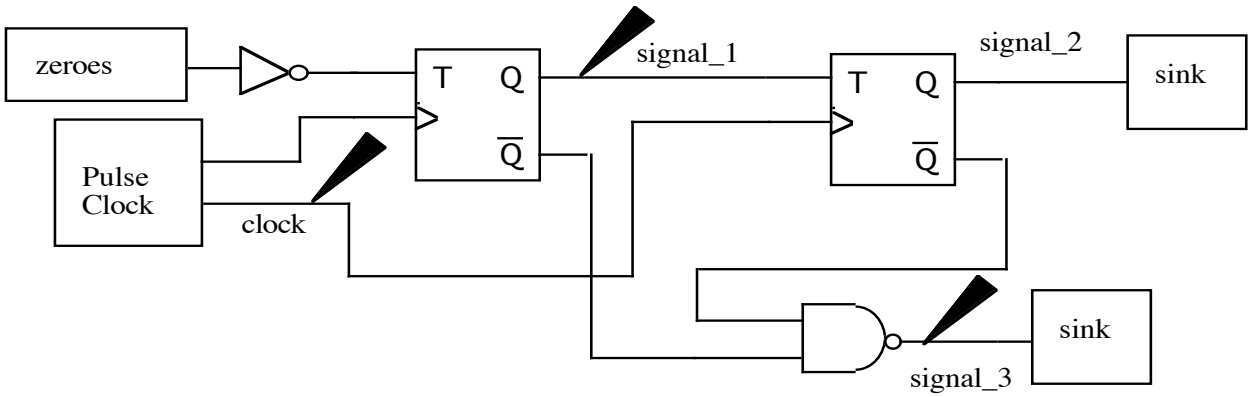


logican

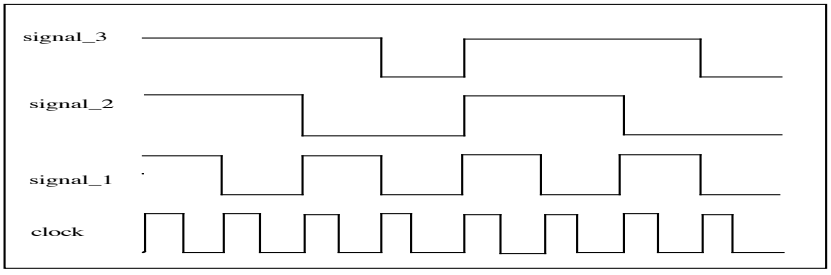
Description

This probe is used to generate traces in a logic analyzer for the signal for which the probe is connected. Each trace is labeled by the signal name. The signal name may be over written by specifying a graph title. If there are, say, 4 logican probes in a simulation, then four traces stacked on top of each other will be plotted. Capsim will automatically collect all traces from each probe and put them in a single plot .

Consider the following topology,



In this simulation the clock is implemented usinging a pulse generator with 128 samples, 16 samples per period and with a pulse width of 8 samples. The amplitude is set at 1. The probes are *logican* blocks. The simulation will produce the logic analyzer plot shown below. Note the signal names by each trace. To over ride these signal names, specify a title in your logican title parameter.



Plot produced by logican probes

Note 1: If capsim is run in non-graphic mode, this block will not produce any results.

Note 2: To display the plot you must click on the plot menu in the Capsim pulldown menu.

Programmer: Sasan Ardalan
Date: 11/16/90

Parameters:
(1) Number of points.
int npts = 100; / number of buffer points in each plot */*
(2) Number of points to skip.
int skip = 0; / points to skip before first plot */*
(3) Title.
file graphTitle = "LogicAnalyzer";
(4) y Axis.Label
file y_axis = "Y";

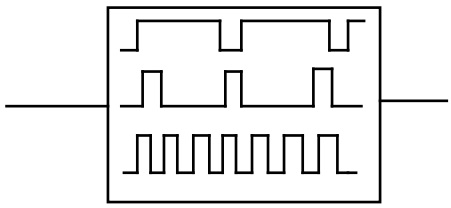
(5) DC Offset
float dcOffsets = 0;

(6) Gain applied after DC offset
float gain = 1.0;

(7) Control : 1= On, 0 = Off.
int control= 1;

Buffer:
Inputs:
Buffer 0: Input date (float)
Outputs:
(optional feed-through of input samples)

Graphic



image

Description

This probe collects pixels and displays the image using X windows. The displayed image is manipulated using qplot. A PostScript hardcopy can be obtained from the displayed image. The image probe determines the depth of the display (1 bit versus, say, 8 bit). Using the color pulldown menu of capsim, various RGB color tables can be loaded to view the image. Many images can be viewed simultaneously.

A gain, DC Offset and threshold may be applied to the pixels before display. If the threshold parameter is zero then it is ignored.

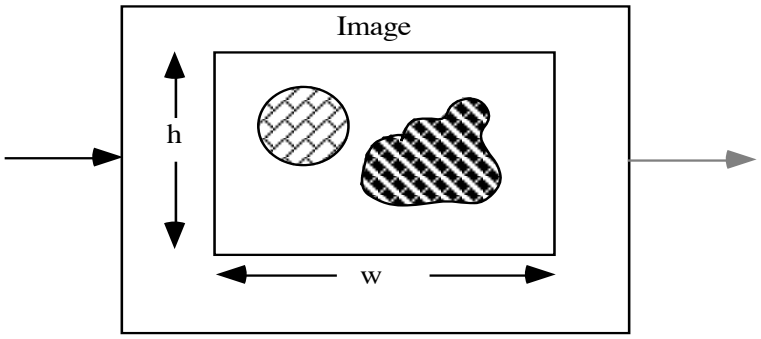
Note: If capsim is run in non graphic mode, this block will produce a file called *title.img* where *title* is set by the title parameter.

Programmer: Sasan Ardalan
Date: November 1990.

Parameters:
(1) Image width in pixels
int pwidth = 1;
(2) Image height in pixels.
int height = 1.0;
(3) Threshold (set to o if below, 1 if larger, used for 1 bit display). If 0 ignore.
float thresh=0;
(4) DC offset before applying threshold.
float dcOffset = 0;
(5) Gain, applied after adding DC offset.
float gain = 1;
(6) Image Title.
file imageTitle="Image"";
(6) X axis label.
file x_axis = " ";
(7) Y axis label.
file x_axis = "";
(8) Control: 1=on, 0=off.
int control=1;
Buffer:

Inputs:
Buffer 0: pixels. (float)
Outputs:
Feed Through

Graphic



imgdisp

Description

This probe collects pixels and displays the image using X windows. The displayed image is manipulated using qplot. A PostScript hardcopy can be obtained from the displayed image. The image probe determines the depth of the display (1 bit versus, say, 8 bit). Using the color pulldown menu of capsim, various RGB color tables can be loaded to view the image. Many images can be viewed simultaneously.

A gain, DC Offset and threshold may be applied to the pixels before display. If the threshold parameter is zero then it is ignored.

Note: If capsim is run in non graphic mode, this block will produce a file called *title.img* where *title* is set by the title parameter.

Programmer: Sasan Ardalan
Date: November 1990.

Parameters:

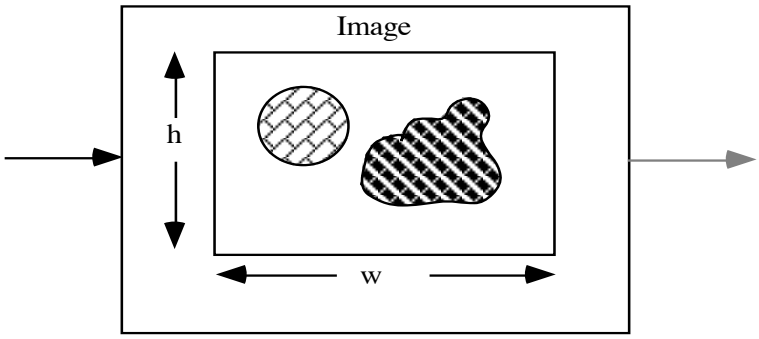
(1) Threshold
float thresh = 0;
(2) DC Offset applied before threshold
float dcOffset=0;
(3) Gain
float gain=1;
(4) Image title
file imageTitle = "Image";
(5) X-Axis label
file x_axis = "";
(6) Y-Axis label
file y_axis = "";
(7) Control: 1=On, 0=Off
int control=1;
(8) Display in same window (Animation) (1=TRUE)
int displaySequence=1;
(9) Fixed Bounds:0= None 1=Use
int fixedBoundsFlag=0;
(10) X Min
float xMin=0;
(11) X Max
float xMax=1;
(12) Y Min
float yMin=0;
(13) Y Max
float yMax=1;

(14) Enable Colormap Legend: 1=Enable, 0=Disable
int enableLegend=0;
(15) Legend Title
file legendTitle="Legend";
(16) Legend Min
float legendMin=0;
(17) Legend Max
float legendMax=256;
(18) Display Axis Flag: 0:Hide,1:Show
float axisDisplay=0;

Buffer:

Inputs:
Buffer 0: pixels. (float)
Outputs:
Feed Through

Graphic



primage

Description

Stores the input samples into a file organized as an image with specified width and height.

The file created by *primage* can be read using *rdimage*.

Programmer: Sasan Ardalan

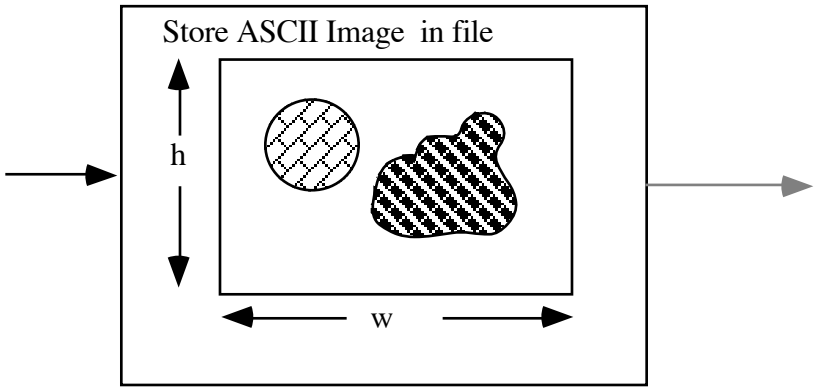
Parameters:

(1) Name of output file
file file_name = "stdout";
(2) Image Width
int width=1;
(3) Image Height
int height=1;

Buffer:

Inputs:
Auto fanin.
Outputs:
Feed Through.

Graphic



prbinimage

Description

Stores the input samples into a file organized as a binary image. Each input pixel is stored as a byte in the file.

The file created by *prbinimage* can be read using *rdbinimg*.

Programmer: Sasan Ardalan

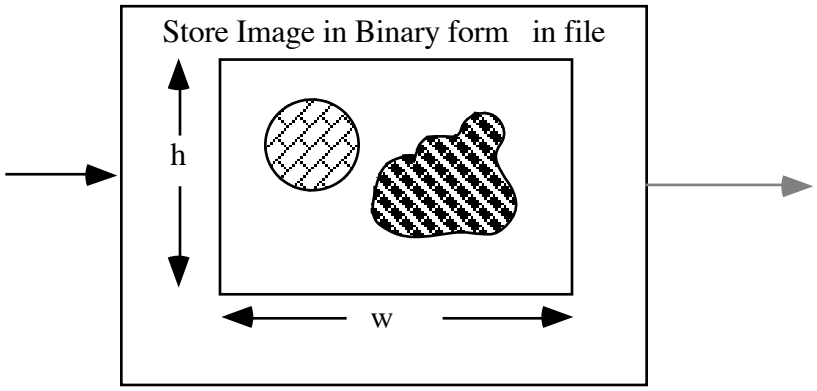
Parameters:

(1) Name of output file
file file_name = "stdout";
(2) Image Width
int width=1;
(3) Image Height
int height=1;

Buffer:

Inputs:
Auto fanin.
Outputs:
Feed Through.

Graphic



Complex Blocks and Vector Processing (FFTs)

cmxfft

Description

This block produces the FFT of the complex input signal.

Programmer: Prayson Pate, S. Ardalan
Date: April 15, 1988

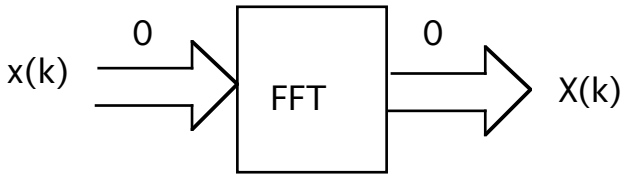
Parameters:
(1) The FFT length (prefer power of two but not necessary).
int npts=128.

Buffer:

Inputs:
Buffer 0: Complex input samples. (complex)

Outputs:
Buffer 0: Complex FFT of complex input samples. (complex)

Graphic



cmxifft

Description

This block produces the inverse FFT of the complex input signal.

Programmer: Prayson Pate, S. Ardalan
Date: April 15, 1988

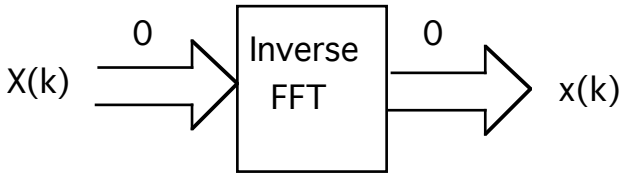
Parameters:
(1) The number of points (prefer power of two)
int npts=128.

