Agilent Technologies

Advanced Design System 2011.01

Feburary 2011
Numeric Components

Acknowledgments
Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries. Mentor products and processes are registered trademarks of Mentor Graphics Corporation. * Calibre is a trademark of Mentor Graphics Corporation in the US and other countries. "Microsoft®, Windows®, MS Windows®, Windows NT®, Windows 2000® and Windows Internet Explorer® are U.S. registered trademarks of Microsoft Corporation. Pentium® is a U.S. registered trademark of Intel Corporation. PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated. UNIX® is a registered trademark of the Open Group. Oracle and Java and registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission. MATLAB® is a U.S. registered trademark of The Math Works, Inc.. HiSIM2 source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code in its entirety, is owned by Hiroshima University and STARC. FLEXlm is a trademark of Globetrotter Software, Incorporated. Layout Boolean Engine by Klaas Holwerda, v1.7 http://www.xs4all.nl/~kholwerd/bool.html . FreeType Project, Copyright (c) 1996-1999 by David Turner, Robert Wilhelm, and Werner Lemberg. QuestAgent search engine (c) 2000-2002, JObjects. Motif is a trademark of the Open Software Foundation. Netscape is a trademark of Netscape Communications Corporation. Netscape Portable Runtime (NSPR), Copyright (c) 1998-2003 The Mozilla Organization. A copy of the Mozilla Public License is at http://www.mozilla.org/MPL/ . FFTW, The Fastest Fourier Transform in the West, Copyright (c) 1997-1999 Massachusetts Institute of Technology. All rights reserved.

The following third-party libraries are used by the NlogN Momentum solver:

"This program includes Metis 4.0, Copyright © 1998, Regents of the University of Minnesota", http://www.cs.umn.edu/~metis , METIS was written by George Karypis (karypis@cs.umn.edu).

Intel@ Math Kernel Library, http://www.intel.com/software/products/mkl

SerDes Example Designs

This section includes the following Serializer/Deserializer (SerDes) application example designs:

- 8b10b Coder and Decoder
- 64b66b Coder and Decoder
- Blind Adaptive Decision Feedback Equalizer
- Adaptive Decision Feedback Equalizer with Training Sequence

8b10b Coder and Decoder

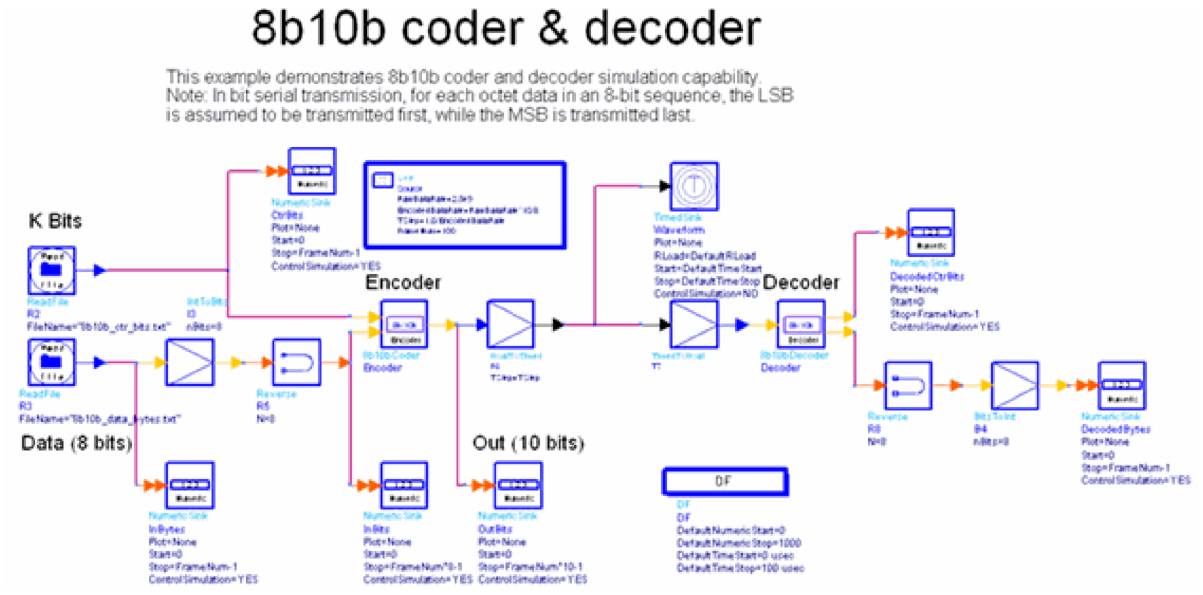
Location: /examples/DSP/serdes_wrk

Objective

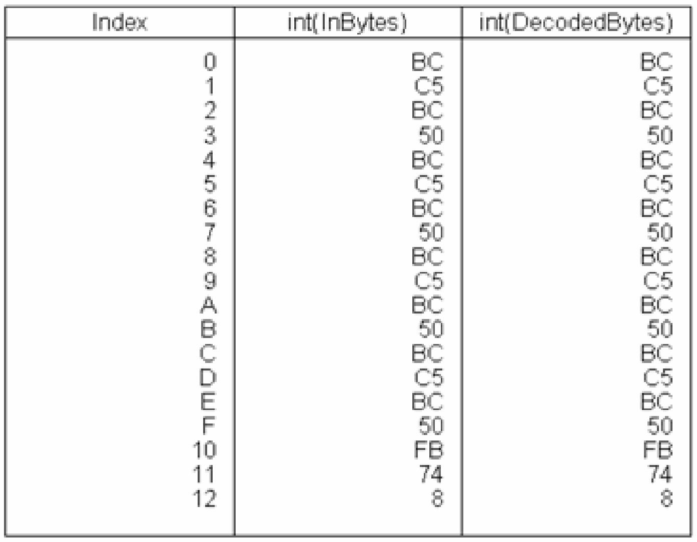
This example demonstrates 8b10b coder and decoder (8B/10B Encoding and 8B/10B Decoding) simulation capability.

Setup

Data bytes and Control bits (K) are read from files. In bit serial transmission, for each octet data in an 8-bit sequence, the LSB is assumed to be transmitted first, while the MSB is transmitted last.

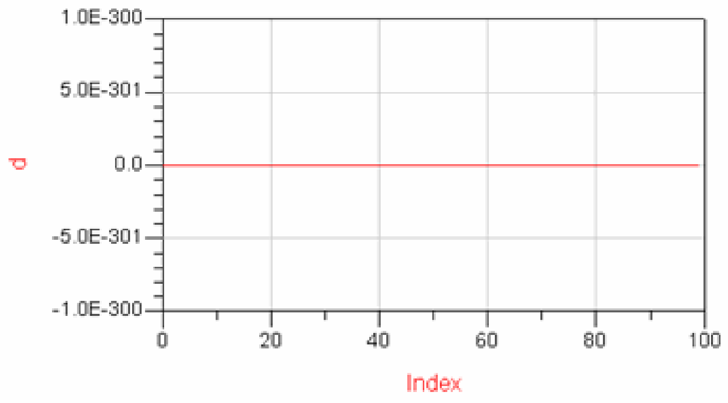

8b10b coder & decoder
This example demonstrates 8b10b coder and decoder simulation capability.
Note: In bit serial transmission, for each octet data in an 8-bit sequence, the LSB is assumed to be transmitted first, while the MSB is transmitted last.
K Bits
ReadFile R2 FileName="8b10b_ctrl_bits.txt"
ReadFile R3 FileName="8b10b_data_bytes.txt"
Data (8 bits)
IntToBits I3 nBits=8
Reverse R5 N=8
Numeric Sink CtrlBits Plot=None Start=0 Stop=FrameNum-1 ControlSimulation=YES
Encoder
8b10bCoder Encoder
Out (10 bits)
Numeric Sink InBytes Plot=None Start=0 Stop=FrameNum-1 ControlSimulation=YES
Numeric Sink InBits Plot=None Start=0 Stop=FrameNum*8-1 ControlSimulation=YES
Numeric Sink OutBits Plot=None Start=0 Stop=FrameNum*10-1 ControlSimulation=YES
Timed Sink Waveform Plot=None RLoad=DefaultRLoad Start=DefaultTimeStart Stop=DefaultTimeStop ControlSimulation=NO
Decoder
8b10bDecoder Decoder
Numeric Sink DecodedCtrlBits Plot=None Start=0 Stop=FrameNum-1 ControlSimulation=YES
Reverse R8 N=8
BitsToInt B4 nBits=8
Numeric Sink DecodedBytes Plot=None Start=0 Stop=FrameNum-1 ControlSimulation=YES
DF
DF DF DefaultNumericStart=0 DefaultNumericStop=1000 DefaultTimeStart=0 usec DefaultTimeStop=100 usec

Analysis

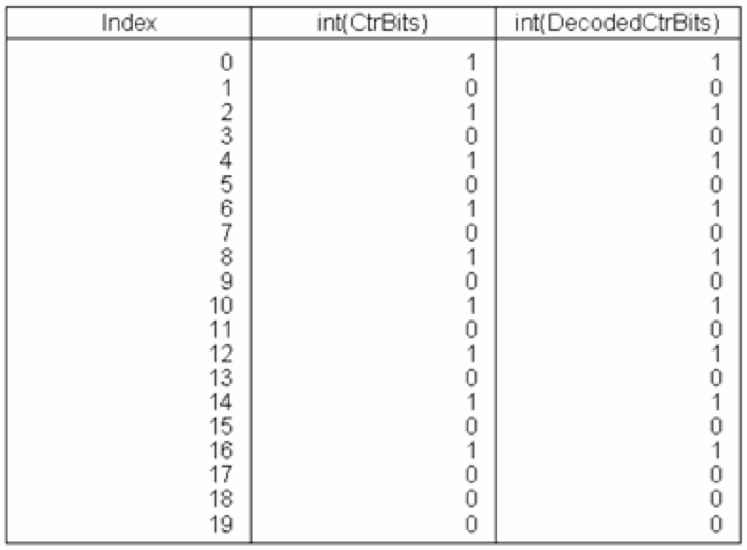


Index	int(InBytes)	int(DecodedBytes)
0	BC	BC
1	C5	C5
2	BC	BC
3	50	50
4	BC	BC
5	C5	C5
6	BC	BC
7	50	50
8	BC	BC
9	C5	C5
A	BC	BC
B	50	50
C	BC	BC
D	C5	C5
E	BC	BC
F	50	50
10	FB	FB
11	74	74
12	8	8

Eqn d=InBytes-DecodedBytes

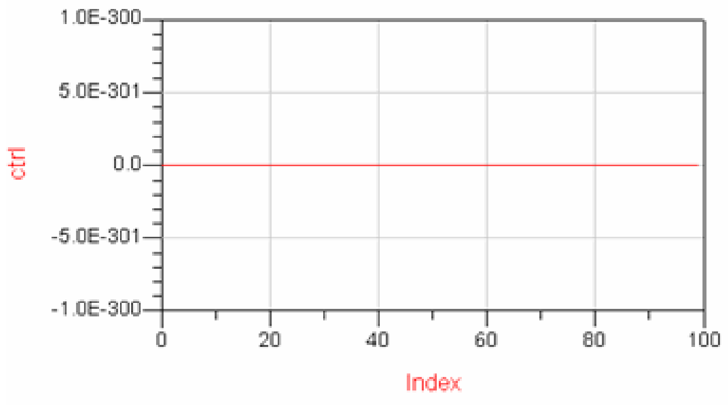


Source bytes before 8b10b Encoder and Decoded bytes after 8b10b Decoder



Index	int(CtrBits)	int(DecodedCtrBits)
0	1	1
1	0	0
2	1	1
3	0	0
4	1	1
5	0	0
6	1	1
7	0	0
8	1	1
9	0	0
10	1	1
11	0	0
12	1	1
13	0	0
14	1	1
15	0	0
16	1	1
17	0	0
18	0	0
19	0	0

Eqn ctrl=CtrBits-DecodedCtrBits



Control bits before 8b10b Encoder and Decoded Control bits after 8b10b Decoder

Notes/Equations

Read the dds to see the transmitted bytes and control bits are correctly decoded.

64b66b Coder and Decoder

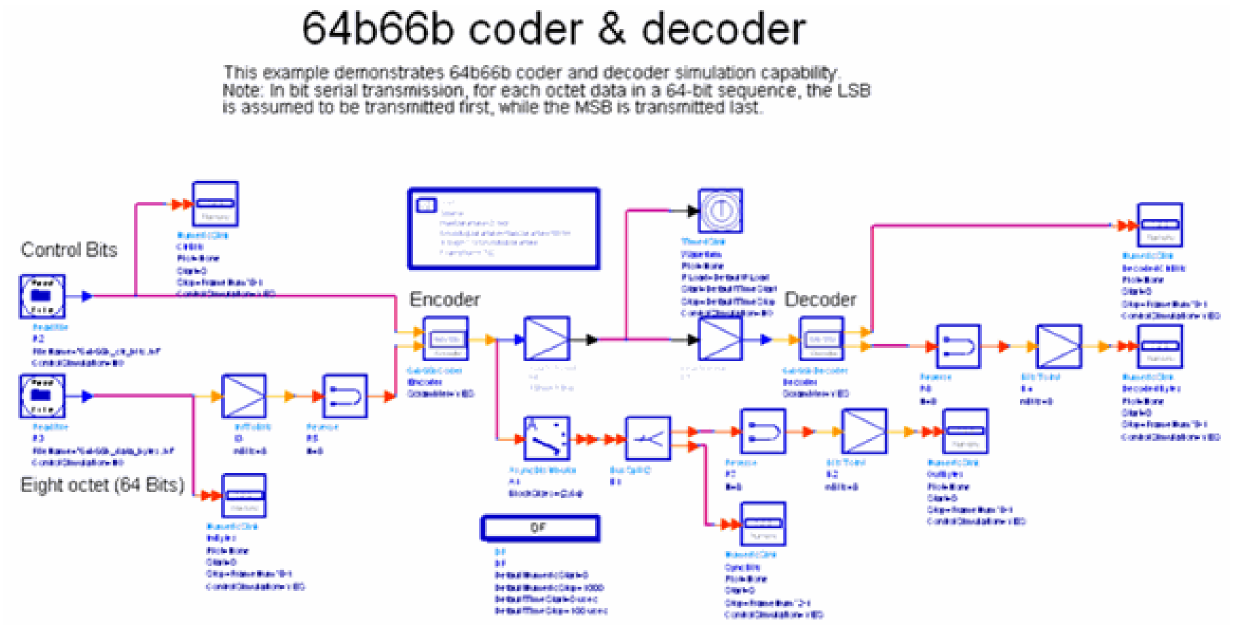
Location: /examples/DSP/serdes_wrk

Objective

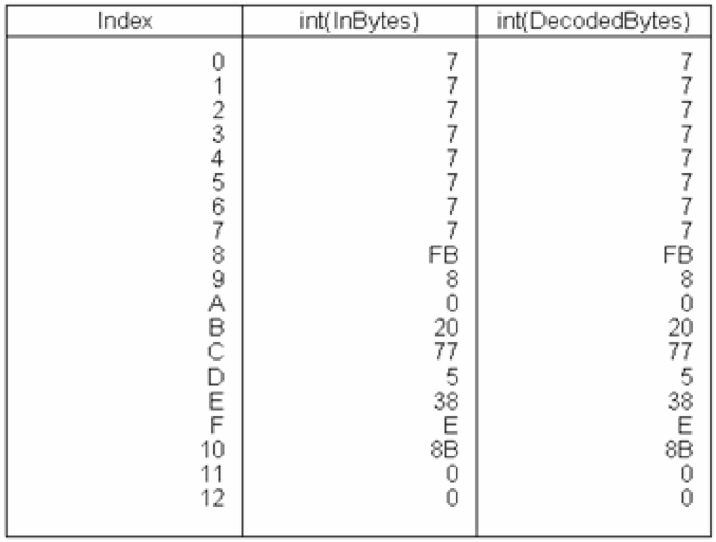
This example demonstrates 64b66b coder and decoder (64B/66B Encoding and 64B/66B Decoding) simulation capability.

Setup

Data bytes and Control bits are read from files. In bit serial transmission, for each octet data in an 64-bit sequence, the LSB is assumed to be transmitted first, while the MSB is transmitted last.

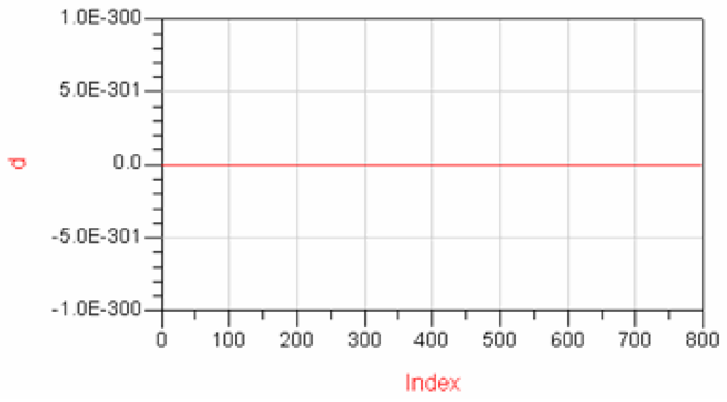

64b66b coder & decoder
This example demonstrates 64b66b coder and decoder simulation capability.
Note: In bit serial transmission, for each octet data in a 64-bit sequence, the LSB is assumed to be transmitted first, while the MSB is transmitted last.
Control Bits
Encoder
Decoder
Eight octet (64 Bits)
DF

Analysis

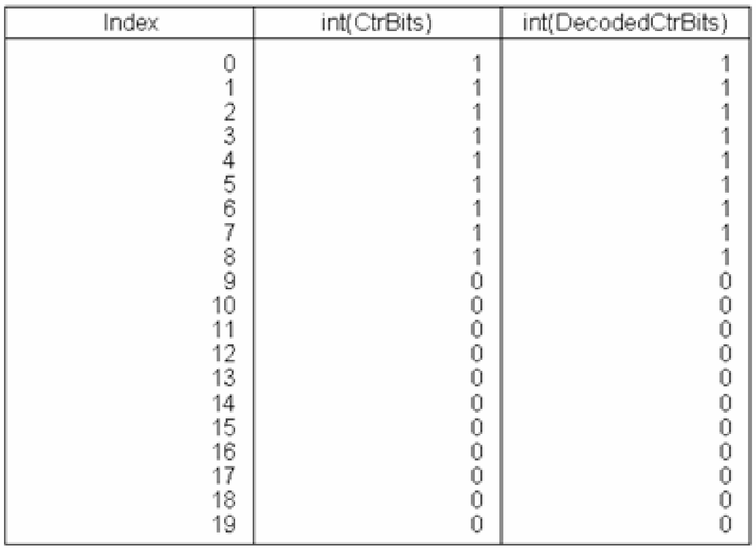


Index	int(InBytes)	int(DecodedBytes)
0	7	7
1	7	7
2	7	7
3	7	7
4	7	7
5	7	7
6	7	7
7	7	7
8	FB	FB
9	8	8
A	0	0
B	20	20
C	77	77
D	5	5
E	38	38
F	E	E
10	8B	8B
11	0	0
12	0	0

Eqn d=InBytes-DecodedBytes

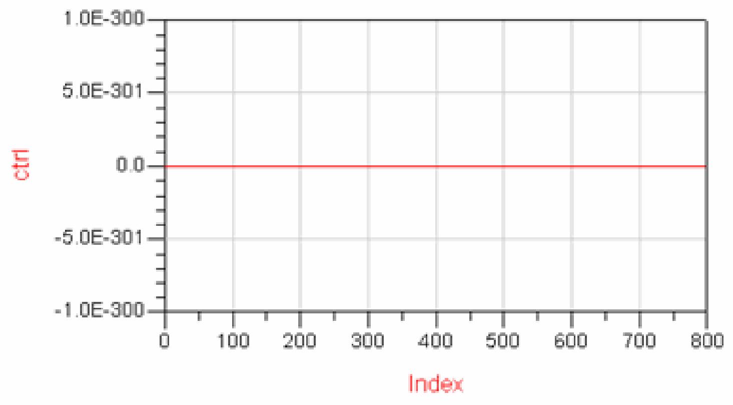


Source bytes before 64b66b Encoder and Decoded bytes after 64b66b Decoder



Index	int(CtrBits)	int(DecodedCtrBits)
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0

Eqn ctrl=CtrBits-DecodedCtrBits



Control bits before 64b66b Encoder and Decoded Control bits after 64b66b Decoder

Notes/Equations

Read the dds to see the transmitted bytes and control bits are correctly decoded.

Blind Adaptive Decision Feedback Equalizer

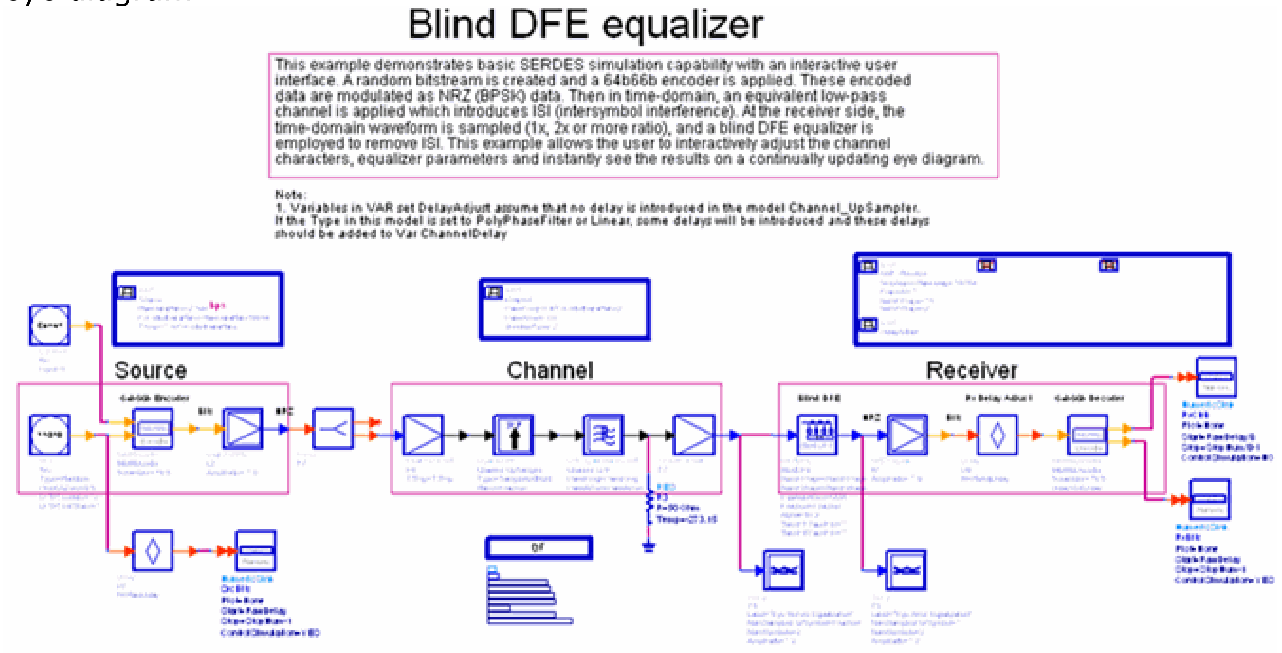
Location: /examples/DSP/serdes_wrk

Objective

This example demonstrates basic SerDes simulation capability with an interactive user interface. 64b66b encoder, decoder and Blind DFE are illustrated.

Setup

A random bitstream is created and a 64b66b encoder is applied. These encoded data are modulated as NRZ (BPSK) data. Then in time-domain, an equivalent low-pass channel is applied which introduces ISI (intersymbol interference). At the receiver side, the time-domain waveform is sampled (1x, 2x or more ratio), and a blind DFE equalizer is employed to remove ISI. This example allows the user to interactively adjust the channel characters, equalizer parameters and instantly see the results on a continually updating eye diagram.

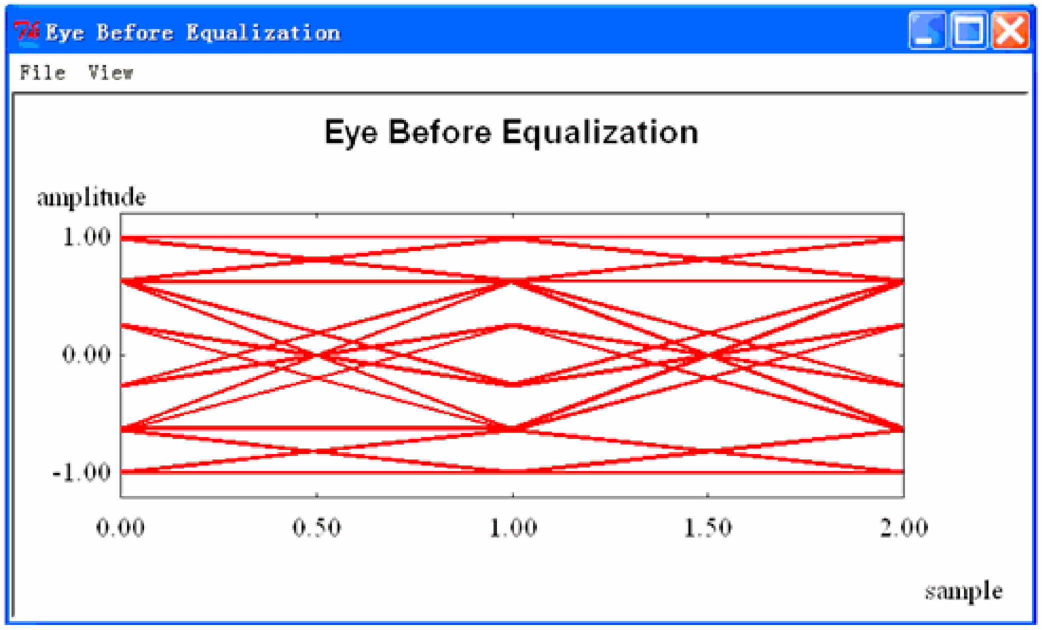


Blind DFE equalizer

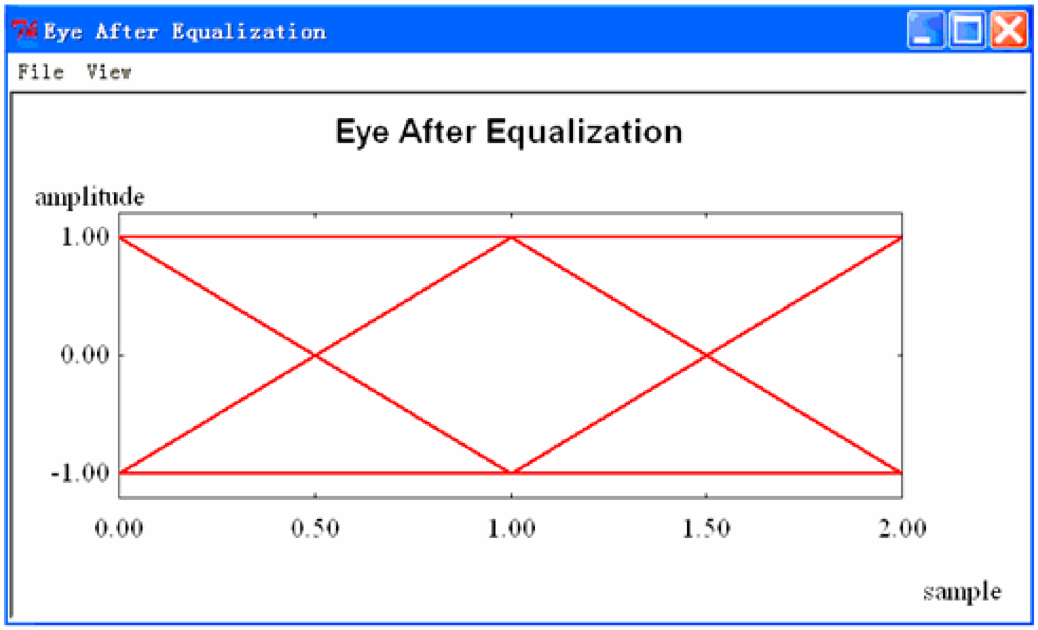
This example demonstrates basic SERDES simulation capability with an interactive user interface. A random bitstream is created and a 64b66b encoder is applied. These encoded data are modulated as NRZ (BPSK) data. Then in time-domain, an equivalent low-pass channel is applied which introduces ISI (intersymbol interference). At the receiver side, the time-domain waveform is sampled (1x, 2x or more ratio), and a blind DFE equalizer is employed to remove ISI. This example allows the user to interactively adjust the channel characters, equalizer parameters and instantly see the results on a continually updating eye diagram.

Note:
1. Variables in VAR set DelayAdjust assume that no delay is introduced in the model Channel_UpSampler.
If the Type in this model is set to PolyPhaseFilter or Linear, some delays will be introduced and these delays should be added to Var ChannelDelay

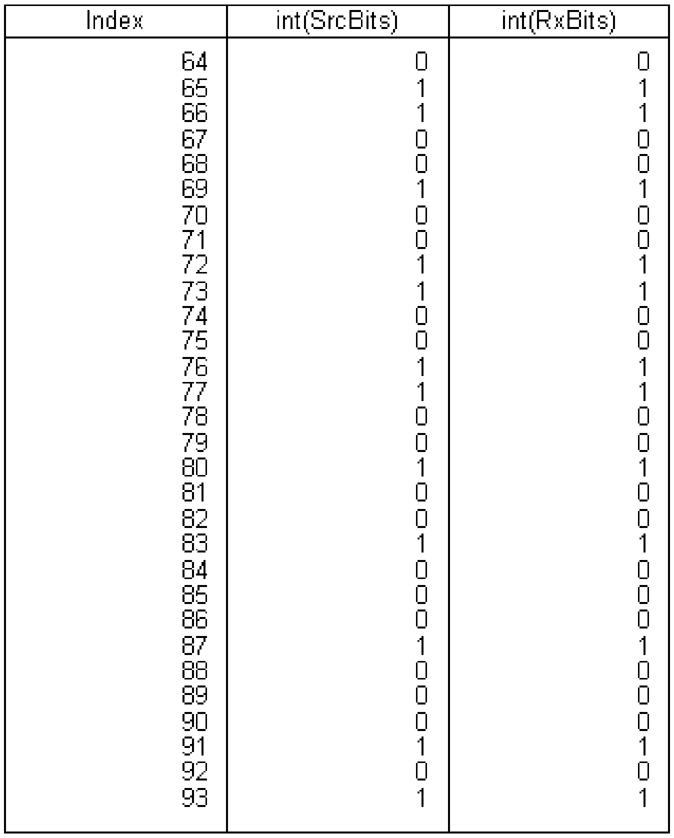
Analysis



Eye before Equalization.

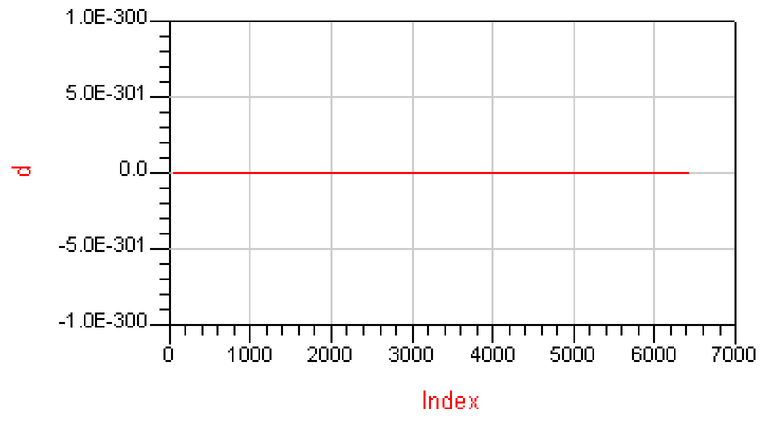


Eye after Equalization.



Index	int(SrcBits)	int(RxBits)
64	0	0
65	1	1
66	1	1
67	0	0
68	0	0
69	1	1
70	0	0
71	0	0
72	1	1
73	1	1
74	0	0
75	0	0
76	1	1
77	1	1
78	0	0
79	0	0
80	1	1
81	0	0
82	0	0
83	1	1
84	0	0
85	0	0
86	0	0
87	1	1
88	0	0
89	0	0
90	0	0
91	1	1
92	0	0
93	1	1

Eqn d=SrcBits-RxBits



The SrcBits are correctly decoded after equalization.

Source bits before 64b66b Encoder and Decoded bits by 64b66b Decoder

Notes/Equations

Observe the eye diagram change before and after equalizer. Read the dds to see the transmitted bits are correctly decoded.

Adaptive Decision Feedback Equalizer with Training Sequence

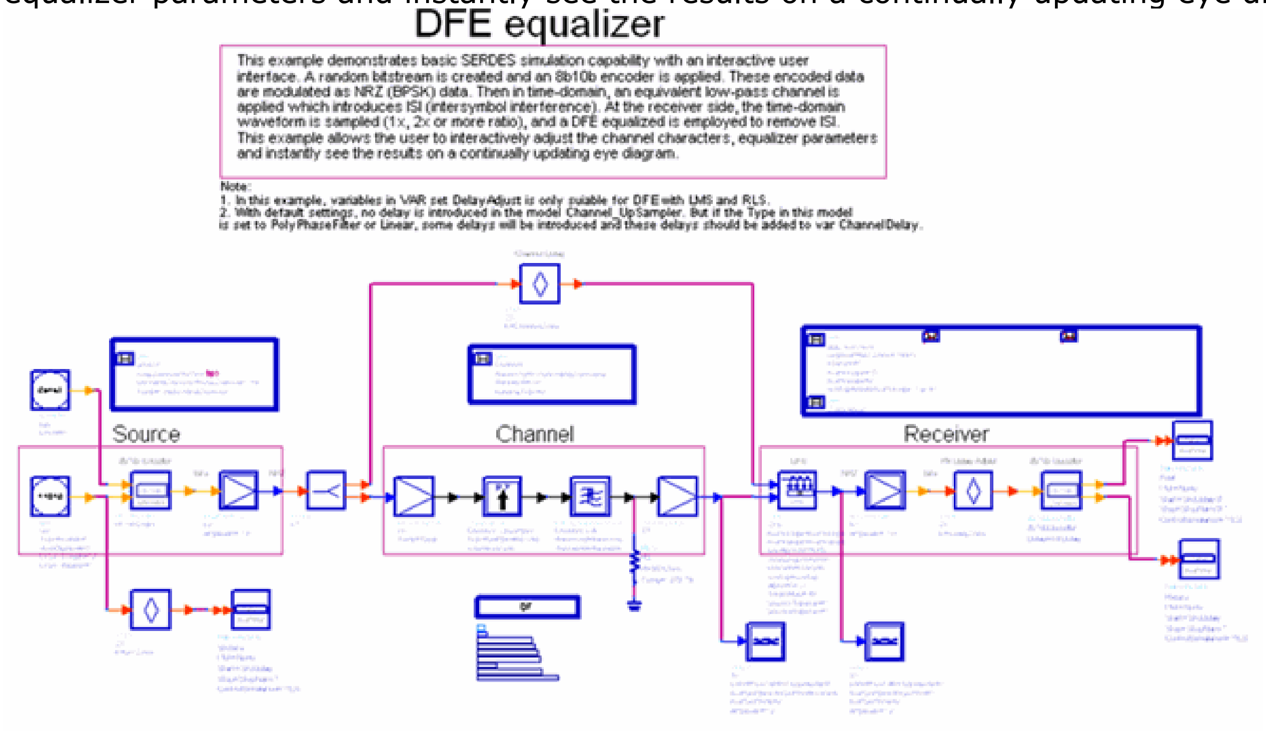
Location: /examples/DSP/serdes_wrk

Objective

This example demonstrates basic SerDes simulation capability with an interactive user interface. 8b10b encoder, decoder and DFE with training sequence are illustrated.

Setup

A random bitstream is created and an 8b10b encoder is applied. These encoded data are modulated as NRZ (BPSK) data. Then in time-domain, an equivalent low-pass channel is applied which introduces ISI (intersymbol interference). At the receiver side, the time-domain waveform is sampled (1x, 2x or more ratio), and a DFE equalizer is employed to remove ISI. This example allows the user to interactively adjust the channel characters, equalizer parameters and instantly see the results on a continually updating eye diagram.