Buffer:

Inputs:
Buffer 0: input samples. (complex)

Outputs:
Buffer 0: Complex FFT of complex input samples. (complex)

Graphic



cmxfftfile

Description

This block reads a file and computes its FFT during the initialization phase. (This produces H(k)). During execution, the block performs a complex multiplication of the FFT of the file with the input complex data blocks (The input fft, X(k)). It then outputs the complex result. This block multiplies the two complex data streams as follows: Each complex sample is assumed to be composed of a real sample followed by an imaginary sample. This block operates like a "butterfly," i.e.

c1 = a + jb = x1(0) + x1(1)
c2 = c + jd = x2(0) + x2(1)

r = c1 * c2 = (ac-bd) + j(bc+ad) = y(0) + y(1)

Inputs: The FFT of the signal to be filtered, X(k)

Outputs: The FFT of the impulse response (from a file) times the input signal, Y(k) = X(k)H(k)

Programmer: Prayson W. Pate,Sasan Ardalan
Date: March 12, 1989

Parameters:

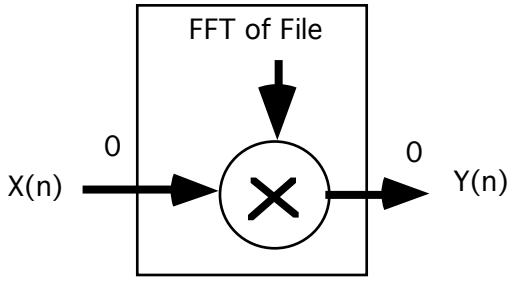
(1) Exponent of FFT length;
int fftexp=8;
(2) File name with impulse response;
file file_name = "imp.dat"

Buffer:

Inputs:
Buffer 0: Complex samples (complex)

Outputs:
Buffer 1: Complex samples (complex)

Graphic


FFT of File
0
X(n)
0
Y(n)

cxadd

Description

Function adds all its complex input samples to yield a complex output sample;
the number of input buffers is arbitrary and determined at run time.
The number of output buffers is also arbitrary (auto-fanout).

Programmer: D.G.Messerschmitt March 7, 1985
Modified: 1/89 ljfaber. add auto-fanout
Modified: 9/91 SHA complex buffers

Parameters

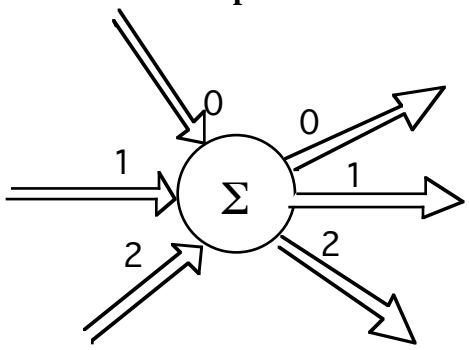
None

Buffers

inputs: auto fan-in (complex)

outputs: auto fan-out (complex)

Graphic



cxconj

Description

Function has a single complex input buffer, and outputs the conjugate of each complex input sample to an arbitrary number of complex output buffers.

Parameters

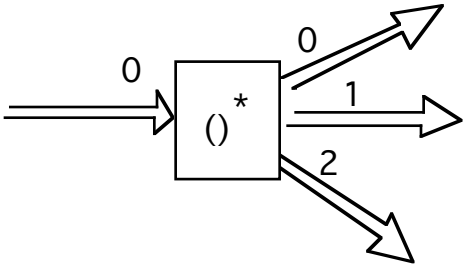
None

Buffers:

Input : x (complex)

Output: auto fan-out (complex)

Graphic



cxdelay

Description

Function delays its complex input samples by any number of samples, N.

Modified: April, 1988 L.J.Faber: add auto-fanout.
Modified: June, 1988 L.J.Faber: add default value; kludge fix for feedback problems...set delay_max.

Parameters:

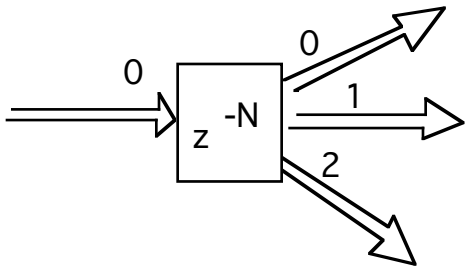
(1) Number of samples to delay.
int samples_delay = 1;

Buffer:

Inputs:
Buffer 0: Samples to delay.(complex)

Outputs:
Auto fanout (complex).

Graphic



cxgain

Description

This block multiplies the incoming complex data stream by the complex parameter coefficient, and outputs the resulting data values.
Auto fanout is supported.

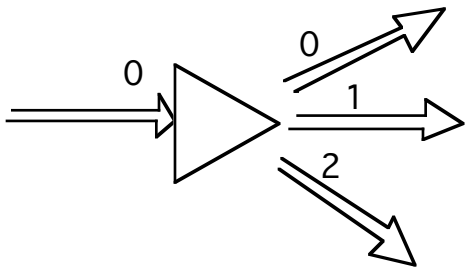
Parameters:
(1) Gain factor real part
float factorReal = 1.0;
(2) Gain factor imaginary part
float factorImag = 0.0;

Buffer:

Inputs:
Buffer 0: Samples (complex).

Outputs:
Auto fanout. (complex)

Graphic



cxmag

Description

This block finds the magnitude of a complex data stream. The magnitude is output as a real floating point sample.

Programmer: Prayson W. Pate
Date: April 18, 1988

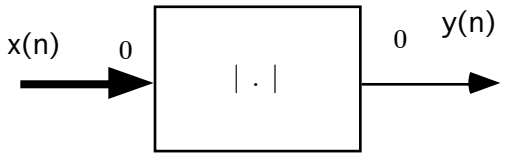
Parameters:

None

Buffer:

Inputs:
Buffer 0: Complex samples. (complex)
Outputs:
Buffer 0: Real magnitude samples. (float)

Graphic



cxmakecx

Description:

Inputs: one or two channels

Outputs: the complex channel

Parameters: None

This block creates a complex buffer from one or two input buffers. If one input buffer(buffer 0) is connected, it is assumed to be the real part. The imaginary part of the complex output is set to zero. If two input channels exist then the second channel (buffer 1) is assumed to be the imaginary part of the complex output sample.

Programmer: Sasan Ardalan
Date: September 4, 1991

Parameters

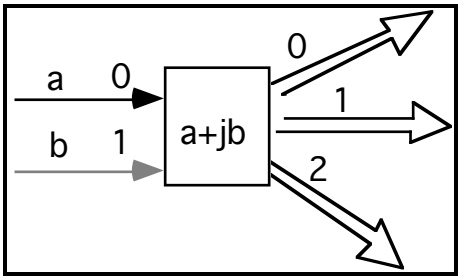
None

Buffers

Inputs: If one(float) then set imaginary part of output to zero.
If two (float) then real part of complex output is buffer 0 and imaginary part is buffer 1.

Outputs: auto fan-out (complex)

Graphic



cxmakereal

Description

Inputs: one complex channels

Outputs: two real channels for the real and imaginary parts

Parameters: None

This block creates a two real buffers from one complex input buffer.
If one output buffer(buffer 0) is connected, only the real part is output.
If two input channels exist then the second channel (buffer 1) is
the imaginary part of the complex input sample.

Programmer: Sasan Ardalan
Date: September 4, 1991

Parameters

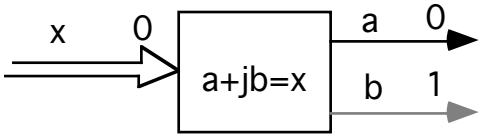
None

Buffers

Input: x (complex)

Output: Buffer 0, Real part of x (float)
Buffer 1 (Optional) iamginary part of x (float)

Graphic



cxmult

Description

Inputs: x1, the first complex signal
x2, the second complex signal

Outputs: y, the complex output

Parameters: none

This block multiplies two complex data streams. Each complex sample is assumed to be composed of a real sample followed by an imaginary sample. This block operates like a "butterfly," i.e.

c1 = a + jb = x1(0) + x1(1)
c2 = c + jd = x2(0) + x2(1)

r = c1 * c2 = (ac-bd) + j(bc+ad) = y(0) + y(1)

Programmer: Prayson W. Pate
Date: April 13, 1988
Modified: April, 1988

Parameters

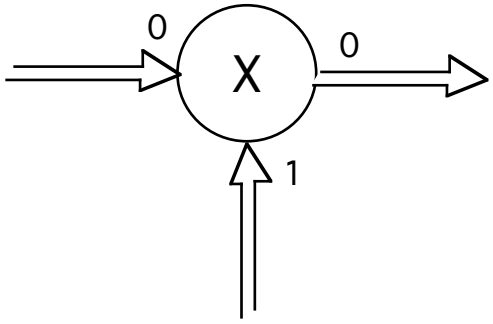
None

Buffers

Inputs: Buffer 0, x1 (complex)
Buffer 1, x2 (complex)

Output: Buffer 0 y=x1*x2 (complex)

Graphic



cxnode

Description

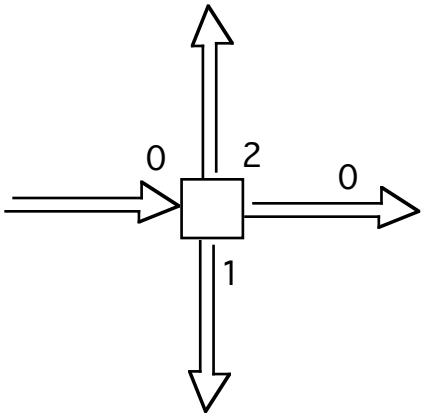
Function has a single complex input buffer, and outputs each complex input sample to an arbitrary number of complex output buffers.

Parameters:
None.

Buffer:

Inputs:
Buffer 0: Input to be forked out (complex).
Outputs:
Auto fanout(complex.

Graphic



cxphase

Description

This block finds the phase of a complex data stream. The phase is in degrees.

Programmer: Prayson W. Pate
Date: April 18, 1988

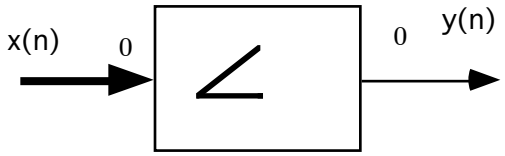
Parameters:
None

Buffer:

Inputs:
Buffer 0: Complex samples.(complex)

Outputs:
Buffer 1: Phase in degrees. (float)

Graphic



freqimp

Description

Calculate the impulse response from frequency response data.
The frequency response data is supplied as floating point data with the real part the even sample and the imaginary part as the odd sample.

The frequency data is assumed to be up to fs/2 where fs is the sampling rate. Also let the number of points be a power of two.

The impulse response will be real and will contain 2*npts samples.

Programmer: Sasan Ardalan
Date: Sometime in 1990

Parameters

(1) Number of frequency points (even odd pairs) to input
int nfft=64;
(2) Conjugatate (0=No, 1=Yes)
int conjFlag=0;

Buffers:

inputs:
Buffer0: xfreq (float)

outputs:
Buffer 0: ximp (float)

Adaptive Filters

predfft

Description

This block implements a multichannel input/output FIR predictor, which is adapted using the least squares Fast Transversal Filter algorithm. It can be used as an equalizer, FSE, DFE, or echo canceller. An arbitrary number p input channels are transversal filtered to produce an arbitrary number q output estimate signals.

Note: each output buffer connected to this block implies a separate output channel, and identically numbered error input channel. Input signal channels are then connected to higher numbered buffers. It is assumed that the estimate error is computed externally.

Do NOT implement an external (causality) unit-delay from output estimate to input error; this delay is handled automatically.

Param. 1 - Name of ASCII input specification file. Filter orders
and initial tap values are given. default => prfile

The proper specification file format is:

```
(int) # output channels, q
(int) # input channels, p

(int) order of in ch.#1  . . .  (int) order of in ch.#p

(float) ch.#1, tap 1 . . .  (float) ch.#1, tap last
   .                                {output ch.1}
   .
(float) ch.#p, tap 1 . . .  (float) ch.#p, tap last
 .
 .
 .
(float) ch.#1, tap 1 . . .  (float) ch.#1, tap last
   .                                {output ch.q}
   .
(float) ch.#p, tap 1 . . .  (float) ch.#p, tap last
```

If you have problems reading this file it is because of an improper number of initial tap values. For various channel orders and output channels the number of initial conditions is the number of output channels times the sum of the orders of the input channels. So specify a lot of zeroes!

Param. 2 - Name of output file, for final adapted filter values.
default => prfileo. The file is written in proper
input-file format. This file can then be used to initialize
the filter for the next run, if desired.

It is assumed that each output prediction filter will create one estimate output for EACH input sample/error sample pair. Any decimation, etc. must occur externally.

Param. 3 - (float) lambda. data forgetting factor. default => 1.0
Lambda = 1.0 implies no long term adaptation occurs.
4 - (float) delta. initial value, forward prediction energy.
default => 1e-4

5 - (int) wait. number of samples to skip before starting adaptation. The predictor still inputs samples, and outputs a zero estimate. default => 0
6 - (int) adapt. number of samples to adapt filter. After this number, filter taps are fixed, and estimates are still produced. default => -1 (implies always adapt)

Programmer: L.J. Faber
Date: April 1988
Modified: May 1988 add multichannel output
Modified: June 1988 estimate-referenced prediction energy
Modified: Aug 1988 est. input power. new parameter delta.
Modified: Sept 1988 add parameters 5,6 and associated.