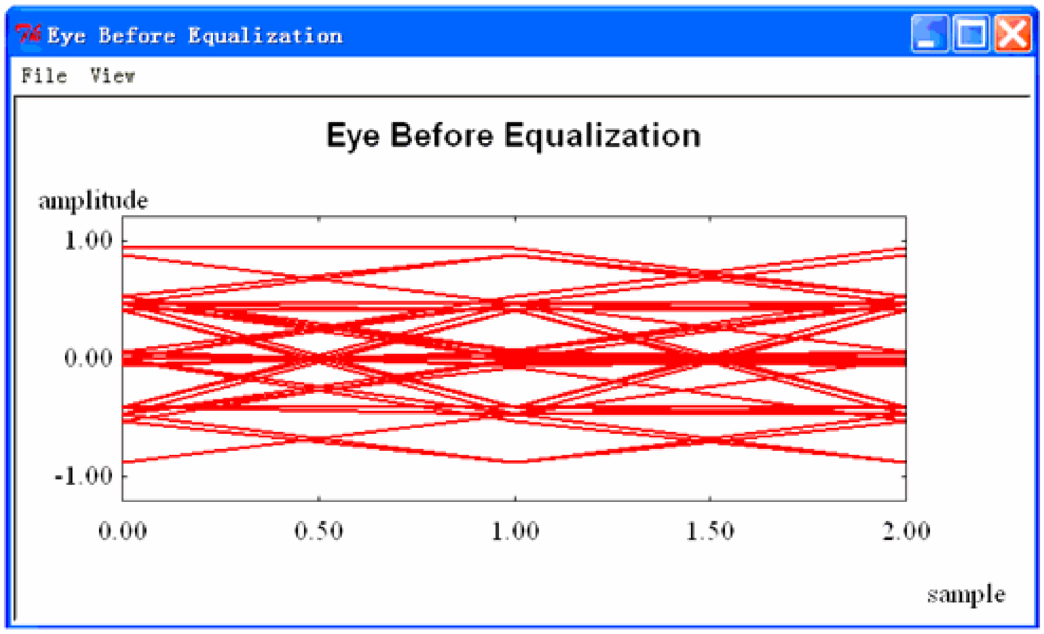


DFE equalizer

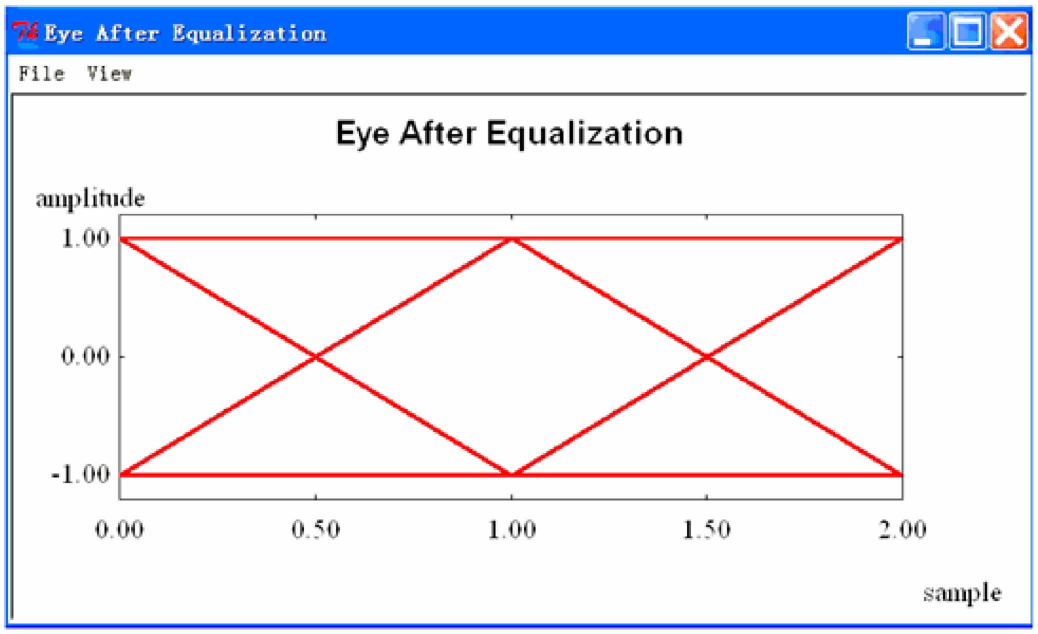
This example demonstrates basic SERDES simulation capability with an interactive user interface. A random bitstream is created and an 8b10b encoder is applied. These encoded data are modulated as NRZ (BPSK) data. Then in time-domain, an equivalent low-pass channel is applied which introduces ISI (intersymbol interference). At the receiver side, the time-domain waveform is sampled (1x, 2x or more ratio), and a DFE equalized is employed to remove ISI. This example allows the user to interactively adjust the channel characters, equalizer parameters and instantly see the results on a continually updating eye diagram.

Note:
1. In this example, variables in VAR set DelayAdjust is only suiable for DFE with LMS and RLS.
2. With default settings, no delay is introduced in the model Channel_UpSampler. But if the Type in this model is set to PolyPhaseFilter or Linear, some delays will be introduced and these delays should be added to var ChannelDelay.

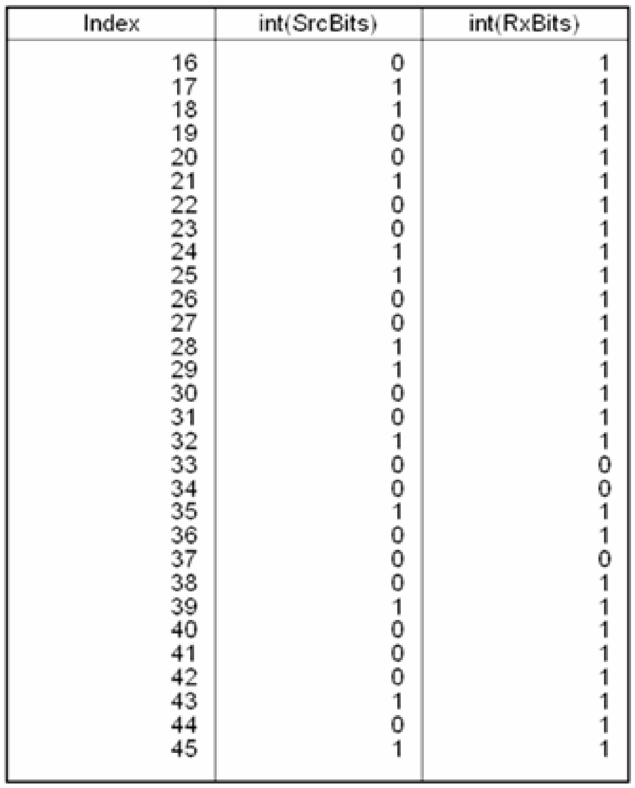
Analysis



Eye before Equalization.

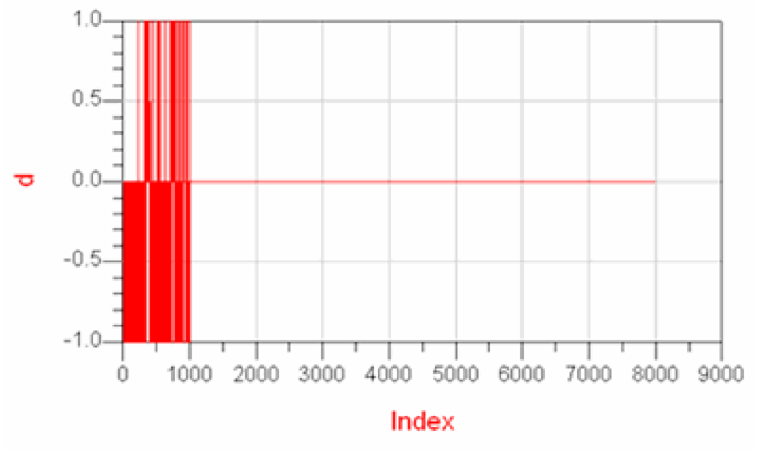


Eye after Equalization.



Index	int(SrcBits)	int(RxBits)
16	0	1
17	1	1
18	1	1
19	0	1
20	0	1
21	1	1
22	0	1
23	0	1
24	1	1
25	1	1
26	0	1
27	0	1
28	1	1
29	1	1
30	0	1
31	0	1
32	1	1
33	0	0
34	0	0
35	1	1
36	0	1
37	0	0
38	0	1
39	1	1
40	0	1
41	0	1
42	0	1
43	1	1
44	0	1
45	1	1

Eqn d=SrcBits-RxBits



Source bits before 8b10b Encoder and Decoded bits by 8b10b Decoder

Notes/Equations

Observe the eye diagram change before and after equalizer. Read the dds to see the transmitted bits are correctly decoded.

WMAN Example Designs

WMAN example designs created in ADS are based on the IEEE 802.16d Standard. These designs (constructed using the new Numeric Advanced Comm components, basic ADS components, and *Matlab* components) focus on the physical layer of WMAN systems. These are intended to be a baseline system for designers to get an idea of what nominal or ideal system performance would be. Evaluations can be made regarding degraded system performance due to system impairments that may include nonideal component performance.

Access the designs from the ADS Main window: **File > Open > Example > Com_Sys > WMAN_802_16d_TX_wrk**.

The ADS2004A designs focus on transmitters: *Test_WMAN_RFSource* for testing a DUT under a WMAN frequency division duplex downlink system; *Test_WMAN_CodedSignals* for generating fully-coded signals; and, *Test_WMAN_ESG* for downloading WMAN data to an ESG. Receiver designs will be addressed beyond ADS2004A.

Agilent Instrument Compatibility

These WMAN designs can be used for downloading data to Agilent instrument through ESG_E4438C_Sink or CM_ESG_E4438C_Sink. WMAN data can drive Agilent ESG instruments such as E443xB or E4438C to generate RF signals. Using these RF WMAN signals from an E4438C, WMAN device under test (DUT) can be tested. Basic system performances can be measured using Agilent 89600 Series Vector Signal Analyzer (VSA) for spectrum as well as waveforms.

The table below lists instrument models and Firmware revisions.

Agilent Instrument Compatibility Information

WMAN Designs	ESG Models	VSA Models
SpecVersion=802.16d,Dec. 2003	E443xB, Firmware Revision B.03.75	89600 Series, software version 5.0

For more information about the ESG series digital and analog RF signal generators, visit http://www.agilent.com/find/ESG

For more information about the 89600 series vector signal analyzers, visit http://www.agilent.com/find/89600

WMAN IEEE 802.16 Specifications

IEEE 802.16a was initiated for WMAN systems. The revised version IEEE 802.16d [1] specifies the air interface of a fixed (stationary) point-to-multipoint broadband wireless access system providing multiple services in a wireless metropolitan area network. The standard includes a particular PHY specification applicable to systems operating at 2- to 11-GHz. The 2- to 11-GHz air interface has options such as WirelessMAN-SCa, WirelessMAN-OFDM, WirelessMAN-OFDMA, and WirelessHUMAN.

WMAN standards for both WirelessMAN-OFDM and WirelessMAN-OFDMA have physical layers based on OFDM. OFDM transmits data simultaneously over multiple, parallel frequency sub-bands and offers robust performance under severe radio channel conditions. OFDM also provides a convenient method for mitigating delay spread effects. A cyclic extension of the transmitted OFDM symbol can be used to achieve a guard interval between symbols. Provided that this guard interval exceeds the excess delay spread of the radio channel, the effect of the delay spread is constrained to frequency selective fading of the individual sub-bands. This fading can be canceled by means of a channel compensator, which takes the form of a single tap equalizer on each sub-band.

IEEE 802.16d OFDM physical layer settings are listed in the table below.

OFDM Physical Layer Specifications

Specification	**Settings**
Information data rate	4-70 Mbps
Modulation	QPSK OFDM, 16QAM OFDM, and 64QAM OFDM
Error correcting code	Reed-Solomon plus Convolutional Code
Overall Coding rate	1/2, 3/4, 2/3
Basic FFT Size	256
Number of subcarriers	200, DC nulled
Number of Pilot tones	8
Cyclic Prefix (or Guard Interval)	1/32,1/16,1/8 and 1/4 symbol period

WMAN System Designs

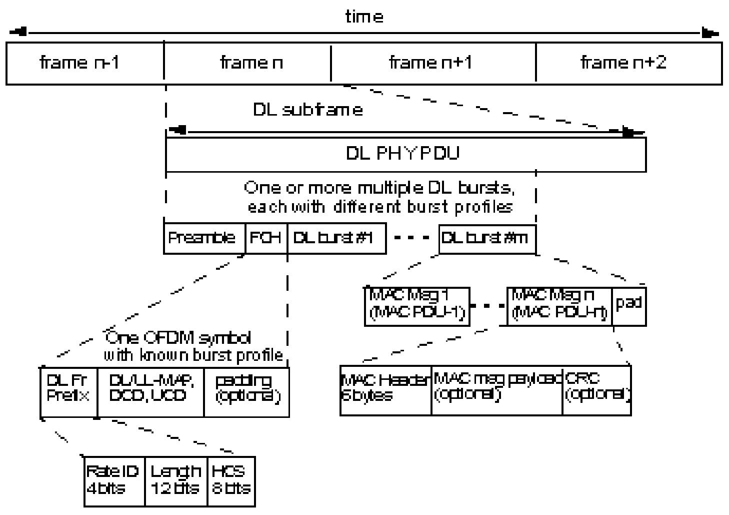
WMAN system design basic components include signal sources, channels, receivers, and measurements. Signal sources and measurements based on WirelessMAN-OFDM are the focus in ADS2004A.

Signal Sources

IEEE 802.16d FDD DL signal sources are provided in the example workspace. Based on the 16d Standard, a WMAN 16d downlink PHY PDU is defined (see OFDM Frame Structure with FDD DL) that starts with a long preamble for PHY synchronization. The preamble is followed by a frame control header (FCH) burst. The FCH burst is one OFDM symbol long and is transmitted using QPSK rate 1/2 with the mandatory coding scheme.

The FCH is followed by one or multiple downlink bursts, each transmitted with different burst profiles. Each downlink burst consists of an integer number of OFDM symbols, and its burst profiles are specified by a 4-bit DIUC in the DL-MAP. DIUC encoding is defined in the DCD messages.

OFDM Frame Structure with FDD DL

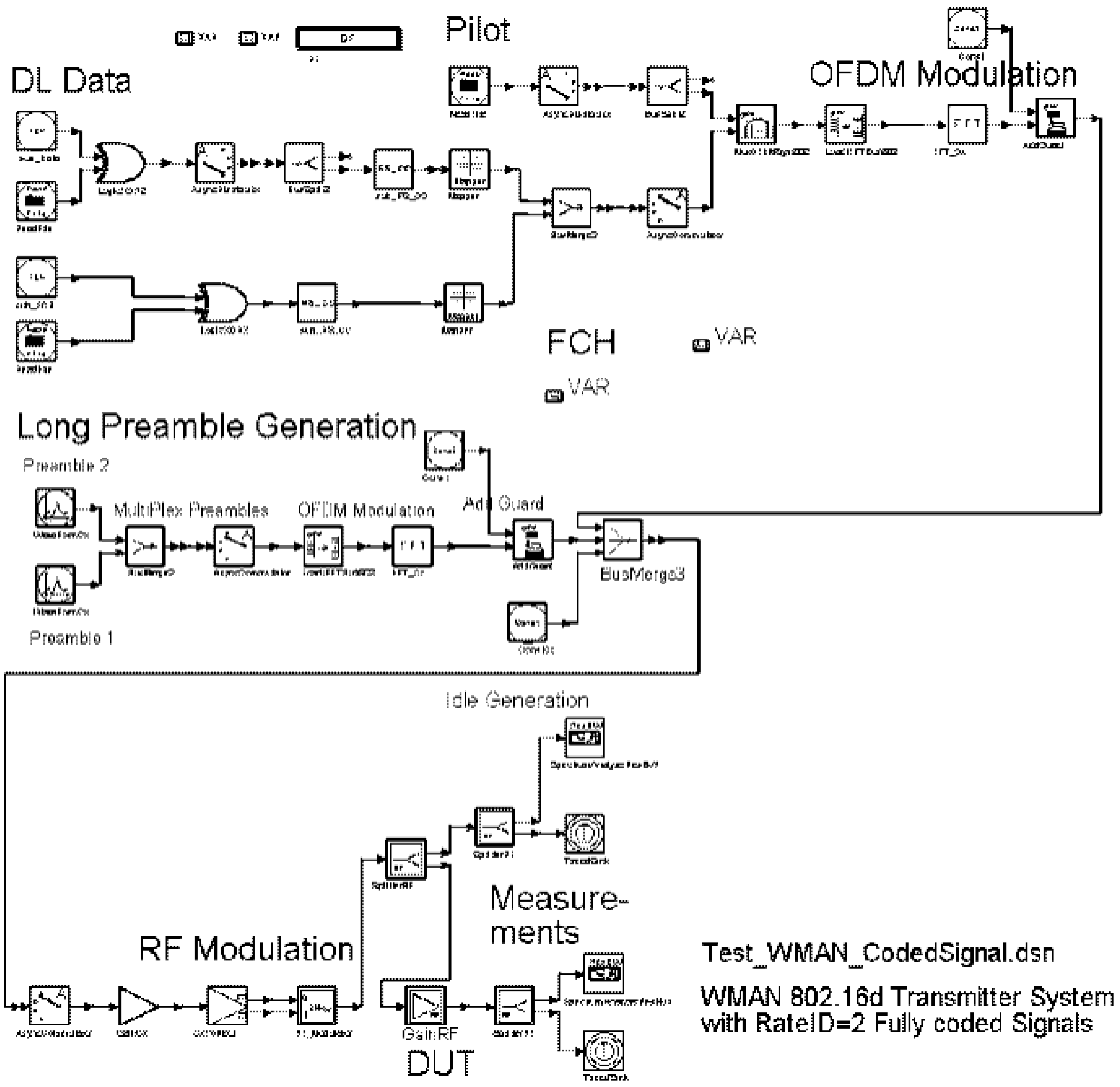


With the OFDM PHY, a PHY burst (downlink or uplink), consists of an integer number of OFDM symbols carrying medium access control (MAC) messages, i.e., MAC PDUs. To form an integer number of OFDM symbols, a burst payload can be padded by the bytes 0xFF. The payload is then scrambled, encoded, and modulated using the burst PHY parameters specified by the 16d Standard.

The example designs are to aid in understanding the WMAN 802.16d transmission system and to find its basic performance in the physical layer. Simulation will generate single bursts of data, formatted for downlink in the mandatory coding schemes.

The figure below shows an OFDM frame structure for the WMAN FDD DL system in the *Test_WMAN_CodedSignal* example; this figure highlights the main components at the sub-system level. (Refer to Fully-Coded Signal Generation for details regarding this design.)

WMAN FDD DL System in ADS: *Test_WMAN_CodedSignal*



To understand WMAN FDD DL signal generation, basic components for constructing sub-systems will be described, then sub-system components such as preamble generation, FCH channel, data generation, OFDM modulation, multiplexing, and measurements for WMAN systems will be described.

Basic Components

This section describes the basic components used in the designs; for details regarding each design, refer to WMAN Design Example Descriptions.

Data Modulation

After bit interleaving, data bits in both FCH and DL data channels are entered serially to the constellation mapper. Gray-mapping is needed for data modulation and the constellations are specified in Section 8.3.3.4 in 802.16d. In the WMAN examples, Mapper (Numeric Advanced Comm library) provides Gray-mapped QPSK, 16QAM and 64QAM modulations.

Pilot Modulation

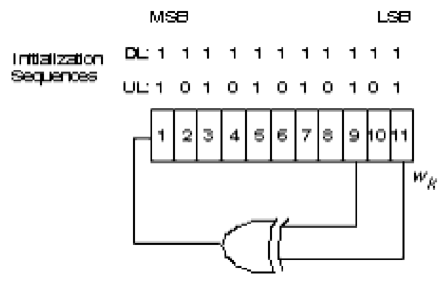
Pilot subcarriers are inserted into each data burst in order to constitute the symbol and these are modulated according to their carrier location within the OFDM symbol. A PRBS generator will be used to produce a sequence. The polynomial for the PRBS generator is $X^{11} + X^{9} + 1$.

The pilot modulation value for OFDM symbol k is derived from w_k. On the downlink, index k represents the symbol index relative to the beginning of the downlink subframe; on the uplink, index k represents the symbol index relative to the beginning of the burst. For uplink and downlink, the first symbol of the preamble is denoted by k=1. Downlink and uplink initialization sequences are shown in PRBS for Pilot Modulation. For the downlink, this results in the sequence 11111111111000000000110... where the third 1 (w_3 =1) will be used in the first OFDM downlink symbol following the frame preamble. For each pilot (indicated by frequency offset index), BPSK modulation will be derived as follows:

$$\text{DL: } C_{-88} = C_{-38} = C_{63} = C_{88} = 1 - 2w_k \text{ and } C_{-63} = C_{-13} = C_{13} = C_{38} = 1 - 2\overline{w_k}$$

$$\text{UL: } C_{-88} = C_{-38} = C_{13} = C_{38} = C_{63} = C_{88} = 1 - 2w_k \text{ and } C_{-63} = C_{-13} = 1 - 2\overline{w_k}$$

PRBS for Pilot Modulation



To implement the pilot PRBS sequence in ADS, an LFSR component is used with parameter settings: Seed=2047 (corresponding to the initial sequence: 1 1 1 1 1 1 1 1 1 1 1) and FeedbackList="11 9". The random data generated from the LFSR can be recorded as a data file; a WaveFormCx component is used to read this data and output as the PRBS sequence for pilot modulation.

Multiplexing for Frame Structure

In the WMAN examples, the AsyncCommutator component with BusMerge2 is used to multiplex 2 different data/signals/preambles as shown in WMAN FDD DL System in ADS: Test_WMAN_CodedSignal. With BusMerge3, AsyncCommutator can be used for multiplexing 3 data/signals/preambles and with BusMerge4 for multiplexing 4 data/signals/preambles.

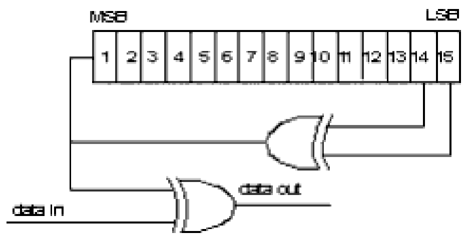
Channel Coding Components

Channel coding components will be used for both FCH and data channels. Key components for channel coding include a scrambler component, forward error correction (FEC) component, and an interleaver component.

The Scrambler component scrambles data with the appropriate LFSR initialization for uplink or downlink.

The shift-register of the randomizer is initialized for each new allocation. The PRBS generator is shown in Scrambling Data Generation. Each data byte to be transmitted is sequentially entered into the randomizer, MSB first. Preambles are not randomized. The seed value is used to calculate the randomization bits, which are combined in an XOR operation with the serialized bit stream of each burst. The randomizer sequence is applied only to information bits.

Scrambling Data Generation



The bits issued from the randomizer are applied to the encoder. On the downlink, the randomizer is re-initialized at the start of each frame with the sequence: 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0.

To implement the scrambler, an LFSR component is used with parameter settings: Seeds=38144 (corresponding to initial sequence 1 0 0 1 0 1 0 0 0 0 0 0 0) and

FeedbackList="15 14". The random data generated from the LFSR can be recorded as a data file; a WaveFormCx component is used to read this data and output as the scramble sequence.

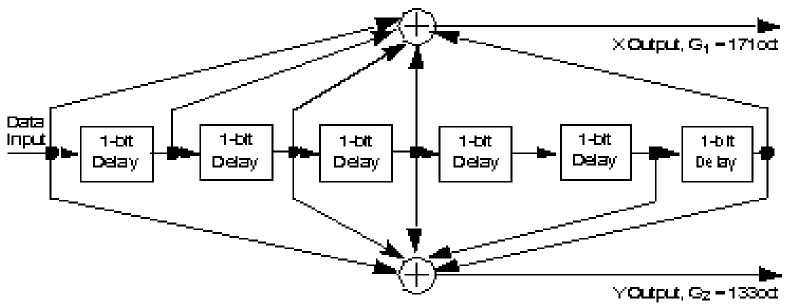
WMAN FEC, consisting of the concatenation of a Reed-Solomon outer code and a rate-compatible convolutional inner code, supports uplink and downlink. BTC and CTC support is optional. The Reed-Solomon convolutional coding rate 1/2 is used as the coding mode when requesting access to the network and in the FCH burst. Encoding is performed by first passing data in block format through the RS encoder.

Reed-Solomon encoding is derived from a systematic RS(N=255, K=239, T=8) code using GF(2^{8}), where N is the number of overall bytes after coding, K is the number of data bytes before coding and T is the number of the data bytes that can be corrected. 802.16d systems uses much smaller code blocks by puncturing the large code blocks down to the required size.

In the *WMAN_CodedSignals* example a CoderRS component is used to generate the RS code based on 802.16d.

Each RS block is followed by the binary convolutional encoder with native rate of 1/2, a constraint length of 7, using polynomial codes to drive its code bits; the encoder is illustrated below.

Convolutional Encoder, Rate 1/2



Convolutional coded data will be punctured before interleaving. Puncturing patterns and serialization order used to realize different code rates are given in Inner Convolutional Code with Puncturing, where 1 denotes a transmitted bit, 0 denotes a removed bit, and X and Y are in reference to Convolutional Encoder, Rate 1/2.

Inner Convolutional Code with Puncturing

	Code Rates			
Rate	1/2	2/3	3/4	5/6
dfree	10	6	5	4
X	1	10	101	10101
Y	1	11	110	11010
XY	X1Y1	X1Y1Y2	X1Y1Y2X3	X1Y1Y2X3Y4X5

Channel Coding Rates gives the block sizes and the code rates used for different modulations and code rates. As 64QAM is optional, modulation codes are implemented only if modulation is implemented.

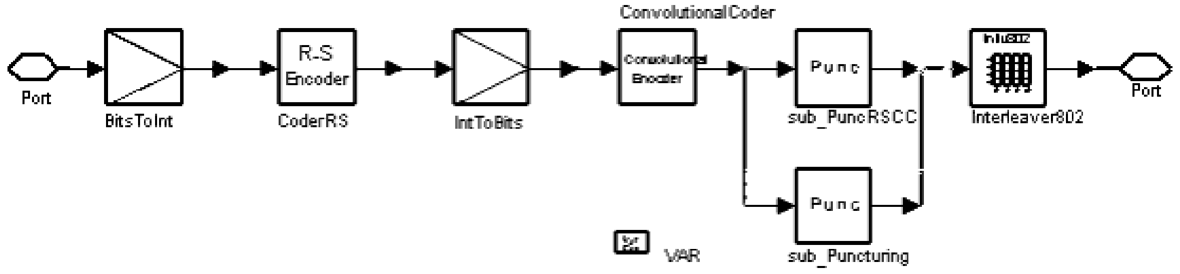
Channel Coding Rates

Modulation	Uncoded Block Size (bytes)	Coded Block Size (bytes)	Overall Coding Rate	RS Code	CC Code Rate
QPSK	24	48	1/2	(32, 24, 4)	2/3
QPSK	36	48	3/4	(40, 36, 2)	5/6
16QAM	48	96	1/2	(64, 48, 8)	2/3
16QAM	72	96	3/4	(80, 72, 4)	5/6
64QAM	96	144	2/3	(108, 96, 6)	3/4
64QAM	108	144	3/4	(120, 108, 6)	5/6

An interleaver is used for coded signals. All encoded data bits are interleaved by a block interleaver with a block size corresponding to the number of coded bits per the allocated subchannels per OFDM symbol N_{cbps}. The interleaver is defined by a 2-step permutation: the first ensures that adjacent coded bits are mapped onto nonadjacent subcarriers; the second ensures that adjacent coded bits are mapped alternately onto less or more significant bits of the constellation, thus avoiding long runs of low reliable bits. The Interleaver802 component performs the 2-step interleaving for the WMAN system.

The *sub_RS_CC* shown in FEC Subnetwork sub_RS_CC demonstrates how to generate the fully-coded signal using an RS-CC coding scheme based on 802.16d.

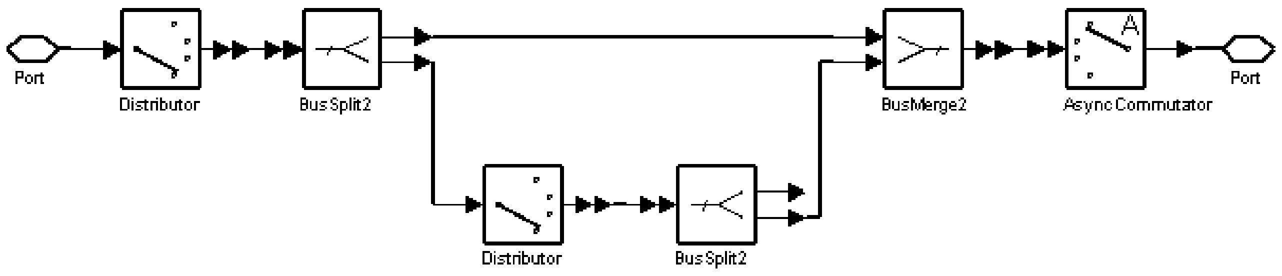
FEC Subnetwork *sub_RS_CC*



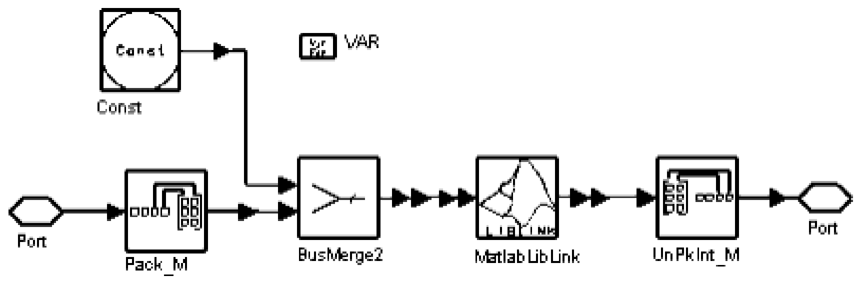
This subnetwork includes a Reed-Solomon encoder component CoderRS, a convolutional encoder component ConvolutionalCoder, and interleaver component Interleaver802. Two subnetwork puncturing components were built for this design; by default *sub_PuncRSCC* is activated and *sub_Puncturing* is deactivated.

- The *sub_PuncRSCC* subnetwork shown in sub_PuncRSCC Puncturing for CC Code Rate 2/3 is used for puncturing coded data for CC code rate 2/3 only (see *Channel Coding Rates*). If all CC code rates need to be supported, several subnetworks are needed using an IfElse component to switch the subnetwork for different RateID.
- The *sub_Puncturing* subnetwork shown in sub_Puncturing Puncturing for All CC Code Rates supports all CC code rates defined in 802.16d.
  To import Matlab functions for puncturing, a MatlabLibLink Function parameter is specified to the Matlab function *rsccpunc.m* that is created based on the puncturing given in *Channel Coding Rates*. This simple Matlab *m* file can be found in *WMAN_802_16d_TX_wrk/data*. For details regarding MatlabLibLink, refer to *MATLAB Cosimulation Introduction* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

***sub_PuncRSCC* Puncturing for CC Code Rate 2/3**



***sub_Puncturing* Puncturing for All CC Code Rates**

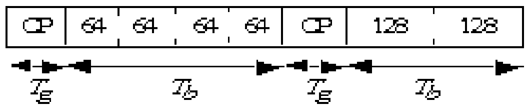


Preambles

All preambles are structured as either one of two OFDM symbols as specified in Section 8.3.3.6 Draft IEEE 802.16d Std [1].

The first preamble in the downlink PHY PDU (as well as the initial ranging preamble) consists of two consecutive OFDM symbols (the combination of the two OFDM symbols is referred to as the long preamble). The first OFDM symbol uses only subcarriers indices that are a multiple of 4. As a result, the time domain waveform of the first symbol consists of 4 repetitions of 64-sample fragment, preceded by a cyclic prefix (CP). The second OFDM symbol uses only even subcarriers, resulting in a time domain structure with 2 repetitions of a 128-sample fragment, preceded by a CP. The time domain structure is illustrated below.

Downlink and Network Entry Preamble Structure



The frequency domain sequences for all full-bandwidth preambles are derived from the sequence:
Pall(-100:100)={1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1+j, 1+j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, 1+j, 1+j, -1+j, 1-j, -1+j, -1+j, 1-j, -1+j, 1-j, 1-j,1+j, -1-j, -1-j, -1-j, -1+j, 1-j, -1-j, -1-j, 1+j, -1-j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1+j, 1-j, -1+j,-1+j, -1-j, 1+j, 0, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, 1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, -1-j, -1+j, 1-j, 1+j, 1+j, -1+j, 1-j, 1-j, 1-j, -1+j, 1-j, -1-j, -1-j, -1-j, 1+j,1+j, 1+j, 1+j, -1-j, -1+j, -1+j, 1+j, -1-j, 1-j, 1-j, 1+j, -1-j, -1-j, -1-j, 1+j, -1-j, -1+j, -1+j, -1+j, 1-j, 1-j, 1-j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, -1+j, -1+j, -1-j, 1+j, 1+j, 1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j, -1+j, -1+j, 1-j, -1-j, -1-j, 1-j, -1+j, -1-j, -1-j, 1-j, -1+j, -1+j, -1+j, 1-j, -1+j,1+j, 1+j, 1+j, -1-j, -1-j, -1-j, -1-j, 1+j, 1-j, 1-j}}

The frequency domain sequence for the *4 times 64 sequence* P 4x64 is defined by:

$$P_{4\times 64(k)} = \begin{cases} \sqrt{2}\times\sqrt{2}\times conj(P_{ALL}(k)) & k_{mod4} = 0 \\ 0 & k_{mod4} \neq 0 \end{cases}$$

The frequency domain sequence for the *2 times 128 sequence* P EVEN is defined by:

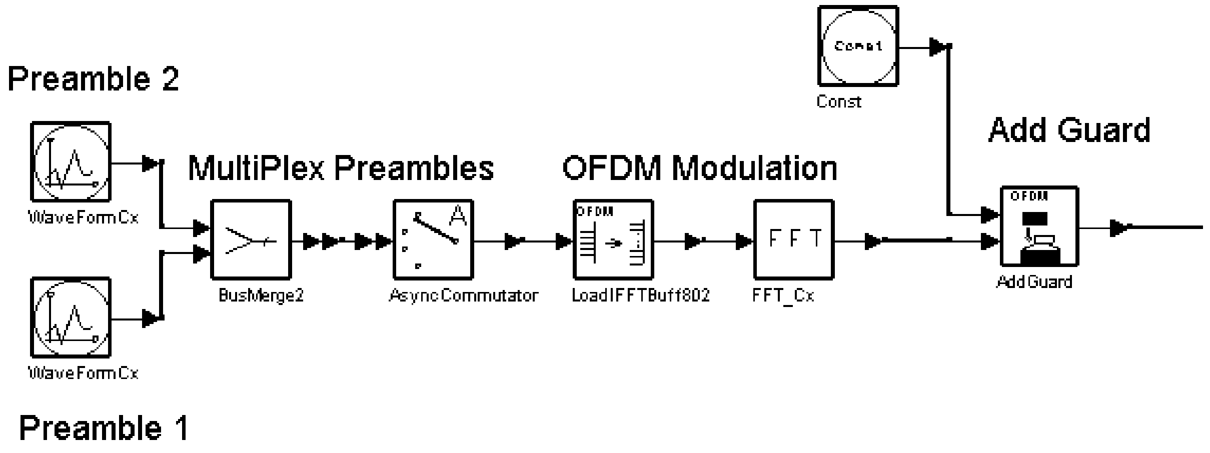
$$P_{EVEN(k)} = \begin{cases} \sqrt{2}\times P_{ALL}(k) & k_{mod2} = 0 \\ 0 & k_{mod2} \neq 0 \end{cases}$$

Long Preamble Generation shows generation of the long preamble for a WMAN FDD downlink transmitter.

- Data file *Preamble_1_16d.txt* (located at *WMAN_802.16d_TX_wrk/data)* based on the *full-bandwidth preambles* and *4 times 64 sequence* equations can be used for the frequency preamble with 4 times 64 sequence. This Preamble 1 will be generated by using a WaveFormCx component referring to data file *Preamble_1_16d.txt*.
- Using another WaveFormCx component referring to data file *Preamble_2_16d.txt* based on *full-bandwidth preambles* and *2 times 128 sequence* equations, Preamble 2 with 2 times 128 sequence will also be generated.

BusMerge2 and AsyncCommutator components are used to multiplex Preamble 1 and Preamble 2. The long preamble through LoadIFFTBuff802, FFT_Cx, and AddGuard form OFDM symbols with guard interval.