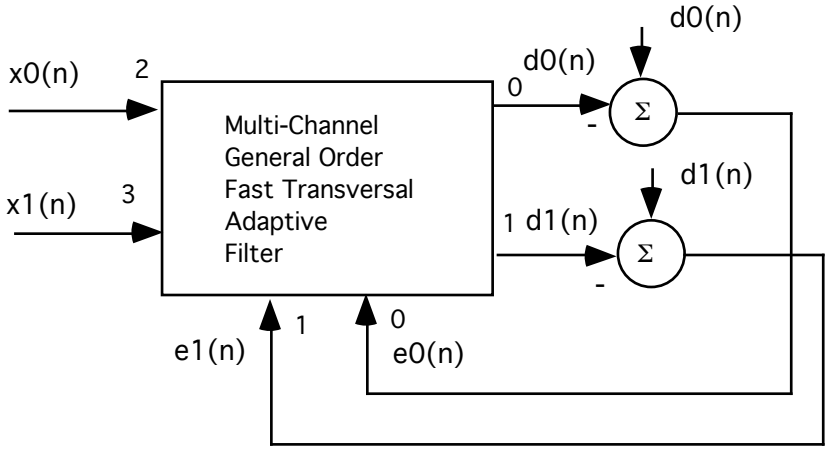
Parameters:

(1) Enter file name with order and initial weights spec.
file ifile_name = "prfile";
(2) Enter file name to store final weights and info.
file ofile_name = "prfileo";
(3) Enter forgetting factor <= 1.
float lambda = 1.0;
(4) Enter soft constraint << 1.
float delta = 1e-4;
(5) Enter number samples to wait before starting adaptation.
int wait = 0;
(6) Enter number of samples to stop adaptation and freeze.
int adapt = -1;

Buffer:

Inputs:
Auto fanin. See description (float).
Outputs:
Auto fanout. See description (float).

Graphic


d0(n)
x0(n)
2
d0(n)
0
Multi-Channel
General Order
Fast Transversal
Adaptive
Filter
Σ
-
d1(n)
x1(n)
3
1 d1(n)
Σ
-
1
0
e1(n)
e0(n)

predlms

Description

This block implements a multichannel input/output FIR predictor, which is adapted using the power normalized LMS algorithm. It can be used as an equalizer, FSE, DFE, or echo canceller. An arbitrary number p input channels are transversal filtered to produce an arbitrary number q output estimate signals.

Note: each output buffer connected to this block implies a separate output channel, and identically numbered error input channel. Input signal channels are then connected to higher numbered buffers. It is assumed that the estimate error is computed externally.

Do NOT implement an external (causality) unit-delay from
output estimate to input error; this delay is handled automatically.

Param. 1 - Name of ASCII input specification file. Filter orders and initial tap values are given. default => prfile

The proper specification file format is:
(int) # output channels, q
(int) # input channels, p

(int) order of in ch.#1 . . . (int) order of in ch.#p

(float) ch.#1, tap 1 . . . (float) ch.#1, tap last
. {output ch.1}
.
(float) ch.#p, tap 1 . . . (float) ch.#p, tap last
.
.
.
(float) ch.#1, tap 1 . . . (float) ch.#1, tap last
. {output ch.q}
.
(float) ch.#p, tap 1 . . . (float) ch.#p, tap last

Param. 2 - Name of output file, for final adapted filter values.
default => prfileo. The file is written in proper
input-file format. This file can then be used to initialize
the filter for the next run, if desired.

If you have problems reading this file it is because of an improper number of initial tap values. For various channel orders and output channels the number of initial conditions is the number of output channels times the sum of the orders of the input channels. So specify a lot of zeroes!

It is assumed that each output prediction filter will create one estimate output for EACH input sample/error sample pair. Any decimation, etc. must occur externally.

Param. 3 - (float) gamma. It is a multiplicative factor to control adaptation rate. default => 1.0
Param. 4 - (float) delta. Tap leakage factor. default => 1.0
Default implies no tap leakage occurs.

5 - (int) wait. number of samples to skip before starting adaptation. The predictor still inputs samples, and outputs a zero estimate. default => 0
6 - (int) adapt. number of samples to adapt filter. After this number, filter taps are fixed, and estimates are still produced. default => -1 (implies always adapt)

Programmer: L.J. Faber
Date: April 1988
Modified: May 1988 add multichannel output
Modified: June 1988 estimate-referenced prediction energy
Modified: Aug 1988 est. input power. new parameter delta.
Modified: Sept 1988 add parameters 5,6 and associated.

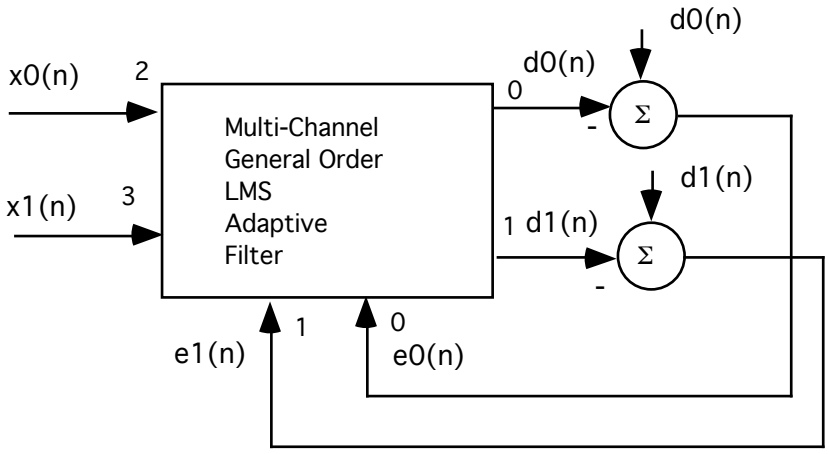
Parameters:

(1) Enter file name with order and initial weights spec.
file ifile_name = "prfile";
(2) Enter file name to store final weights and info.
file ofile_name = "prfileo";
(3) Enter LMS adaptation gain constant.
float gamma = 0.1;
(4) Enter tap leakage factor.
float delta = 1.0;
(5) Enter number of samples to skip before adaptation.
int wait = 0;
(6) Enter number of samples to stop adaptation and freeze.
int adapt = -1;

Buffer:

Inputs:
Auto fanin. See description.
Outputs:
Auto fanout. See description.

Graphic


d0(n)
x0(n)
2
d0(n)
0
Multi-Channel
General Order
LMS
Adaptive
Filter
Σ
-
d1(n)
x1(n)
3
1 d1(n)
Σ
-
1
0
e1(n)
e0(n)

lms

Description

This block implements a simple LMS adaptive filter. Below, x(n) is the input sample and d(n) is the desired response.

$\mathbf{x}(n) = [\ x_0(n)\ \ x_1(n)\ ...\ x_{N-1}(n)\ ]^T$

$\mathbf{w}(n) = [w_0(n)\ \ w_1(n)\ \ ...\ w_{N-1}(n)\ ]^T$

$dest(n) = \mathbf{x}^T(n)\mathbf{w}(n-1)$

$e(n) = d(n) - dest(n)$

$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\mathbf{x}(n)e(n)$

Programmer: Adali, Tulay
Date: September 23, 1988

parameters
(1) Filter order
int N = 10 ;
(2) LMS gain constant
float mu=0.1;

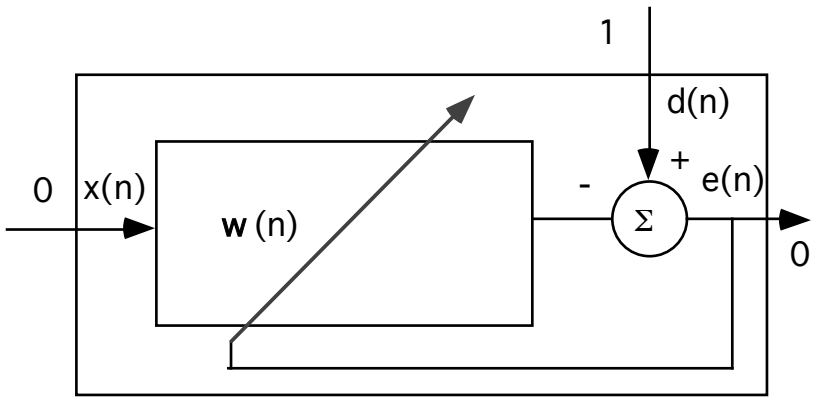
Buffers
Input Buffers

Buffer 0: input samples x(n) (float)
Buffer 1: desired response d(n) (float)

Output Buffer

Buffer 0: estimation dest(n) (float)

Graphic



Processing

autoxcorr

Description

If one input then compute autocorrelation.

If two inputs then compute crosscorrelation.

Both are computed in the frequency domain. Thus, this block operates on a vector of data.

Programmer: Sasan Ardalan
Date: Dec. 27, 1990

parameters

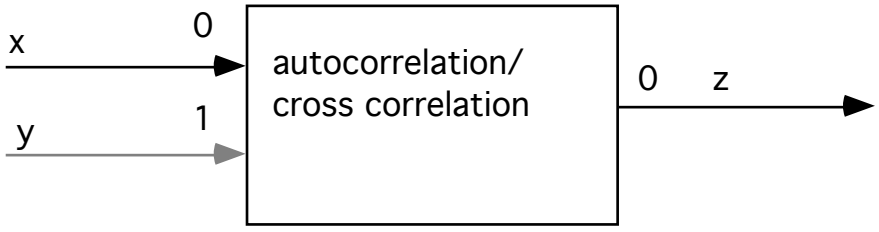
Number of samples
int npts=128;

Buffers

Inputs:
Buffer 0: x samples (float)
Buffer 1: y samples (float) optional. Connected only if cross correlation desired.

Outputs:
Buffer 0: auto(cross) correlation samples (float)

Graphic



autoeigen

Description

The block computes the autocorrelation matrix of the input data stream x and stores the eigenvalues and eigenvectors of the autocorrelation matrix in a file.

The autocorrelation matrix is: Rxx = E{ **x**(n) **x**T(n) } where
x(n) = [x(n) x(n-1) ... x(n-N+1)]T

The biased estimate #3, given in "Adaptive Filters and Equalisers"
(Mulgrew and Cowan ,page 27) is used for computing the autocorrelation function.

The eigenvalues and eigenvectors are computed using routines from "Numerical Recipes in C".

Programmers: Sasan Ardalan, Tulay Adali
Date: January 10, 1991

parameters

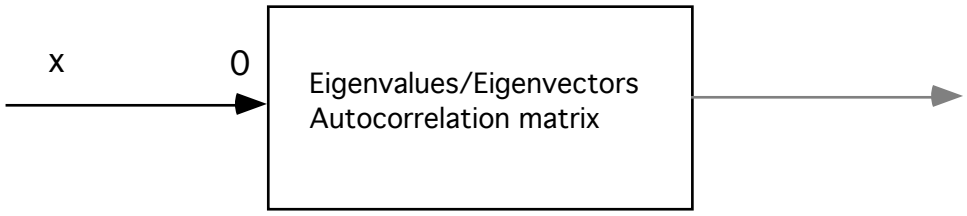
N : Size of the input vector (dimension of autocorrelation matrix)
int N=2 ;
K : Summation index
int K=100 ;
File to store eigenvalues,eigenvectors
file fileName="stdout";

Buffers

Inputs:
Buffer 0: x samples (float)

Outputs:
Auto feed through

Graphic


x
0
Eigenvalues/Eigenvectors
Autocorrelation matrix

divby2

Description

This block produces a square wave at half the frequency of the incomming wave.

Programmer: Sasan Ardalan

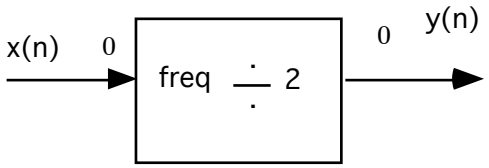
Parameters:
None

Buffer:

Inputs:
Buffer 0: Input signal. (float)

Outputs:
Buffer 0: Square wave at half frequency of input buffer 0.(float)

Graphic



ang

Description

parameters: none

inputs: x, numerator
y, denominator

outputs: z, atan(x/y)

description: This block find the inverse tangent of x/y

Parameters

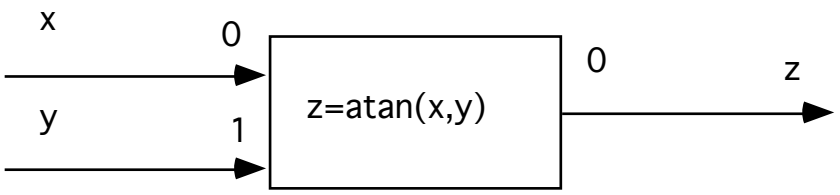
None

Buffers

Inputs: buffer 0 x (float)
buffer 1 y (float)

Outputs: buffer 0 z (float)

Graphic



scrambler

Description

This routine expects a sequence of input bits (0.0 or 1.0) and gives as an output another sequence of bits (also 0.0 or 1.0).

The scrambler and descrambler implemented here, which are self-synchronizing, are given in the CCITT recommendation V35.

The input parameters are:

mode: The operation done on the input sequence is either scrambling
(if mode = 0) or descrambling (if mode = 1).
seed: Can be used to force two scramblers out of phase
by choosing two values for the seed
Programmer: O. E. Agazzi / D.G.Messerschmitt
Date: March 31, 1982.
Converted to V2.0 by DGM March 11, 1985

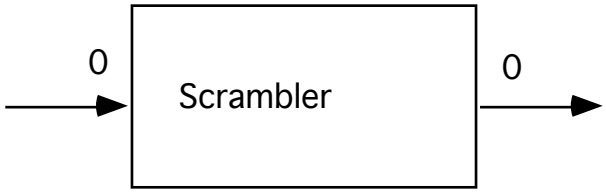
Parameters:

(1) Operation: 0=scrambling. 1=descrambling.
int mode;
(2) Shift register initial seed.
int seed = 12;

Buffer:

Inputs:
Buffer 0: Input bits.(float)
Outputs:
Buffer 0 : Scrambled bits.(float)

Graphic



sqr

Description

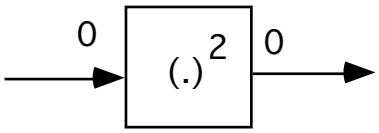
Square the input samples.