Long Preamble Generation



FCH Structure

As specified in Section 8.3.4.1 Draft IEEE 802.16d Std [1], the FCH contains downlink frame prefix to specify the burst profile and length of downlink burst 1. Downlink frame prefix fields are:

- Rate_ID Defines the burst profile of the following burst. Encoding is specified in *OFDM Rate ID Encoding*.
- Length Number of OFDM symbols (PHY payload) in the burst immediately following the FCH burst.
- HCS An 8-bit header check sequence used to detect errors in the downlink frame prefix.

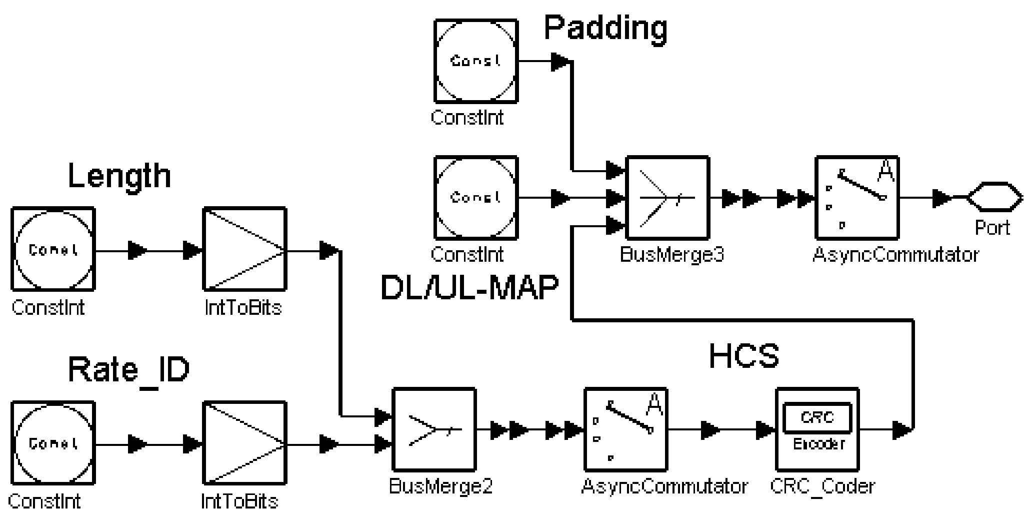
OFDM Rate ID Encoding

Rate_ID	Modulation RS-CC Rate
0	QPSK 1/2
1	QPSK 3/4
2	16QAM 1/2
3	16QAM 3/4
4	64QAM 2/3
5	64QAM 3/4
6 - 15	Reserved

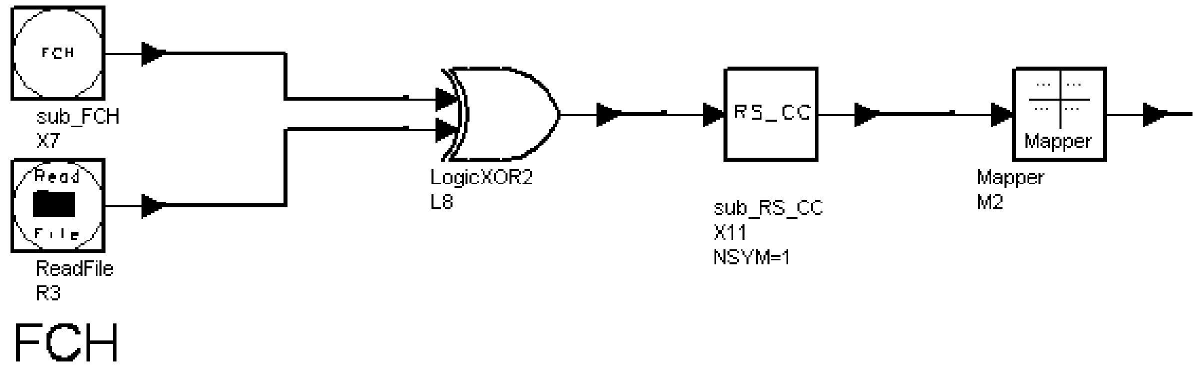
The basic content of the FCH symbol is the downlink frame prefix implemented in *sub_FCH* (sub_FCH FCH Structure). In the FCH, key parameters RateID and Length are included in the header. The HCS generation can be modeled by a CRC check, where the transmitter takes the Rate_ID and Length bytes as the input of the CRC encoder and outputs the HCS code.

As can be seen in Scrambling, Channel Coding, and Mapping for FCH Symbol, the FCH symbol from sub_FCH will be scrambled by the scramble sequence from ReadFile and LogicXOR2, channel coded through sub_RS_CC channel coder, mapped by Mapper, then ready for framing the WMAN signal. (Scrambler, RS-CC channel coding, and mapping were discussed in the section Basic Components.)

***sub_FCH* FCH Structure**



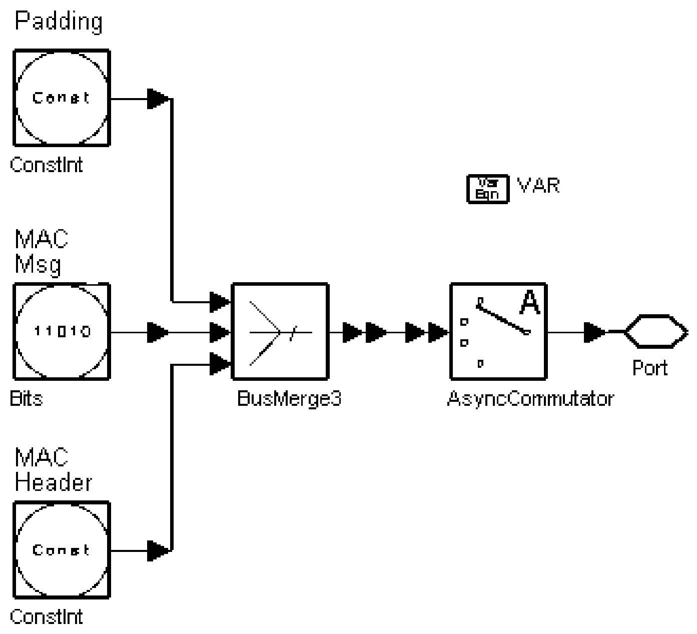
Scrambling, Channel Coding, and Mapping for FCH Symbol



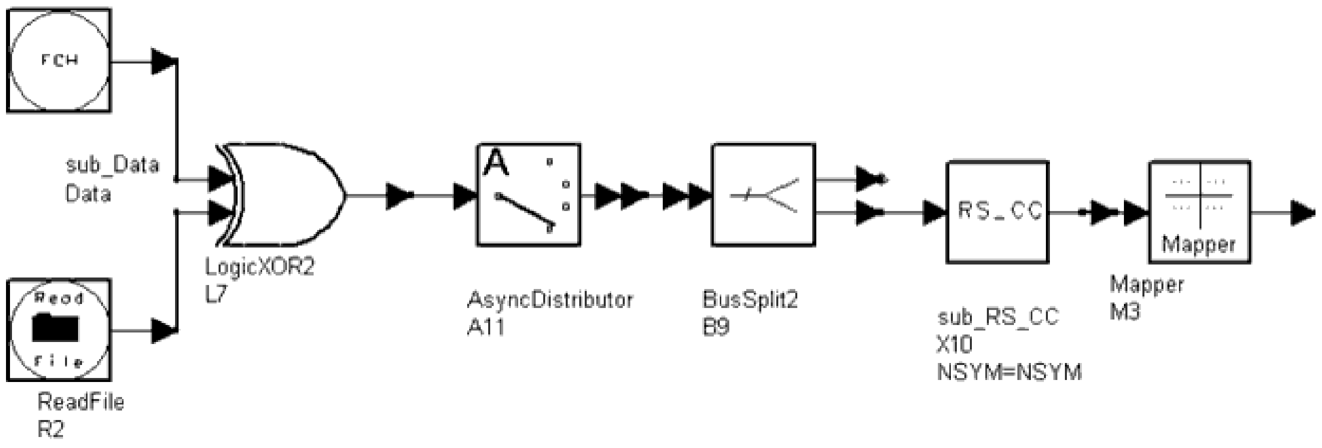
Downlink Burst Generation

The *sub_Data,* shown in sub_Data Downlink Burst Generation, generates the WMAN downlink burst (formed by *MAC Header*, *MAC Msg*, and *Padding*). The input data stream to the modulation is selected as random data with a specific data length. In Scrambling, Channel Coding, and Mapping for Data Symbols, packed data is scrambled by ReadFile and LogicXOR2, channel-coded through sub_RS_CC, mapped with Mapper, and ready for framing the WMAN signal. (Scrambler, RS-CC channel coding, and mapping were discussed in the section Basic Components.)

***sub_Data* Downlink Burst Generation**



Scrambling, Channel Coding, and Mapping for Data Symbols



OFDM Modulation

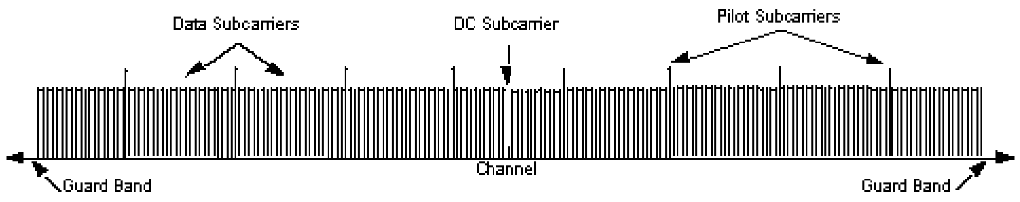
The WMAN physical layer is based on OFDM modulation.

An OFDM symbol is made up of subcarriers, the number of which determines the FFT size as illustrated in OFDM Symbol. WMAN subcarriers types include:

- Data subcarriers for data transmission.
- Pilot subcarriers for various estimation purposes.
- Null subcarriers (no transmission at all) for guard band and DC subcarrier.

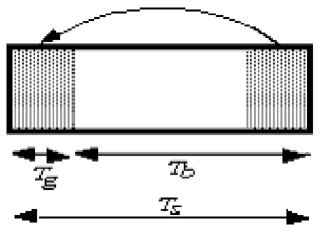
The guard band (illustrated in OFDM Symbol Time Structure) enables the signal to naturally decay and create FFT *brick wall* shaping.

OFDM Symbol



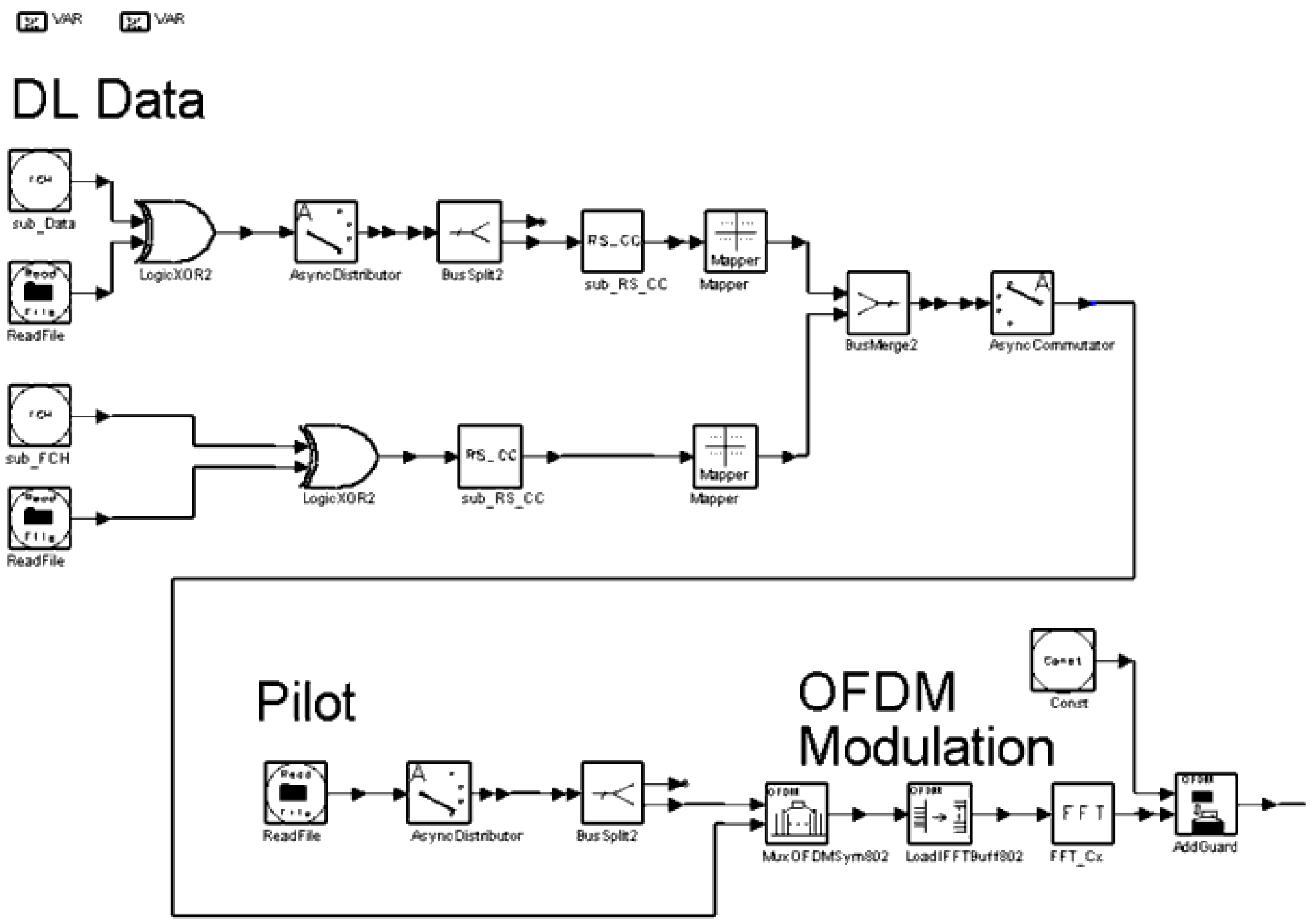
Inverse-Fourier-transforming creates the OFDM waveform; this time duration is referred to as the useful symbol time T_b. A copy of the last T_g of the useful symbol period CP is used to collect multipath while maintaining the orthogonality of the tones. OFDM Symbol Time Structure illustrates this OFDM symbol structure in the time domain.

OFDM Symbol Time Structure



OFDM Modulation shows OFDM modulation in ADS. Downlink data and FCH signal through channel coding and mapping are multiplexed. MuxOFDMSym802 then multiplexes pilot and data carriers to form WMAN OFDM symbols in the frequency domain. LoadIFFTBuff802 and FFT_Cx then perform an inverse-FFT to form the WMAN OFDM symbols in the time domain. AddGuard adds a guard interval to complete the OFDM symbols.

OFDM Modulation



Measurements

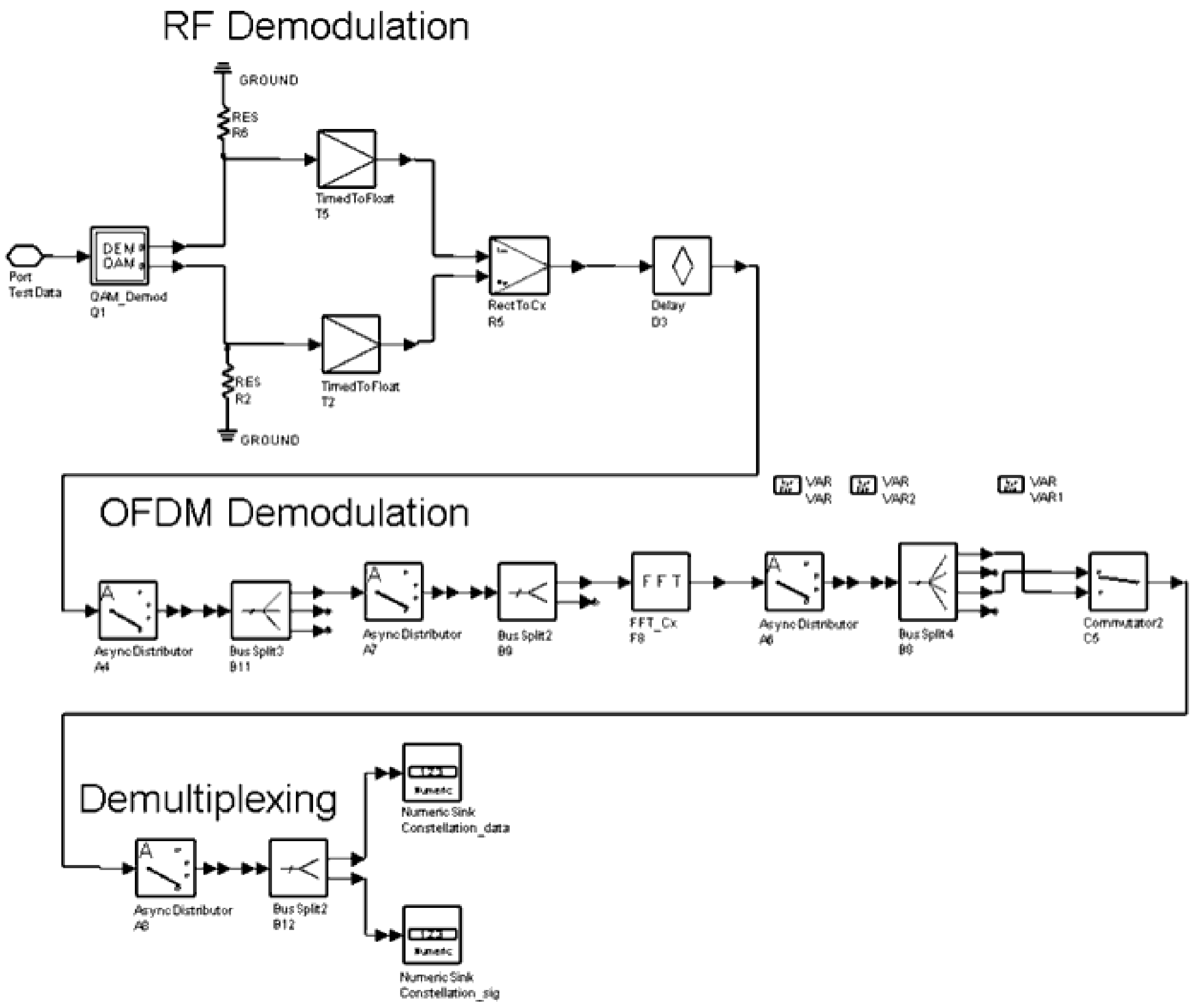
Measurements are provided for waveforms, spectrum, power, and constellation.

TimedSink models are directly used to display waveforms for preamble, FCH, medium access control data, and whole framed signals.
SpectrumAnalyzerResBW is used to measure the spectrum for the WMAN signals.

Signal power is measured in the region that does not include signal idle. The *total_pwr* expression in the data display window is used with two data display markers for specifying region. For CCDF, WMAN downlink frame can be measured by using *power_ccdf* in the data display window with two data display markers for specifying the region to be measured as shown in Examples.

For the WMAN constellation measurement, *sub_WMAN_Constellation* is used. As shown in sub_WMAN_Constellation Constellation Measurement this design integrates RF demodulation, OFDM demodulation, demultiplexing for Data and SIGNAL, and sinks for displaying Data as well as Signal constellations. NumericSink *Constellation_data* displays 16QAM constellation for data and BPSK Constellation for the pilot; NumericSink *Constellation_sig* displays FCH SIGNAL constellations.

***sub_WMAN_Constellation* Constellation Measurement**



WMAN Design Example Descriptions

The *WMAN_802_16d_wrk* includes: *Test_WMAN_CodedSignals* for fully-coded signal generation; *Test_WMAN_RFSource* for transmitter test; and *Test_WMAN_ESG* for downloading a WMAN signal to an ESG. These designs are described in the following sections. Simulation will generate single bursts of data, formatted for downlink in the mandatory coding schemes. (The optional FEC features are not supported.)

Fully-Coded Signal Generation

Test_WMAN_CodedSignals demonstrates how to build an OFDM frame structure for the WMAN frequency division duplex downlink (FDD DL) system in ADS; the schematic is shown in Test_WMAN_CodedSignal Schematic.

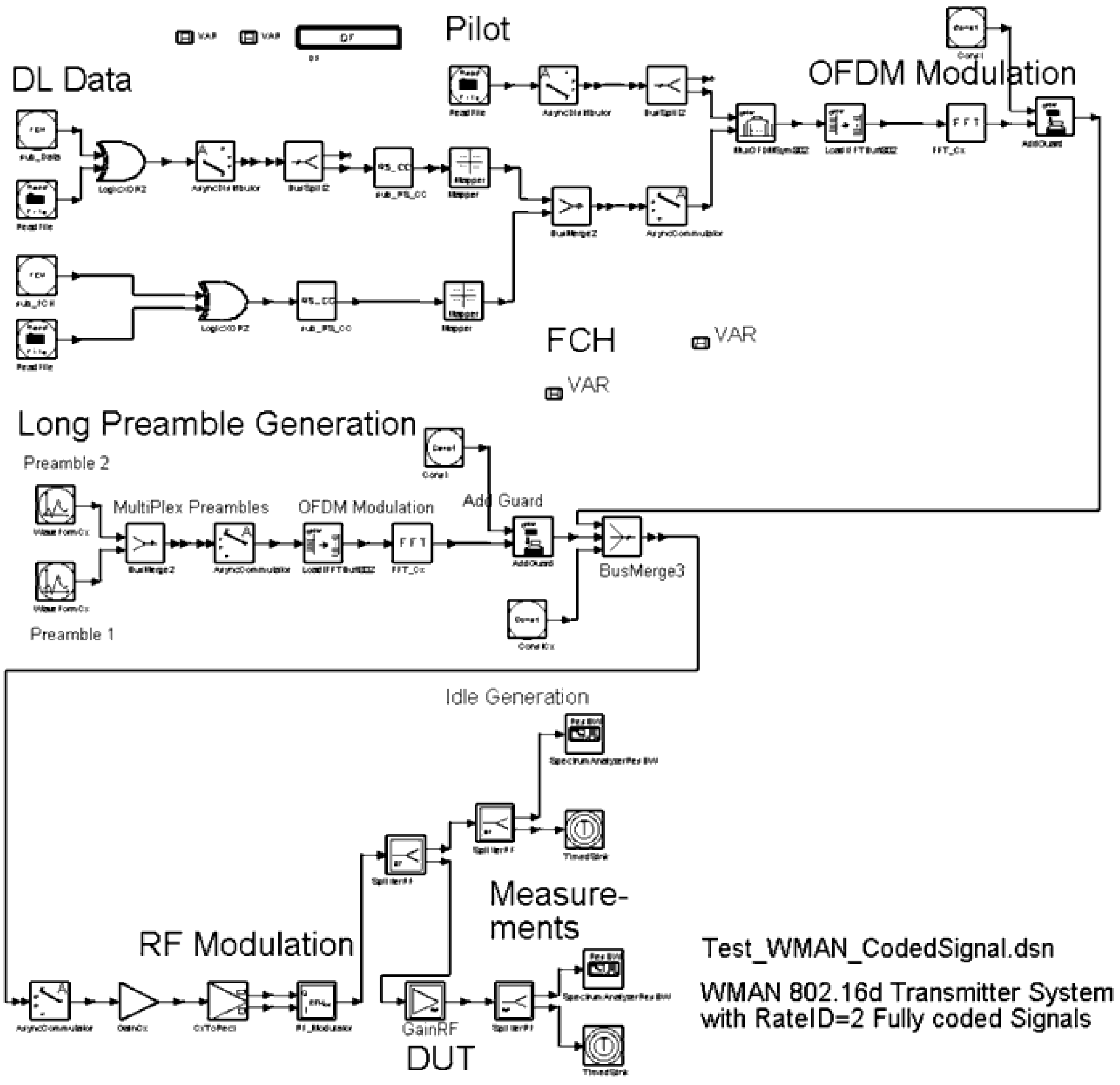
The main components are provided at the subsystem level and include long preamble, frame control header (FCH) and FDD DL data generation, OFDM modulation, multiplexing, RF modulation, and measurements. Signals are fully coded by RS-CC encoding and framed based on the 16d Standard.

An RF modulator for modulation of the fully-coded WMAN signal to the RF carrier frequency is followed by an RFGain power amplifier as the DUT.

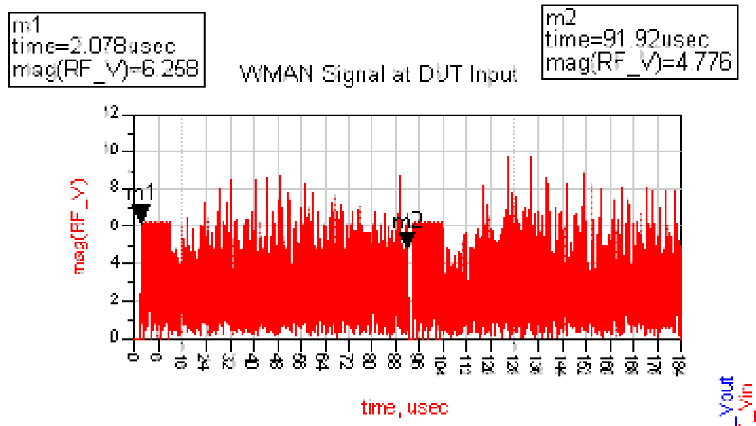
To show system performance in time as well as frequency domains, TimeSink and SpectrumAnalyzerResBW are used for both input and output of the DUT.

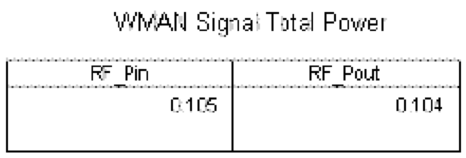
- In the time domain, the amplitude of the framed WMAN signal is displayed first, total power and CCDF are then measured using *total_pwr* and *power_ccdf* expressions; simulation results are shown in Power and CCDF Measurement Results.
- In the frequency domain, WMAN signal spectrum is measured for both input and output of the DUT; simulation results are shown in Spectrum Measurement Results.

***Test_WMAN_CodedSignal* Schematic**

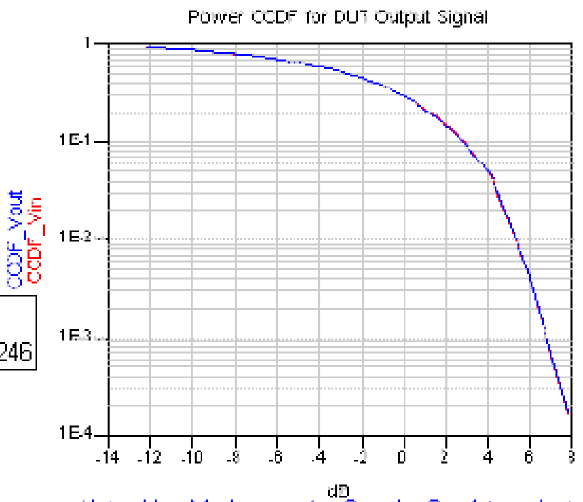


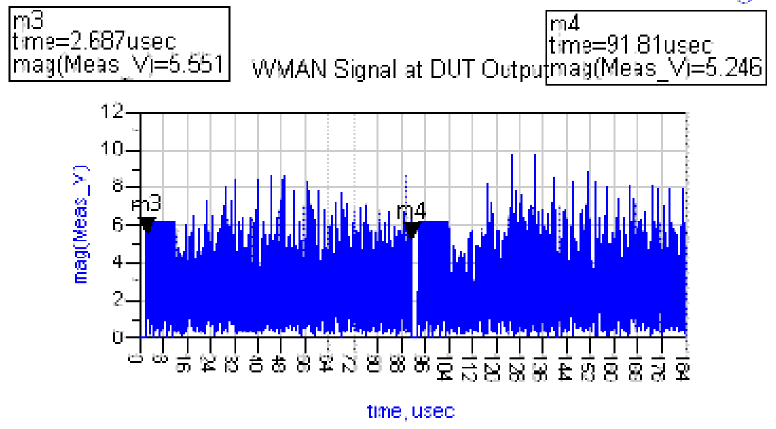
Power and CCDF Measurement Results




WMAN Signal Total Power

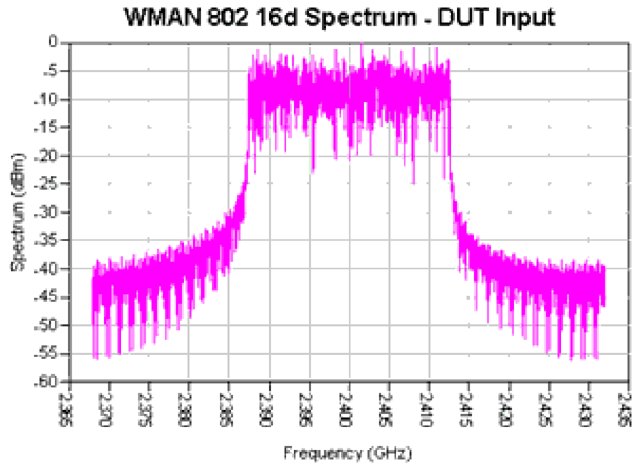
RF_Pin	RF_Pout
0.105	0.104

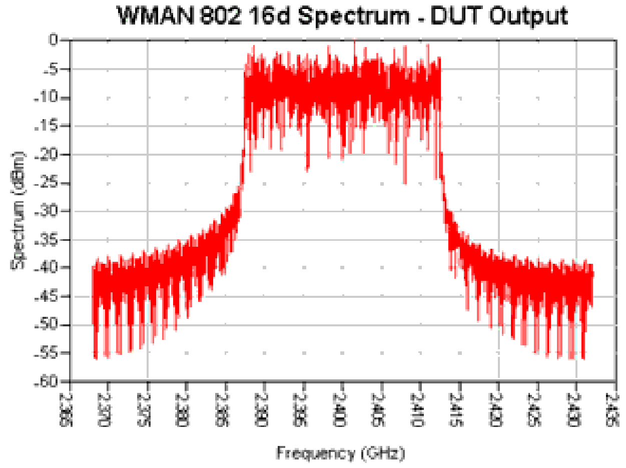




Note: Use Markers, m1-m2 and m3-m4 to select a section of data for measuring CCDF and total Power

Spectrum Measurement Results





Default settings for basic signal information are listed below.

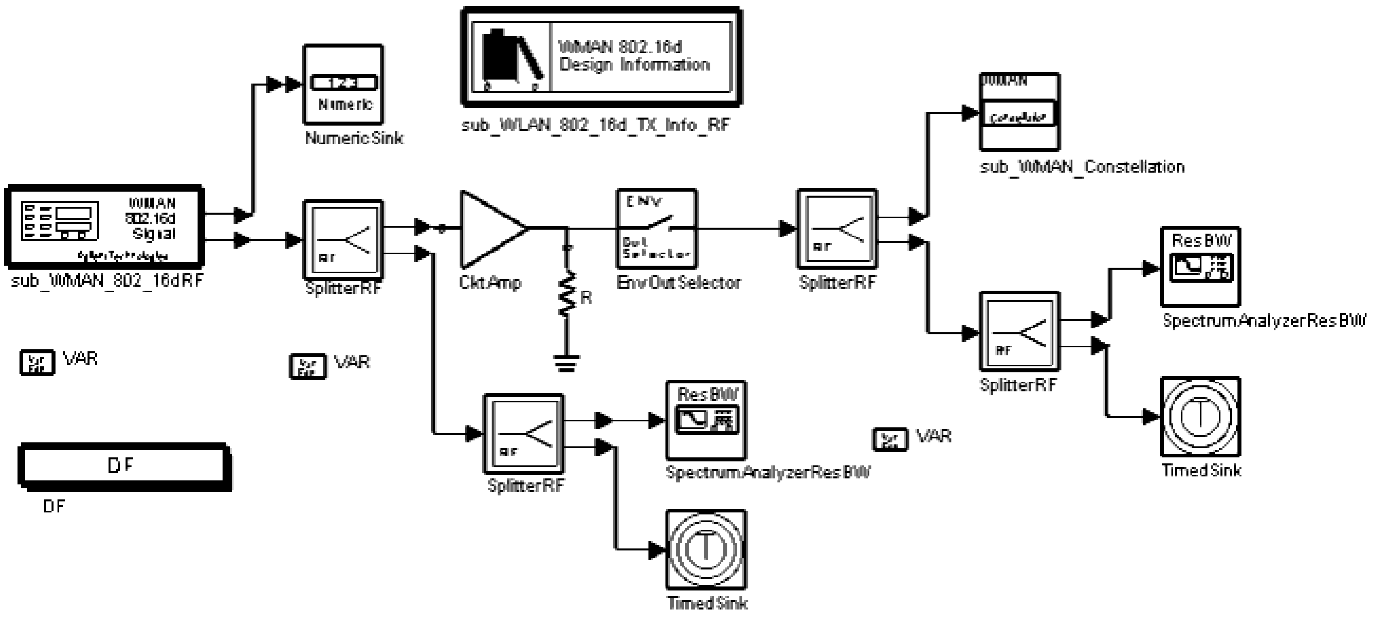
Default Settings for WMAN Measurements

Parameter	Descriptions	Default Setting
FSource	Source carrier frequency	2.4 GHz
SourceR	Source resistor	50 Ohm
Source Power	Source power	20 dBm
Bandwidth	System bandwidth	20m MHz
RateID	Rate ID	2 16QAM, coded block size 48, uncoded block size 96, overall coding rate 1/2
Data Length	Data length in bytes	256
FFT size	FFT size	512
DL Frame Time	FDD Downlink frame time	92 us
Guard Interval	Guard interval	1/4
Idle Interval	Idle interval time	2 us
Data Sub-carriers	Number of subcarriers for data	200
Pilot Carriers	Number of subcarriers for pilot	8
Measured Frames	Number of frames measured	2

Transmission Test

Test_WMAN_RFSource tests WMAN transmission; the schematic is shown in Test_WMAN_RFSource Schematic.

***Test_WMAN_RFSource* Schematic**



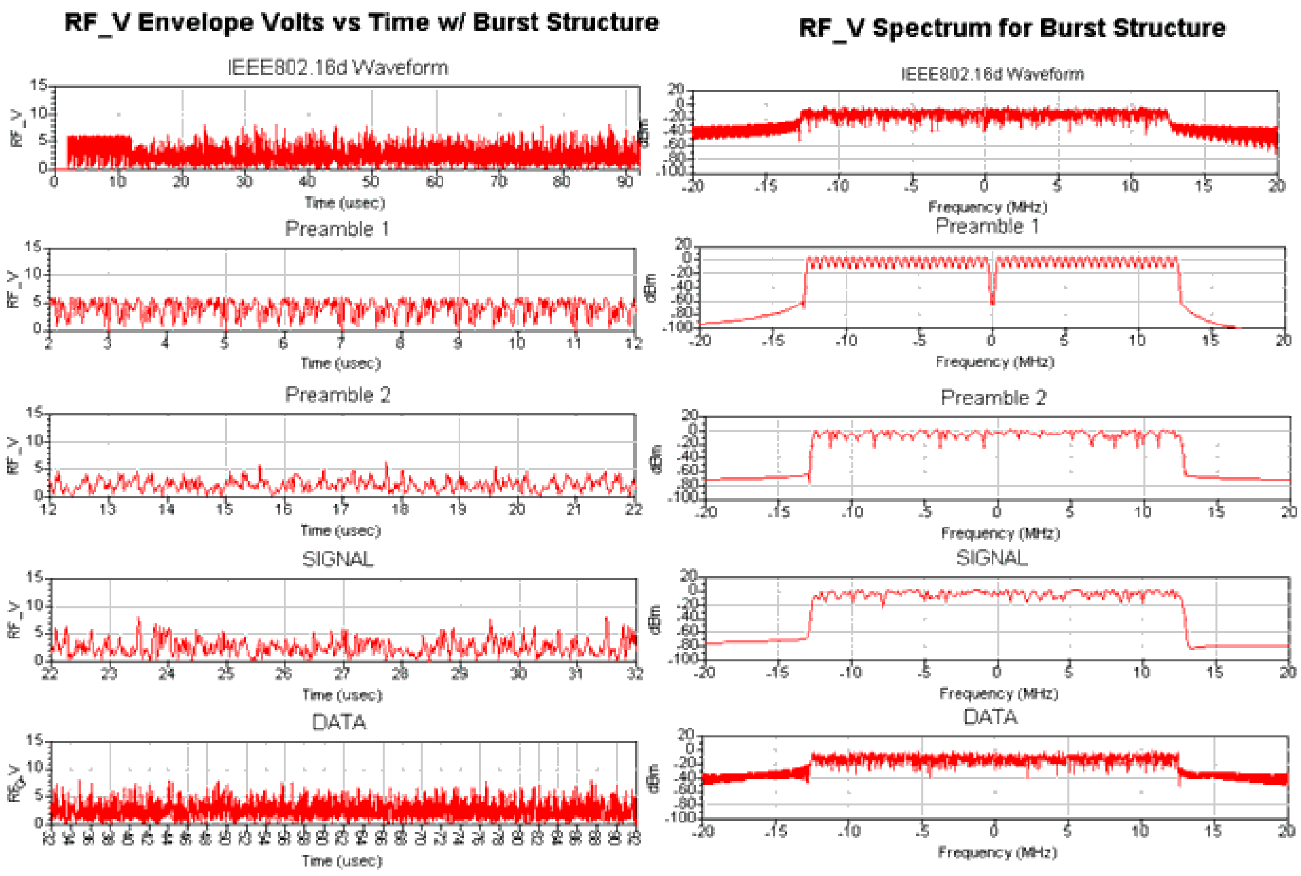
The top level of this schematic consists of: WMAN source (*sub_WMAN_802_16dRF*); DUT (CktAmp with EnvOutSelector); and measurements.

sub_WMAN_802_16dRF is a local subnetwork component to generate a partially-coded WMAN signal. By pushing into this subnetwork, we can see the design is the same as the signal source in Test_WMAN_CodedSignal Schematic, except there is no FEC in *sub_WMAN_802_16dRF*. For the transmission test, basic performance including spectrum, power, CCDF, and constellation measurements will produce the same results with or without FEC.

Key parameters defined in *Signal_Generation_Vars* and *Measurement_Vars*, provide an easy way to configure the transmitter at the top-level design. The DUT can be replaced by customer's DUT that will then be measured for performance.

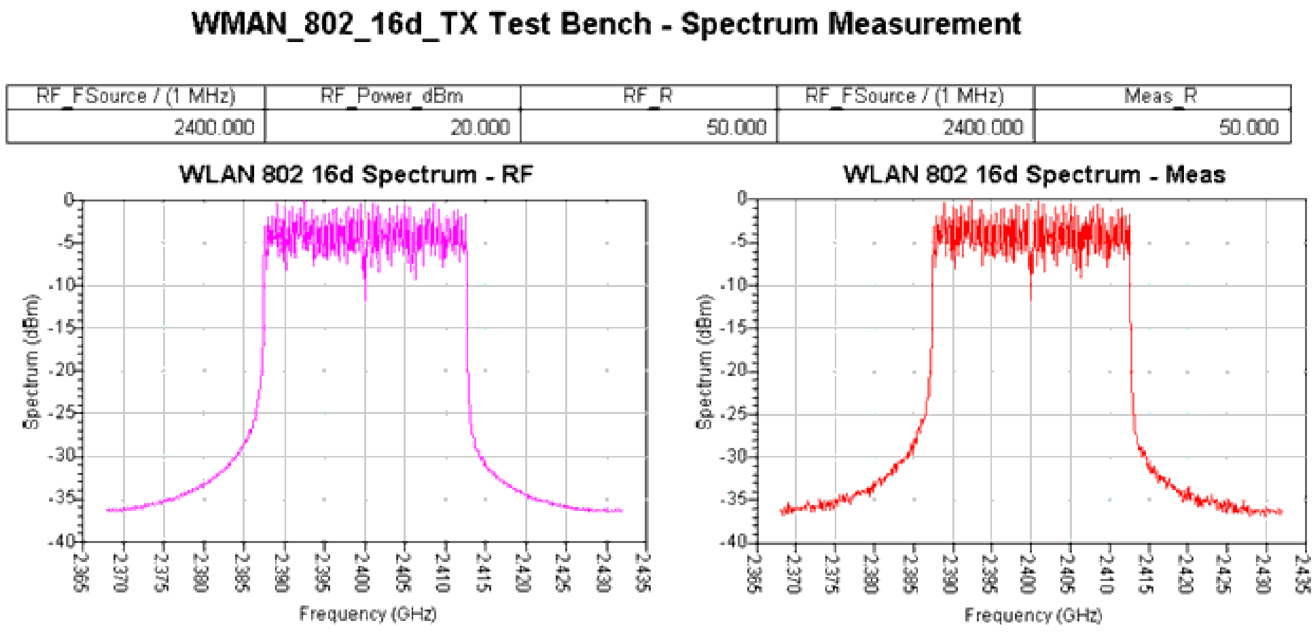
The RF Envelope measurement is used to show the time envelope and spectrum of each field in the 802.16d RF signal frame: preambles, FCH and DL Data fields. Two signals are tested, the RF source signal at the input of the RF DUT and the Meas signal at the output of the RF DUT. RF envelope time and spectrum measurements are implemented for each signal. Results are shown below.

Time Envelope and Spectrum of Each Frame Field



SpectrumAnalyzerResBW is used to measure the spectrum for the WMAN signals. Results are shown below.

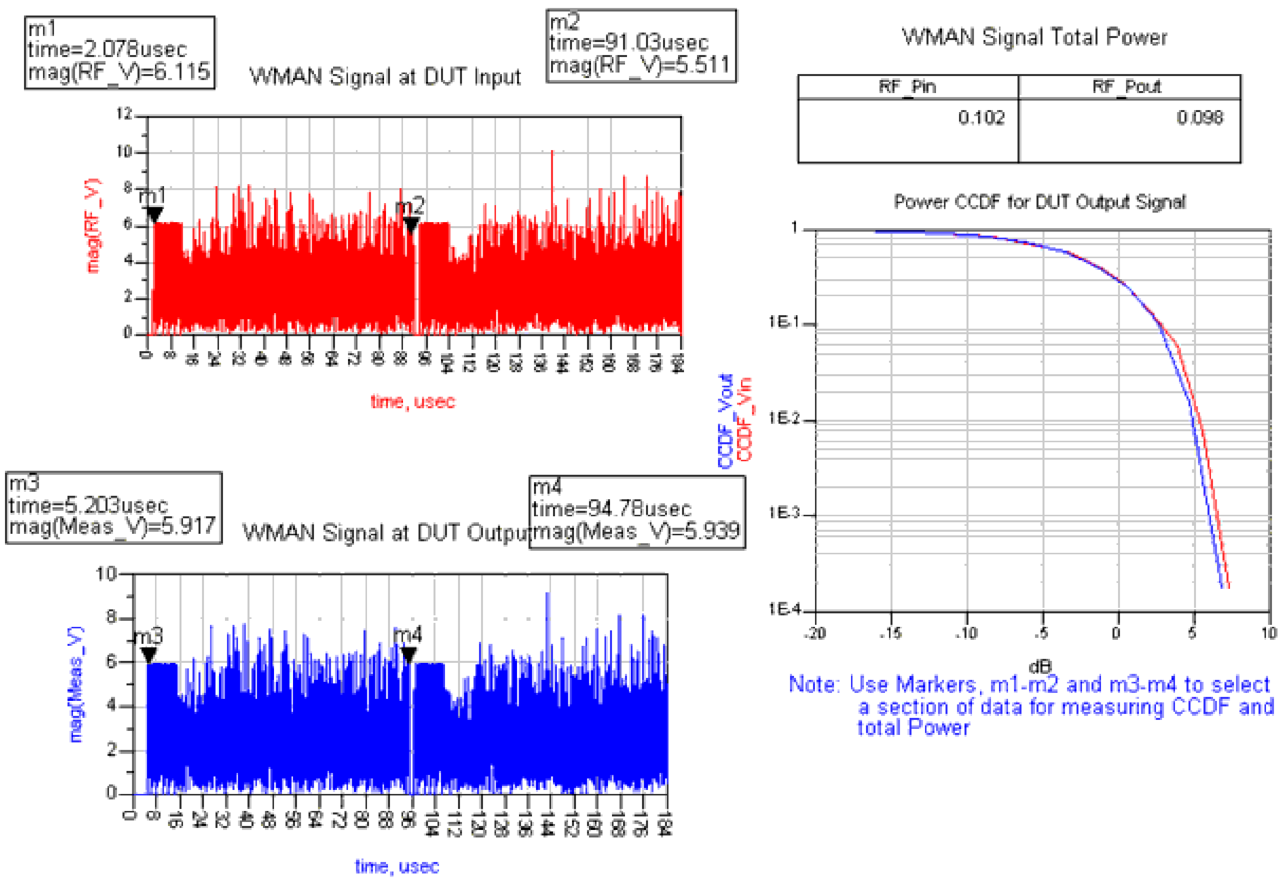
Spectrum Measurement Results



RF_FSource / (1 MHz)	RF_Power_dBm	RF_R	RF_FSource / (1 MHz)	Meas_R
2400.000	20.000	50.000	2400.000	50.000

Power and CCDF measurement results are shown in Power and CCDF Measurement Results. The downlink burst can be measured by using the *power_ccdf* measurement expression based on the DUT input and output waveforms.

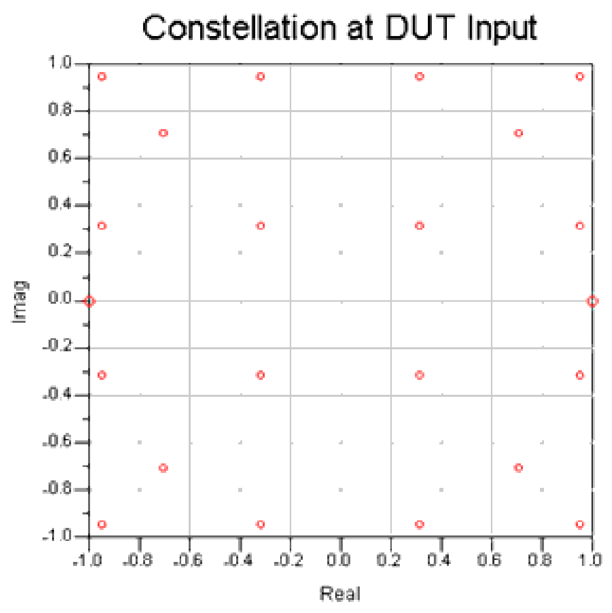
Power and CCDF Measurement Results

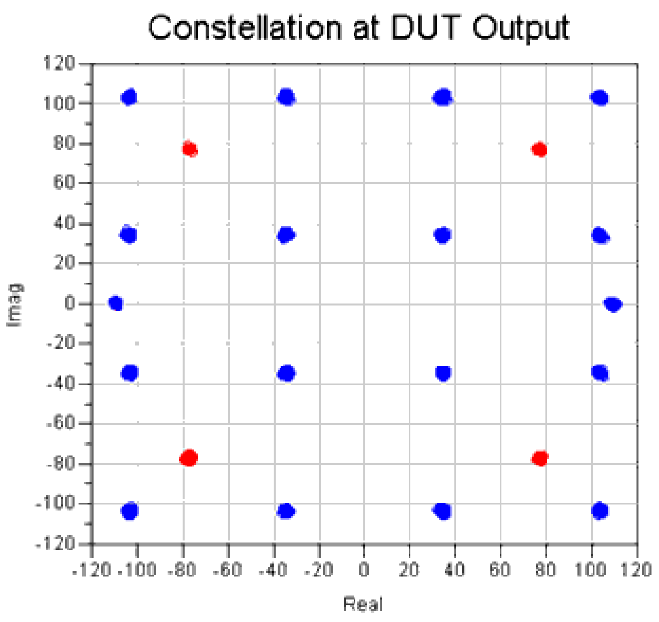


RF_Pin	RF_Pout
0.102	0.098

Constellation measurement results shown below include BPSK constellation for pilot signal, QPSK for FCH, and 16QAM for medium access control data.

Constellation Measurement Results

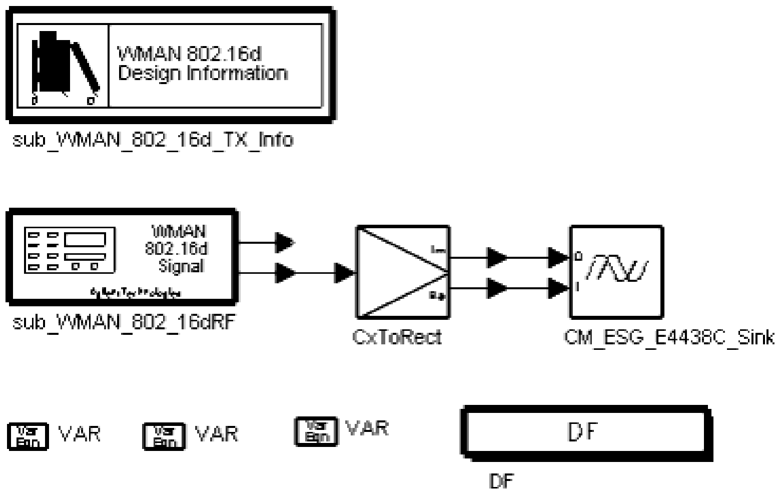




Signal Downloading to ESGc

Test_WMAN_ESG generates and downloads a WMAN signal to an Agilent ESG signal generator; the schematic is shown below.

***Test_WMAN_ESG* Schematic**



The RF signal generated by *sub_WMAN_802_16dRF* is converted to I and Q data through CxToRect and sent to CM_ESG_E4438C_Sink to download data to the ESGc (E4438C). The downloaded framed signal can drive ARB signal generator in ESGc for generating a test signal for WMAN system, sub-system, and component tests.

A WMAN power amplifier DUT can be tested using this WMAN signal. Basic system performances can be measured using Agilent 89600 Series Vector Signal Analyzer (VSA) for spectrum as well as waveforms.

Key Parameters

Each design in *WMAN_802_16d_TX_wrk* contains VAR blocks for ease of setting key parameters. Parameter settings are described here.

Signal_Generation_Vars:

- FSource specifies RF carrier frequency.
- SourcePower specifies source output power in dBm or W.
- BandOption specifies system bandwidth 1.75, 3.5, 7, 14, or 28 MHz; values are BandOption=0, 1, 2, 3, 4, respectively. Other bandwidths are not supported. If bandwidth < 0, set BandOption=0; if bandwidth >4, set BandOption=4.
- Rate_ID specifies data modulation and channel coding types. *Channel Coding Rates* lists RateID parameters of 802.16d associate with coding rate per modulation. For

example for RateID=2, modulation type is specified as 16QAM and overall coding rate is 1/2.

- DataLength is used to set the number of data bytes in a frame (or burst). There are 8 bits per byte.
- OversamplingOption sets the oversampling ratio of 802.16d RF signal source. Options from 0 to 4 result in oversampling ratio 1, 2, 4, 8, 16 where oversampling ratio = $2^{OversamplingOption}$. If oversampling ratio < 0, set OversamplingOption=0; if oversampling ratio >4, set OversamplingOption=4. If the oversampling ratio = 2^2 = 4 and the simulation RF bandwidth is larger than the system bandwidth by a factor of 4 (e.g. for Bandwidth=14 MHz, the simulation RF bandwidth = 14 MHz × 4 = 16 MHz). The FFT size is determined by OversamplingOption. FFTsize=256 × $2^{OversamplingOption}$. When OversamplingOption=0, 1,2,3,4, FFTsize=256,512,1024,2048 and 4096.
- IdleInterval specifies the idle interval between two consecutive frames when generating an 802.16d signal source.
- GuardInterval is used to set cyclic prefix in an OFDM symbol. The value range of GuardInterval is [0.0,1.0]. The cyclic prefix is a fractional ratio of the IFFT length. In 802.16d, GuardInterval=1/32, 1/16, 1/8, 1/4 of the useful OFDM symbol time.

Measurement_Vars (*Test_WMAN_RFSouce* and *Test_WMAN_CodedSignals*)

- FMeasure specifies the carrier frequency for the measurement.
- Carriers specifies the number of subcarriers for an OFDM signal.
- MeasFrames specifies the number of frames for measuring the Constellation.

ESG_Setting_Vars (*Test_WMAN_ESG*)

- NumberOfSubFrames specifies the number of frames measured.
- SubFrameTime specifies the signal frame time.
- Stop specifies the signal stop time to be sent to the ESG.

References

1. Draft IEEE Standard for Metropolitan Area Networks IEEE P802.16-REVd/D2-2003, Dec, 2003.

Numeric Advanced Comm Components

- *AddGuard* (numeric)
- *ConvolutionalCoder* (numeric)
- *CRC Coder* (numeric)
- *CRC Decoder* (numeric)
- *Deinterleaver802D* (numeric)
- *Demapper* (numeric)
- *Interleaver802* (numeric)
- *LoadIFFTBuff802* (numeric)
- *Mapper* (numeric)
- *MuxOFDMSym802* (numeric)
- *RMSE* (numeric)
- *ViterbiDecoder* (numeric)

Numeric Advanced Communications components provide functions for simulation of advanced communication systems based on the latest communication technologies including wireless metropolitan access networks (WMAN), wireless local access networks (WLAN), and wireless personal access networks (WPAN).

The MuxOFDMSym802, LoadIFFTBuff802, and AddGuard components provide orthogonal frequency division multiplexing (OFDM) modulation. These components can be used for OFDM modulation based on IEEE.802.11a/g, IEEE 802.153a, and IEEE 802.16d standards.

The Mapper and Demapper components provide basic modulation/demodulation and mapping/demapping types BPSK, QPSK, 8PSK, 16QAM, 64QAM, 128QAM, and 256QAM.

The ConvolutionalCoder and ViterbiDecoder components provide convolutional encoding and decoding.

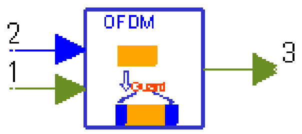
The CRC_Coder and CRC_Decoder components provide code error checking.

The Interleaver802 and Deinterleaver802 components provide interleaving/deinterleaving functionality based on IEEE 802 standards.

The RMSE component provides EVM calculations for designers who want to create subnet measurements.

ADS examples (accessed from the ADS Main window: *File > Open > Example > Com_Sys > WMAN_802_16d_TX_wrk*) demonstrate the use of these components for simulation as well as WLAN and WMAN system testing. *WMAN Example Designs* (numeric) discusses designs in this workspace.

AddGuard



Description: Guard insertion of OFDM symbol
Library: Numeric, Advanced Comm
Class: SDFAddGuard

Parameters

Name	Description	Default	Unit	Type	Range
IFFTSize	IFFT size	64		int	[1, ∞)
PreGuard	Pre-guard length	16		int	[0:IFFTSize]
PostGuard	Post-guard length	0		int	[0:IFFTSize]
Intersection	Guard intersection length	0		int	[0:IFFTSize]

Pin Inputs

Pin	Name	Description	Signal Type
1	In	Transmitted signal after IFFT	complex
2	Window	Window function	real

Pin Outputs

Pin	Name	Description	Signal Type
3	Out	OFDM output data	complex

Notes/Equations

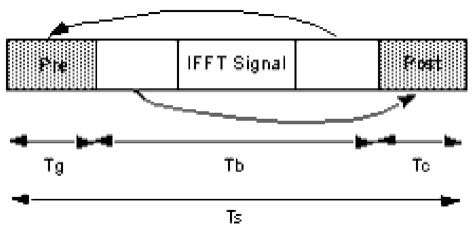
1. This component is used to add a guard interval to IFFT signals, which forms an OFDM symbol. Pre- and post-guard intervals are implemented; all OFDM systems are supported.
2. IFFTSize specifies the input IFFT signal length.
   PreGuard specifies the pre-guard length; PostGuard specifies the post-guard length.
   If PreGuard = 0, there is no pre-guard; if PostGuard = 0, there is no post-guard.
   Intersection specifies the intersect length of two consecutive OFDM symbols. If Intersection = 0, there is no intersect between symbols. To protect the IFFT signals, Intersection cannot exceed PreGuard + PostGuard.
   IEEE 802 series (802.11a, 802.11g, 802.15.3a, 802.16a, 802.16d) and DVB-T standards do not include post-guard and intersection.
3. Each firing IFFTSize tokens are input from pin In.
   PreGuard + IFFTSize + PostGuard tokens are input from pin Window.
   PreGuard + IFFTSize + PostGuard-Intersection tokens are output.
   Pin In is the IFFT signal input, pre-guard and post-guard are added accordingly, which forms an OFDM symbol.
   Pin Window is used to add a window function to the current OFDM symbol; length is PreGuard + IFFTSize + PostGuard. Designers can specify the window values and

input to this pin. The input of this pin can also be set as a constant value.

- If an intersect does not exist, the OFDM symbol multiplies the window, then outputs at pin Out.
- If an intersect does exist, the OFDM symbol multiplies the window; results are output after adding the intersecting parts of the previous OFDM symbol. Then the intersecting parts of the OFDM symbol are stored as intersecting parts for the next OFDM symbol.

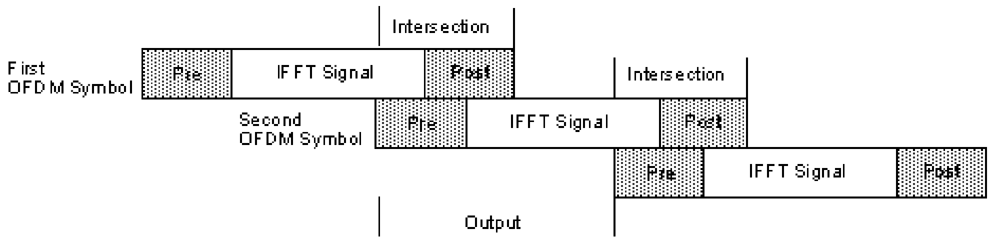
4. An OFDM symbol is formed as described here.
Inverse-Fourier-transforming creates the IFFT signal; time duration is Tb. A copy of the last time duration Tg of the useful symbol period is added before the IFFT signal (this pre-guard is also called *cyclic prefix*). A copy of the last time duration Tc of the useful symbol period is added after the IFFT signal (this post-guard is also called *cyclic postfix*). The combined duration is referred to as symbol time Ts. OFDM Symbol Time with Guard Interval illustrates this sequence.

OFDM Symbol Time with Guard Interval



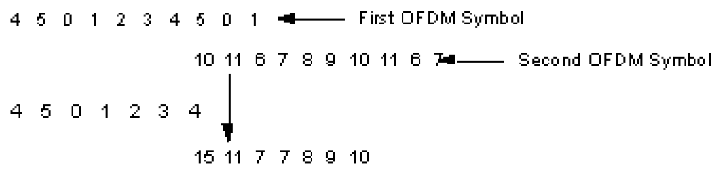
5. Intersection, PreGuard and PostGuard values form consecutive OFDM symbols.
   - Case 1: Intersection > PreGuard, Intersection > PostGuard

Intersection > PreGuard, Intersection > PostGuard



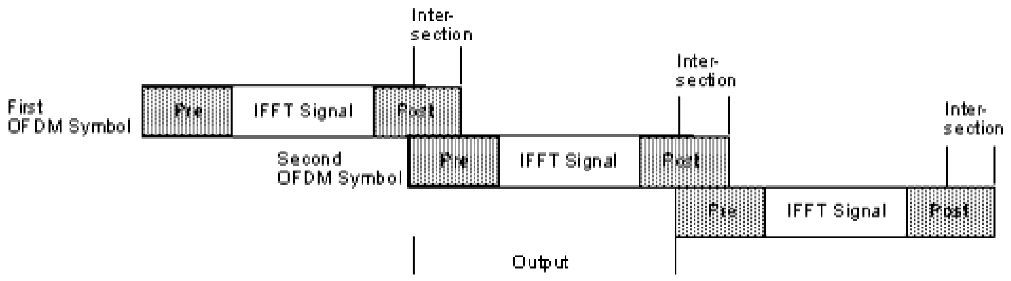
For the IFFT signal of the second OFDM symbol, pre-guard, and post-guard are added. Thus, the second OFDM symbol are formed and multiplied by window. The points with Intersection length of the first and second OFDM symbols are then summed and output first. The points of the second OFDM symbol with length of PreGuard + IFFTSize + PostGuard-Intersection are then output. The points with Intersection length of the second OFDM symbol are stored as intersecting parts for the next OFDM symbol, as described next.
Let the input be {0, 1, 2, 3, 4, 5} and {6, 7, 8, 9, 10, 11}, window is 1, IFFTSize = 6, PreGuard = 2, PostGuard = 2, Intersection = 3. With calculation steps above, the output of the first and second OFDM symbol are {4, 5, 0, 1, 2, 3, 4} and {15, 11, 7, 7, 8, 9, 10}, respectively. Case 1: Calculation for Output illustrates the calculation.

Case 1: Calculation for Output



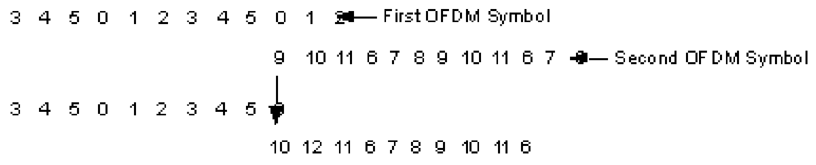
- Case 2: Intersection≤ PreGuard, Intersection≤ PostGuard

Intersection ≤ PreGuard, Intersection ≤ PostGuard



his calculation is similar to Case 1. Let the input be {0, 1, 2, 3, 4, 5} and {6, 7, 8, 9, 10, 11}, window is 1, IFFTSize = 6, PreGuard = 3, PostGuard = 3, Intersection = 2. The output of the first and second OFDM symbols are {3, 4, 5, 0, 1, 2, 3, 4, 5, 0} and {10, 12, 11, 6, 7, 8, 9, 10, 11, 6}, respectively. Case 2: Calculation for Output illustrates the calculation.

Case 2: Calculation for Output

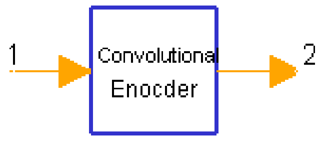


References

1. IEEE Standard 802.11a-1999, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band," 1999.
2. ETSI TS 101 475 v1.1.1, "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer," April, 2000.
3. ARIB-JAPAN, Terrestrial Integrated Services Digital Broadcasting (ISDB-T); Specification of Channel Coding, Frame Structure and Modulation, Sept.1998.
4. ETSI, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. EN300 744 v1.2.1, European Telecommunication Standard, July 1999.
5. IEEE P802.15-03/268r1, "*Multi-band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a*," September 2003.
6. IEEE P802.16-REVd/D2-2003, "*Draft IEEE Standard for Local and metropolitan area*

networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2003.

ConvolutionalCoder



Description: Convolutional coder
Library: Numeric, Advanced Comm
Class: SDFConvolutionalCoder
Derived From: ConvolutionalCodeBase

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
CodingRate	Coding rate: rate_1_2, rate_1_3, rate_1_4, rate_1_5, rate_1_6, rate_1_7, rate_1_8	rate_1_2	R		enum	
ConstraintLength	Constraint length	7	K		int	[3, 14]
Polynomial	Generator polynomial	{0133, 0171}			int array	{2^(K-1)+2*n-1}, n=1,2,3,...2^(K-2).
ZeroTail	Zero tail used to convert convolutional code to block code: NO, YES	NO			enum	
BitSequenceLength	Length of bit squence not including tail bits, valid when ZeroTail=YES	88	N		int	[1,65535]

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input	int

Pin Outputs

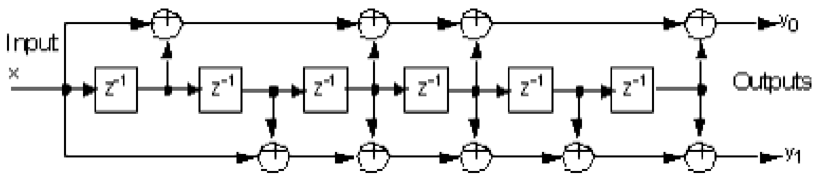
Pin	Name	Description	Signal Type
2	Out	output	int

Notes/Equations

1. This component is used to convolute the input information sequence bit-by-bit. Each firing, 1/CodingRate Out tokens are produced when one In token is consumed. A convolutional code is generated by passing the information sequence to be transmitted through a linear finite-state shift register. The shift register generally consists of K(*k*-bit) stages and *n* linear algebraic function generators. Input data to the encoder (assumed to be binary) is shifted into (and along) the shift register *k* bits at a time. The number of output bits for each *k*-bit input sequence is *n* bits. Therefore, the code rate is defined as $R_c = k/n$, which is consistent with the code rate definition for a block code. The K parameter is called the constraint length of the convolutional code.

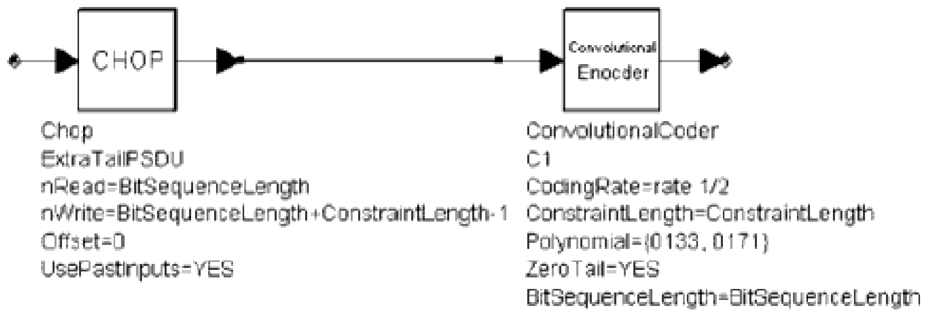
2. CodingRate (R_c) is the ratio of input bit (*k*) and output bits (*n*). ConvolutionalCoder supports the 1/*n* coding rate only, which implements an R_c = 1/*n* rate(*n* = 2, 3, 4, 5, 6, 7, 8) convolution for input data.
   Convolutional codes with *k* /*n* (*k* > 1) are not supported by this component because: coding and decoding will be more complex; and, even convolutional codes with a *k* /*n* (*k* > 1) coding rate are used that are typically implemented by puncturing the convolutional code with a 1/*n* coding rate.
3. ConstraintLength (*K*) represents shift register stages.
4. Polynomial is the generator function of the convolutional code. In general, the generator matrix for a convolutional code is semi-infinite since the input sequence is semi-infinite. As an alternative to specifying the generator matrix, a functionally equivalent representation is used in which a set of n vectors is specified, one vector for each *n* modulo-2 adder. A 1 in the ith position of the vector indicates that the corresponding stage in the shift register is connected to the modulo-2 adder; 0 in a given position indicates that no connection exists between that stage and the modulo-2 adder.
   For example, consider the binary convolutional encoder with constraint length *K* = 7, *k* = 1, and *n* = 2; refer to Convolutional Code CC(2, 1 ,7). The connection for y0 is (1, 1, 0, 1, 1, 0, 1) from Input to Outputs; the connection for y1 is (1, 0, 1, 1, 1, 1, 1). The generators for this code are more conveniently given in octal form as (0155, 0137). So, when *k* = 1, *n* generators, each of dimension *K* specify the encoder.

Convolutional Code CC(2, 1 ,7)



5. ZeroTail specifies the character of encoder input sequence. If ZeroTail = YES, the input sequence of encoder is divided into blocks. The length of the block is BitSequenceLength. After each block, *K* − 1 zeros need to be appended as tail bits. That is, the total block length of encoder is (BitSequenceLength + *K* − 1), referring to Tail bits appending for ZeroTail = YES. The information will be used in the decoder to obtain better performance.

Tail bits appending for ZeroTail = YES



Chop
ExtraTailPSDU
nRead=BitSequenceLength
nWrite=BitSequenceLength+ConstraintLength-1
Offset=0
UsePastInputs=YES

ConvolutionalCoder
C1
CodingRate=rate 1/2
ConstraintLength=ConstraintLength
Polynomial={0133, 0171}
ZeroTail=YES
BitSequenceLength=BitSequenceLength

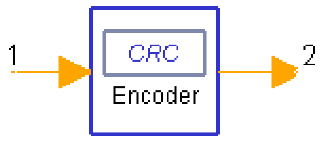
6. BitSequenceLength (valid only if ZeroTail = YES) is used to specify the information bit length, which indicates the length of uncoded bits. This parameter can be used to

set the same value for the encoder and the decoder.

References

1. John G. Proakis, Digital Communications (Third edition), Publishing House of Electronics Industry, Beijing, 1998.

CRC_Coder



Description: CRC generator
Library: Numeric, Advanced Comm
Class: SDFCRC_Coder
Derived From: CRC_Base

Parameters

Name	Description	Default	Unit	Type	Range
ParityPosition	Parity bits position: Tail, Head	Tail		enum	
ReverseData	reverse the data sequence or not: NO, YES	NO		enum	
ReverseParity	reverse the parity bits or not: NO, YES	NO		enum	
ComplementParity	complement parity bits or not: NO, YES	NO		enum	
MessageLength	input message length	172		int	[1, inf)
InitialState	initial state of encoder	0x0		int	[0, inf)
Polynomial	generator polynomial	0x1f13		int	[3, inf)

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input data	int

Pin Outputs

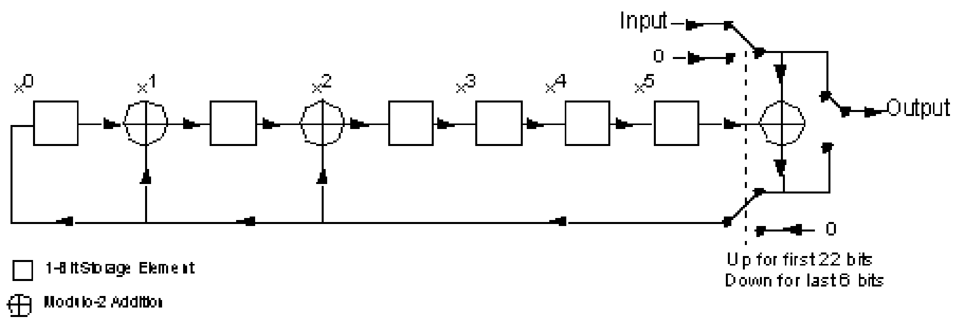
Pin	Name	Description	Signal Type
2	Out	output data	int

Notes/Equations

1. This component is used to add CRC bits to the input information. Each firing, (MessageLength + CRCLength) tokens are produced when MessageLength tokens are consumed. CRCLength is the length of CRC bits that is determined by Polynomial, where $2^{CRCLength} \leq$ Polynomial $\leq 2^{CRCLength+1}$.
2. CRC bits can be added to the head or the tail of the information bits by setting ParityPosition. The order of CRC bits and the order of information bits can be reversed by setting ReverseData and ReverseParity.
3. CRC Bit Calculation is an example of a CRC encoder in CDMA2000, where $g(x) = x^6 + x^2 + x + 1$, and Polynomial is hex 0x47. The CRC bits are added after the information bits; the order of the CRC and information bits are not reversed.
   - Initially, all shift register elements are set to the InitialState and the switches are set in the up position.
   - The register is clocked the number of times equal to MessageLength.
   - Switches are then set in the down position so that the output is a modulo-2

addition with a 0 and the successive shift register inputs are 0.
- The register is clocked an additional number of times equal to CRCLength and the CRC bits are output.

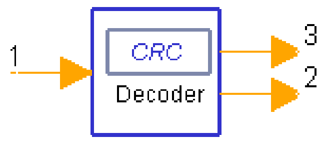
CRC Bit Calculation



References

1. TIA/EIA/IS-2000.2 (PN-4428), Physical Layer Standard for cdma2000 Spread Spectrum Systems, July 1999.

CRC_Decoder



Description: CRC Decoder
Library: Numeric, Advanced Comm
Class: SDFCRC_Decoder
Derived From: CRC_Base

Parameters

Name	Description	Default	Unit	Type	Range
ParityPosition	Parity bits position: Tail, Head	Tail		enum	
ReverseData	reverse the data sequence or not: NO, YES	NO		enum	
ReverseParity	reverse the parity bits or not: NO, YES	NO		enum	
ComplementParity	complement parity bits or not: NO, YES	NO		enum	
MessageLength	input message length	172		int	[1, inf)
InitialState	initial state of encoder	0x0		int	[0, inf)
Polynomial	generator polynomial	0x1f13		int	[3, inf)

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input data	int

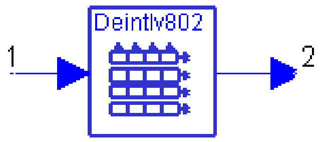
Pin Outputs

Pin	Name	Description	Signal Type
2	Out	output data	int
3	Parity	Parity check	int

Notes/Equations

1. This component is used to check the CRC bits for CRC frame errors.
   Each firing, (MessageLength + CRCLength) tokens are consumed when MessageLength tokens and one parity token are produced. CRCLength is the CRC bit length determined by Polynomial, where 2CRCLength ≤ Polynomial ≤ 2CRCLength+1.
2. The message part of the input data is sent to a CRC encoder that has the same Polynomial value as the encoder (CRC_Coder). The CRC bits are then compared with the CRC bits in the input data. If these are the same, the CRC check is passed.