Parameters:
None.

Buffer:

Inputs:
Buffer 0: Input samples.
Outputs:
Buffer 0: Squared samples.

Graphic



cubepoly

Description

Implements a cubic polynomial nonlinearity of the form:
output = a*x + b*x^2 + c*x^3,
where x is the input
There are 3 parameters: a, b, c.

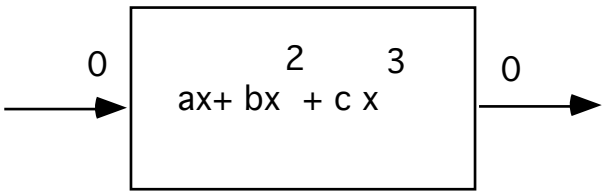
Parameters

(1) linear coefficient
float a;
(2) quadratic coefficient
float b;
(3) cubic coefficient
float c;

Buffer:

Inputs:
Buffer 0: Input samples (float).
Outputs:
Buffer 0: Output samples (float).

Graphic



limiter

Description

This block implements a hard limiter.

Programmer: Ray Kassel
Date: October 29,1990

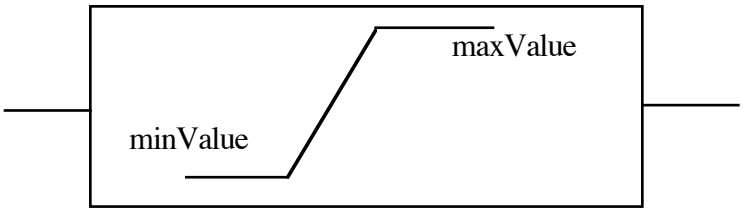
Parameters:

Minimum value.
float min=0.0;
Maximum value.
float max=1.0;

Buffer:

Inputs:
Buffer 0: Input signal.
Outputs:
Auto fanout.

Graphic



quot

Description

quotient

parameters: none

inputs: x, numerator
y, denominator

outputs: z, the quotient of x and y

This block takes two inputs and produces their quotient

Parameters

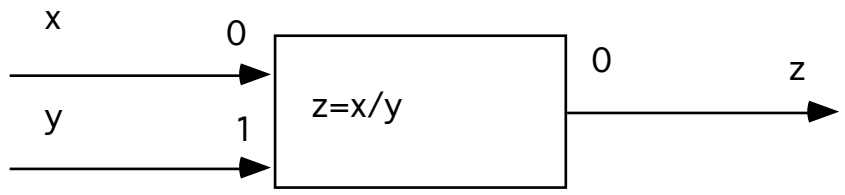
None

Buffers:

Inputs
Buffer 0: float x;
Buffer 1: float y;

Output
Buffer 0: float z;

Graphic



trig

Description

Perform various operations on the input data stream.

Programmer: Sasan Ardalan
Date: Dec. 29, 1990

Parameters

(1) Gain
float gain = 1.;
(2) DC Offset
float offset = 0.;
(3) Operation:0=sin,1=cos,2=x**2,3=sqrt(x), 4=10 log x**2";
int operation = 0;

Buffers:

Inputs:
Buffer 0: x (float)

Outputs:
Buffer 0: xmod (float)

Decimation/Interpolation/ Multiplexing

cmux

Description

This block ACTIVELY selects one input data channel to send to its output. Input channel 0 is always the control port; its input stream of numbers selects which input data channel to route to the output. The number N of input data channels is arbitrary (>= 1). In the event that the control port sample does not correspond to an attached input buffer number (i.e. 1-N), a zero sample is output. Since the control buffer is (float), rounding is used to generate an integer buffer number. Note: ALL input buffers are incremented (it_in) at each time, whether their sample is chosen for output or not.

Auto-fanout is supported.
There are no parameters.

Programmer: L.J. Faber
Date: April, 1988.

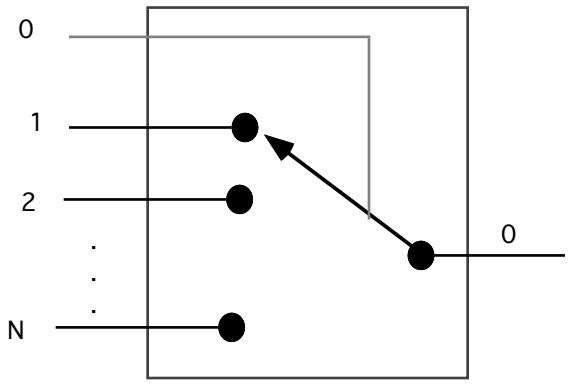
Parameters:
None

Buffer:

Inputs:
Buffer 0: Control port. Selects what input channel to output. (float)
Buffer 1,2,... N channels to output based on buffer 0. (float)

Outputs:
Auto fanout (float)

Graphic



resmpl

Description

This block performs interpolation or decimation on an input data stream, in order to change the output data rate. Polynomial or sinc interpolation is used to create output values that occur "between" input points. An initial time offset between the input/output streams can be entered.

Param. 1 - (float) ratio: output data rate/input data rate.
2 - (float) phi: delay of first output sample, relative to first input sample; expressed in units of input data period. Normally -1 < phi < 1. default: 0.
3 - (int) intype: type of interpolation:
0: sinc (3 point)
1: 1rst order (line) (default)
2: 2nd order (parabola)
3: 3rd order polynomial

Warning: although any output/input rate ratio > 0 will work, some spectral problems can occur. Time interpolation is not optimal, since there is no access to an infinite number of points! This problem is magnified as ratio deviates farther from unity. If ratio < 1 (decimation mode), aliasing can occur if the input signal is not properly bandlimited.

Programmer: L.J. Faber
Date: June, 1988.

Parameters:
(1) ratio: output data rate/input data rate.
float ratio = 1.;
(2) phi: delay of first output sample, relative to first input sample; expressed in units of input data period. Normally -1 < phi < 1. default: 0.
float phi = 0;
(3) Type of interpolation:
0: sinc (3 point)
1: 1rst order (line) (default)
2: 2nd order (parabola)
3: 3rd order polynomial
int intype = 1;

Buffer:
Inputs:
Buffer 0: Input samples. delay_max = 3;
Outputs:
Buffer 0 : Output samples.

Graphic


O
Resample
Change sampling rate
Interpolate/
Decimate
O

mux

Description

This block actively selects one input data channel to send to its output. Input channel 0 is always the control port; its input stream of numbers selects which input data channel to route to the output. The number N of input data channels is arbitrary (>= 1). In the event that the control port sample does not correspond to an attached input buffer number (i.e. 1-N), a zero sample is output. Since the control buffer is (float), rounding is used to generate an integer buffer number.
Note: ALL input buffers are incremented (it_in) at each time, whether their sample is chosen for output or not.

The output supports auto-fanout (automatic "forking").
There are no parameters.

Programmer: L.J. Faber
Date: April, 1988.

Parameters:

None.

Buffer:

Inputs:
Buffer 0: Control signal (float).
Buffer 1 through Buffer N (float).

Outputs:
Auto fanout (float).

demux

Description

This block provides periodic demultiplexing of an input data stream. It is appropriate for sub-sampling (integer decimation) or creating data streams for fractionally-spaced equalization (FSE). For every N (integer) input samples, 1 sample is sent to each output. The number of outputs and their phases are selectable.

Programmer: L.J. Faber
Date: April 1988

Parameters:

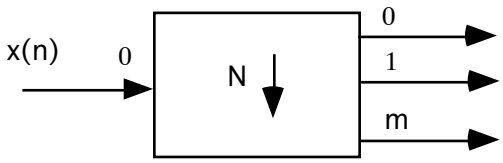
(1) Ratio input rate/output rate, N.
int N = 8;
(2) (array) specifies the "phase" (delay in samples relative to first input sample) for each output (10 Max). All phases must be non-negative.
array phases;

Buffers:

Inputs:
Buffer 0: Samples to be decimated.(float)

Outputs:
Auto fanout. Determined by number of elements in array. (float)

Graphic



toggle

Description

This block selects one of two input data channels to output. It always begins with channel 0. After a delay, it switches to ch.1. Parameter 1 selects the number of samples of ch.0 to output before switching to ch.1.

Note: Both input buffers are incremented (it_in) at each time, whether their sample is chosen for output or not.

The output supports auto-fanout (automatic "forking").

Programmer: L.J. Faber
Date: April, 1988.

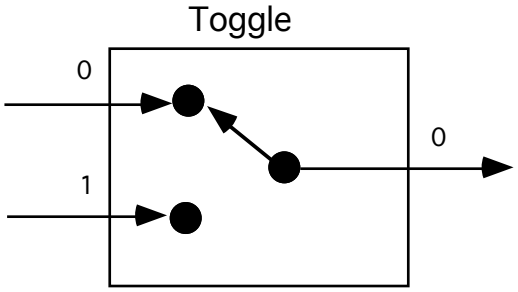
Parameters:

(1) Selects the number of samples of ch.0 to output before switching to ch.1.
int switch_time = 0;

Buffer:

Inputs:
Buffer 0: Channel 0.(float)
Buffer 1 : Channel 1.(float)
Outputs:
Buffer 0: Channel 0 then Channel 1.(float)

Graphic



hold

Description

This block simulates a sample and hold circuit. The lone parameter specifies the number of samples to hold the value. Triggering is on the positive edge of the clock.

Programmer: Sasan Ardalan
Date: February 22, 1988

Parameters:

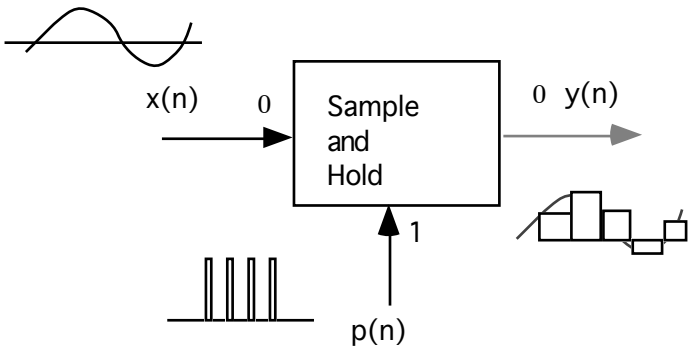
(1) Hold time in samples.
int holdTime = 1;

Buffer:

Inputs:
Buffer 0: Samples to be held.(float)
Buffer 1 : Clock signal. Hold on rising edge.(float)

Outputs:
Buffer 0: Sampled samples.(float)

Graphic



stcode

Description

This block inputs data and stretches it with zeros. The code output oversampling rate (samples per baud interval) is selected by the parameter `smplbd'.

Programmer: A. S. Sadri
Date: June 4, 1990

parameters

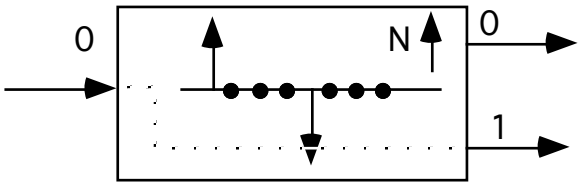
(1) Samples per baud
int smplbd = 8;

Buffer:

Inputs:
Buffer 0: Samples to be held.(float)
delay_max = 1;

Outputs:
Buffer 0: Stretched line code.(float)
Buffer 1: Input symbols at symbol rate.(float)

Graphic



Building Blocks

add

Description

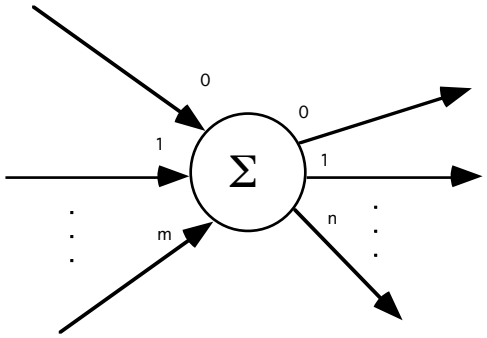
Function adds all its input samples to yield an output sample;
the number of input buffers is arbitrary and determined at run time.
The number of output buffers is also arbitrary (auto-fanout).

Programmer: D.G.Messerschmitt March 7, 1985
Modified: 1/89 ljfaber. add auto-fanout

Parameters:
None

Buffer:
Inputs: Auto fanin.
Outputs: Auto fanout

Graphic



delay

Description

Function delays its input samples by any number of samples, N.

Modified: April, 1988 L.J.Faber: add auto-fanout.
Modified: June, 1988 L.J.Faber: add default value; kludge fix for feedback problems...set delay_max.

Parameters:

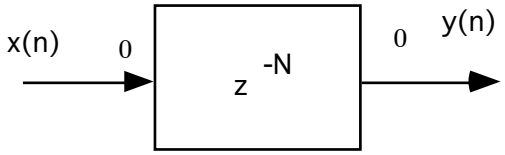
(1) Number of samples to delay.
int samples_delay = 1;

Buffer:

Inputs:
Buffer 0: Samples to delay.(float)

Outputs:
Auto fanout (float).

Graphic



gain

Description

This block multiplies the incoming data stream by the parameter coefficient, and outputs the resulting data values.
Auto fanout is supported.