Deinterleaver802D



Description: Deinterleave the input data
Library: Numeric, Advanced Comm
Class: SDFDeinterleaver802

Parameters

Name	Description	Default	Type	Range
s	modular factor of interleaving	1	int	[1, ∞)
l	divisor factor of interleaving	16	int	[1, ∞)
NCBPS	Number of coded bits per OFDM symbol	48	int	[1, ∞)

† The configuration of parameters s,l and NCBPS should be considered carefully or unexpected result will occur.

Pin Inputs

Pin	Name	Description	Signal Type
1	In	Input	real

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	Output	real

Notes/Equations

1. Deinterleaver802 performs deinterleaving based on IEEE 802 standards. This component deinterleaves (the inverse of Interleaver802) input bits with a block size corresponding to the number of bits in a single OFDM symbol N_{CBPS}.

   Each firing, N_{CBPS} tokens are consumed and N_{CBPS} tokens are produced.
2. Deinterleaving is defined by a two-step permutation; *j* is used to denote the index of the original received bit before the first permutation; *i* is used to denote the index after the first (and before the second) permutation; *k* is used to denote the index after the second permutation, before delivering the coded bits to the convolutional (Viterbi) decoder.
   The first permutation is defined by
   $i = s \times \text{floor}(j/s) + (j + \text{floor}(l \times j/ N_{CBPS})) \bmod s \; j = 0, 1, \ldots N_{CBPS} - 1$

   The function floor (.) denotes the largest integer not exceeding the parameter
   The second permutation is defined by

$k = l \times i - (N_{CBPS} - 1)\text{floor}(l \times i/N_{CBPS})$ $i = 0, 1, \ldots N_{CBPS} - 1$

In the equations, *s is the modular factor and l is the divisor factor*; these are variable parameters and their values depend on which standard the model is used for.
If this model is used for IEEE 802.11 and HIPERLAN/2
$s = \max(N_{BPSC}/2, 1)$, $l = 16$

where
N_{BPSC} and N_{CBPS} are determined by data rates given in IEEE 802.11 and HIPERLAN/2 Rate Dependent Values.
If this model is used for IEEE 802.16
$s = N_{BPSC}/2, 1)$ $l = 12$

where N_{BPSC} and N_{CBPS} are determined by block sizes given in IEEE 802.16 Bit Interleaver Block Sizes (NCBPS / NBPSC).

IEEE 802.11 and HIPERLAN/2 Rate Dependent Values

Data Rate (Mbps)	Modulation	Coding Rate (R)	Coded Bits per Subcarrier (NBPSC)	Coded Bits per OFDM Symbol (NCBPS)	Data Bits per OFDM Symbol (NDBPS)
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24 (IEEE 802.11a)	16QAM	1/2	4	192	96
27 (HIPERLAN/2)	16QAM	9/16	4	192	108
36	16QAM	3/4	4	192	144
48 (IEEE 802.11a)	64QAM	2/3	6	288	192
54	64QAM	3/4	6	288	216

IEEE 802.16 Bit Interleaver Block Sizes (N_{CBPS} / N_{BPSC})

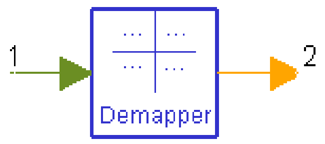
Modulation	16 Subchannels (Default)	8 Subchannels	4 Subchannels	2 Subchannels	1 Subchannel
QPSK	384/2	192/2	96/2	48/2	24/2
16QAM	768/4	384/4	192/4	96/4	48/4
64QAM	1152/6	576/6	288/6	144/6	72/6

References

1. IEEE Standard 802.11a-1999, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band," 1999.
2. ETSI TS 101 475 v1.1.1, "Broadband Radio Access Networks (BRAN); HIPERLAN Type

2; Physical (PHY) layer," April, 2000.
3. IEEE P802.16-REVd/D2-2003," Part 16 Air Interface for Fixed Broadcast Wireless Access Systems".

Demapper



Description: Demodulator for BPSK, QPSK, 8PSK, 16QAM, 32QAM, 64QAM, 128QAM, and 256QAM or demapping according to user defined table.
Library: Numeric, Advanced Comm
Class: SDFDemapper

Parameters

Name	Description	Default	Unit	Type	Range
ModType	Modulation type: BPSK, QPSK, PSK8, QAM16, QAM32, QAM64, QAM128, QAM256, User_Defined	QPSK		enum	
MappingTable	Constellation table			complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input symbol sequence	complex

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	output bit sequence	int

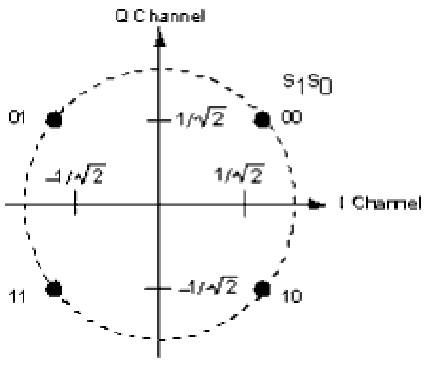
Notes/Equations

1. This component is used for BPSK, QPSK, 8PSK, 16QAM, 32QAM, 64QAM, 128QAM and 256QAM symbol demodulation or for demapping bits according to the mapping table.
2. The input signal is assumed to be modulated using the Mapper component. For QAM modulations, the input signal amplitude must be normalized before input to the model according to the constellations.
   Each firing, when one In token is consumed:
   - 1 Out token is produced for BPSK
   - 2 Out tokens are produced for QPSK
   - 3 Out tokens are produced for 8PSK
   - 4 Out tokens are produced for 16QAM
   - 5 Out tokens are produced for 32QAM
   - 6 Out tokens are produced for 64QAM
   - 7 Out tokens are produced for 128QAM
   - 8 Out tokens are produced for 256QAM
     For the user-defined mapping table, assuming the size of the array is A, log2(A) Out tokens are produced when one In token is consumed.

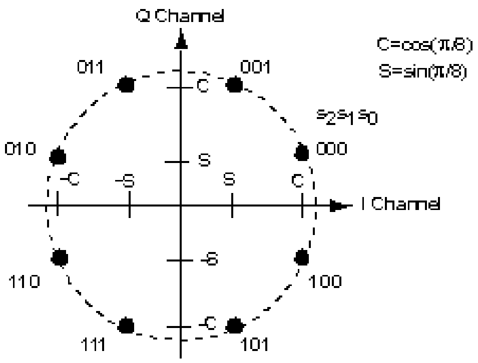
   For BPSK, bit 0 is mapped to 1 and bit 1 is mapped to −1.

3.
4. The QPSK constellation is illustrated in QPSK Modulation Constellation. The 8PSK constellation is illustrated in 8PSK Modulation Constellation.

QPSK Modulation Constellation



8PSK Modulation Constellation

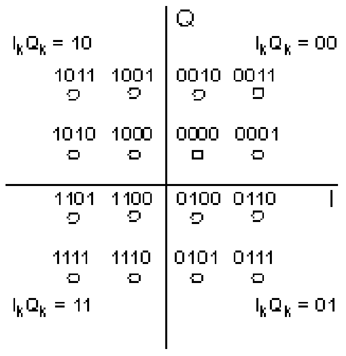


5. For 16QAM, 32QAM, 64QAM, 128QAM and 256QAM, the constellation points in quadrant 1 are converted to quadrants 2, 3 and 4 by changing the two most significant bits (Ik and Qk) and by rotating the q least significant bits according to Conversion of Constellation Points.
   Constellation diagrams are illustrated in 16 and 32QAM Constellations through 256QAM Constellation.
6. For user-defined mapping, the input binary bit sequence is mapped to a constellation point with the corresponding decimal index specified in the MappingTable parameter.

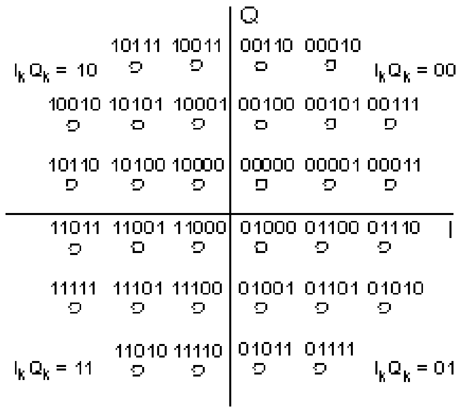
Conversion of Constellation Points

Quadrant	Most Significant Bit	Least Significant Bit Rotation
1	00	
2	10	π/2
3	11	π
4	01	3π/2

16 and 32QAM Constellations

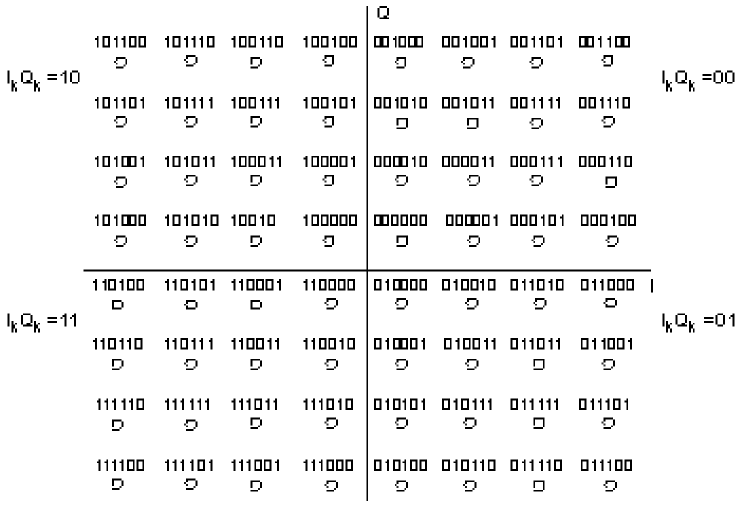


16-QAM



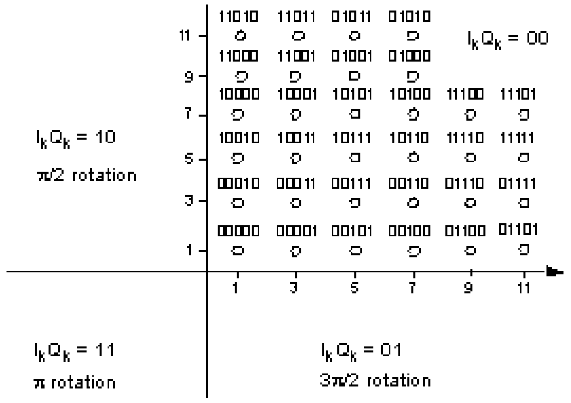
32-QAM

64QAM Constellation

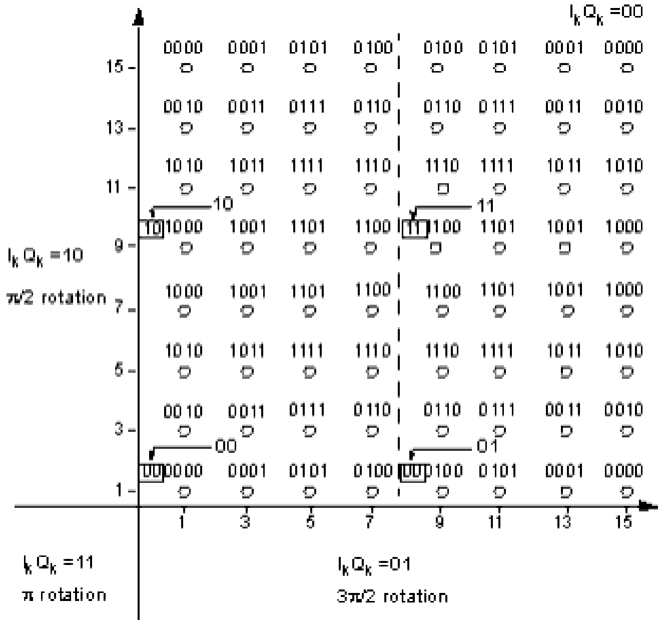


I_kQ_k are the two MSBs in each quadrant

128QAM Constellation



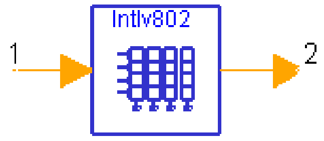
256QAM Constellation



References

1. EN 300 429, "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems," V1.2.1, 1998-04.

Interleaver802



Description: Interleave the input bits
Library: Numeric, Advanced Comm
Class: SDFInterleaver802

Parameters

Name	Description	Default	Type	Range
s	modular factor of interleaving	1	int	[1, ∞)
l	divisor factor of interleaving	16	int	[1, ∞)
NCBPS	Number of coded bits per OFDM symbol	48	int	[1, ∞)

† The configuration of parameters s,l and NCBPS should be considered carefully or unexpected result will occur.

Pin Inputs

Pin	Name	Description	Signal Type
1	In	Input	int

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	Output	int

Notes/Equations

1. Interleaver802 performs interleaving based on IEEE 802 standards. Encoded data bits are interleaved by this block interleaver with a block size corresponding to the number of bits in a single OFDM symbol N_{CBPS}.

   Each firing, N_{CBPS} tokens are consumed and N_{CBPS} tokens are produced.
2. Interleaving is defined by a two-step permutation. The first permutation ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second permutation ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation, thereby avoiding long runs of low reliability bits.
   In the following, k denotes the index of the coded bit before the first permutation; i denotes the index after the first and before the second permutation; j denotes the index after the second permutation, just prior to modulation mapping.
   The first permutation is defined by

i = (N_{CBPS} /l) (k mod l) + floor(k/l) k = 0, 1, ..., N_{CBPS} - 1

The function floor (.) denotes the largest integer not exceeding the parameter.
The second permutation is defined by
j = s × floor(i/s) + (i + N_{CBPS} - floor(l × i/N_{CBPS})) mod s i = 0, 1, ... N_{CBPS} - 1

In the equations, *s is the modular factor and l is the divisor factor* ; these are variable parameters and their values depend on which standard the model is used for.
If this model is used in IEEE 802.11 and HIPERLAN/2,
s = max (N_{BPSC} /2, 1), l = 16;

where N_{BPSC} and N_{CBPS} are determined by data rates given in IEEE 802.11 and HIPERLAN/2 Rate-Dependent Values.
If this model is used in IEEE 802.16,
s = N_{BPSC} /2, l = 12;

where N_{BPSC} and N_{CBPS} are determined by block sizes given in IEEE 802.16 Bit Interleaver Block Sizes (NCBPS /NBPSC).

IEEE 802.11 and HIPERLAN/2 Rate-Dependent Values

Data Rate (Mbps)	Modulation	Coding Rate (R)	Coded Bits per Subcarrier (NBPSC)	Coded Bits per OFDM Symbol (NCBPS)	Data Bits per OFDM Symbol (NDBPS)
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24 (IEEE 802.11a)	16QAM	1/2	4	192	96
27 (HIPERLAN/2)	16QAM	9/16	4	192	108
36	16QAM	3/4	4	192	144
48 (IEEE 802.11a)	64QAM	2/3	6	288	192
54	64QAM	3/4	6	288	216

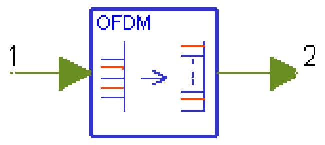
IEEE 802.16 Bit Interleaver Block Sizes (N_{CBPS} / N_{BPSC})

Modulation	16 Subchannels (Default)	8 Subchannels	4 Subchannels	2 Subchannels	1 Subchannel
QPSK	384/2	192/2	96/2	48/2	24/2
16QAM	768/4	384/4	192/4	96/4	48/4
64QAM	1152/6	576/6	288/6	144/6	72/6

References

1. IEEE Standard 802.11a-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band, 1999.
2. ETSI TS 101 475 v1.1.1, Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer, April, 2000.
3. IEEE P802.16-REVd/D2-2003, *Part 16 Air Interface for Fixed Broadcast Wireless Access Systems*.

LoadIFFTBuff802



Description: Subcarriers loader into IFFT buffer
Library: Numeric, Advanced Comm
Class: SDFLoadIFFTBuff802

Parameters

Name	Description	Default	Unit	Type	Range
Carriers	Number of subcarriers per OFDM symbol	52		int	[1:8192]
DCCarrier	DC carrier: OFF, ON	OFF		enum	
DCPilotValue	DC Pilot Value	1.333333+j*0.0		complex	
FullSubcarriers	Active all sub-carriers: NO, YES	YES		enum	
SubcarrierList	Sub-carrier list	{-21, -7, 7, 21}		int array	
Order	IFFT points in 2^Order	7		int	[(logCarriers/log2), ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	In	Transmitted signal before IFFT	complex

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	IFFT input signal, zero padded	complex

Notes/Equations

1. This component is used to load transmission data into the IFFT buffer. Each firing, Carriers tokens are consumed and 2^{Order} tokens are generated. For example, if Carriers = 52, Order = 7, 52 tokens are consumed and 128 tokens are generated.
2. Data loading is performed as follows.
   Assume $x(0), x(1), \ldots, x(N-1)$ are the inputs that generally represent active subcarriers defined by designers, where N = Carriers. $y(0), y(1), \ldots, y(M-1)$ are the outputs, M = 2^{Order}.
   when N is even
   $$y(i) = x\left(\frac{N}{2}+i-1\right) \quad i = 1, \ldots, \frac{N}{2}$$

$$y(i) = 0 \qquad i = 0, \frac{N}{2}+1, \ldots, M-\frac{N}{2}-1$$

$$y(i) = x\left(i - M + \frac{N}{2}\right) \qquad i = M-\frac{N}{2}, \ldots, M-1$$

when N is odd

$$y(i) = x\left(\frac{N-1}{2} + i - 1\right) \qquad i = 1, \ldots, \frac{N+1}{2}$$

$$y(i) = 0 \qquad i = 0, \frac{N+1}{2}+1, \ldots, M-\frac{N+1}{2}$$

$$y(i) = x\left(i - M - \frac{N-1}{2}\right) \qquad i = M-\frac{N-1}{2}, \ldots, M-1$$

For example, if Order = 4 and Carriers = 7, the input carriers are x(0), x(1), x(2), x(3),x(4),x(5),x(6), and the output carrier sequence would be:
0 , x(3) , x(4) , x(5) , x(6) , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , x(0) , x(1) , x(2)
which will be loaded into the IFFT model for the IFFT transformation.

3. DCCarrier and DCPilotValue specify whether DC carrier is used; if DCCarrier = ON, the DC carrier value is set by DCPilotValue.
   In the example provided in *note 2*, DCCarrier = OFF.
   While DCCarrier = ON and DCPilotValue = 4/3, the output carriers sequence would be:
   4/3, x(3), x(4), x(5), x(6), 0, 0, 0, 0, 0, 0, 0, 0, x(0), x(1), x(2)
   in which the first carrier is 4/3 instead of 0.
4. If FullSubcarriers = YES, all input carriers will be used. If FullSubcarriers = NO, some of the input carriers will be used; SubcarrierList specifies which input carriers will be used.
5. SubcarrierList (valid when FullSubcarriers = NO) specifies the positions of the input carriers to be used as active subcarriers (all subcarriers are 0 except those carriers specified).
   Assume *x*(0), *x*(1), ... , *x*(*N*−1) are the input signals that generally represent active subcarriers defined by designers, where N = Carriers. *y*(0), *y*(1), ... , *y*(*M*−1) are the output of the model *M* = 2^{Order}. The corresponding indices of *x*(0), *x*(1), ... , *x*(*N*−1) are {int(−Carriers/2), int(−Carriers/2) + 1, ... , −1, 1, ... , int(Carriers/2)−1, int(Carriers/2)}.
   The active subcarrier loading procedure is performed as follows: assume index is an element of {int(−Carriers/2), int(−Carriers/2) + 1, ... , −1, 1, ... , int(Carriers/2)−1, int(Carriers/2)}:
   when N is even

$$y(index) = x\left(\frac{N}{2} + index - 1\right) \qquad index > 0$$

$$y(M + index) = x\left(index + \frac{N}{2}\right) \qquad index < 0$$

when N is odd

$$y(index) = x\left(\frac{N-1}{2} + index - 1\right) \qquad index > 0$$

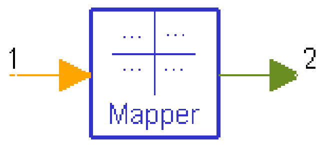
$$y(M + index) = x\left(index + \frac{N-1}{2}\right) \qquad index < 0$$

For example, SubcarrierList = {−2, −1, 2, 3}, and input carriers are x(0), x(1), x(2), x(3), x(4), x(5), x(6). Indices of the input carriers are −3, −2, −1, 1, 2, 3, 4. Elements in SubcarrierList must be integer and in (−Carriers/2, Carriers/2), in which Carriers is the number of carriers of input, here, it is 7 and index should be in [−3, 3]. In this case, the carrier with index is −2, −1, 2, 3 is used, these are x(1), x(2), x(4), x(5). The output subcarriers are then:
4/3, 0, x(4), x(5), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, x(1), x(2).

References

1. IEEE Standard 802.11a-1999, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band," 1999.
2. ETSI TS 101 475 v1.1.1, "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer," April, 2000.
3. ARIB-JAPAN, Terrestrial Integrated Services Digital Broadcasting (ISDB-T); Specification of Channel Coding, Frame Structure and Modulation, Sept.1998.
4. ETSI, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. EN300 744 v1.2.1, European Telecommunication Standard, July 1999.
5. IEEE P802.15-03/268r1, "Multi-band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a," September 2003.
6. IEEE P802.16-REVd/D2-2003, "Draft IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2003.

Mapper



Description: Modulator for BPSK, QPSK, 8PSK, 16QAM, 32QAM, 64QAM, 128QAM, and 256QAM or mapping according to user defined table.
Library: Numeric, Advanced Comm
Class: SDFMapper

Parameters

Name	Description	Default	Unit	Type	Range
ModType	Modulation type: BPSK, QPSK, PSK8, QAM16, QAM32, QAM64, QAM128, QAM256, User_Defined	QPSK		enum	
MappingTable	Constellation table			complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input bit sequence	int

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	output symbol sequence	complex

Notes/Equations

1. The Mapper is a generic element performing a Mapping/Modulation for an input bit sequence. When ModType is specified to BPSK, QPSK, 8-PSK, 16-QAM, 32-QAM, 64-QAM, 128-QAM or 256-QAM, the input bit sequence will be mapped/modulated to BPSK, QPSK, 8-PSK, 16-QAM, 32-QAM, 64-QAM, 128-QAM or 256-QAM symbols as described in the section 9 of [1], in which Constellations have been defined in figure 7-8 of [1]. For each Mapped/Modulated symbols, two most significant bits (MSB) are deferential encoded and least significant bits are rotated based on the specification in Conversion of Constellation Points. In this case, the Mapper just gives the exact same Mapping/Modulation as what shown in Figure 7-8 of [1]. When ModType is specified to User_Defined, users can customize the Constellation by putting their own symbols in the MappingTable.
2. This component is used to generate BPSK, QPSK, 8PSK, 16QAM, 32QAM, 64QAM, 128QAM and 256QAM modulation symbols or bit mapping according to the mapping table.
   Each firing, one Out token is produced when:
   - 1 In token is consumed for BPSK
   - 2 In tokens are consumed for QPSK
   - 3 In tokens are consumed for 8PSK
   - 4 In tokens are consumed for 16QAM

- 5 In tokens are consumed for 32QAM
- 6 In tokens are consumed for 64QAM
- 7 In tokens are consumed for 128QAM
- 8 In tokens are consumed for 256QAM
  For user-defined mapping table, assuming the size of the array is A, one Out token is produced when log2(A) In tokens are consumed. For more than one input token the input sequence is LSB first and MSB last.

3. For BPSK, bit 0 is mapped to 1; bit 1 is mapped to −1.
4. For QPSK, the constellation diagram is illustrated in QPSK Constellation.
5. For 8PSK, the constellation diagram is given in 8PSK Constellation.
6. For 16QAM, 32QAM, 64QAM, 128QAM and 256QAM, the constellation points in quadrant 1 are converted to quadrants 2, 3 and 4 by changing the two most significant bits (Ik and Qk) and by rotating the q least significant bits according to Conversion of Constellation Points.

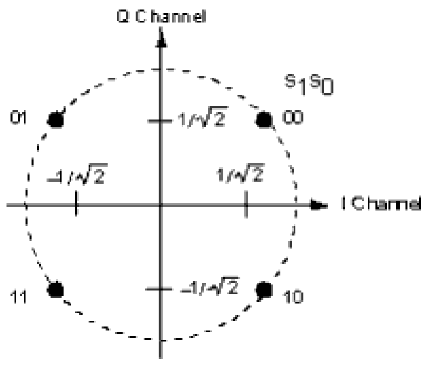
Conversion of Constellation Points

Quadrant	Most Significant Bit	Least Significant Bit Rotation
1	00	
2	10	π/2
3	11	π
4	01	3π/2

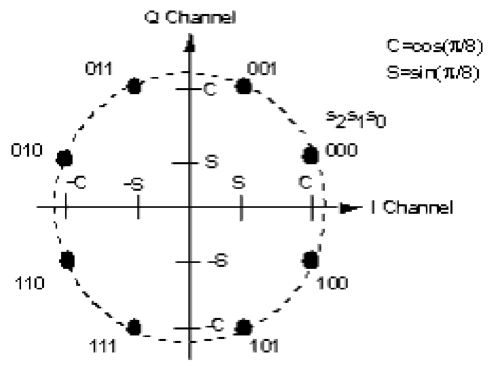
16QAM, 32QAM, 64QAM, 128QAM and 256QAM constellation diagrams are illustrated in 16 and 32QAM Constellation through 256QAM Constellation.

7. For user-defined mapping, the input binary bit sequence is mapped to a constellation point with corresponding decimal index in the MappingTable.

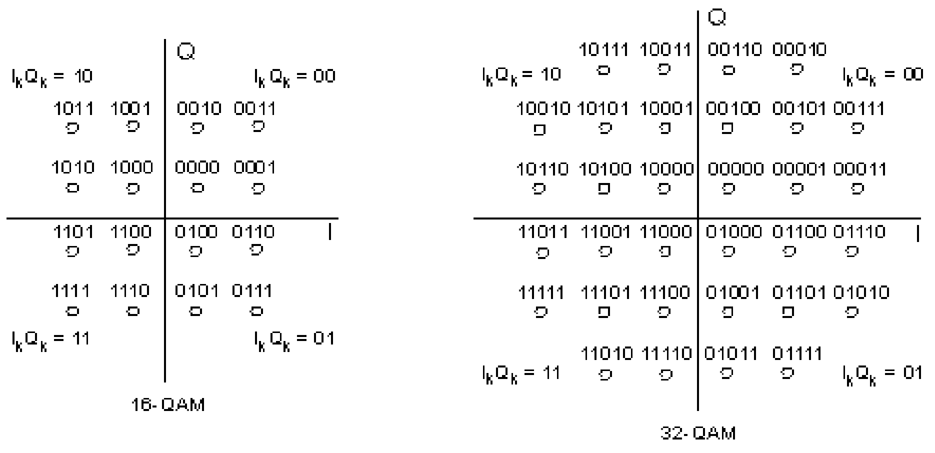
QPSK Constellation



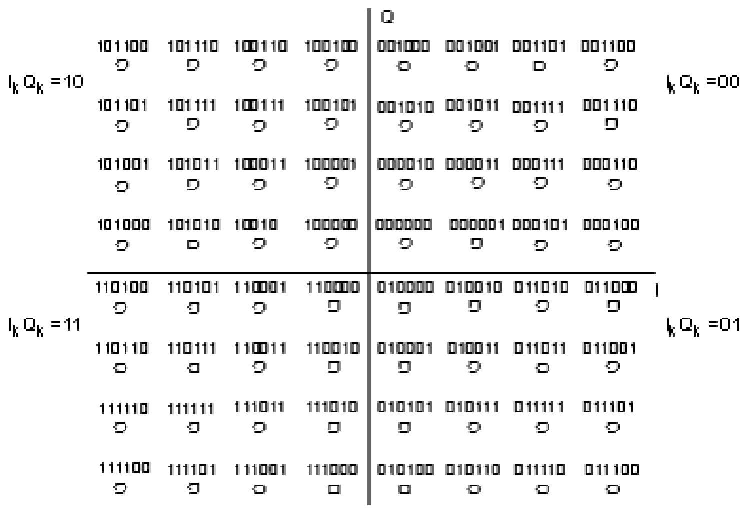
8PSK Constellation



16 and 32QAM Constellation

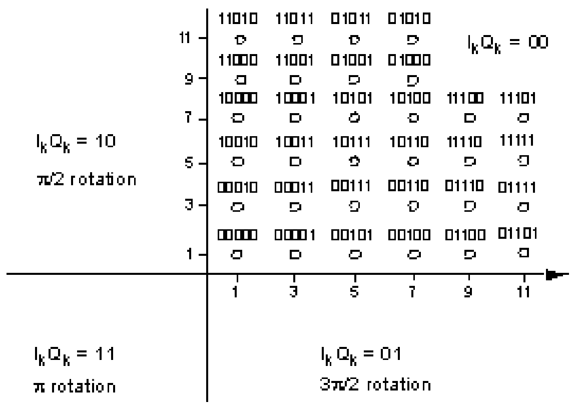


64QAM Constellation

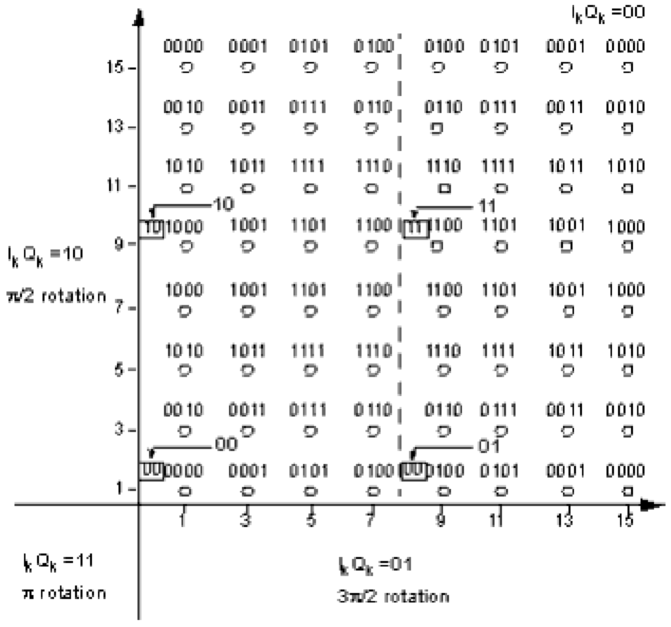


$I_k Q_k$ are the two MSBs in each quadrant

128QAM Constellation



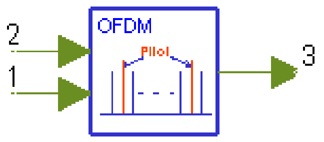
256QAM Constellation



References

1. EN 300 429, "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems," V1.2.1, 1998-04.

MuxOFDMSym802



Description: generic OFDM symbol multiplexer
Library: Numeric, Advanced Comm
Class: SDFMuxOFDMSym802

Parameters

Name	Description	Default	Unit	Type	Range
Carriers	Number of subcarriers per OFDM symbol	52		int	[1:8192]
DataCarriers	Number of data subcarriers per OFDM symbol	48		int	[1:8192]
PilotPosition	Standard pilots positions	{-21, -7, 7, 21}		int array	
PilotValue	Standard pilots values	{1.0, 1.0, 1.0, -1.0}		complex array	
GuardCarrierPosition	Guard carriers positions			int array	
GuardCarrierValue	Guard carriers values			complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	data subcarriers input	complex
2	Pilot	continual pilot value	complex

Pin Outputs

Pin	Name	Description	Signal Type
3	Out	OFDM symbol output	complex

Notes/Equations

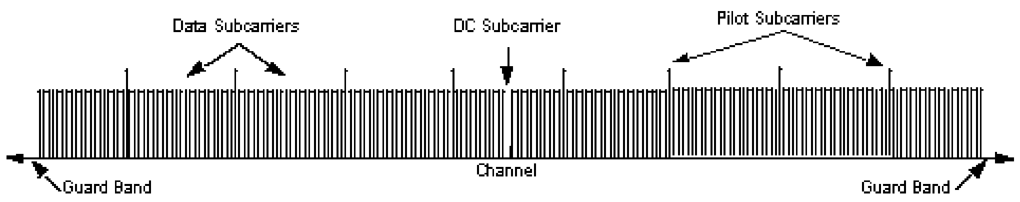
1. This component is used to multiplex data and pilot subcarriers into the OFDM symbol for IEEE 802 standards 802.11a, 802.11g, 802.15.3a, 802.16a, and 802.16d.

   Note
   OFDM symbols generally consist of continual pilots (CP) and scattered pilots (SP). Current IEEE 802 standards use CP only. Even though some DAB, DVB-T, and ISDB-T OFDM systems may use both CP and SP, MuxOFDMSym802 supports CP only.

2. The basic OFDM symbol structure is introduced in the frequency domain. The symbol (illustrated in OFDM Symbol) consists of subcarriers that determine the size of the FFT. There are several subcarrier types:
   - Data subcarriers for data transmission
   - Pilot subcarriers for estimations

- Null subcarriers for no transmission, for guard bands and DC subcarrier.
  Guard bands in most OFDM systems (DVB-T, ISDB-T, 802.11a, 802.11g, 802.16a, and 802.16d) are inserted zeros.
  IEEE 802.15.3a has additional guard carriers defined between data subcarriers and guard bands. The guard subcarriers can be used for various purposes, including relaxing the specification on transmit and receive filters. The magnitude level of the guard tones is not specified, so reduced power levels for these subcarriers can be used. The all-zeros guard bands allow the signal to naturally decay and create the FFT *brick wall* shaping.

OFDM Symbol



  This component multiplexes data and pilot subcarriers into one OFDM symbol according to the positions of data and pilot subcarriers defined in the standards. The null subcarriers (guard bands and DC subcarrier) are inserted into an OFDM symbol by the LoadIFFTBuff802 component. (Both MuxOFDMSym802 and LoadIFFTBuff802 components implement an OFDM symbol in the frequency domain.)

3. MuxOFDMSym802 parameter settings enable designers to generate a variety of OFDM symbol formats, in accordance with IEEE standards or not.
   Carriers specifies the number of active subcarriers (data subcarriers, pilot subcarriers and guard subcarriers) in one OFDM symbol.

   Note
   Carriers = DataCarriers + PilotPosition + GuardCarrierPosition.

   DataCarriers specifies the number of data subcarriers in one OFDM symbol.
   PilotPosition specifies continual pilot positions; PilotPosition is the number of pilot subcarriers in one OFDM symbol.
   PilotValue specifies values for continual pilot positions.
   GuardCarrierPosition specifies guard carriers positions (default = NULL);
   GuardCarrierPosition is the number of guard carrier subcarriers in one OFDM symbol.
   GuardCarrierValue specifies values for guard carrier positions (default = NULL).
4. Each firing, one Pilot token and DataCarriers tokens are consumed and Carriers tokens are output.
   The complex Data input signal is directly multiplexed into the OFDM symbol.
   The continual pilots are multiplexed into OFDM symbols as follows:
   p_k is the input in Pilot pin for kth OFDM symbol (or kth firing)

   $a_0, a_1, \ldots, a_n$ are n+1 pilot values defined by PilotValue

   The actual pilot values of kth OFDM symbol are $p_k \times a_0, p_k \times a_1, \ldots, p_k \times a_n$

   . The continual pilot subcarrier values are multiplexed into the OFDM symbol according to PilotPosition.
   The guard carriers are multiplexed into the OFDM symbol like continual pilot as follows:
   $b_0, b_1, \ldots, b$ are m+1 guard carriers values specified by GuardCarrierValue.