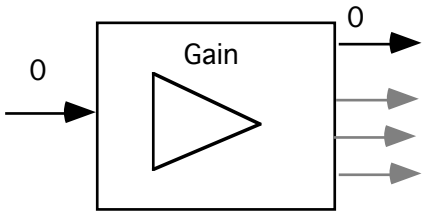
Parameters:
(1) Gain factor.
float factor = 1.0;

Buffer:

Inputs:
Buffer 0: Samples.

Outputs:
Auto fanout.

Graphic



mixer

Description

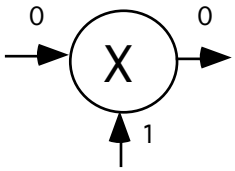
This block takes two inputs and produces their product.

Parameters:
None.

Buffer:

Inputs:
Buffer 0: Input Samples X. (float)
Buffer 1: Input Samples Y.(float)
Outputs:
Buffer 0: Product X*Y (float)

Graphic



multiply

Description

Function multiplies all its input samples to yield an output sample.

The number of input buffers is arbitrary and determined at run time (auto fan-in)

The number of output buffers is also arbitrary (auto-fanout).

Parameters

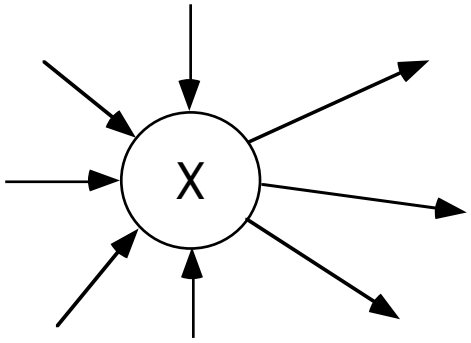
None

Buffers

Inputs: Auto fan-in (float)

Outputs: Auto fan-out (float)

Graphic



node

Description

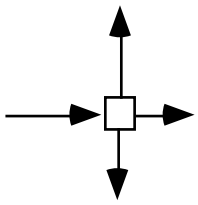
Function has a single input buffer, and outputs each input sample to an arbitrary number of output buffers.

Parameters:
None.

Buffer:

Inputs:
Buffer 0: Input to be forked out (float).
Outputs:
Auto fanout(float).

Graphic



sum

Description

This block is an extension of "add". It creates a Weighted Sum of all input channels and sends it to the output buffer(s). (This is convenient for negating inputs, for example.) Parameter one is an array for input channel weights.

The number of input buffers is determined at run time, 10 maximum.
The number of output buffers is determined at run time (auto-fanout).

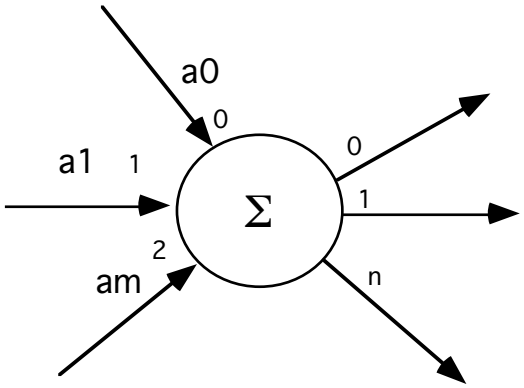
Programmer: L.J. Faber
Date: April 20, 1988.

Parameters:
(1) Array of weights.
array weights;

Buffer:

Inputs:
Automatic. Maximum 10.(float)
Outputs:
Auto fanout.

Graphic



operate

Description

This block performs a number of operations on its input samples x to produce the output samples y.

y= f[(x-DC)*gain]
where
f(x) = x,
f(x) = abs(x),
f(x) = x^2,
f(x) = sqrt(x),
f(x) = 10 log10(x*x).

Programmer: Sasan Ardalan
Date: January 10, 1991

parameters

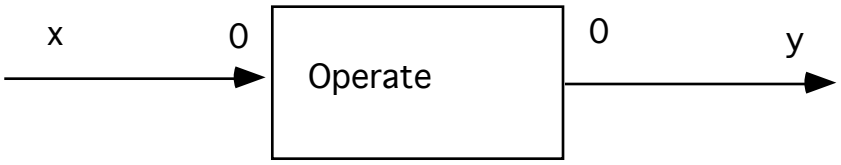
Number of samples to output
int N = 30000;
First sample to start from
int first = 0;
Gain
float gain = 1.;
DC Offset
float offset = 0.;
Operation:0=none,1=abs,2=square,3=sqrt,4=dB
int operation = 0;

Buffers

Inputs:
Buffer 0: x samples (float)

Outputs:
Buffer 0: y samples (float)

Graphic


x
0
Operate
0
y

Synchronization

loopfilt

Description

Loop filter for PLL

parameters: gain#1, gain#2, gain#3 of the filter

inputs: in, the signal to be filtered

outputs: out, the filtered signal

description: This block implements a loop filter as a recursive equation:

y(n)=2*y(n-1)-y(n-2)
+g1*[x(n)-2*x(n-1)+x(n-2)]
+g2*[x(n)-x(n-1)]
+g3*x(n)

This filter can be used as a PLL loop filter.

Programmer: Ray Kassel
August 9, 1990

Parameters

(1) Gain1
float g1=1.0;
(2) Gain2
float g2=0.0;
(3) Gain3
float g3=0.0;

Buffers:

Input : (float) x

Output:
delay_max = 3;
(float) y

pump

Description

Charge pump and loop filter for PLL

This block implements a charge pump loop filter as a recursive equation:
y(n)=y(n-1)+g1*[x(n)-x(n-1)]

This filter can be used as a PLL loop filter.

parameters: g1, integrate gain of the filter
vs, voltage step magnitude

inputs: up, of phase detector
down, of phase detector

outputs: out, the filtered signal

Parameters

(1) Integrate gain
float g1=1.0;
(2) Voltage step
float vs=1.0;

Buffers

Inputs: Buffer 0, delay_max = 2; up (float)
Buffer 1, delay_max = 2; down (float)

Outputs Buffer 0, delay_max = 2; y (float)

vcm

Description

voltage-controlled multivibrator

parameters: fs, sampling frequency
fo, center frequency

inputs: lambda, phase update term

output: square wave equivalent of cos(2*PI*fo*t+theta)

This block produces samples of the outputs every 1/fs seconds with frequency of fo. The phase is updated as theta=theta+lambda (integrates the input)

Parameters

(1) Sampling Rate
float fs=32000.;
(2) Center Frequency
float fo=1000.;

Buffers:

Inputs: Buffer 0 lambda(float)

Output: Buffer 0 data_out (float)

dco

Description

digitally controlled oscillator

This block produces samples of the outputs every 1/fs seconds. The dco behaves just like an FM modulator. The phase is updated as theta=theta+lambda (integrates the input)
$\theta(n) = \theta(n-1) + \lambda$

parameters:
fs, sampling frequency
fo, center frequency
A, amplitude

inputs: lambda, phase update term
outputs:
A*cos(2*PI*fo*t+theta)
A*sin(2*PI*fo*t+theta)

Programmer: **John T. Stonick**

Parameters:

(1) fs, sampling frequency.
float fs=1.;
(2) fo, center frequency.
float fo=1.;
(3) A, amplitude.
float A=1.;

Buffer:

Inputs:
Buffer 0: lambda; (float)
Outputs:
Buffer 0: In phase output (cosine); (float)
Buffer 1: Quadrature output (sine); (float)

Graphic


0
DCO
0
1

zc (zero crossing detector)

Description

This block generates impulses at the zero crossings of the input signal.

Written by: Sasan Ardalan, October 1989.

Parameters:

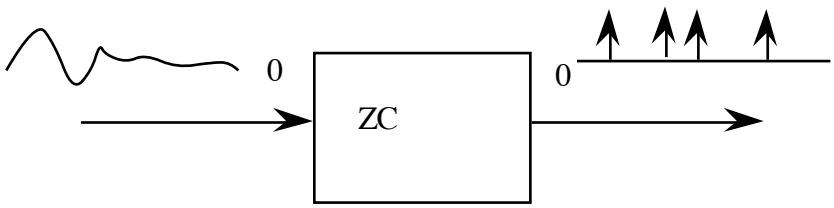
(1) Trigger edge: 1= Rising, 0 = Falling.
int edge = 1;

Buffer:

Inputs:
Buffer 0: Input signal(float).

Outputs:
Buffer 0: Impulses at zero crossing (float)

Graphic



Miscellaneous

null

Description

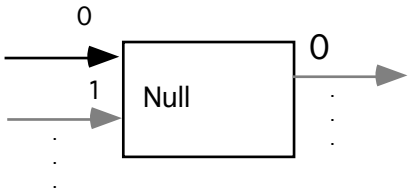
This block does nothing, simply puts its input samples on its output buffer. It is useful as a temporary substitute for a block.

Parameters:
None.

Buffer:

Inputs:
Auto fanin.
Outputs:
Auto fanout

Graphic



tee

Description

This block is a programmable tap-off for data lines. It can be used for example as a connector to print or plot blocks. It is inserted into a connection line between two blocks; input data flows to Output 0 unchanged. Output 1 is a modified version of the input:

Programmer: L.J. Faber
Date: May 1988

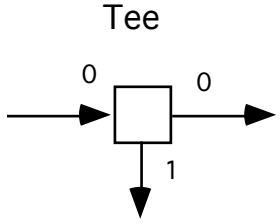
Parameters:
(1) Number samples to output; default => all
int N = 30000;
(2) Index first sample; default => start from first
int first = 0;
(3) Gain; default => unity gain
float gain = 1.;
(4) DC offset; default => no offset
float offset = 0.;

Buffer:

Inputs:
Buffer 0: Input samples.

Outputs:
Buffer 0: Feed Through samples.
Buffer 1: Tapped samples.

Graphic



skip

Description

This block eats the first N values, useful for skipping transient periods.

-Parameter one: the number of samples to skip

Programmer: John T. Stonick
Date: January 1988

Parameters

(1) Number of samples to skip
int no_skip=100;

Buffers:

Input: Buffer 0 : xin (float)

Output: Buffer 0: xout (float)

threshold

Description:

Compares input with threshold and outputs a 0 if less, and 1 if greater or equal Output occurs on positive edge of control input

parameters: threshold

inputs: x, signal to threshold
in_control, control signal

outputs: y, the threshold decision

This block compares the input with a threshold and outputs a 1 if the input is less than the threshold, or 0 if the input is greater than or equal to the threshold. Output occurs on positive edge of control input.

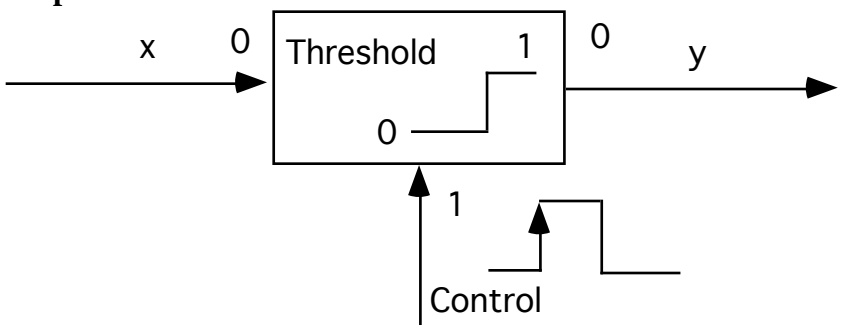
Parameters
(1) Threshold
float thh=0.0;

Buffers

Inputs: Buffer 0: delay_max = 2, x (float)
Buffer 1: delay_max = 2, in_control (float)

Output: Buffer 0 y(float)

Graphic



Channel Models

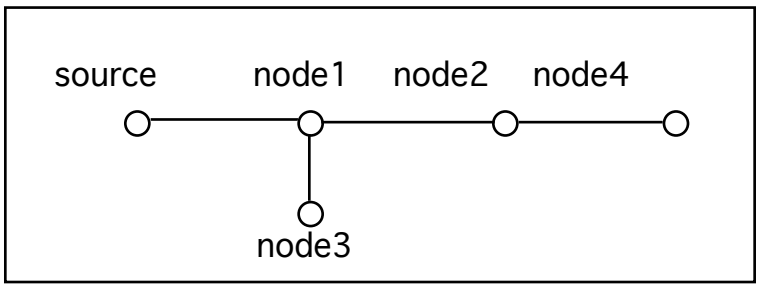
transline

Description

"Transmission Line Network Modeling":
This block produces the impulse response of a transmission line system. Depending on the *type* parameter, the impulse response for point to point nodes or the echo path impulse response is calculated. Once the impulse response is calculated, it then uses the overlap save method for fast convolution to model the network through the calculated impulse response.

The network is described by a toplogy file. An example is given below:

```
node1 node2 node3
node2 node4
node3
node4
node1        transmissionLineType        length
node2        transmissionLineType        length
node3        transmissionLineType        length    100     0
node4        transmissionLineType        length    open
```



A transmission line network is represented by a graph. The graph contains vertices or nodes and edges. The nodes hold information about the edges. Edges can be transmission lines or lumped elements. Nodes are either binary, such as *node1* in the example, or they are unary such as *node2* (a cascade node) or they are loads. Nodes *node4* and *node3* are loads. Therefore, load nodes contain besides the transmission line type and *length*, tinformation about the load. Only real loads or open circuits are allowed. For general loads see lumped elements below. *node3* is a 100 Ohm load. *node4* is an open circuit (bridged tap).