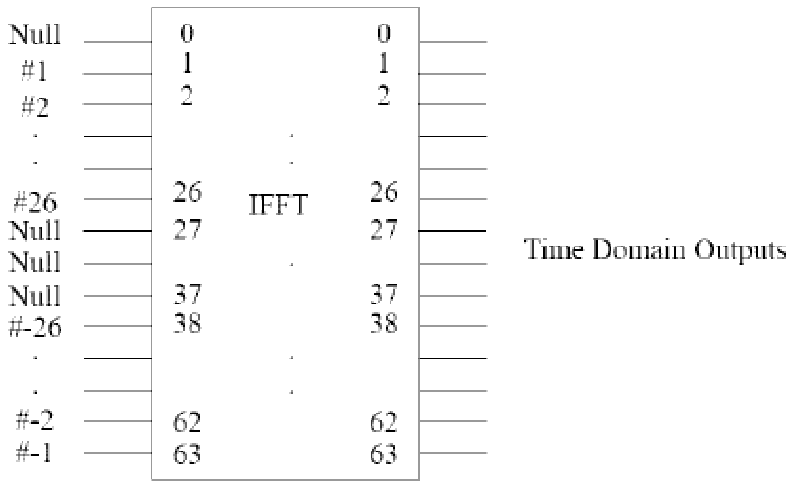
m

The actual guard carrier values of kth OFDM symbol are $p_k \times b_0, p_k \times b_1, \dots, p_k \times b_m$.

These guard carrier subcarriers values are multiplexed into the OFDM symbol according to GuardCarrierPosition.

5. The MuxOFDMSym802 output includes all active data, pilot, and guard carriers subcarriers indexed in the frequency domain:
$[-(Carriers)/2, -(Carriers)/2 + 1, \dots, -1, 1, \dots, (Carriers + 1)/2 -1, (Carriers + 1)/2]$
LoadIFFTBuff802 loads these output signals from MuxOFDMSym802 into the IFFT buffer and inserts zeros into the NULL and DC subcarriers. IFFT Input and Output (802.11a Specification) illustrates the 802.11a IFFT input and output. An OFDM symbol is input in the frequency domain after LoadIFFTBuff802; an OFDM symbol is output in the time domain after IFFT.

IFFT Input and Output (802.11a Specification)

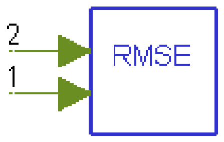


References

1. IEEE Standard 802.11a-1999, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band," 1999.
2. ETSI TS 101 475 v1.1.1, "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer," April, 2000.
3. ARIB-JAPAN, Terrestrial Integrated Services Digital Broadcasting (ISDB-T); Specification of Channel Coding, Frame Structure and Modulation, Sept.1998.
4. ETSI, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. EN300 744 v1.2.1, European Telecommunication Standard, July 1999.
5. IEEE P802.15-03/268r1, "*Multi-band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a*," September 2003.
6. IEEE P802.16-REVd/D2-2003, "*Draft IEEE Standard for Local and metropolitan area*

networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2003.

RMSE



Description: Root Mean Square Error
Library: Numeric, Advanced Comm
Class: SDFRMSE

Parameters

Name	Description	Default	Unit	Type	Range
StartFrame	Start frame	0		int	[0, ∞)
FramesToAverage	Number of frames for the average RMSE	1		int	[1, ∞)
FrameLength	Frame length	4096		int	[1, ∞)
DisplayOption	Display option: RMS, dB	RMS		enum	

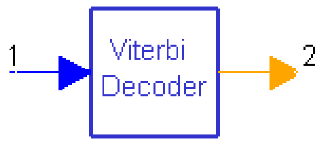
Pin Inputs

Pin	Name	Description	Signal Type
1	InRef	Input reference signal	complex
2	InTest	Input test signal	complex

Notes/Equations

1. This component is used to calculate the root mean square error of the input data. Each firing, one token is consumed; after (FramesToAverage + StartFrame) × FrameLength tokens are consumed, the RMSE of the input signal is sinked.
2. The root mean square error is calculated according to the equation
$$RMSE = \frac{1}{N_f}\sum_{i=1}^{N_f}\sqrt{\frac{1}{L_f}\sum_{j}^{L_f}((I_1(i,j)-I_2(i,j))^2+(Q_1(i,j)-Q_2(i,j))^2)}$$
where,
N_f is the number of frames to average
L_f is the frame length
$I_1(i, j)$, $Q_1(i, j)$ and $I_2(i, j)$, $Q_2(i, j)$ are the in-phase and quadrature parts, respectively, of the input signals

ViterbiDecoder



Description: Viterbi decoder for convolutional code
Library: Numeric, Advanced Comm
Class: SDFViterbiDecoder

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
CodingRate	Coding rate: rate_1_2, rate_1_3, rate_1_4, rate_1_5, rate_1_6, rate_1_7, rate_1_8	rate_1_2	R		enum	
ConstraintLength	Constraint length	7	K		int	[3, 14]
Polynomial	Generator polynomial	{0133, 0171}			int array	{2^(K-1)+2*n-1}, n=1,2,3,...2^(K-2).
ZeroTail	Zero tail used to convert convolutional code to block code: NO, YES	NO			enum	
BitSequenceLength	Length of bit squence not including tail bits, valid when ZeroTail=YES	88	N		int	[1,65535]
MaxSurvivorLength	Maximum length of survivor, in bits	35			int	[5*K, 20*K]
Polarity	Mapping mode from NRZ to logic signal: Negative to logic 1, Negative to logic 0	Negative to logic 1			enum	
InitialState	Initial state of convolutional encoder: Zero state, Non-zero state	Zero state			enum	
IgnoreNumber	Number of data points to be ignored	0			int	[0, 65535]

Pin Inputs

Pin	Name	Description	Signal Type
1	In	input	real

Pin Outputs

Pin	Name	Description	Signal Type
2	Out	output	int

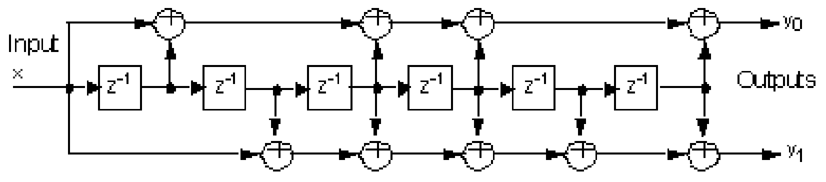
Notes/Equations

1. This component is used for convolutional code decoding with a Viterbi algorithm. Generally, there are two ways to implement convolutional code in communications

system: code a semi-infinite bit sequence length where the initial encoder state could be zero- or non-zero with any final state; or, code block-by-block by appending zero tails after bit blocks so that the initial and the final encoder states are both zero. The ZeroTail parameter specifies this implementation; if ZeroTail = YES, then zero tails must be appended before input to this component.
Each firing, if ZeroTail = YES, (*N* + *K* − *1*) Out tokens are produced, when (*N* + *K* − 1)/ *R* In tokens are consumed; If ZeroTail = NO, 1 Out token is produced for 1/ *R* In tokens consumed.
For example, in CDMA access channel, CC(3, 1, 9) with zero tail is used in which the convolutional code rate *R* is 1/3 and the bit sequence length is 88. CodingRate is set to *rate 1/3*, ZeroTail = YES and BitSequenceLength = 88.Each firing, 96 Out tokens are produced when 288 In tokens are consumed.
ViterbiDecoder supports the 1/ *n* coding rate only. Convolutional codes with *k*/ *n* (*k* >1) are not supported by this component because: the coding and decoding will be more complex (this is also the reason why convolutional codes with a k/n (*k* >1) coding rate are seldom used in real communication systems); and, even convolutional codes with a *k*/ *n* (*k* >1) coding rate are used that are typically implemented by puncturing the convolutional code with a 1/ *n* coding rate.

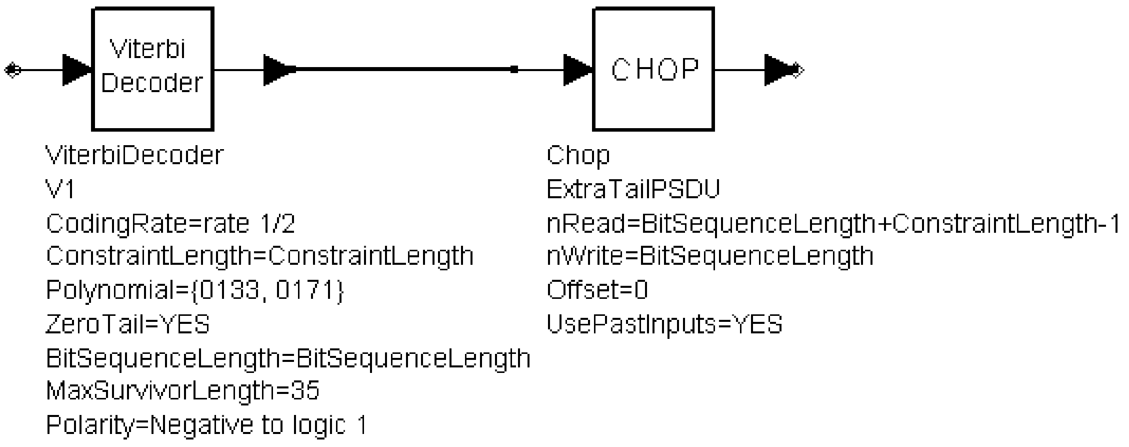
2. Polynomial is the convolutional code generator function. The generator matrix for a convolutional code is generally semi-infinite because the input sequence is semi-infinite. As an alternative to specifying the generator matrix, a functionally equivalent representation is used in which a set of n vectors is specified, one vector for each of the n modulo-2 adder. 1 in the ith position of the vector indicates that the corresponding stage in the shift register is connected to the modulo-2 adder; 0 in a given position indicates that no connection exists between that stage and the modulo-2 adder.
For example, consider the binary convolutional encoder with constraint length K = 7, k = 1, and n = 2, illustrated in Convolutional Code CC(2,1,7). The connection for y0 is (1, 0, 1, 1, 0, 1, 1) from Outputs to Input, while the connection for y1 is (1, 1, 1, 1, 1, 0, 1). Generators for this code are conveniently given in octal form as (0133, 0175). So, when k=1, n generators (each of dimension K) are required to specify the encoder.

Convolutional Code CC(2,1,7)



3. ZeroTail is used to specify the encoder input sequence character. If ZeroTail = YES, the encoder input sequence is divided into blocks; block length is *N* . After each block, K−1 zeros are appended as tail bits. The total block length of the encoder is (*N* + K − 1), referring to Tail bits removal for ZeroTail = YES. In the decoder, known information can be used to obtain better performance.

Tail bits removal for ZeroTail = YES



ViterbiDecoder
V1
CodingRate=rate 1/2
ConstraintLength=ConstraintLength
Polynomial={0133, 0171}
ZeroTail=YES
BitSequenceLength=BitSequenceLength
MaxSurvivorLength=35
Polarity=Negative to logic 1

Chop
ExtraTailPSDU
nRead=BitSequenceLength+ConstraintLength-1
nWrite=BitSequenceLength
Offset=0
UsePastInputs=YES

4. BitSequenceLength (valid only when ZeroTail = YES) is used to specify the information bit length, which indicates the length of uncoded bits. This parameter can be set to the same value in the encoder and the decoder.
5. MaxSurvivorLength is the maximum length of the survivor that is stored in memory. The delay in decoding a long information sequence that has been convolutionally encoded is usually too long for most practical applications; moreover, memory required to store the entire length of surviving sequences is large and expensive. A solution for this is to modify the Viterbi algorithm in such a way that results in a fixed decoding delay without significantly affecting the optimal performance of the algorithm.
   The modification is to retain at any given time t only the most recent δ decoded informations bits in each surviving sequence. As each new information bit is received, a final decision is made on the bit received δ branches back in the trellis, by comparing the metrics in the surviving sequences and determining in favor of the bit in the sequence having the largest metric. If the δ chosen is sufficiently large, all surviving sequences will contain the identical decoded bit δ branches back in time. That is, with high probability, all surviving sequences at time t stem from the same one as t−δ. Experimental simulation has determined that a delay δ ≥ 5 *K* results in a negligible degradation in the performance relative to the optimum Viterbi algorithm.
6. Polarity is used to specify the mapping mode from bit (0, 1) to the NRZ signal level. Generally, bit 0 is mapped to level 1 and bit 1 is mapped level −1. An alternative is to map bit 0 to level −1 and bit 1 to level 1.
7. InitialState is used to specify the coded sequence character. If the initial state of encoder is zero-state, the known information can be used to obtain better performance. If the initial state is not known to be zero, InitialState must be set to a non-zero state.
8. IgnoreNumber is used to specify how much data will be ignored by this component. Delays in communications systems can be caused by devices or transmission. And, the delay may be inserted between the encoder and decoder in the form of meaningless data, so the information must be set in IgnoreNumber.
   - If ZeroTail = YES, the value of IgnoreNumber is $n \times (N + K - 1)/R$ (n is an integer and $n \geq 0$), and no extra delay will be introduced because it is assumed the sequence is frame synchronized before input to ViterbiDecoder.
   - If ZeroTail = NO, the delay is an integer number n ; this means the symbol synchronization is achieved before ViterbiDecoder. If n / R is also an integer, then the delay of output bit sequence will be n / R bits. Otherwise, the delay will be the minimum integer larger than n / R.
     Input sequence requirements are:
     If ZeroTail = YES

- The input sequence must be frame synchronized; that is, IgnoreNumber must be $n \times N / R$ (n is an integer and $n \geq 0$) and the first valid data must be the first symbol of the first codeword in that frame.
- The input sequence must be encoded from blocks, each having K−1 zero tails so that the initial state and final state are all zero-state.
  If ZeroTail = NO
- The input sequence must be bit synchronized; that is, the first valid data must be the first symbol of a codeword.
- If InitialState is set to Zero state, the first valid symbol must be encoded with zero initial state.

9. The Viterbi algorithm is an optimal method of decoding convolutional codes. Optimal decoding decisions cannot be made on a symbol-by-symbol basis; instead, the entire received sequence must be compared with all possible transmitted sequences. The number of possible transmitted sequences increases exponentially with time, so an efficient method of comparing sequences is necessary.
   The Viterbi algorithm is computationally efficient, but its complexity increases exponentially with the constraint length of the code. The Viterbi decoder measures how similar the received sequence is to a transmitted sequence by calculating a number called *path metric* (*path metric* of a sequence is calculated by adding numbers known as *symbol metric*, which is a measure of how close a received symbol is to each of the possible transmitted symbols). The transmitted sequence corresponding to the smallest path metric is declared to be the most likely sequence.
   The Viterbi algorithm for a CC(n, k, K) code is described in the following paragraphs.
   Branch Metric Calculation
   The branch metric $m^{(a)}_{j}$, at the J th instant of the α path through the trellis is defined as the logarithm of the joint probability of the received n-bit symbol $r_{j}1$, $r_{j}2$ ... , r_{jn} conditioned on the estimated transmitted n-bit symbol $c_{j}1^{(a)}$, $c_{j}2^{(a)}$ ... , $c_{jn}^{(a)}$ for the α path. That is,

$$m^{(\alpha)}_{j} = \ln\left(\prod_{i=1}^{n} P(r_{ji}|c_{ji}^{(\alpha)})\right)$$

$$= \sum_{i=1}^{n} \ln P(r_{ji}|c_{ji}^{(\alpha)}).$$

   If Rake receiver is regarded as a part of the channel, for the Viterbi decoder the channel can be considered to be an AWGN channel. Therefore,

$$m^{(\alpha)}_{j} = \sum_{i=1}^{n} r_{ji}c_{ji}$$

   Path Metric Calculation
   The path metric $M^{(a)}$ for the α path at the J th instant is the sum of the branch metrics belonging to the α path from the first instant to the J th instant. Therefore,

$$M^{(\alpha)} = \sum_{j=1}^{J} m^{(\alpha)}_{j}$$

   Information Sequence Update
   There are 2^{k} merging paths at each node in the trellis and the decoder selects from

paths α1, α2, ... , α2k the one having the largest metric, namely:

$$max(M^{(\alpha_1)}, M^{(\alpha_2)}, \ldots, M^{(\alpha_{2^k})})$$

This path is known as the survivor.

Decoder Output

When the two survivors have been determined at the *J* th instant, the decoder outputs from memory the (*J-L*)th information symbol survivor with the largest metric.

10. **ViterbiDecoder Component Validation**

BER Measurements lists BER measurements for a rate 1/2 code (g_0 = 171, g_1 = 133) and a memoryless additive white Gaussian channel. Simulations were made with hard decision decoding (binary quantization) and soft decision decoding (no quantization). Simulation results are listed along with results published in QUALCOMM Technical Data Sheet Q0256; note that the published data and simulation results agree.

BER Measurements

Eb/No(dB)	Hard Decision		Soft Decision	
	Simulated BER	**QUALCOMM BER**	**Simulated BER**	**QUALCOMM BER(3 bits)**
3.0			3.62e-04	8.00e-04
3.5			7.56e-05	2.00e-04
4.0	5.01e-03	6.50e-03	1.11e-05	3.50e-05
4.5	1.79e-03	1.80e-03	2.12e-06	7.00e-06
5.0	5.71e-04	5.50e-04		
5.5	1.25e-04	9.00e-05		
6.0	2.81e-05	4.00e-05		

References

1. S. Lin and D. J. Costello, Jr., *Error Control Coding Fundamentals and Applications*, Prentice Hall, Englewood Cliffs NJ, 1983.
2. J. G. Proakis, Digital Communications (Third edition), Publishing House of Electronics Industry, Beijing, 1998.

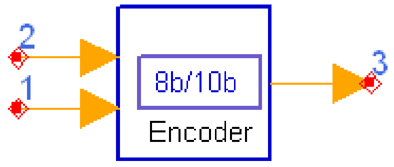
Numeric Communications Components

The numeric communications components provide basic communication functions on single data points or arrays of data that are integer, double precision floating point (real), fixed-point (fixed), or complex values. Each component accepts a specific class of signal and outputs a resultant signal.

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. These components do not accept any matrix class of signal. The auto conversion from timed, complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the real value of 1.0 creates a fixed-point value with precision of 2.30, and a value of 0.5 would create one of precision of 1.31. For signal conversion rules, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components accept parameter values that are arrays of data. The syntax for referencing arrays of data as parameter values includes an explicit list of values, a reference to a file that contains those values, or a combination of explicit values along with file references. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

8b10bCoder



Description: 8b/10b coder
Library: Numeric, Communications
Class: SDF8b10bCoder

Pin Inputs

Pin	Name	Description	Signal Type
1	Din	bits to be coded	int
2	Kin	control of Din (encoded as data (Kin=0) or encoded as a special character (Kin=1)	int

Pin Outputs

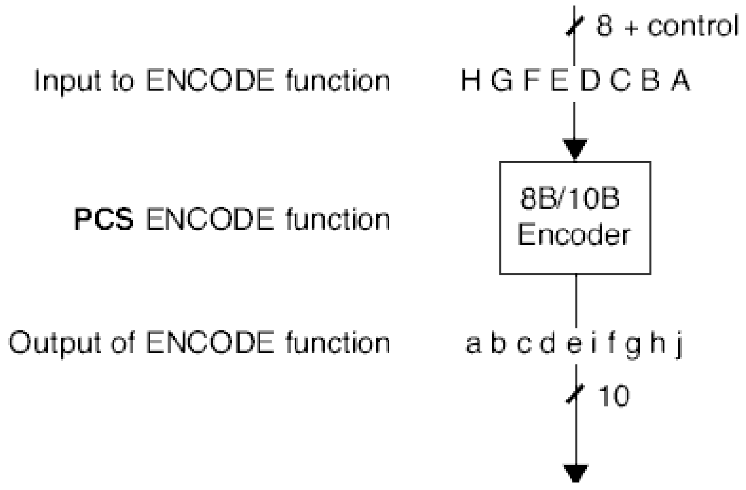
Pin	Name	Description	Signal Type
3	output	coded bits	int

Notes/Equations

1. The 8B/10B transmission code is used to improve the transmission characteristics of information. The encodings defined by the transmission code ensure that sufficient transitions are present in the PHY bit stream to make clock recovery possible at the receiver. Such encoding also greatly increases the likelihood of detecting any single or multiple bit errors that may occur during transmission and reception of information. In addition, some of the special code-groups of the transmission code contain a distinct and easily recognizable bit pattern that assists a receiver in achieving code-group alignment on the incoming PHY bit stream. The 8B/10B transmission code has a high transition density, is a run-length-limited code, and is dc-balanced. The transition density of the 8B/10B symbols ranges from 3 to 8 transitions per symbol.
2. 8B/10B transmission code uses letter notation for describing the bits of an unencoded information octet and a single control variable. Each bit of the unencoded information octet contains either a binary zero or a binary one. A control variable, Z, has either the value D or the value K. When the control variable associated with an unencoded information octet contains the value D, the associated encoded code-group is referred to as a data code-group. When the control variable associated with an unencoded information octet contains the value K, the associated encoded code-group is referred to as a special code-group.
The bit notation of A, B, C, D, E, F, G, H for an unencoded information octet is used in the description of the 8B/10B transmission code. The bits A, B, C, D, E, F, G, H are translated to bits a, b, c, d, e, i, f, g, h, j of 10-bit transmission code-groups. The 8B/10B encoder is illustrated in 8B/10B Encoder. Each valid code-group has been given a name using the following convention: /Dx.y/ for the 256 valid data code-groups, and /Kx.y/ for special control code-groups, where x is the decimal value of

bits EDCBA, and y is the decimal value of bits HGF. For detailed information, refer to *Tables 36-1* and *36-2* in *IEEE Std 802.3, 2000 Edition, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Section 36.2.4.*

8B/10B Encoder

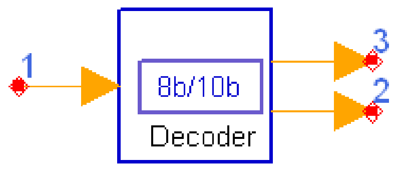


3. Each firing,
   - Eight tokens are consumed at pin *Din*, and one token is consumed at pin *Kin* (control character). Ten tokens are produced at pin output.
   - All the bits are input and output serially.
   - The input at pin *Kin* is the control variable Z, in which 0 means the value D and 1 means the value K.
   - The input at pin *Din* is the unencoded information octet. The LSB bit (A) is input first, while the MSB (H) is input last.
   - The output at pin output is the 10-bit transmission code-group. The LSB bit (a) is output first, while the MSB (j) is output last.

References

1. IEEE Std 802.3, 2000 Edition, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Section 36.2.4.

8b10bDecoder



Description: 8b/10b decoder
Library: Numeric, Communications
Class: SDF8b10bDecoder

Parameters

Name	Description	Default	Type
Delay	number of 10-bit symbol delay	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	input	bits to be decoded	int

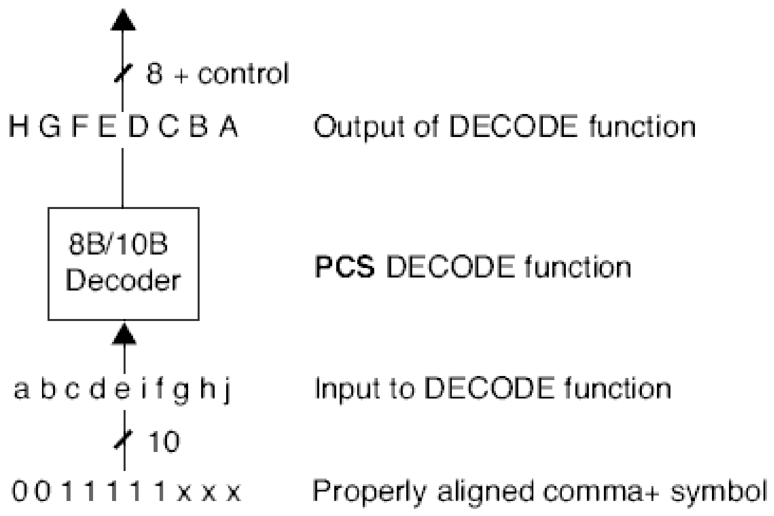
Pin Outputs

Pin	Name	Description	Signal Type
2	output	decoded data bits	int
3	Kout	decoded control bits	int

Notes/Equations

1. The 8B/10B decoder is the reverse procedure of 8B/10B encoder. It's illustrated in 8B/10B Decoder.

8B/10B Decoder



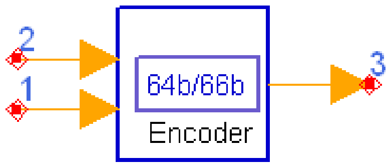
For more information on the 8B/10B Coder, refer to *8b10bCoder* (numeric).

2. Parameter Description:
   Delay specifies the number of 10-bit symbol delay. The decoder begins to work after 10*Delay input tokens.
3. Each firing,
   - Ten tokens are consumed at pin input. One token is produced at pin Kout (control character), and eight tokens are produced at pin output.
   - All the bits are input and output serially.
   - The input at pin input is the 10-bit transmission code-group. The LSB bit (a) is input first, while the MSB (j) is input last.
   - The output at pin Kout is the decoded control variable Z, in which 0 means the value D and 1 means the value K.
   - The output at pin output is the decoded information octet. The LSB bit (A) is output first, while the MSB (H) is output last.

References

1. IEEE Std 802.3, 2000 Edition, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Section 36.2.4.

64b66bCoder



Description: 64b/66b coder
Library: Numeric, Communications
Class: SDF64b66bCoder

Parameters

Name	Description	Default	Type
Scrambler	scramble or not: NO, YES	NO	enum
ScramblerInit	initial state of scrambler	{1,1,1,1,1,1,1, 1,1,1,1,1,1,1 ,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1}	int array

Pin Inputs

Pin	Name	Description	Signal Type
1	input	data to be coded	int
2	CtrlBits	control bits	int

Pin Outputs

Pin	Name	Description	Signal Type
3	output	coded bits	int

Notes/Equations

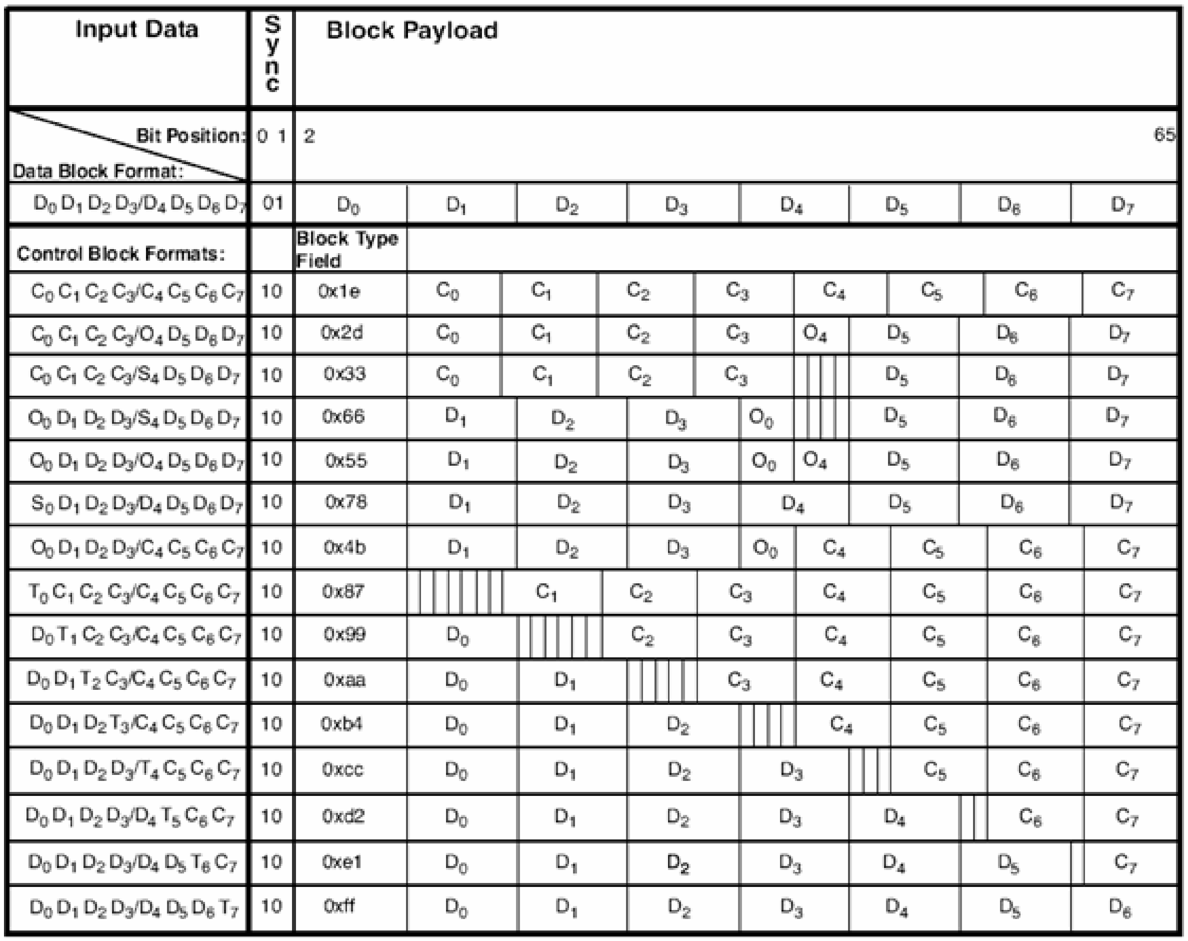
1. The 64B/66B transmission code is used to improve the transmission characteristics of information and to support transmission of control and data characters. The encodings defined by the transmission code ensure that sufficient transitions are present in the PHY bit stream to make clock recovery possible at the receiver. Such encoding also greatly increases the likelihood of detecting any single or multiple bit errors that may occur during transmission and reception of information. In addition, the synchronization headers of the code enable the receiver to achieve block alignment on the incoming PHY bit stream. The 64B/66B transmission code has a high transition density and is a run-length-limited code.
2. 64B/66B encodes 8 data octets or control characters into a block. Blocks containing control characters also contain a block type field. Data octets are labeled D_0 to D_7.

Control characters other than /O/, /S/ and /T/ are labeled C0 to C $_7$. The control character for ordered_set is labeled as O $_0$ or O $_4$ since it is only valid on the first octet of the XGMII. The control character for start is labeled as S $_0$ or S $_4$ for the same reason. The control character for terminate is labeled as T $_0$ to T $_7$.

Two consecutive XGMII transfers provide eight characters that are encoded into one 66-bit transmission block. The subscript in the above labels indicates the position of the character in the eight characters from the XGMII transfers.
Contents of block type fields, data octets and control characters are shown as hexadecimal values. The LSB of the hexadecimal value represents the first transmitted bit. For instance, the block type field 0x1e is sent from left to right as 01111000. The bits of a transmitted or received block are labeled TxB<65:0> and RxB<65:0> respectively where TxB<0> and RxB<0> represent the first transmitted bit. The value of the sync header is shown as a binary value. Binary values are shown with the first transmitted bit (the LSB) on the left.

3. Blocks consist of 66 bits. The first two bits of a block are the synchronization header (sync header). Blocks are either data blocks or control blocks. The sync header is 01 for data blocks and 10 for control blocks. Thus, there is always a transition between the first two bits of a block. The remainder of the block contains the payload. The payload is scrambled and the sync header bypasses the scrambler. Therefore, the sync header is the only position in the block that always contains a transition. This feature of the code is used to obtain block synchronization.
Data blocks contain eight data characters. Control blocks begin with an 8-bit block type field that indicates the format of the remainder of the block. For control blocks containing a Start or Terminate character, that character is implied by the block type field. Other control characters are encoded in a 7-bit control code or a 4-bit O Code. Each control block contains eight characters.
The format of the blocks is as shown in 64B/66B Encoder. In the figure, the column labeled Input Data shows, in abbreviated form, the eight characters used to create the 66-bit block. These characters are either data characters or control characters and, when transferred across the XGMII interface, the corresponding TXC or RXC bit is set accordingly. Within the Input Data column, D0 through D7 are data octets and are transferred with the corresponding TXC or RXC bit set to zero. All other characters are control octets and are transferred with the corresponding TXC or RXC bit set to one. The single bit fields (thin rectangles with no label in the figure) are sent as zero and ignored upon receipt.
Bits and field positions are shown with the least significant bit on the left. Hexadecimal numbers are shown in normal hexadecimal. For example the block type field 0x1e is sent as 01111000 representing bits 2 through 9 of the 66 bit block. The least significant bit for each field is placed in the lowest numbered position of the field.

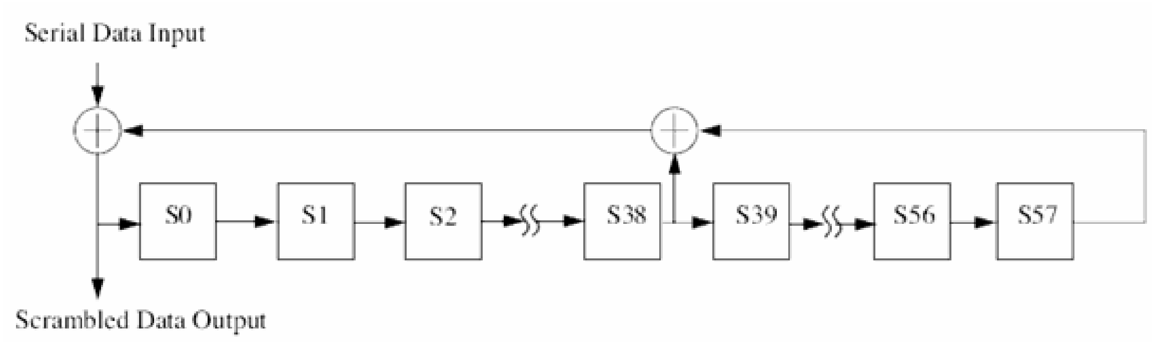
64B/66B Encoder



Input Data	Sync	Block Payload								
Bit Position: / Data Block Format:	0 1	2								65
$D_0 D_1 D_2 D_3/D_4 D_5 D_6 D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
Control Block Formats:		Block Type Field								
$C_0 C_1 C_2 C_3/C_4 C_5 C_6 C_7$	10	0x1e	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$C_0 C_1 C_2 C_3/O_4 D_5 D_6 D_7$	10	0x2d	C_0	C_1	C_2	C_3	O_4	D_5	D_6	D_7
$C_0 C_1 C_2 C_3/S_4 D_5 D_6 D_7$	10	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
$O_0 D_1 D_2 D_3/S_4 D_5 D_6 D_7$	10	0x66	D_1	D_2	D_3	O_0		D_5	D_6	D_7
$O_0 D_1 D_2 D_3/O_4 D_5 D_6 D_7$	10	0x55	D_1	D_2	D_3	O_0	O_4	D_5	D_6	D_7
$S_0 D_1 D_2 D_3/D_4 D_5 D_6 D_7$	10	0x78	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
$O_0 D_1 D_2 D_3/C_4 C_5 C_6 C_7$	10	0x4b	D_1	D_2	D_3	O_0	C_4	C_5	C_6	C_7
$T_0 C_1 C_2 C_3/C_4 C_5 C_6 C_7$	10	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0 T_1 C_2 C_3/C_4 C_5 C_6 C_7$	10	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0 D_1 T_2 C_3/C_4 C_5 C_6 C_7$	10	0xaa	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0 D_1 D_2 T_3/C_4 C_5 C_6 C_7$	10	0xb4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0 D_1 D_2 D_3/T_4 C_5 C_6 C_7$	10	0xcc	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0 D_1 D_2 D_3/D_4 T_5 C_6 C_7$	10	0xd2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0 D_1 D_2 D_3/D_4 D_5 T_6 C_7$	10	0xe1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0 D_1 D_2 D_3/D_4 D_5 D_6 T_7$	10	0xff	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

4. Ordered sets are used to extend the ability to send control and status information over the link such as remote fault and local fault status. Ordered sets consist of a control character followed by three data characters. Ordered sets always begin on the first octet of the XGMII. 10 Gigabit Ethernet uses one kind of ordered_set: the sequence ordered_set. The sequence ordered_set control character is denoted /Q/. An additional ordered_set, the signal ordered_set, has been reserved and it begins with another control code. The 4-bit O field encodes the control code. See Table 49-1 in *IEEE Std 802.3ae-2002, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation, Section 49.2.* for the mappings.
5. A block is invalid if any of the following conditions exists:
   a) The sync field has a value of 00 or 11.
   b) The block type field contains a reserved value.
   c) Any control character contains a value not in Table 49-1.
   d) Any O code contains a value not in Table 49-1.
   e) The set of eight XGMII characters does not have a corresponding block format in 64B/66B Encoder.
6. If parameter *Scrambler* is set as NO, the payload of the block is not scrambled. If it is set as *YES*, the payload of the block is scrambled with a self-synchronizing scrambler. The scrambler shall produce the same result as the implementation shown in Scrambler. This implements the scrambler polynomial: G(x) = 1 + x39 + x58. The parameter *ScramblerInit* is the initial value of the scrambler according to Scrambler. Note that, in this 58-element array parameter *ScramblerInit* , the first element is the initial value in S0 while the 58th element is the initial value in S57. The scrambler is run continuously on all payload bits. The sync header bits bypass the scrambler.

Scrambler

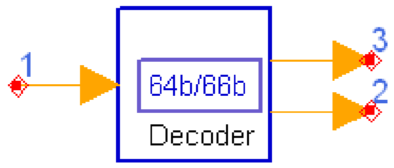


7. Each firing,
    - 64 tokens are consumed at pin input, and 8 tokens are consumed at pin CtrlBits. 66 tokens are produced at pin output.
    - The input at pin input are 8 data octets or control characters. For each data octet or control character, the LSB is input first.
    - Each token at pin CtrlBits indicates the type of corresponding octet at pin input. 0 indicates data octet while 1 indicates control character.
    - All the bits are input and output serially.

References

1. IEEE Std 802.3ae-2002, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation, Section 49.2.

64b66bDecoder



Description: 64b/66b decoder
Library: Numeric, Communications
Class: SDF64b66bDecoder

Parameters

Name	Description	Default	Type
Scrambler	scramble or not: NO, YES	NO	enum
ScramblerInit	initial state of scrambler	{1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1}	int array
Delay	number of 66-bit symbol delayed for descrambler	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	input	bits to be decoded	int

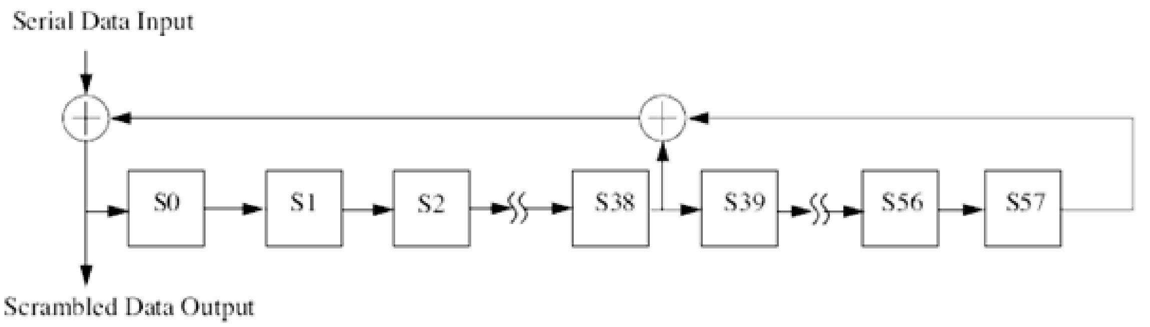
Pin Outputs

Pin	Name	Description	Signal Type
2	output	decoded bits	int
3	CtrlBits	control bits	int

Notes/Equations

1. The 64B/66B decoder is the reverse procedure of the 64B/66B encoder. For more information on the 64B/66B Coder, refer to *64b66bCoder* (numeric).
2. Parameter Description:
   If parameter *Scrambler* is set as *NO*, the payload of the block is not scrambled. If it is set as *YES*, the payload of the block is scrambled with a self-synchronizing scrambler. The scrambler shall produce the same result as the implementation shown in Scrambler. This implements the scrambler polynomial: G(x) = 1 + x39 + x58. The parameter *ScramblerInit* is the initial value of the scrambler according to Scrambler. Note that, in this 58-element array parameter *ScramblerInit*, the first element is the initial value in S0 while the 58th element is the initial value in S57. The scrambler is run continuously on all payload bits. The sync header bits bypass the scrambler.

Scrambler



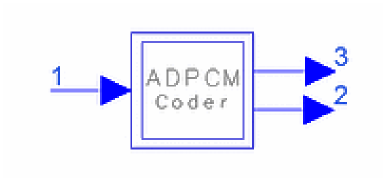
Parameter *Delay* specifies the number of 66-bit symbol delay. The decoder begins to work after 66* *Delay* input tokens.

3. Each firing,
    - 66 tokens are consumed at pin input. 64 tokens are produced at pin output, and 8 tokens are produced at pin CtrlBits (with each corresponding to 8 decoded bits).
    - The output at pin output are 8 data octets or control characters. For each data octet or control character, the LSB is input first.
    - Each token at pin CtrlBits indicates the type of corresponding output octet at pin output. 0 indicates data octet while 1 indicates control character.
    - All the bits are input and output serially.

References

1. IEEE Std 802.3ae-2002, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation, Section 49.2.

ADPCM_Coder



Description: Adaptive Differential Pulse-Code Modulation Encoder
Library: Numeric, Communications
Class: SDFADPCM_Coder

Parameters

Name	Description	Default	Unit	Type	Range
StepSize	Step size of adaptive LMS prediction filter	1.0e-12		real	(∞, ∞)
InitialLMS_Taps	initial taps of adaptive LMS prediction filter	1.0 0.0 [15]		real array	
Range	range of PCM signal level	800		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	analog input signal	real

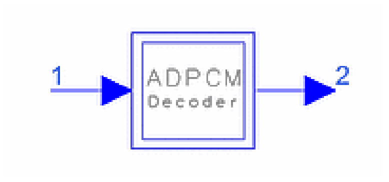
Pin Outputs

Pin	Name	Description	Signal Type
2	d	unquantized ADPCM prediction error signal	real
3	u	quantized ADPCM prediction error signal	real

Notes/Equations

1. ADPCM_Coder is an adaptive differential pulse-code modulation encoder that quantizes to 4-bit (2^4 levels). The adaptive prediction is done with an LMS (least-mean square) adaptive filter.
2. The number of taps in the InitialLMS_Taps parameter sets the order of the LMS filter. The InitialLMS_Taps default value (1.0 0.0 [15]) specifies 16 taps; therefore, the order of the prediction filter is also 16.
3. ADPCM_Coder works with ADPCM_Decoder and ADPCM_ToBits; the Range parameter must be set to the same value in each ADPCM component used.
4. Also see: ADPCM_Decoder, ADPCM_FromBits, ADPCM_ToBits, and LMS.
5. For general information regarding numeric communications components, refer to *Numeric Communications Components* (numeric).

ADPCM_Decoder



Description: Adaptive Differential Pulse-Code Modulation Decoder
Library: Numeric, Communications
Class: SDFADPCM_Decoder

Parameters

Name	Description	Default	Unit	Type	Range
StepSize	step size of adaptive LMS prediction filter	1.0e-12		real	(-∞, ∞)
InitialLMS_Taps	initial taps of adaptive LMS prediction filter	1.0 0.0 [15]		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	quantized ADPCM prediction error signal	real

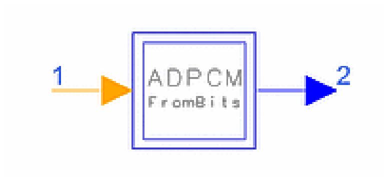
Pin Outputs

Pin	Name	Description	Signal Type
2	output	decoded signal	real

Notes/Equations

1. ADPCM_Decoder is an adaptive differential pulse-code modulation decoder. The adaptive prediction is done with an LMS (least-mean square) adaptive filter.
2. The number of taps in the InitialLMS_Taps parameter sets the order of the LMS filter. The InitialLMS_Taps default value 1.0 0.0 [15] specifies 16 taps; therefore, the order of the prediction filter is also 16.
3. The predicted error signal is internally limited to the range -12000 to $+12000$. This prevents the LMS algorithm from overflowing the floating-point (real) range in the event the algorithm becomes unstable. Instability will still be observable, however, as the output will approach infinity.
4. ADPCM_Decoder works with ADPCM_Coder and ADPCM_FromBits.
5. Also see: ADPCM_Coder, ADPCM_FromBits, ADPCM_ToBits, and LMS.
6. For information regarding numeric communications component signals, refer to the *Numeric Communications Components* (numeric).

ADPCM_FromBits



Description: Adaptive Differential Pulse-Code Modulation Error Signal Decoder
Library: Numeric, Communications
Class: SDFADPCM_FromBits

Parameters

Name	Description	Default	Unit	Type	Range
Range	range of PCM signal level	800		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	4-bit encoded ADPCM error signal	int

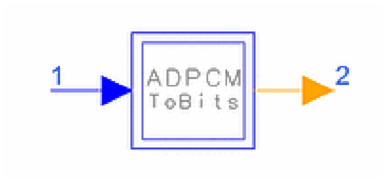
Pin Outputs

Pin	Name	Description	Signal Type
2	output	quantized ADPCM error signal	real

Notes/Equations

1. ADPCM_FromBits decodes a previously encoded quantized ADPCM error signal. For each set of four input bits received, a single quantized ADPCM error signal value is produced.
2. ADPCM_FromBits works with ADPCM_ToBits and ADPCM_Decoder; the Range parameter must be set to the same value in each ADPCM component used.
3. Also see: ADPCM_Coder, ADPCM_Decoder, ADPCM_ToBits.
4. For information regarding numeric communications component signals, refer to *Numeric Communications Components* (numeric).

ADPCM_ToBits



Description: 4-Bit Adaptive Differential Pulse-Code Modulation Error Signal Decoder
Library: Numeric, Communications
Class: SDFADPCM_ToBits

Parameters

Name	Description	Default	Unit	Type	Range
Range	range of PCM signal level	800		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	quantized ADPCM error signal	real

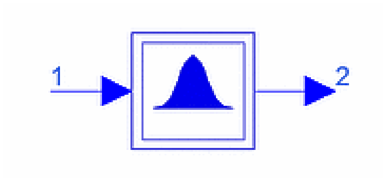
Pin Outputs

Pin	Name	Description	Signal Type
2	output	4-bit code for the received ADPCM error signal value	int

Notes/Equations

1. ADPCM_ToBits encodes a previously quantized ADPCM error signal into a set of 4 bits. For each input value received, four 1-bit outputs are produced.
2. ADPCM_ToBits works with ADPCM_FromBits and ADPCM_Coder; the Range parameter must be set to the same value in each ADPCM component used.
3. Also see: ADPCM_Coder, ADPCM_Decoder, ADPCM_FromBits.
4. For information regarding numeric communications component signals, refer to *Numeric Communications Components* (numeric).

AWGN_Channel



Description: Additive White Gaussian Noise Channel
Library: Numeric, Communications
Class: SDFAWGN_Channel

Parameters

Name	Description	Default	Unit	Type	Range
FwdTaps	forward FIR filter tap to model linear distortion	1		real array	
FdbkTaps	feedback FIR filter tap to model linear distortion	0		real array	
NoisePwr	variance of the additive white Gaussian noise	0.5		real	[0.0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

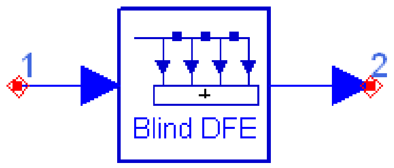
Notes/Equations

1. AWGN_Channel simulates a channel with white Gaussian noise and optional linear distortion. To simulate linear distortion, the input signal is filtered through an FIR filter and fed back through a second FIR filter. White Gaussian noise with zero mean and variance NoisePwr is then added to the signal. The default values of FwdTaps and FdbkTaps cause the signal to be passed through without distortion.
2. AWGN_Channel can be represented as Y = X + G, where G is a zero mean Gaussian random variable with variance σ^2 and X = x_K, k = 0, 1, ... q − 1. For a given X, it follows that Y is Gaussian with mean x_K and variance σ^2. That is,
$$P(Y/X = x_K) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-(Y - x_K)^2/2\sigma^2}$$
For any given input sequence, X_i, i-1, 2 ... , n, there is a corresponding output sequence $Y_i = X_i + G_i$, i = 1, 2, ... n.
3. Also see: NoiseChannel.
4. For information regarding numeric communications component signals, refer to *Numeric Communications Components* (numeric).

References

1. J. G. Proakis, Digital Communications, McGraw-Hill, 1989.

BlindDFE



Description: Blind decision feedback equalizer
Library: Numeric, Communications
Class: SDFBlindDFE

Parameters

Name	Description	Default	Type
NumFFtaps	number of feed-forward taps	5	int
FFinitial	feed-forward filter taps are initialized by users or not: NO, YES	NO	enum
FFtaps	initial feed-forward filter taps (only valid when FFinitial is YES)	{0, 0, 1, 0, 0}	real array
NumFBtaps	number of feedback filter taps	2	int
FBinitial	feedback filter taps are initialized by users or not: NO, YES	NO	enum
FBtaps	initial feedback filter taps (only valid when FBinitial is YES)	{0, 0}	real array
EquAlgorithm	adaptive algorithm: None, CMA, DD	CMA	enum
Fraction	number of samples per symbol at input, range [1, 16]. Fraction=1: symbol-spaced equalizer; Fraction=2~16: fractionally spaced equalizer	1	int
Alpha	step size for tap adjustment	1e-4	real
SaveFFTapsFile	filename in which to save final FF tap values		string
SaveFBTapsFile	filename in which to save final FB tap values		string

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal before equalizer	real

Pin Outputs

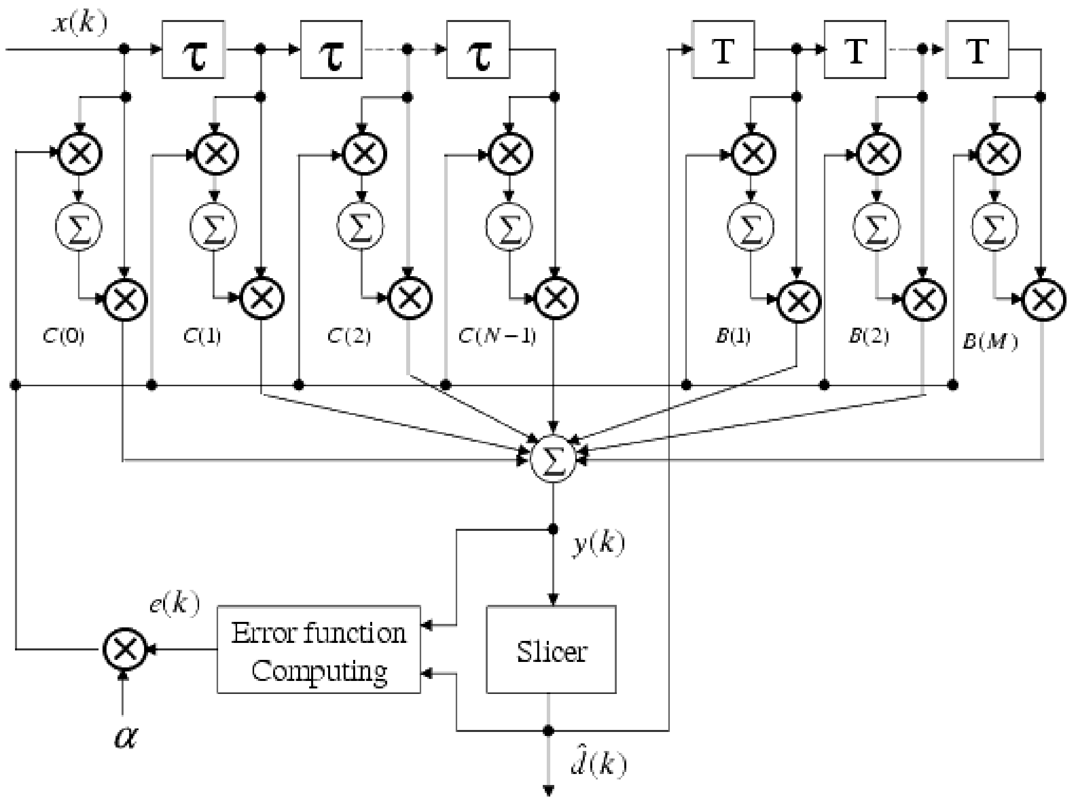
Pin	Name	Description	Signal Type
2	output	output signal after blind decision feedback equalizer	real

Notes/Equations

1. Time-dispersive channels can cause intersymbol interference (ISI). For example, in a multipath scattering environment, the receiver sees delayed versions of a symbol transmission, which can interfere with other symbol transmissions. An equalizer attempts to mitigate ISI and thus improve the receiver's performance. This model is a blind Decision Feedback Equalization (DFE), and it operates in blind equalization algorithm whether the eyes are closed or opened.
2. In each firing, it consumes Fraction input tokens while produces one output token.

3. Implementation:
   A block diagram of the equalizer is shown in Block Diagram of the DFE. This equalizer works in blind equalization modes. During start-up and tracking, this equalizer operates in blind algorithm whether the eyes are opened or closed.

Block Diagram of the DFE



   In Block Diagram of the DFE, {x(k)} is the received sequence before equalizer, {y(k)} is the equalized sequence, (C(0), C(1), ..., C(N-1)) is the taps of feed-forward transversal filter, (B(1), B(2), ..., B(M)) is the taps of feedback filter. e(k) is the error signal of the blind equalization algorithm, the decision signal
   $\hat{d}(k)$
   which is the output of the Slicer, α is the step size to adjust the equalizer taps. If *Fraction=1*,
   $\tau = T$
   the feed-forward transversal filter is a linear equalizer. Otherwise
   $\tau = Fraction \times T$
   where *Fraction* is a parameter, and the feed-forward transversal filter is a Fractionally Spaced Equalizer (FSE). N+M is the number of taps for this equalizer; N is parameter NumFFtaps; M is parameter NumFBtaps; T is the sampling time.
4. Equalization Algorithm
   The blind equalizer algorithm works well when the eyes closed. The adaptive algorithm of CMA and DD are adopted. The difference between LMS algorithm and the blind algorithm is only the error signal e(k). The error signal e(k) is also the difference in all of the blind equalization algorithms.

The error signal of CMA algorithm is as follows

$$e_{CMA}(k) = y(k)(y^2(k) - R^2)$$

where R2=1 in binary case.

$$\hat{d}(k)$$

is the decision signal. The error signal of DD algorithm is as follows:

$$e_{DD}(k) = y(k) - \hat{d}(k)$$

where

$$\hat{d}(k) = \begin{cases} 1.0, if(y(k) \geq 0) \\ -1.0, if(y(k) < 0) \end{cases}$$

The blind equalization algorithm is as follows:

$$C_{k+1} = C_k - \alpha e(k)x(k)$$

$$B_{k+1} = B_k + \alpha e(k)y(k)$$

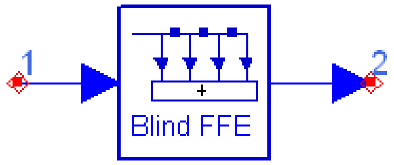
α is parameter Alpha.

5. For blind equalizers, no reference tap is defined to specify the delay introduced by the equalizer. A common way for determining the delay is to compare the equalizer output with the source empirically after the equalizer has converged.
6. Parameter Details:
    - *NumFFtaps* specifies the number of feed-forward taps.
    - *FFinitial* indicates whether the feed-forward filter taps are initialized by users or not. If users don't want to set FFtaps, FFinitial is selected as NO and the FFtaps are generated in code automatically.
    - *FFtaps* specifies the initial value of feed-forward filter taps if FFinitial is YES.
    - *NumFBtaps* specifies the number of feedback filter taps.
    - *FBinitial* indicates whether the feedback filter taps are initialized by users or not. If users don't want to set FBtaps, FBinitial is selected as NO and the FBtaps are generated in code automatically.
    - *FBtaps* specifies the initial value of feedback filter taps if FBinitial is YES.
    - *EquAlgorithm* selects the adaptive algorithm. If NONE is selected, it's used as a non-adaptive equalizer.
    - *Alpha* specifies the step size for tap adjustment.
    - *Fraction* specifies the number of samples per symbol at input, range [1, 16].
    - *SaveFFTapsFile* specifies the filename in which to save final feed-forward tap values. If the SaveFFTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.
    - *SaveFBTapsFile* specifies the filename in which to save final feedback tap values. If the SaveFBTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.

References

1. John G. Proakis, *Digital Communications*, Third Edition, McGraw-Hill, 1995.
2. Dimitris G. Manolakis et.al, *Statistical and Adaptive Signal Processing*, McGraw-Hill, 2000.

BlindFFE



Description: Blind feed-forward equalizer
Library: Numeric, Communications
Class: SDFBlindFFE

Parameters

Name	Description	Default	Type
NumFFtaps	number of feed-forward taps	5	int
FFinitial	feed-forward filter taps are initialized by users or not: NO, YES	NO	enum
FFtaps	initial feed-forward filter taps (only valid when FFinitial is YES)	{0, 0, 1, 0, 0}	real array
EquAlgorithm	adaptive algorithm: None, CMA, DD	CMA	enum
Fraction	number of samples per symbol at input, range [1, 16]. Fraction=1: symbol-spaced equalizer; Fraction=2~16: fractionally spaced equalizer	1	int
Alpha	step size for tap adjustment	1e-4	real
SaveTapsFile	filename in which to save final tap values		string

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal before equalizer	real

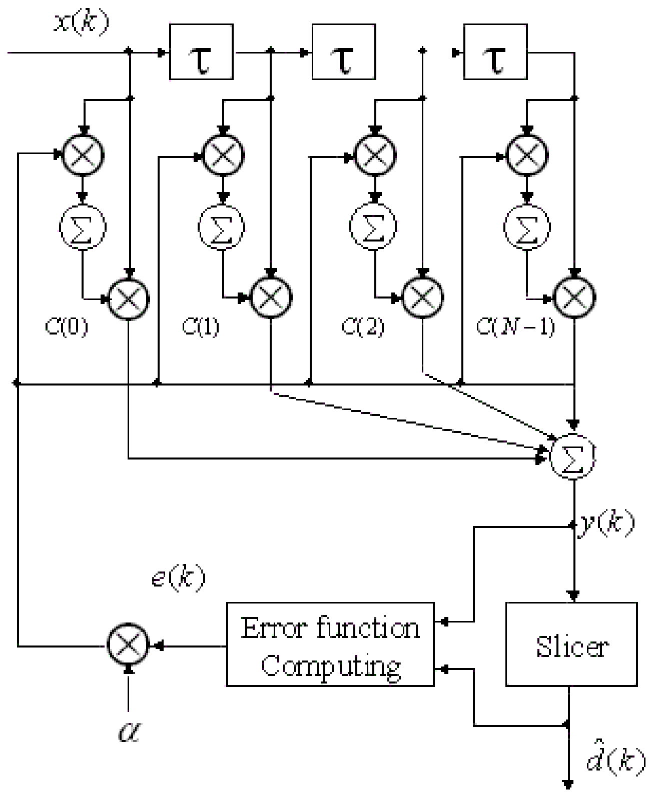
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal after blind feed-forward equalizer	real

Notes/Equations

1. Time-dispersive channels can cause intersymbol interference (ISI). For example, in a multipath scattering environment, the receiver sees delayed versions of a symbol transmission, which can interfere with other symbol transmissions. An equalizer attempts to mitigate ISI and thus improve the receiver's performance. This model is a blind Feed-Forward Equalization (FFE), and it operates in blind equalization algorithm whether the eyes are closed or opened.
2. In each firing, it consumes *Fraction* input tokens while produces one output token.
3. Implementation:
   A block diagram of the equalizer is shown in Block Diagram of the FFE. This equalizer works in blind equalization modes. During start-up and tracking, this equalizer operates in blind algorithm whether the eyes are opened or closed.

Block Diagram of the FFE



In Block Diagram of the FFE, {x(k)} is the received sequence before equalizer, {y(k)} is the equalized sequence, (C(0), C(1), ..., C(N-1)) is the taps of feed-forward transversal filter. e(k) is the error signal of the blind equalization algorithm, the decision signal

$$\hat{d}(k)$$

which is the output of the Slicer, α is the step size to adjust the equalizer taps. If Fraction=1,

$$\tau = T$$

the feed-forward transversal filter is a linear equalizer. Otherwise

$$\tau = Fraction \times T$$

where Fraction is a parameter, and the feed-forward transversal filter is a Fractionally Spaced Equalizer (FSE). N is the number of taps for this equalizer (parameter NumFFtaps), T is the sampling time.

4. Equalization Algorithm
   The blind equalizer algorithm works well when the eyes closed. The adaptive algorithm of CMA and DD are adopted. The difference between LMS algorithm and the blind algorithm is only the error signal e(k). The error signal e(k) is also the difference in all of the blind equalization algorithms.
   The error signal of CMA algorithm is as follows

$$e_{CMA}(k) = y(k)(y^2(k) - R^2)$$

where R2=1 in binary case.

$$\hat{d}(k)$$

is the decision signal. The error signal of DD algorithm is as follows:

$e_{DD}(k) = y(k) - \hat{d}(k)$

where

$$\hat{d}(k) = \begin{cases} 1.0, if(y(k) \geq 0) \\ -1.0, if(y(k) < 0) \end{cases}$$

The blind equalization algorithm is as follows:

$C_{k+1} = C_k - \alpha e(k) x(k)$

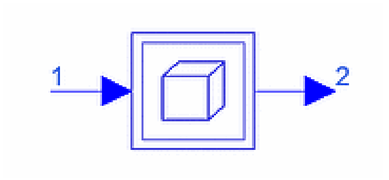
α is parameter Alpha.

5. For blind equalizers, no reference tap is defined to specify the delay introduced by the equalizer. A common way for determining the delay is to compare the equalizer output with the source empirically after the equalizer has converged.
6. Parameter Details:
   - *NumFFtaps* specifies the number of feed-forward taps.
   - *FFinitial* indicates whether the feed-forward filter taps are initialized by users or not. If users don't want to set FFtaps, FFinitial is selected as NO and the FFtaps are generated in code automatically.
   - *FFtaps* specifies the initial value of feed-forward filter taps if FFinitial is YES.
   - *EquAlgorithm* selects the adaptive algorithm. If NONE is selected, it's used as a non-adaptive equalizer.
   - *Alpha* specifies the step size for tap adjustment.
   - *Fraction* specifies the number of samples per symbol at input, range [1, 16].
   - *SaveTapsFile* specifies the filename in which to save final feed-forward tap values. If the SaveTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.

References

1. John G. Proakis, *Digital Communications*, Third Edition, McGraw-Hill, 1995.
2. Dimitris G. Manolakis et.al, *Statistical and Adaptive Signal Processing*, McGraw-Hill, 2000.

BlockPredictor



Description: Block Linear Predictor
Library: Numeric, Communications
Class: SDFBlockPredictor

Parameters

Name	Description	Default	Unit	Type	Range
Order	order of the regression (also number of reflection coefficients to generate)	1		int	(0, ∞)
BlockSize	number of input that use each reflection coefficient set	64		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input random process	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

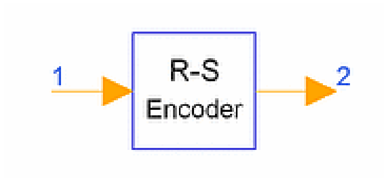
Notes/Equations

1. BlockPredictor consists of Burg's algorithm to estimate the linear predictor coefficients of an input random process and a block lattice to implement forward lattice filter with reflection coefficients that are periodically updated from the output of Burg's algorithm.
2. The BlockSize parameter tells how often the updates occur. This parameter specifies how many input samples are to be processed using each set of reflection coefficients from the output of Burg's algorithm.
3. The Order parameter tells how many reflection coefficients there are. The order of the autoregressive model (all-pole signal model) in Burg's algorithm is also given by the Order parameter.
4. The coefficients of autoregressive modeling in the BlockPredictor are the estimated coefficients of the all-pole filter that could have produced the observations (input data) given a white noise input.The definition of reflection coefficients varies in the literature.
5. The reflection coefficients are the negative of the ones generated by Burg's algorithm in the BlockPredictor, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
6. See also: Burg, BlockLattice, BlockAllPole
7. For information regarding numeric communications component signals, refer to *Numeric Communications Components* (numeric).

References

1. J. Makhoul, "Linear Prediction: A Tutorial Review," *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

CoderRS



Description: Reed Solomon Encoder
Library: Numeric, Communications
Class: SDFCoderRS

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
GF	Define a Galois Field (2^GF)	8	m		int	[2,30]
CodeLength	Length of output codeword	255	n		int	[3,2 ^{m} -1]
MessageLength	Length of input message symbols	223	k		int	[1,CodeLength-2]
PrimPoly	Coefficients of Primitive Polynonial	1 0 1 1 1 0 0 0 1	p(x)		int array	†
Root	The first root of generator polynomial	1	m $_{0}$		int	[0,2 ^{m} -1 - (n - k)]

† PrimPoly must be the coefficients of the m order of polynomial

Pin Inputs

Pin	Name	Description	Signal Type
1	in	information symbol	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	systematical code	int

Notes/Equations

1. This model is used to perform Reed-Solomon (RS) encoding. RS codes are a class of block codes that operate on non-binary symbols. The symbols are formed from *m* bits of a binary data stream. A code block is then formed with $n = 2^{m} - 1$ symbols. In each block, *k* symbols are formed from the encoder input and (*n* - *k*) parity symbols are added. The code is thus a systematic code. The rate of the code is *k/n*, and the code is able to correct up to $t = (n - k - 1)/2$ or $(n - k)/2$ symbol errors in a block, depending on whether *n* - *k* is odd or even. For example, the code used in the WCDMA [1] data transmission system is a (36,32) code shortened from RS code (255,251) defined on Galois Field (2^{8}). A shortened code can be formed by taking 32 input symbols, padding them out with 219 all zero symbols to form 251 symbols, and then encoding with a RS code (255,251). The 219 fixed symbols are then discarded prior to transmission. The input pin consumes k tokens and the output pin produces n

tokens for each firing.

2. Implementation

   The code format is: RS code (n, k), defined on Galois Field (2^m).

   Galois Field Generator

   Galois Fields are set up according to the number of bits per symbol and the number of symbols per block.

   Generate GF (2^m) from the irreducible primitive polynomial. It is defined as the polynomial of least degree, with coefficients in GF(2) and a highest degree coefficient equal to 1. The polynomial is always of degree *m*.

   The elements of Galois Field can have two representations: exponent or polynomial.

   Let α represent the root of the primitive polynomial p(x). Then in GF(2^m), for any $0 \le i \le 2^m - 2$

   $$\alpha^i = b_i(0) + b_i(1)\alpha + b_i(2)\alpha^2 + \cdots + b_i(m-1)\alpha^{m-1}$$

   where the binary vector (bi(0), bi(1),..., bi(m-1)) is the representation of the integer polynomial[i]. Now exponent[i] is the element whose polynomial representation is (bi(0), bi(1),..., bi(m-1)), and exponent[polynomial[i]] = i.

   Polynomial representation is convenient for addition, exponent representation for multiplication.

   RS Encoder

   The RS generator polynomial is generally defined as

   $$g(x) = (x - a^{m_0})(x - a^{m_0+1})\ldots(x - a^{m_0+2t-1})$$

   where t is the correctable error number. It can be reduced to a 2t order of polynomial

   $$g(x) = x^{2t} + g_{2t-1}x^{2t-1} + \ldots + g_0$$

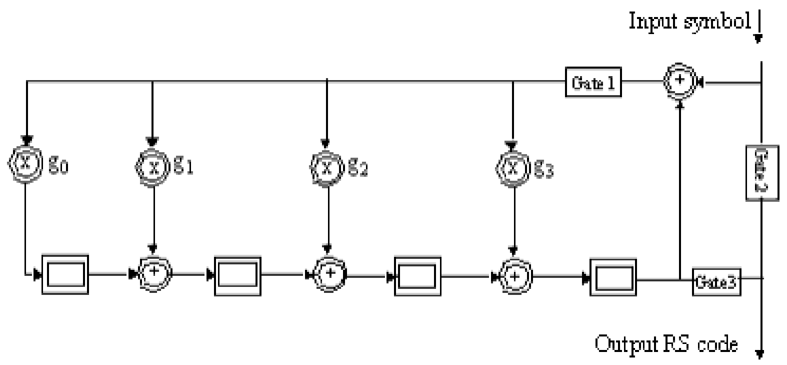
   Encoding is done by using a feedback shift register with appropriate connections specified by the element g_i . The encoded symbol is then

   $$in(x) \times x^{(n-k)} + parity(x)$$

   where in(x) is the polynomial representation of the input data, parity(x) is the polynomial of the parity symbol.

   The RS encoder diagram is illustrated in Reed Solomon Encoder.

   Reed Solomon Encoder

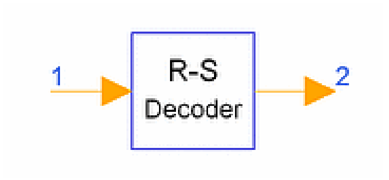


3. For information regarding numeric communications component signals, refer to *Numeric Communications Components* (numeric).

References

1. NTT Mobile Communications Network Inc. "Specifications for W-CDMA Mobile Communication System Experiment", October 9, 1997.
2. S. Lin, D. J. Costello, Error Control Coding Fundamentals and Applications, 1983.

DecoderRS



Description: Reed Solomon Decoder
Library: Numeric, Communications
Class: SDFDecoderRS

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
GF	Define a Galois Field (2^GF)	8	m		int	[2,30]
CodeLength	Length of input codewords	255	n		int	[3,2 ^{m} -1]
MessageLength	Length of output message symbols	223	k		int	[1,CodeLength-2]
PrimPoly	Coefficients of primitive polynomial	1 0 1 1 1 0 0 0 1	p(x)		int array	†
Root	First root of generator polynomial	1	m_0		int	[0,2 ^{m}-1 - (n - k)]

† PrimPoly must be the coefficients of the m order of polynomial

Pin Inputs

Pin	Name	Description	Signal Type
1	in	received symbol	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	decoded symbol	int

Notes/Equations

1. This model is used to perform RS decoding via the Berlekamp iterative algorithm [2].
2. The Berlekamp iterative algorithm locates the error in RS code and generates an error location polynomial. By finding the root of the error location polynomial, the error position can be determined. If decoding is successful, the information symbols are output; otherwise, the received data is unaltered. The input pin consumes n tokens and the output pin produces k tokens.
3. Decoding routines are described here.
   For the shortened code, the same number of symbols 0 is inserted into the same position as CoderRS and a Reed Solomon decoder is used to decode the block. After decoding, the padded symbols are discarded, leaving the desired information symbols.
   Syndromes indicate erroneous situations. When the generator polynomial g(x) and

the received codeword represented by r(x) are given, one or more errors have occurred during transmission of an encoded block.
Let

$$v(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_{n-1} x^{n-1}$$

where v(x) is the polynomial representation of the transmitted symbol.

$$r(x) = r_0 + r_1 x + r_2 x^2 + \ldots + r_{n-1} x^{n-1}$$

where r(x) is the polynomial representation of the received symbol.
Then

$$r(x) = v(x) + e(x)$$

where e(x) denotes the error patterns.
If r_i - v_i, then $e_i = 0$; else $e_i = 1$.

Remember that
$v(x) = g(x)Q(x)$
where Q(x) is the quotient.
So if α^i is the root of g(x), then $v(\alpha^i) = 0$ and $r(\alpha^i) = e(\alpha^i)$.
Now there is a simple procedure for checking the occurrence of errors at the receiver:
Calculate syndromes $s(i)$, the syndromes are decided by the error patterns:

$$s(i) = e(\alpha^{m_0 + i})$$

If one or more of the syndromes are not equal to zero, one or more symbol errors occur in the received data. For example, if

$$\alpha^{m_0}, \alpha^{m_0 + 1}, \ldots, \alpha^{m_0} + 2^{t-1}$$

are roots of g(x), then

$$s(1) = r(\alpha^{m_0})$$

$$s(1) = r(\alpha^{m_0 + 1})$$

.
.
.

$$s(2t) = r(\alpha^{m_0 + 2t - 1})$$

Syndromes are used to find the error location polynomial.
Given the syndromes s(i), the decoding algorithm will synthesize an error location polynomial. The roots of the polynomial indicate the error positions.
Assuming the received symbols have v symbol errors, the syndromes are represented as:

$$s(1) = \beta_1^{m_0} + \beta_2^{m_0} + \ldots + \beta_v^{m_0}$$

$$s(2) = \beta_1^{m_0 + 1} + \beta_2^{m_0 + 1} + \ldots \beta_v^{m_0 + 1}$$

.
.
.

$$s(2t) = \beta_1^{m_0 + 2t - 1} + \beta_2^{m_0 + 2t - 1} + \ldots \beta_v^{m_0 + 2t - 1}$$

where the error location is

$\beta_l = a^{i_l}$

and

$a^{i_l}(1 \le l \le v)$

Now the error location polynomial is defined as

$$\Omega(x) = \left(1 + \beta_1^{m_0} x\right)\left(1 + \beta_2^{m_0} x\right)\ldots\left(1 + \beta_v^{m_0} x\right)$$

$$= \Omega_0 + \Omega_1 x + \Omega_2 x^2 + \ldots + \Omega_v x^v$$

The Berlekamp iterative algorithm is used to construct this polynomial, which is the key to RS decoding.
The algorithm is described here without proof; for more information, see Ref. [1].
An iterative table will be filled.