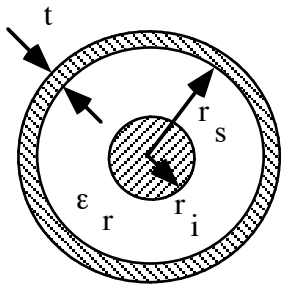
In the topology describing the graph, if a node is followed by two nodes it is a binary node (*node1*). If it is followed by a single node (*node2* above) it is a cascade node. If no

node follows the node name then the node is a load. The nodes *node3* and *node4* are loads. The source node is implied.

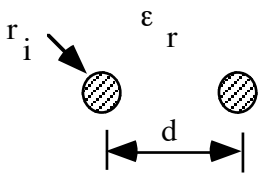
Note that the topology file consists of two distinct sections. The first section describes the node interconnection. The second section describes the node parameters such as transmission line type, lengths and load. All dimensions are in meters. Including the length.

Supported Transmission Line Types

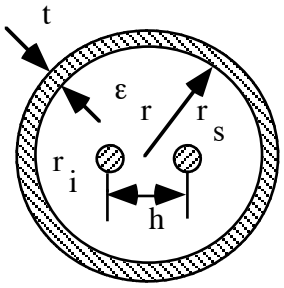
The transmission line types are specified in the file *trans_types.dat*. The following transmission lines are supported:



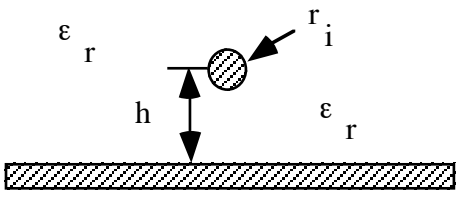
Coax



Parallel Conductors



Balanced Shielded Line



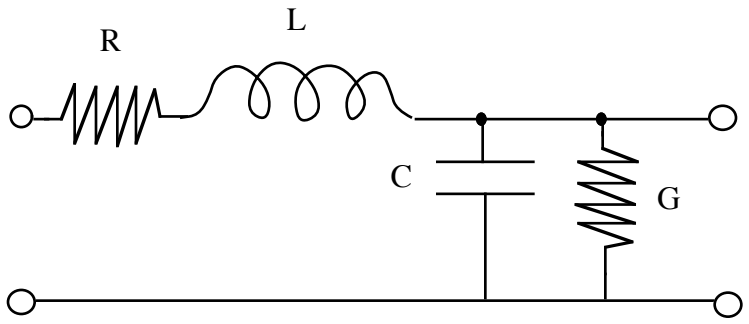
Single Wire Near Ground

Supported Transmission Lines

Also supported but not shown are any transmission line for which the R.L,C, and G parameters may be specified.

Lumped Element

Also supported is the lumped element section shown below:



Lumped Element Model

The transmission line types or lumped element values for each node are specified in the topology file following the node name.

The transmission line types are described in a data base in a file called *trans_types.dat*. This file is organized as follows:

name1 parameter1 parameter2 parameter3 parameter4
name2 parameter1 parameter2 parameter3 parameter4
name3 parameter1 parameter2 parameter3 parameter4
...
nameN parameter1 parameter2 parameter3 parameter4

The first component is the name of the transmission line. The name is formed as follows: The first few characters describe the transmission line type. The next characters in the name provide a distinct name for the transmission line with the specified parameters. For example,

coaxthin 0.001 0.005 2.25 0.001
coaxthick 0.002 0.01 2.25 0.001

specify two transmission lines of type coax. Their geometric parameters are different, however. One has a shield radius of 1 cm (*coaxthick*) while *coaxthin* has a shield radius of 0.5 cm. In this manner we can specify for example an ethernet thick and thin coax segment. All dimensions are in meters. The dielectric constant for both lines is 2.25.

For the supported transmission lines, the parameters are (refer to the figure above):

(1) Coax

*coax*Name r_i r_s ε_r t

(2) Parallel Conductors

*paral*Name r_i d ε_r 0

(3) Balanced Shielded Line

*balsh*Name r_i r_s ε_r h

(4) Wire Near Ground

*wireabg*Name r_i h ε_r 0

(5) Default

name r l c g

In the above, the italic letters are mandatory. The Name changes to distinguish the same line type but with different parameters. The 0 in the fourth parameter for 2 and 3 is mandatory.

Specifying RLCG parameters:

By default, the 4 parameters describe the r, l, c and g parameters of any line. This is used for twisted pair or any other type of transmission line. In this case, the name is arbitrary. That is, no required letters are needed. However, the names cannot begin with the italicized names for the supported transmission lines specified.

Specifying Lumped Element Parameters

Lumped element parameters are specified in the topology file following the node name. That is, instead of specifying a transmission line type, the values for the lumped elements are specified. For example,

node1 R100_L0.001_C1e-12_G0.010

and

node1 r100_l0.001_c1e-12_g0.01 0

specify a lumped element with R= 100 Ohms, L = 1 mH, C= 1 pF, G = 0.01 MHOs. Note that you can use upper or lower case letters for R,L,C and G. If you omit an element its value is assumed to be zero. Thus,

node2 R100_G0.01 0

Specifies a voltage divider where L=0 and C = 0. The use of admittance for G is useful since G=0 implies an open circuit.

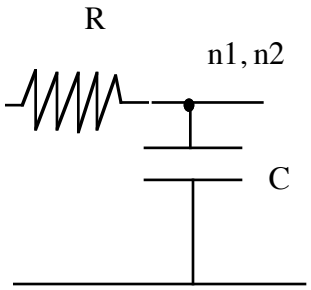
Note the zero following the lumped element value specification. This is **mandatory** indicating a length of 0 which is ignored.

A lumped element can be used as a load by following the node with an open circuit.

Hence,

```
n1 n2
n2
n1 R100        0
n2 C100e-12    0        open
```

specifies a network with series 100 Ohm resistor and a 100 pF capacitor for a load.



Plane Wave Propagation

Normal incidence plane wave propagation through inhomogeneous media can be modeled using the transmission line block. In this case, if we assume lossless transmission through a
medium with dielectric constant ε_r and permeability μ, then the medium can be specified in the trans_types.dat file as follows:

name 0 μ $\varepsilon_r\varepsilon_0$ 0

where ε_0 is the permittivity of free space.

Essentially, we replace the RLCG parameters with their free space equivalents.

To model plasmas use the following specification:

*palsma*Name collision_frequency electron_density 0 0

The electron density is in electrons per cubic cm.

Example:

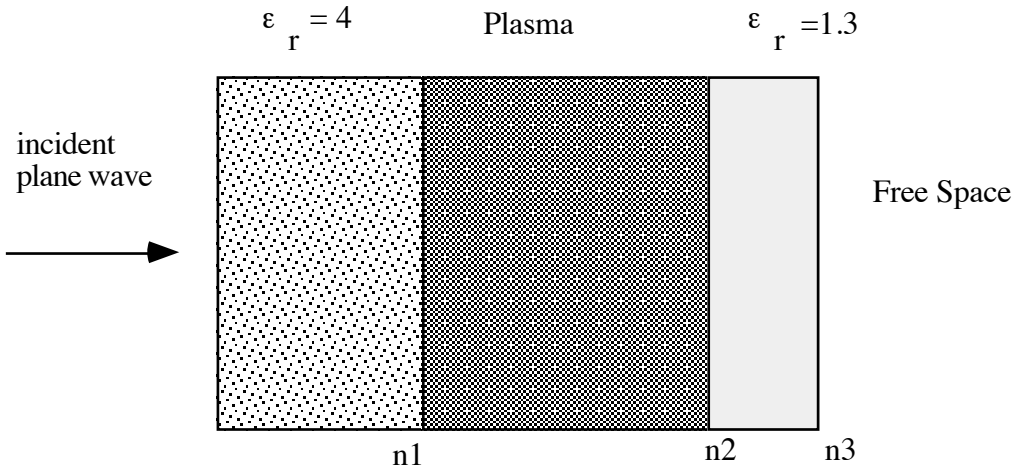
Consider the topology,

```
n1      n2
n2      n3
n3
n1      material1       1000
n2      plasmaTest      2000
n3      material2       500     377     0
```

and let the *trans_types.dat* file contain,

```
material1       0       1.257e-6        4*8.85e-12      0
material2       0       1.257e-6        1.3*8.85e-12    0
plasmaTest      0       10e10           0               0
```

Note that free space is indicated by a real matched termination of 377 Ohms at the "load" node n3.



Example of plane wave propagation

Block Parameters:

The parameters are as follows:
Param.

1 - (int) typeImpulse: point to point, echo path, or bandpass
point to point
2 - (int) implexp: log2 [length of impulse response in samples].
3 - (float) fsamp: sampling rate in Hz.
4 - (float) sourceImp: real source impedance, Ohms.
5 - (file) netFileName: ASCII file which contains network topology.
6 - (file) nodeName: node name to be analyzed (impulse response
from source to this node).
7 - (float) fstart: Start frequency, Hz for bandpass system.
8 - (float) fdel: Frequency step, Hz for bandpass system.

Convolution is performed by the fft overlap-save method (described in Oppenheim & Schafer, Digital Signal Processing, pp. 113).

The FFT length must be greater than the impulse response length.
For efficiency, it should probably be more than twice as long.

Note: The impulse response, complex frequency response and input impedance are stored in the files *nodeName.imp*, *nodeName.fre*, and *nodeName.zin* respectively.

Programmer: Sasan H. Ardalan, Overlap-save method by M. R. Civanlar
Date: July 26, 1990

Parameters

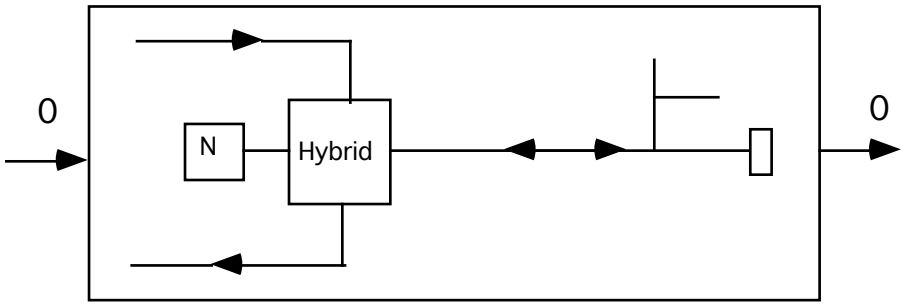
(1) 0: point to point, 1: echo path, 2: bandpass
int typeImpulse=0;
(2) log2(length of impulse response in samples
int implexp =8;
(3) Sampling Rate, Hz
float fsamp=1.e6;
(4) Source Impedance
float sourceImp = 100.0;
(5) (file)ASCII file which holds description of network
file netFileName = "net.top";
(6) Name of node for impulse response
file nodeName = "src";
(7) Start frequency for bandpass system (Hz)
float fstart=0;
(8) Frequency step for bandpass system (Hz)
float fdel=0;

Buffers

Inputs:
Buffer 0: Input signal(float).

Outputs:
Buffer 0: Channel Response (float)

Graphic



doppler

Description

"Doppler Shift":
This block takes a real input and performs a doppler shift. This is done in the frequency domain, where each frequency is shifted by

df(k) = (vm/c)f(k) = (vm/c)(fb + k*df)

where

df(k) is the shift of the kth bin in the frequency domain (after FFT of block of data),
vm is the velocity of the target,
c is the speed of light
fb is the beginning frequency of the band
df is fs/N where fs is the sampling rate and N is the number of
FFT points.

Parameters

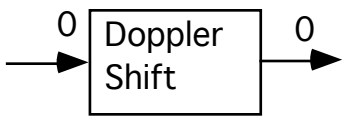
1 - (int) numberSamples: Total number of input samples.
2 - (float) fs: The sampling rate.
3 - (float) fb: The beginning frequency.
4 - (float) vm: The velocity of the target.

Programmer: Sasan H. Ardalan
Date: June 14, 1990

parameters

(1) Total number of input samples
int numberSamples =128;
(2) Sampling rate, Hz
float fs=10240e6;
(2) Beginning frequency
float fb=20e9;
(3) Target velocity
float vm=0;

Graphic



fade

Description

This block models multipath fading channels for mobile radio applications. The block accepts a complex baseband equivalent input and produces complex baseband equivalent samples.

The method is based on William C. Jakes, "Microwave Mobile Communications," John Wiley & Sons, 1974 in particular pp. 13-65.

Multipath is modeled using the technique presented by Nader Farahati, "A Software Multipath Fading Channel Simulator", Technophone Limited, July 1989. Nader Farahati is now with Scientific Generics, Cambridge U.K. Each multipath is associated with a time delay. The time delays are incorporated by transforming the problem into the frequency domain.

This block first reads all samples, u(t), at its input. It then multiplies the complex input samples by the complex fading channel amplitude with doppler shift for path i, ri(t), and transforms them into the frequency domain, Yi(f). The various delays are incorporated by multiplying the frequency domain data, Yi(f), by $\exp\{-2\pi j(fc+f)ti\}$ where fc is the carrier frequency, f is the frequency, and ti is the time delay of the ith multipath.

The various multipaths with independent fading channel amplitudes are added in the frequency domain and transformed back into the time domain.

The block then outputs the complex data as two channels (in-phase and quad-phase) in 128 sample chunks. This helps in limiting the size of buffers.

Note that other doppler spectrums and Rician distributions will be supported later. The block can easily be changed.

Programmer: Sasan Ardalan
Date: Dec. 27, 1990

parameters

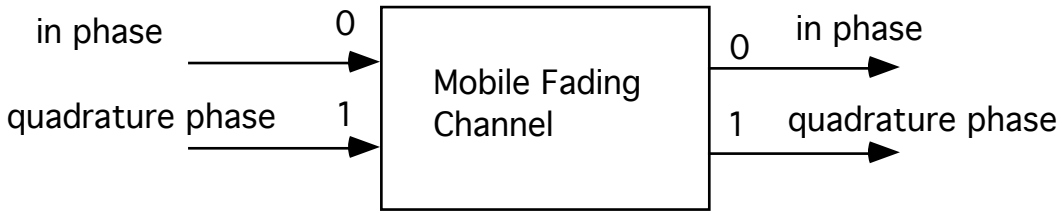
Number of points (preferably a power of 2)
int npts = 128;
Doppler Spectrum, only Ez supported at this time.
int type=0;
Sampling Rate
float fs=1.0;
Carrier frequency
float fc=1000e6;
Vehicle Velocity, m/s
float v= 0;
Power
float p= 1.0;
Array of multipath delays microsec: number_of_paths t0 t1 ...
array delays;
Array of multipath powers: number_of_paths p0 p1 ...
array powers;
Number of Plane Waves arriving plane waves, N where N >=34
int numberArrivals=40;

Buffers

Inputs:
Buffer 0: inPhase (float)
Buffer 1: quadPhase (float)

Outputs:
Buffer 0: inPhase (float)
Buffer 1: quadPhase (float)

Graphic



Fixed Point/Floating Point Models

fti

Description

Convert floating point buffer to 32 bit integer buffer. Auto fan-out.

Programmer: Sasan Ardalan
Date: July 1990

parameters

None

Buffers

Inputs:
Buffer 0 (float)

Outputs:
Auto fan-out (int)

Graphic

itf

Description

Convert 32 bit integer buffer to floating point buffer. Auto fan-out.

Programmer: Sasan Ardalan
Date: July 1990

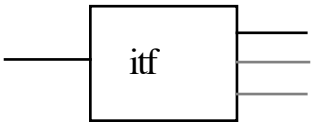
parameters

None

Buffers
Inputs:
Buffer 0 (int)

Outputs:
Auto fan-out (float)

Graphic



fxadd

Description

This block adds all of its double precision input samples. The input is in double precision (two packaged 32 bit integers) normally coming from the fxgain.s block. The output is rounded by the number of bits specified by the parameter roundoff_bits. Saturation can be turned on or off. Auto fan-in and fan-out supported. The output is single precision integers.

Although the inputs are double precision they represent numbers anywhere from 2 to 64 bits in wordlength. The output is a 32 bit integer, however, it represents a 1 to 32 bit number.

The structure of the double precision cells in the input buffer is:

```
typedef struct {
        int     lowWord;
        int     highWord;
} doublePrecInt;
```

for auto fan-in we use *indi*(i,k) for "input double precision integer (buffer i, delay k)"

Programmer: Jeyhan Karaoguz
Date: October 29, 1990

parameters

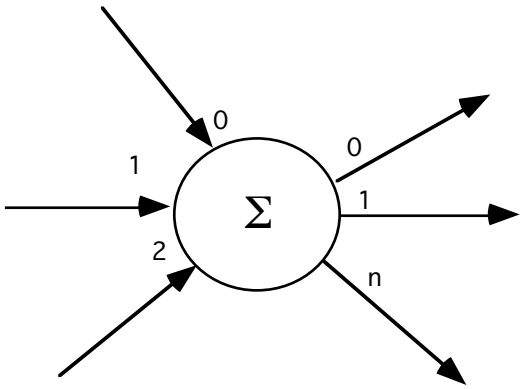
(1) Roundoff bits
int roundoff_bits =8;
(2) Word length
int size =32;
(3) Output size
int output_size = 32 ;
(4) Saturation mode: 1 = On , 0= Off
int saturation_mode = 1 ;

Buffers

Inputs:
(doublePrecInt). Auto fan-in

Outputs:
(doublePrecInt) Auto fan-out

Graphic



fxgain

Description

This block multiplies the incoming data stream by the parameter "Gain factor" in fixed-point arithmetic.The block is capable of doing extended precision arithmetic up to 64 bits result which is to be rounded to at least 32 bits after the fxadd.s block. The output buffer cells are doublePrecInt where two 32 bit integers are packaged into a cell. See fxadd.s.

Programmer: Jeyhan Karaoguz
Date: October 29, 1990

parameters

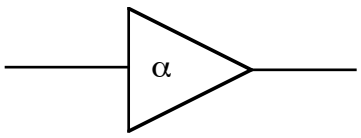
(1) Gain factor (float)
float factor =1.0;
(2) Number of bits to represent fraction
int qbits =8;
(3) Word length
int size = 32 ;

Buffers

Inputs:
Buffer 0: x (int).

Outputs:
(doublePrecInt) Auto fan-out

Graphic



fxnode

Description

Exactly as node block but with integer buffers.

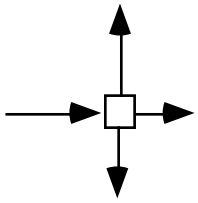
Programmer: Jeyhan Karaoguz
Date: September , 1990

parameters
None

Buffers
Inputs:
Buffer 0: (int).

Outputs:
(int) Auto fan-out

Graphic



fxdelay

Description

Exactly as delay block but with integer buffers.

Programmer: Jeyhan Karaoguz
Date: September , 1990

parameters

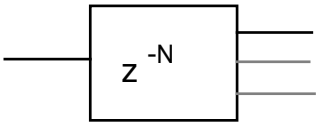
(1) Number of samples to delay";
int samples_delay = 1;

Buffers

Inputs:
Buffer 0: (int).delay_min = samples_delay; delay_max = samples_delay + 1;

Outputs:
(int) Auto fan-out

Graphic



fxnl

Description

Fixed Point Normalized Lattice Filter
fxnl()
This block implements a fixed point normalized lattice filter.

All variables used are of type integer so that the algorithm can be directly implemented on a Digital Signal Processor.

This block supports both floating point and integer buffers.

If floating point buffers are used, the input samples are quantized with the input quantizer range and number of bits specified as parameters. The output is also converted to floating point within the original quantizer range. e.g. +- 5 volts.

If integer buffers are used, no quantization is used and the
integer input samples are processed directly.

This block can be replace the floating point normalized lattice block *nl.s* to analyze the effects of fixed point implementation with different word sizes.

Programmer: Sasan H Ardalan
Date: September 7, 1991

parameters

(1) File with normalized lattice filter parameters
file file_name = "tmp.lat";
(2) Fixed point precision for coefficients, bits
int regBits=16;
(3) Input/output quantization bits
int quantBits=16;
(4) Input/output Range e.g. +- 5 volts
int quantRange=10.0;
(5) 0=Float Buffers,1=Integer Buffer
int bufferType=0;

Buffers

Inputs:
Buffer 0: input samples (int or float see parameter 5);

Outputs:

Buffer 0: output samples(int or float see parameter 5);

pri

Description

Stores integer samples in a file as hex.

Programmer: Jeyhan Karaoguz
Date: September , 1990

parameters

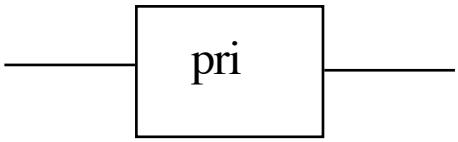
(1) File to store data;
file filename;

Buffers

Inputs:
Buffer 0: input samples (int);

Outputs:
Buffer 0: output samples(int);

Graphic



Logic Models

and

Description

Function performs logical "and" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

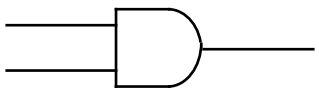
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



invert

Description

This block logically inverts incoming data stream.
Auto fanout is supported.

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

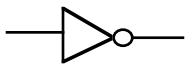
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Buffer 0: Input bits(float).

Outputs:
Output bits (float) Auto fan-out

Graphic



nand

Description

Function performs logical "nand" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

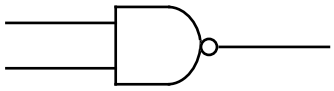
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



nor

Description

Function performs logical "nor" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

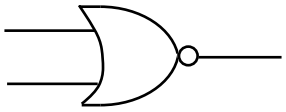
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



or

Description

Function performs logical "or" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

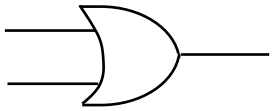
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



xnor

Description

Function performs logical "xnor" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

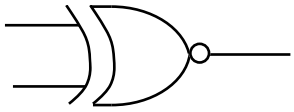
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



xor

Description

Function performs logical "xor" of all its input samples to yield an output sample; the number of input buffers is arbitrary and determined at run time.The number of output buffers is also arbitrary (auto-fanout).

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

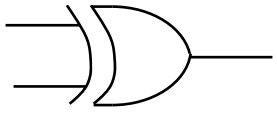
(1) Length of data in bits
int b_length =1;

Buffers

Inputs:
Input bits(float). Auto fan-in

Outputs:
Output bits (float) Auto fan-out

Graphic



jkff

Description

JK Flip Flop. Positive edge triggered.

Characteristic Table

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

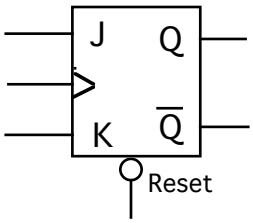
None

Buffers

Inputs:
Buffer 0: j (float). delay_max = 1
Buffer 1: k (float). delay_max = 1
Buffer 2: cp (float). delay_max = 1
Buffer 3: re (float). delay_max = 1

Outputs:
Buffer 0: q (float)
Buffer 1: qp (float)

Graphic


J
Q
K
$\overline{Q}$
Reset

srff

Description

Set/Reset SR Flip Flop. Positive edge triggered.

Characteristic Table

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	-
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	-

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

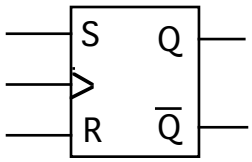
None

Buffers

Inputs:
Buffer 0: S delay_max=1;
Buffer 1: R delay_max=1;
Buffer 2: cp delay_max=1;

Outputs:
Buffer 0: Q (float)
Buffer 1: QP (float)

Graphic


S
Q
R
$\overline{Q}$

srlatch

Description

Set/Reset SR latch.

Truth Table

S	*R*	*Q*	*Q'*	
1	*0*	*1*	*0*	
0	*0*	*1*	*0*	*(after S=1, R=0)*
0	*1*	*0*	*1*	
0	*0*	*0*	*1*	*(after S=0,R=1)*
1	*1*	*0*	*0*	

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

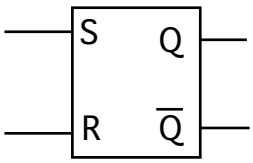
None

Buffers

Inputs:
Buffer 0: S delay_max=1;
Buffer 1: R delay_max=1;

Outputs:
Buffer 0: Q (float)
Buffer 1: QP (float)

Graphic



tff

Description

Toggle T Flip Flop. Positive edge triggered.

Characteristic Table

Q	*T*	*Q(t+1)*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

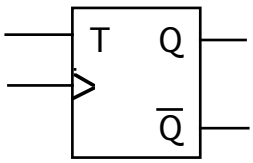
None

Buffers

Inputs:
Buffer 0: T delay_max=1;
Buffer 1: cp delay_max=1;

Outputs:
Buffer 0: Q (float)
Buffer 1: QP (float)

Graphic



dff

Description

D Flip Flop. Positive edge triggered.

Characteristic Table

Q	*D*	*Q(t+1)*
0	*0*	*0*
0	*1*	*1*
1	*0*	*0*
1	*1*	*1*

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

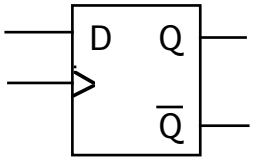
None

Buffers

Inputs:
Buffer 0: T delay_max=1;
Buffer 1: cp delay_max=1;

Outputs:
Buffer 0: Q (float)
Buffer 1: QP (float)

Graphic



divider

Description

Divides input asynchronously by specified parameter.

Programmer: Ray J. Kassel
Date: October 29, 1990

parameters

Value to divide by.
int divide_by = 2;

Buffers

Inputs:
Buffer 0: Data to be divided delay_max=1;

Outputs:
Buffer 0: Divided data (float)

Graphic

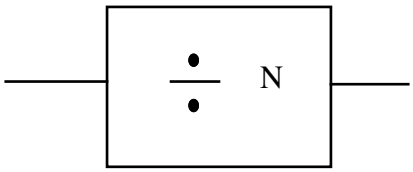


Image Manipulation Blocks

imgaddnoise

Description

Input an image and add the following noise to it:

Uniform noise distributed between a(param1) and b(param2)
Gaussian noise with mean(param1) and std(param2) specified.
Spike noise generated as follows:
a Normal distribution is generated. If its level exceeds
param1, then its value is assigned to x.
Next x is multiplied by param2 to obtain the spike.
The spike value is then added to the matrix.

To generate different outcomes change the expression

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

```
Noise Type:0=none,1=uniform,2=normal,3=spike
        int     type=1;
Expression for seed generation
        file    expression="any_expression";
param1: a(uniform), mean (normal) trigger(spike)
        float   param1=0.0;
param2: b(uniform), std (normal) multiplier(spike)
        float   param2=1.0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgbreakup

Description

The sub images are sequentially output.

Programmer: Sasan Ardalan
Date: August 15, 1993

Parameters

Sub image width
int subWidth=8;
Sub image height
int subHeight=8;
Levels (for inverse)
int levels=256;

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgbuild

Description

This block inputs a sequence of images and creates a larger image with the inputted images forming subimages from left to right top to bottom.

When the height (specified as a parameter) is exceeded the inputted images wrap around.

After the sub images are all gathered, the image is output.

Programmer: Sasan Ardalan
Date: August 15, 1993

Parameters

Image width
    int imageWidth=128;
Image height
    int imageHeight=128;
Levels (for inverse)
    int levels=256;

Buffers

Outputs:
Buffer 0: image_t y

Graphic

imgcalc

Description

Perform a mathematical or logical operation on an image using another image.
Buffers:
Input:
x
y
Output:
z

The image on buffer x is replaced with the result of the calculation.
x=f(x,y)
x is output on buffer z.

Image y may be offset in width and height prior to the operation.

Supported Operations:
0: Multiply
1: Add
2: Subtract
3: Divide
4: AND
5: OR
6: XOR
7: Complement
8: Copy

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

Operation:0=x,1=-,2:*,3:/,4:&,5:|,6:xor,7:cmpl,8:copy
int operation=6;
width offset
int widthOffset=0;
height offset

```
int     heightOffset=0;
```

Buffers

Inputs:
Buffer 0: image_t x
Buffer 0: image_t y

Outputs:
Buffer 0: image_t z

Graphic

imgcolorsep

Description

This block inputs an image and outputs multiple images representing
color components.

For RGB, three images are output for each image inputted.
The first image is the red color, the second, green and the fourth blue.

For YCbCr, the outputed images are the luminance (Y) and chrominance (Cb,Cr)
images.

The block obtains the RGB values from the current installed color map.

To view the images, connect this block to the imgdisp block.
Set the Animation parameter to 0 (FALSE)
After the simulation, uninstall the color map, next
load the 16 level gray color map.
Finally, select Update All Images from the Image pulldown menu.
This will display the image components in gray scale.

Programmer: Sasan Ardalan
Date: August 15, 1993

Parameters

```
Color Space:0=RGB,1=YCbCr,2=YUB
        int     colorSpace=0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgcxmag

Description

This block inputs a complex image height*(2*width) and creates a new real image height*width of magnitudes

Programmer: Sasan Ardalan
Date: September 9, 1993

Parameters

None

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgcxtrl

Description

This block inputs a complex image height*(2*width) and creates a new real image height*width of only the real part

Programmer: Sasan Ardalan
Date: September 9, 1993

Parameters

1=free input image, 0= don't
int freeImageFlag=0;

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgfft

Description

This block inputs an image and computes its forward or inverse FFT.
For a forward FFT a real matrix is expected and a complex image is produced.
For an inverse FFT a complex image is expected (possible result of a forward FFT).
Complex images store real and imaginary parts as even and odd sample columns. Thus for complex images width =2*height.

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

```
Operation: 0=Forward FFT, 1= Inverse FFT
        int     fftType=0;
Center: 0=None , 1= Yes
        int     centerFlag=0;
1=free input image, 0= don't
    int   freeImageFlag=0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgfilter

Description

Input an image and filter it.
The filter kernel is specified. It must be an ASCII file as follows:

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

```
        Filter Kernel
                file      filterKernel="filt.krn";
    1=free input image, 0= don't
        int     freeImageFlag=0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgfilter

Description

Input an image and filter it.
The filter kernel is specified. It must be an ASCII file as follows:

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

```
        Filter Kernel
                file      filterKernel="filt.krn";
    1=free input image, 0= don't
        int     freeImageFlag=0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imggen

Description

Generate a rectangular image. The image contains a rectangle offset in with and height with specified rectangle width and height.

The image may be complemented in the sense that the value will fill all the image except the rectangle.

For now only one image is generated. However, this can be easily changed so that a sequence is generated.

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

Pixel Value
    float pixel=1.0;
Image Width
    int pwidth=128;
Image Height
    int pheight=128;
Rectangle Width
    int rectWidth=128;
Rectangle Height
    int rectHeight=128;
Rectangle Width Offset
    int widthOffset=0;
Rectangle Height Offset
    int heightOffset=0;
Complex Flag
    int complexFlag=0;
Complement Flag
    int complementFlag=0;

Buffers

Outputs:
Buffer 0: image_t y

Graphic

imghisteq

Description

This block inputs an image and performes an histogram equalization on it.
The original image is overwritten.

Programmer: Sasan Ardalan
Date: September 10, 1993

Parameters

Levels (Power of two)
int levels=256;

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imginterp

Description

This block inputs an image and interpolates it.

NOTE: Input and output buffers are of image_t type.

Programmer: Sasan Ardalan
Date: May 12, 1993

Parameters

width factor
int widthFactor=1;
heightFactor
int heightFactor=1;

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgmanip

Description

This block inputs an image and transposes or flips it.

For transposing, a new image is generated.
All other operations overwrite the input image

Programmer: Sasan Ardalan
Date: April 15, 1993

Parameters

Operation:0=none,1=transpose,2=flipVert,4=flipHorz,3=inverse
int operation=0;
Levels (for inverse)
int levels=256;

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgmux

Description

Multiplex N input image channels to one output channel.
The multiplexing order is from input channel 0 to N-1 for N input channels.

Auto-fanout is supported.
There are no parameters.

Programmer: L.J. Faber
Date: April 15, 1988
Extended to images by Sasan Ardalan, June 2, 1993

Parameters

None

Buffers

Inputs:
Auto fan-in: image_t

Outputs:
Auto fan-out: image_t

Graphic

imgnonlinfilter

Description

Input an image and perform nonlinear filtering on it.

Programmer: L.J. Faber
Date: April 15, 1988
Extended to images by Sasan Ardalan, June 2, 1993

Parameters

```
        Nonliner Filter Type:2=min,3=median,4=max
                int       type=3;
        Order
                int       order=3;
    1=free input image, 0= don't
        int     freeImageFlag=0;
```

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgnonlinfilter

Description

Input an image and perform nonlinear filtering on it.

Programmer: Sasan Ardalan
Date: April 15, 1993

Parameters

Name of output file
file file_name = "output.img";

Buffers

Inputs:
Buffer 0: image_t x

Graphic

imgproc

Description

This block inputs an image and transposes or flips it.

For transposing, a new image is generated.
All other operations overwrite the input image

Programmer: Sasan Ardalan
Date: April 15, 1988

Parameters

Operation: 0=Transpose, 1= flip
int operation=0;

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgrdbin

Description

Description: Read a binary image.
On each visit a row is read from file.
An image sample is output.
Auto fan-out.

Programmer: Sasan Ardalan
Date: November 4, 1990

Parameters

Image width
int pwidth=128;
Image height
int pheight=128;
File that contains binary image
file file_name = "test.img";
Number of bytes to skip
int skip=0;

Buffers

Autofan-out

Graphic

imgrdfptiff

Description

Description: Read a floating point TIFF image.
Auto fan-out.

Programmer: Sasan Ardalan
Date: April 15, 1993

Parameters

File that contains floating point TIFF image
file fileName = "image.tif";

Buffers

Auto fan-out *image_t*

Graphic

imgrtcx

Description

This block inputs a real image widthxheight and creates a new
complex image (2*width)*height with the imaginary part set to zero

Programmer: Sasan Ardalan
Date: September 9, 1993

Parameters

1=free input image, 0= don't
```
        int     freeImageFlag=0;
```

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgserin

Description

This block inputs rows and creates an image and outputs it to the image buffer

Programmer: Sasan Ardalan
Date: April 15, 1988

Parameters

Image Width
int pwidth=1;
Image Height
int pheight=1;

Buffers
Inputs:
Buffer 0: float x

Outputs:
Buffer 0: image_t y

Graphic

imgserout

Description

This block inputs an image and outputs it a row at a time.

Programmer: Sasan Ardalan
Date: April 15, 1988

Parameters

None

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: float y

Graphic

imgshrink

Description

This block inputs an image and interpolates it.

NOTE: Input and output buffers are of image_t type.

Programmer: Sasan Ardalan
Date: May 12, 1993

Parameters

width shrink factor
int widthFactor=1;
height shrink Factor
int heightFactor=1;

Buffers
Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgsubimg

Description

This block inputs an image and transposes or flips it.

Programmer: Sasan Ardalan
Date: May 12, 1988

Parameters

Sub image offset width
int widthOffset=0;
Sub image offset height
int heightOffset=0;
Sub image width
int subWidth=1;
Sub image height
int subHeight=1;

Buffers

Inputs:
Buffer 0: image_t x

Outputs:
Buffer 0: image_t y

Graphic

imgwrfptiff

Description

Writes an input image to a TIFF file as floating point. Also store the current colormap.

If multiple images are received, they overwrite the previous on.
This block can later be modified so that multiple images are stored in a singe TIFF file (with multiple directories). Or stored in multiple TIFF files with the file name changing in some manner.

Auto fan out is supported.

Programmer: Sasan Ardalan
Date: October 14, 1993

Parameters

Name of output file
file fileName = "output.tif";

Buffers

Inputs:
Buffer 0: image_t x

Graphic

imgwrtiff

Description

Writes an input image to a TIFF file as 8 bit samples. Also store the current colormap.

If multiple images are received, they overwrite the previous on.
This block can later be modified so that multiple images are stored in a singe TIFF file (with multiple directories). Or stored in multiple TIFF files with the file name changing in some manner.

Auto fan out is supported.

Programmer: Sasan Ardalan
Date: October 14, 1993

Parameters

Name of output file
file fileName = "output.tif";

Buffers
Inputs:
Buffer 0: image_t x

Graphic

Various

expr

Description

Function evaluates all its input samples through an expression
specified as a parameter to yield an output sample;
the number of input buffers is arbitrary and determined at run time.
The number of output buffers is also arbitrary (auto-fanout).

In the expression refer to the sample on input buffer 0 as in0, for
input buffer 1 as in1 and so on. For example:

```
sin(5*PI*in0/100)*exp(-0.001*in1)
```

Note the PI is predefined.

Programmer: Sasan Ardalan
Date: October 14, 1993

Parameters

Enter expression (buffers are in0,in1,...)
file paramExpr="in0*1";

Buffers
Inputs:
Auto fan-in float

Outputs:
Auto fan-out

Graphic

invcust

Description

This block generates inventory customers.

The customers each have an inter arrival time and a product demand.

The inter arrival time is packaged with the product demand into a complex data structure and output.

The real part of the complex data structure is the inter arrival time.
The imaginary part is the product demand.

The first parameter, which defaults to NUMBER_SAMPLES,
tells how many total samples to send out.

Programmer: Sasan Ardalan
Date: October 14, 1993

Parameters

total number of customers
int num_of_samples = 128;
Type:0=exp,1=gamma
int type = 0;
Inter Arrival Time
float meanArrival = 1.0;
Expression for random number generator
file expression = "any expresssion";
File with demand probabilities
file demandProbDist = "demand_dist.dat";

pace rate to determine how many samples to output
float pace_rate = 1.0;
number of samples on the first call if paced
int samples_first_time = 128;

Buffers

Outputs:
Auto fan-out

Graphic

inventory

Description
This block models an inventory system.

The input buffer is the customer inter arrival times and
product demand.

The input are complex with the real part equal to the inter arrival time
and the imaginary part equal to the product demand.

The inventory does not need to output anything but we have chosen to output
the inventory level. Thus by connecting the output of
the inventory block to the plot block, you can observe the inventory level
over time.

Many other output combinations are possible but this block is used to serve
as an example.

The inventory block implements the C code in "Simulation Modeling and Analysis"
by Averill M. Law and W. David Kelton, Second Edition 1991.

We have included original comments.

The parameters are:

(1) Initial inventory level
(2) Number of months
(3) Set up cost
(4) Incremental cost
(5) Holding cost
(6) Shortage cost
(7) Minimum lag
(8) Maximum lag
(9) Order threshold (s)
(10) Inventory Level (S)
(11) Expression for random number generator
(12) Output Request:0=Inventory Level,1=Demand Size

Notes:
(1) The simulation will end when there are no more customers.
However, the simulation can end using parameter 2 as a condition.

The input buffers are arbitrary so that in the future multiple customer sources may be modeled with a single inventory.

The number of input buffers is arbitrary and determined at run time. The number of output buffers is also arbitrary (auto-fanout).

Programmer: Sasan Ardalan
Date: November 6, 1993

Parameters

Initial inventory level
int initialInventoryLevel = 60;

number of months
int numberMonths = 120;

set up cost
float setupCost = 32.0;

incremental cost
float incrementalCost = 3.0;

holding cost
float holdingCost = 1.0;

shortage cost
float shortageCost = 5.0;

minimum lag
float minLag = 0.5;

maximum lag
float maxLag = 1.0;

Order threshold (s)
float smalls = 20.0;

Inventory Level (S)

float bigs = 60.0;

Expression
file expression = "any expresssion";

Output Request:0=Inventory Level,1=Demand Size
int outputRequest = 0;

Buffers

Inputs:
Auto fan-in

Outputs:
Auto fan-out

Graphic

rangen

Description

This block generates random samples.

The first parameter, which defaults to NUMBER_SAMPLES, tells how many total samples to send out.

Programmer: Sasan Ardalan
Date: October 14, 1993

Parameters

total number of samples to output
int num_of_samples = 128;
Type:0=normal,1=uniform,2=exp,3=gamma
int type = 0;
Parameter 1(mean,a,lambda)
float p1 = 0.0;
Parameter 2(std,b)
float p2 = 0.0;
Expression
file expression = "any expresssion";
pace rate to determine how many samples to output
float pace_rate = 1.0;
number of samples on the first call if paced
int samples_first_time = 128;

Buffers

Outputs:
Auto fan-out

Graphic