μ	$\Omega^{(u)}(x)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	s_1	0	0
1				
2 ... , 2t				

where
μ
is the iterative step number
d_μ
is the μth step iterative difference
l_μ
is the order of $\Omega^{(\mu)}(x)$
If
$d_\mu = 0$
then
$\Omega^{(\mu+1)}(x) = \Omega_\mu^{(\mu)}(x)$
and
$l_{\mu+1} = l_\mu$
If
$d_\mu \neq 0$
search for lines in the table to find step *p* in which $d_p \neq 0$ and the value of $p - l_p$ is the maximum, then

$\Omega^{(\mu+1)}(x) = \Omega^{(\mu)}(x) - d_\mu d_\rho^{-1} x^{(\mu-\rho)} \Omega^{(\rho)}(x)$

and

$l_{\mu+1} = max(l_\mu, l_\rho + \mu - \rho)$

For the two conditions

$d_{\mu+1} = s_{\mu+2} + \Omega_1^{(\mu+1)} s_{\mu+1} + \ldots + \Omega_{l_\mu+1}^{(\mu+1)} s_{\mu+2-l_{\mu+1}}$

Iterate until the last line of the table $\Omega^{(2t)}(x)$ is calculated. If the order of the

polynomial is greater than *t* (which means the received codeword block has more than *t* errors) the error cannot be corrected.
For non-binary codes, the error values must be known.
The minimum order polynomial is iteratively solved to obtain the least number of roots (error location number). The inverse element of the root indicates the error location.
The error value is calculated based on the Ref. [2] equation

$$e_{jl} = \beta_l^{(1-m_0)} \frac{z(\beta_l^{-1})}{\prod_{\substack{i=1 \\ i \neq l}}^{v} (1 + \beta_i \beta_l^{-1})}$$

where

$$z(x) = 1 + (s_1 + \Omega_1)x + (s_2 + \Omega_1 s_1 + \Omega_2)x^2 + \\ \ldots + (s_v + \Omega_1 s_{v-1} + \Omega_2 s_{v-2} + \ldots + \Omega_v)x^v$$

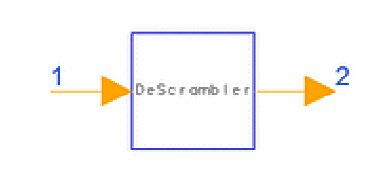
Then,

$$out(x) = r(x) - e(x)$$

References

1. E.R. Berlekamp, Algebraic Coding Theory, McGraw-Hill, New York, 1968.
2. S. Lin, D. J. Costello, Error Control Coding Fundamentals and Applications, 1983.

DeScrambler



Description: Input bit sequence descrambler
Library: Numeric, Communications
Class: SDFDeScrambler
C++ Code: See *doc/sp_items/SDFDeScrambler.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Polynomial	generator polynomial for the shift register - decimal, octal, or hex integer	0440001		int	(0, ∞)
ShiftReg	initial state of the shift register - decimal, octal, or hex integer	1		int	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input bit sequence (zero or nonzero)	int

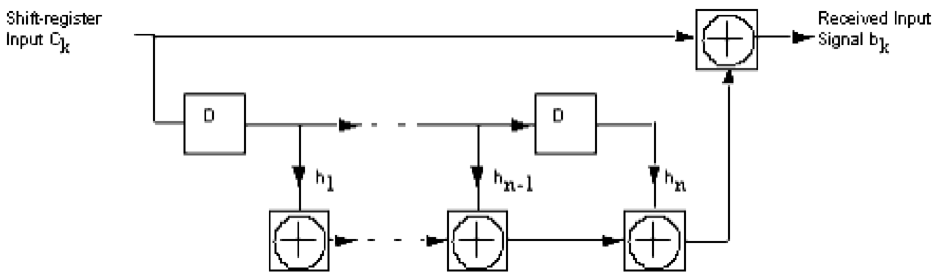
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output bit sequence (zero or one)	int

Notes/Equations

1. This component descrambles the input bit sequence using a feedback shift register. The taps of the feedback shift register are given by the Polynomial parameter. This is a self-synchronizing descrambler that will exactly reverse the operation of the Scrambler component if the corresponding parameter values of Scrambler and DeScrambler are the same.
   A self-synchronized descrambler is shown in Self-Synchronized Descrambler.

Self-Synchronized Descrambler

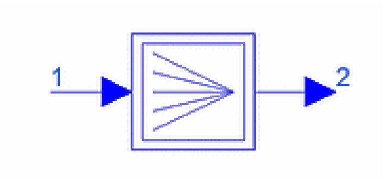


2. See also, *Scrambler* (numeric).

References

1. E. A. Lee and D. G. Messerschmitt, *Digital Communication*, Second Edition, Kluwer Academic Publishers, 1994, pp. 595-603.

DeSpreader



Description: Frame Synchronized Direct-Sequence Spread Spectrum Demodulator
Library: Numeric, Communications
Class: SDFDeSpreader

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input spread spectrum signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	demodulated signal	real

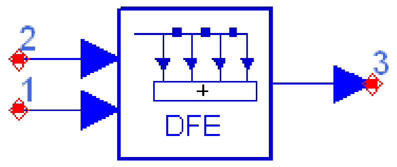
Notes/Equations

1. DeSpreader is a frame synchronized direct-sequence spread spectrum demodulator. Each input sample is demodulated with a 31-bit pseudo-noise spreading code. This *despreads* the signal.
2. See also Spread, and RecSpread.

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 9.

DFE



Description: decision feedback equalizer
Library: Numeric, Communications
Class: SDFDFE

Parameters

Name	Description	Default	Unit	Type	Range
NumFFtaps	number of feed-forward taps	5		int	
FFinitial	feed-forward filter taps are initialized by users or not: NO, YES	NO		enum	
FFtaps	initial feed-forward filter taps (only valid when FFinitial is YES)	{0, 0, 0, 0, 0}		real array	
NumFBtaps	number of feedback filter taps	2		int	
FBinitial	feedback filter taps are initialized by users or not: NO, YES	NO		enum	
FBtaps	initial feedback filter taps (only valid when FBinitial is YES)	{0, 0}		real array	
EquAlgorithm	adaptive algorithm: None, LMS, RLS, ZF	LMS		enum	
TrainSeqLen	length of training sequence	1000		int	
Fraction	number of samples per symbol at input, range [1, 16]. Fraction=1: symbol-spaced equalizer; Fraction=2~16: fractionally spaced equalizer	1		int	
RefTap	index of reference tap for LMS and RLS algorithms, range [1, NumFFtaps]	3		int	
Alpha	step size for LMS algorithm	1e-3		real	
Lambda	weighting factor for RLS algorithm	0.999		real	(0.0, 1.0)
Delta	small positive constant for RLS algorithm	0.001		real	(0.0, 10.0]
TargetMSE	reference MSE in dB for stopping updating coefficients when RLS equalizer reaches this MSE	-40	dB	real	(-100, 100]
SaveFFTapsFile	filename in which to save final FF tap values			string	
SaveFBTapsFile	filename in which to save final FB tap values			string	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal before equalizer	real
2	TrainSeq	input training sequence for equalizer	real

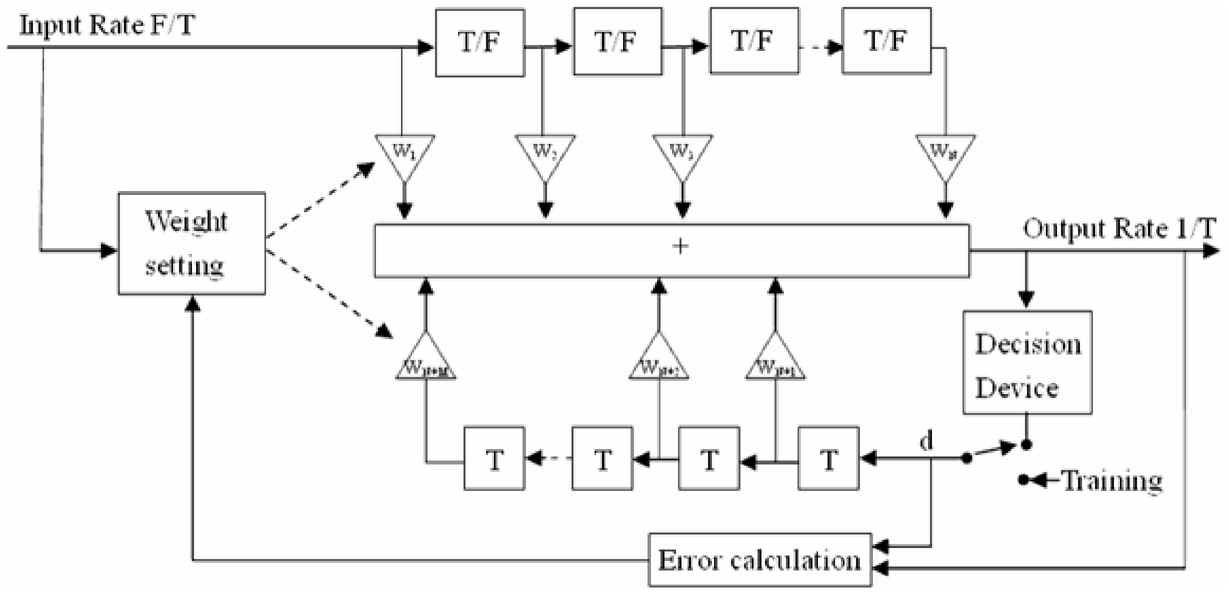
Pin Outputs

Pin	Name	Description	Signal Type
3	output	output signal after decision feedback equalizer	real

Notes/Equations

1. Time-dispersive channels can cause intersymbol interference (ISI). For example, in a multipath scattering environment, the receiver sees delayed versions of a symbol transmission, which can interfere with other symbol transmissions. An equalizer attempts to mitigate ISI and thus improve the receiver's performance. This model is a Decision Feedback Equalization (DFE), and it operates with training sequence. In each firing, the input consumes *Fraction* input token and TrainSeq consumes one input token, while produces one output token.
2. A block diagram of the equalizer is shown in Block Diagram of the DFE. This equalizer works in training sequence mode.

Block Diagram of the DFE



Here F means *Fraction*.

3. If the parameter *EquAlgorithm* is set to ZF, RLS or LMS, the equalizer works in training and tracking modes. In the training mode, the training sequence (from Pin TrainSeq) is used as training sequence. The number of training sequence is *TrainSeqLen*. The error signal ek is from the training signal:

$$e_k = I_k - \hat{I}_k$$

$$I_k$$

is the input training sequence.

$$\hat{I}_k$$

is the equalized output sequence.

After the training mode, the decision feedback equalizer coefficient is converged and the equalizer enters into the tracking mode. When the parameter *EquAlgorithm* is set to ZF, LMS or RLS, the ZF, LMS or RLS adaptive algorithm is used in tracking mode correspondingly. The error signal ek is from the decision signal of the equalized signal:

$$e_k = \tilde{I}_k - \hat{I}_k$$

where $\tilde{I}_k$

is the detected output sequence for binary case:

$$\tilde{I}_k = \begin{cases} 1.0, \hat{I}_k \geq 0 \\ -1.0, \hat{I}_k < 0 \end{cases}$$

If the parameter *EquAlgorithm* is set to NONE, the equalizer works in non-adaptive mode with fixed coefficients.

LMS Algorithm

The criterion most commonly used in the optimization of the equalizer coefficients is the minimization of the mean square error (MSE) between the desired equalizer output and the actual equalizer output.

MSE minimization can be accomplished recursively by use of the stochastic gradient algorithm introduced by Widrow, called the LMS algorithm. This algorithm is described by the coefficient update equation

$$C_{k+1} = C_k + \alpha e_k X_k^*$$

where

C $_k$ is the vector of the equalizer coefficients at the kth iteration *X* $_k$ represents the signal vector.

α is parameter *Alpha*.

This algorithm is applied on both the forward filter and the feedback filter in DFE equalizer.

RLS Algorithm

The convergence rate of the LMS algorithm is slow because a single parameter α controls the rate of adaptation. A fast converging algorithm is obtained if a recursive least squares (RLS) criterion is adopted for adjustment of the equalizer coefficients. The RLS iteration algorithm follows.

Calculate output:

$$\hat{I}_k = C_k \times X_k$$

Calculate Kalman gain vector:

$$K_k = \frac{P_{k-1} \times X_k^*}{\lambda + X_k^T \times P_{k-1} \times X_k^*}$$

Update inverse of the correlation matrix:

$$P_k = \frac{1}{\lambda} \times \left[P_{k-1} - K_k \times X_k^T \times P_{k-1}\right]$$

Update coefficients:

$$C_k = C_{k-1} + P_k \times X_k^* \times e_k$$

λ

is parameter Lamda.

P_k

is a diagonal matrix with initial value Delta*I (here I is a diagonal matrix).

Delta is parameter *Delta*.
This algorithm is applied on both the forward filter and the feedback filter in DFE equalizer.
The updating of coefficients in RLS algorithm will be halted when the MSE averaged over 100 consecutive symbols is less than a reference MSE defined by *TargetMSE*.

ZF Algorithm

The zero-forcing (ZF) solution is achieved by forcing the cross-correlation between the error sequence

e_k

and the desired information sequence $\{I_k\}$ to be zero.

When in the training mode, the coefficients are updated as:

$$C_{k+1} = C_k + \alpha e_k I_k^*$$

When in the tracking mode, the coefficients are updated as:

$$C_{k+1} = C_k + \alpha e_k \tilde{I}_k^*$$

where $\tilde{I}_k$
is the detected output sequence.
Since ZF is a linear equalizer, this algorithm will be applied only on the forward filter in DFE equalizer while LMS algorithm is applied on the feedback filter in DFE equalizer.

4. For LMS and RLS algorithms, the total delay caused by the equalizer is equal to (RefTap-1)/Fraction. Usually the reference tap is set to the center tap in a linear equalizer, or the center tap of the forward filter in a DFE equalizer.
For ZF algorithm, no delay is introduced by this equalizer after the equalizer has converged. Note that ZF algorithm has a condition that the input signal needs to have the eye open prior to equalization. That is, the convergence of ZF algorithm requires

$$\frac{1}{f_0}\sum_{n=1}^{L} |f_n| < 1$$

L is the number of ISI affected symbols and the impulse response {fn} are coefficients of the linear filter model which causes ISI.

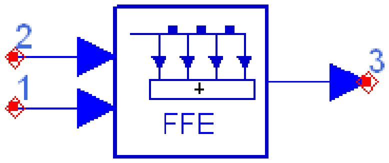
5. Parameter Details:
   - *NumFFtaps* specifies the number of feed-forward taps.
   - *FFinitial* indicates whether the feed-forward filter taps are initialized by users or not. If users don't want to set FFtaps, FFinitial is selected as NO and the FFtaps are generated in code automatically.
   - *FFtaps* specifies the initial value of feed-forward filter taps if FFinitial is YES.
   - *NumFBtaps* specifies the number of feedback filter taps.
   - *FBinitial* indicates whether the feedback filter taps are initialized by users or not. If users don't want to set FBtaps, FBinitial is selected as NO and the FBtaps are generated in code automatically.
   - *FBtaps* specifies the initial value of feedback filter taps if FBinitial is YES.
   - *EquAlgorithm* selects the equalizer algorithm.
   - *TrainSeqLen* specifies the length of training sequence.
   - *Fraction* specifies the number of samples per symbol at input, range [1, 16].
   - *RefTap* specifies the index of reference tap for LMS and RLS algorithms, ranged from 1 to NumFFtaps.
   - *Alpha* specifies the step size for tap adjustment.
   - *Lambda* specifies weighting factor for RLS algorithm.

- *Delta* specifies a small positive constant for RLS algorithm.
- *TargetMSE* specifies the reference MSE in dB for RLS algorithm. RLS equalizer will stop updating coefficients when the MSE averaged over 100 consecutive symbols is less than this reference.
- *SaveFFTapsFile* specifies the filename in which to save final feed-forward tap values. If the SaveFFTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.
- *SaveFBTapsFile* specifies the filename in which to save final feedback tap values. If the SaveFBTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.

References

1. John G. Proakis, Digital Communications, Third Edition, McGraw-Hill, 1995.
2. Dimitris G. Manolakis et.al, Statistical and Adaptive Signal Processing, McGraw-Hill, 2000.

FFE



Description: feed-forward equalizer
Library: Numeric, Communications
Class: SDFFFE

Parameters

Name	Description	Default	Unit	Type	Range
NumFFtaps	number of feed-forward taps	5		int	
FFinitial	feed-forward filter taps are initialized by users or not: NO, YES	NO		enum	
FFtaps	initial feed-forward filter taps (only valid when FFinitial is YES)	{0, 0, 0, 0, 0}		real array	
EquAlgorithm	adaptive algorithm: None, LMS, RLS, ZF	LMS		enum	
TrainSeqLen	length of training sequence	1000		int	
Fraction	number of samples per symbol at input, range [1, 16]. Fraction=1: symbol-spaced equalizer; Fraction=2~16: fractionally spaced equalizer	1		int	
RefTap	index of reference tap for LMS and RLS algorithms, range [1, NumFFtaps]	3		int	
Alpha	step size for LMS algorithm	1e-3		real	
Lambda	weighting factor for RLS algorithm	0.999		real	(0.0, 1.0)
Delta	small positive constant for RLS algorithm	0.001		real	(0.0, 10.0]
TargetMSE	reference MSE in dB for stopping updating coefficients when RLS equalizer reaches this MSE	-40	dB	real	(-100, 100]
SaveTapsFile	filename in which to save final tap values			string	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal before equalizer	real
2	TrainSeq	input training sequence for equalizer	real

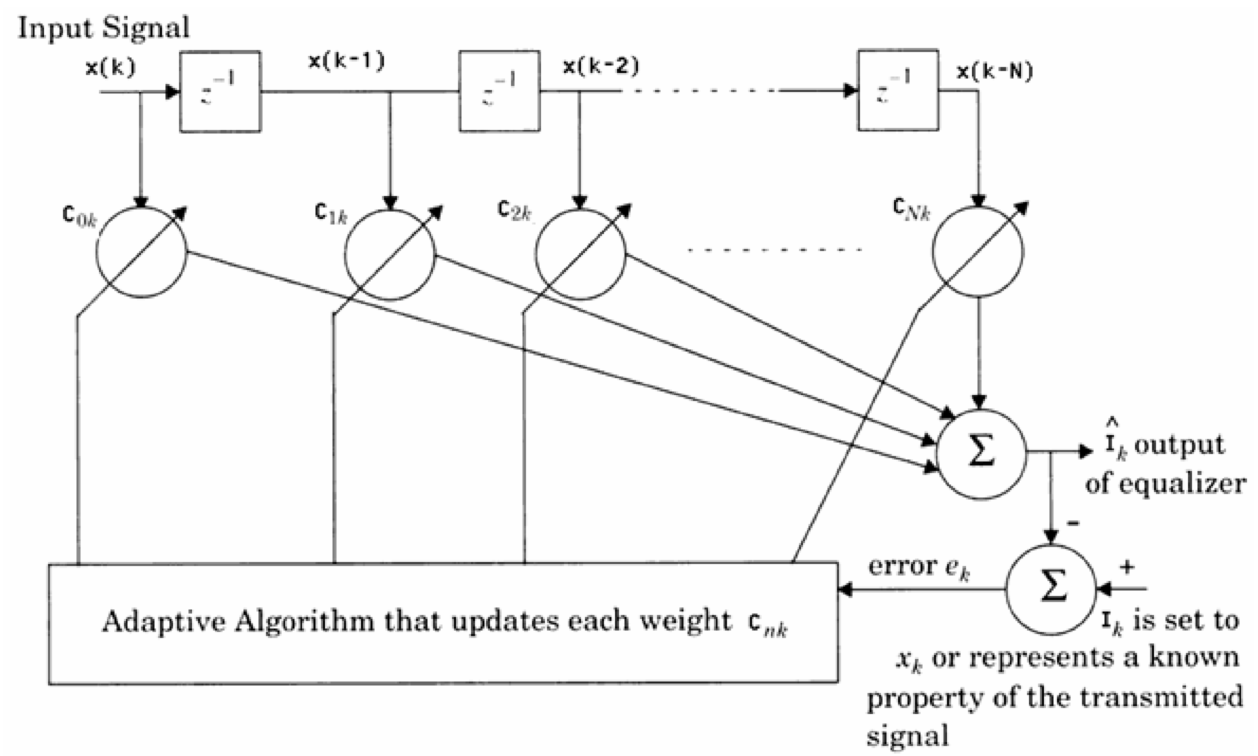
Pin Outputs

Pin	Name	Description	Signal Type
3	output	output signal after decision feedback equalizer	real

Notes/Equations

1. Time-dispersive channels can cause intersymbol interference (ISI). For example, in a multipath scattering environment, the receiver sees delayed versions of a symbol transmission, which can interfere with other symbol transmissions. An equalizer attempts to mitigate ISI and thus improve the receiver's performance. This model is a Feed-Forward Equalization (FFE), and it operates with training sequence. In each firing, the input consumes *Fraction* input token and TrainSeq consumes one input token, while produces one output token.
2. A block diagram of the equalizer is shown in Block Diagram of the FFE. This equalizer works in training sequence mode.

Block Diagram of the FFE


Input Signal
x(k)
x(k-1)
x(k-2)
x(k-N)
z^{-1}
c_{0k}
c_{1k}
c_{2k}
c_{Nk}
Σ
$\hat{I}_k$ output of equalizer
-
+
error e_k
Adaptive Algorithm that updates each weight c_{nk}
I_k is set to x_k or represents a known property of the transmitted signal

3. If the parameter *EquAlgorithm* is set to ZF, RLS or LMS, the equalizer works in training and tracking modes. In the training mode, the training sequence (from Pin TrainSeq) is used as training sequence. The number of training sequence is TrainSeqLen. The error signal e_k is from the training signal:

$$e_k = I_k - \hat{I}_k$$

I_k

is the input training sequence.

$\hat{I}_k$

is the equalized output sequence.

After the training mode, the decision feedback equalizer coefficient is converged and the equalizer enters into the tracking mode. When the parameter *EquAlgorithm* is set to ZF, LMS or RLS, the ZF, LMS or RLS adaptive algorithm is used in tracking mode correspondingly. The error signal e_k is from the decision signal of the equalized signal:

$$e_k = \tilde{I}_k - \hat{I}_k$$

where $\tilde{I}_k$
is the detected output sequence for binary case:

$$\tilde{I}_k = \begin{cases} 1.0, \hat{I}_k \geq 0 \\ -1.0, \hat{I}_k < 0 \end{cases}$$

If the parameter *EquAlgorithm* is set to NONE, the equalizer works in non-adaptive mode with fixed coefficients.

LMS Algorithm

The criterion most commonly used in the optimization of the equalizer coefficients is the minimization of the mean square error (MSE) between the desired equalizer output and the actual equalizer output.

MSE minimization can be accomplished recursively by use of the stochastic gradient algorithm introduced by Widrow, called the LMS algorithm. This algorithm is described by the coefficient update equation

$$C_{k+1} = C_k + \alpha e_k X_k^*$$

where

C $_k$ is the vector of the equalizer coefficients at the kth iteration *X* $_k$ represents the signal vector.

α is parameter *Alpha*.

RLS Algorithm

The convergence rate of the LMS algorithm is slow because a single parameter α controls the rate of adaptation. A fast converging algorithm is obtained if a recursive least squares (RLS) criterion is adopted for adjustment of the equalizer coefficients. The RLS iteration algorithm follows.

Calculate output:

$$\hat{I}_k = C_k \times X_k$$

Calculate Kalman gain vector:

$$K_k = \frac{P_{k-1} \times X_k^*}{\lambda + X_k^T \times P_{k-1} \times X_k^*}$$

Update inverse of the correlation matrix:

$$P_k = \frac{1}{\lambda} \times \left[P_{k-1} - K_k \times X_k^T \times P_{k-1}\right]$$

Update coefficients:

$$C_k = C_{k-1} + P_k \times X_k^* \times e_k$$

λ

is parameter Lamda.

P_k

is a diagonal matrix with initial value Delta*I (here I is a diagonal matrix).

Delta is parameter *Delta*.

The updating of coefficients in RLS algorithm will be halted when the MSE averaged over 100 consecutive symbols is less than a reference MSE defined by *TargetMSE*.

ZF Algorithm

The zero-forcing (ZF) solution is achieved by forcing the cross-correlation between the error sequence

e_k

and the desired information sequence {Ik} to be zero.

When in the training mode, the coefficients are updated as:

$$C_{k+1} = C_k + \alpha e_k I_k^*$$

When in the tracking mode, the coefficients are updated as:

$$C_{k+1} = C_k + \alpha e_k \tilde{I}_k^*$$

where

$\tilde{I}_k$

is the detected output sequence.

4. For LMS and RLS algorithms, the total delay caused by the equalizer is equal to (RefTap-1)/Fraction. Usually the reference tap is set to the center tap in a linear equalizer, or the center tap of the forward filter in a DFE equalizer.
   For ZF algorithm, no delay is introduced by this equalizer after the equalizer has converged. Note that ZF algorithm has a condition that the input signal needs to have the eye open prior to equalization. That is, the convergence of ZF algorithm requires

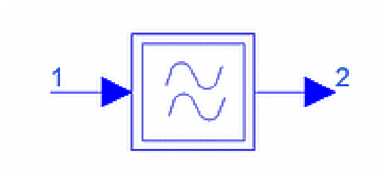
   $$\frac{1}{f_0}\sum_{n=1}^{L}|f_n| < 1$$

   L is the number of ISI affected symbols and the impulse response $\{f_n\}$ are coefficients of the linear filter model which causes ISI.
5. Parameter Details:
   - *NumFFtaps* specifies the number of feed-forward taps.
   - *FFinitial* indicates whether the feed-forward filter taps are initialized by users or not. If users don't want to set FFtaps, FFinitial is selected as NO and the FFtaps are generated in code automatically.
   - *FFtaps* specifies the initial value of feed-forward filter taps if FFinitial is YES.
   - *EquAlgorithm* selects the equalizer algorithm.
   - *TrainSeqLen* specifies the length of training sequence.
   - *Fraction* specifies the number of samples per symbol at input, range [1, 16].
   - *RefTap* specifies the index of reference tap for LMS and RLS algorithms, ranged from 1 to NumFFtaps.
   - *Alpha* specifies the step size for tap adjustment.
   - *Lambda* specifies weighting factor for RLS algorithm.
   - *Delta* specifies a small positive constant for RLS algorithm.
   - *TargetMSE* specifies the reference MSE in dB for RLS algorithm. RLS equalizer will stop updating coefficients when the MSE averaged over 100 consecutive symbols is less than this reference.
   - *SaveTapsFile* specifies the filename in which to save final feed-forward tap values. If the SaveTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.

References

1. John G. Proakis, Digital Communications, Third Edition, McGraw-Hill, 1995.
2. Dimitris G. Manolakis et.al, Statistical and Adaptive Signal Processing, McGraw-Hill, 2000.

FreqPhase



Description: Frequency Offset or Phase Jitter Sampler
Library: Numeric, Communications
Class: SDFFreqPhase

Parameters

Name	Description	Default	Unit	Type	Range
SampleRate	input signal sample rate	2 * PI		real	[0, ∞)
PhaseJitterFrequencyHz	frequency of phase jitter distortion to add to signal	0.0		real	[0.0, ∞)
FrequencyOffsetHz	frequency offset distorion to add to signal	0.0		real	[0.0, ∞)
PhaseJitterAmplitudeDeg	phase jitter peak amplitude, in degrees	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real

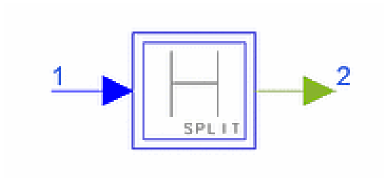
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output signal	real

Notes/Equations

1. FreqPhase can be used to impose a frequency offset or phase jitter, or both, on a signal in order to model channels (such as telephone channels) that suffer these impairments.
2. Very low- and very high-frequency signals (near the Nyquist frequency) will be distorted due to the Hilbert filter.
3. See also, *PhaseShift* (numeric).

HilbertSplit



Description: Real to Analytic Signal Converter
Library: Numeric, Communications
Class: SDFHilbertSplit

Parameters

Name	Description	Default	Unit	Type	Range
Delay	processing delay of this block	32		int	[0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	real input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	analytic output signal	complex

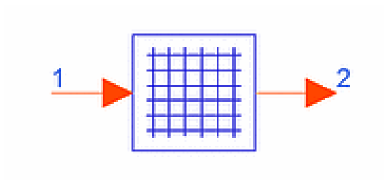
Notes/Equations

1. HilbertSplit converts the real input signal into an analytic signal using a phase splitter. The Delay parameter determines the length and, therefore, the accuracy of the Hilbert filter used. The Hilbert filter has (2 × Delay + 1) taps. A larger value for delay gives a more accurate filter, but increases the processing time and the delay through the system. The component scales the input signal so that input and output signals have the same rms value.
2. See also, *Hilbert* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

InterleaveDeinterleave



Description: Interleaver / Deinterleaver
Library: Numeric, Communications
Class: SDFInterleaveDeinterleave
Derived From: Transpose
C++ Code: See *doc/sp_items/SDFInterleaveDeinterleave.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Rows	number of rows of the interleave/deinterleave matrix	8		int	(0, ∞)
Columns	number of columns of the interleave/deinterleave matrix	8		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. This component is a general purpose interleaver/de-interleaver. Every time it fires it reads (Rows × Columns) samples from its input and writes them to its output in a different order. Its operation is equivalent to writing the samples read from its input in a Rows × Columns matrix row-wise, then reading the matrix elements column-wise and writing them to the output.
   Alternatively, the *Transpose* (numeric) component in the Numeric Control library can be used.

M_PSK

Description: Modulator for M-ary PSK including BPSK, QPSK, 8PSK, 16PSK, 32PSK, 64PSK, 128PSK, 256PSK and 512PSK
Library: Numeric, Communications
Class: SDFM_PSK

Parameters

Name	Description	Default	Unit	Type	Range
ModType	Modulation type: BPSK, QPSK, PSK8, PSK16, PSK32, PSK64, PSK128, PSK256, PSK512	QPSK		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	In	Input bit sequence	int

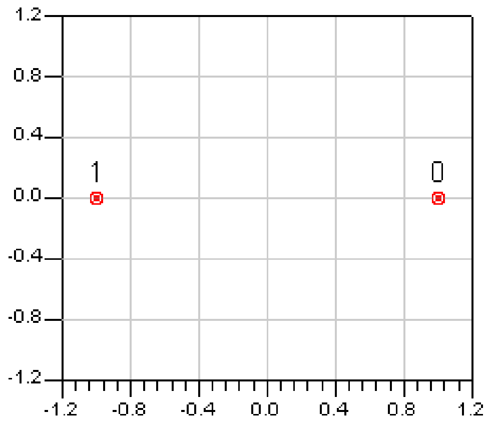
Pin Outputs

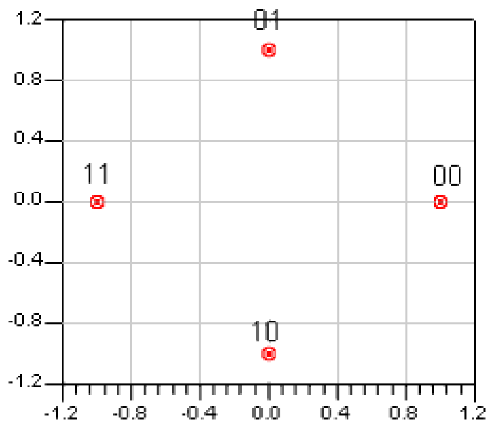
Pin	Name	Description	Signal Type
2	Out	Output complex symbol	complex

Notes/Equations

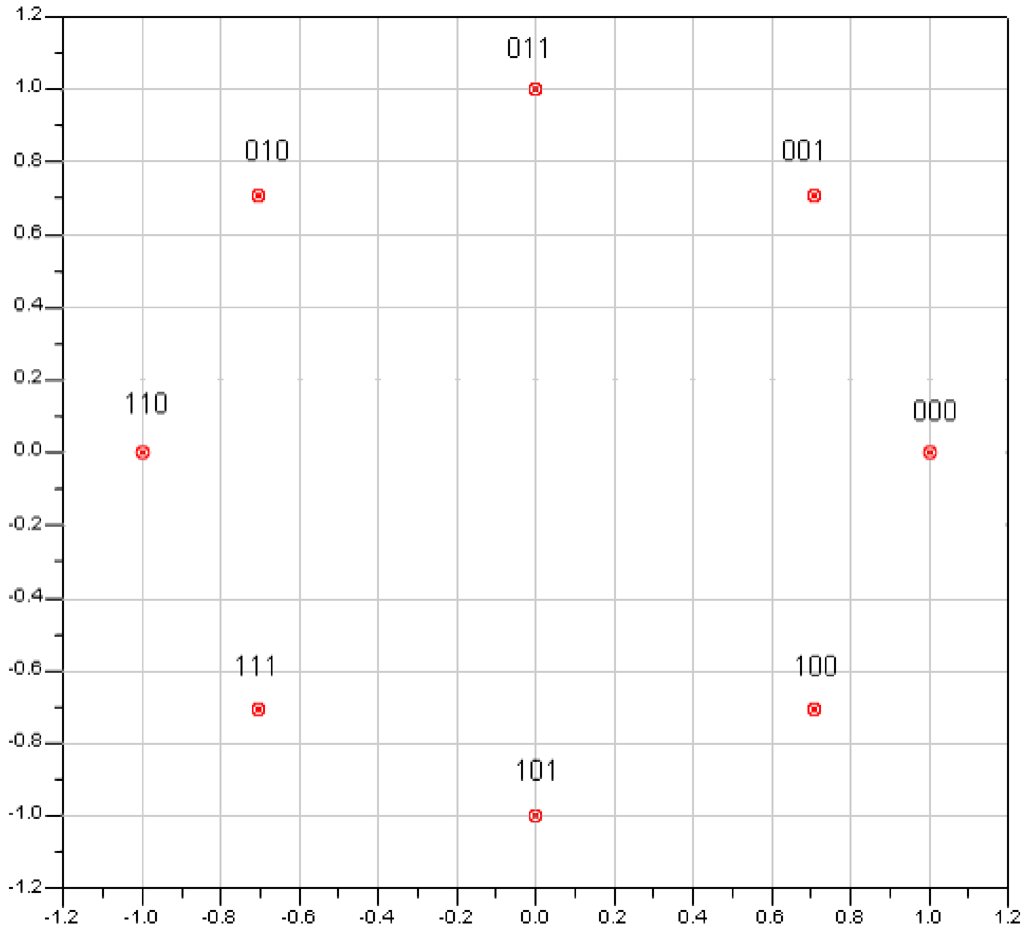
1. M_PSK performs a M-ary phase shift key (PSK) modulation on the input bit stream, producing a Gray coded complex output signal. This component supports all popular M-ary PSK modulations in communication systems, including BPSK (2-BPSK), QPSK (4-PSK), 8-, 16-, 32-, 64-, 128-, 256-, and 512-PSK.
2. This is a multirate component. In general, if an M-ary PSK modulation is selected by using ModeType, the component consumes n = log2(M) bits from the input and produces one modulated complex output. Input bits are Gray encoded and mapped to an output constellation point as shown in BPSK and QPSK Modulation Using Gray Encoding to 32-PSK Modulation Using Gray Coding. For example, if ModType = PSK8, the component consumes log2(8) = 3 bits from the input for Gray coded bits then maps these coded bits to a corresponding constellation point as shown in 8PSK Modulation Using Gray Coding.
3. While there are many ways to encode and map sets of input bits into an M-point PSK constellation, Gray coding is always used for modulations to reduce error probabilities in communication systems. For M_PSK, a generic Labeling Expansion method proposed by E. Agrell [1] is used for Gray-encoding the input bits.
   For specific mapping details, refer to *Mapper* (numeric).

BPSK and QPSK Modulation Using Gray Encoding

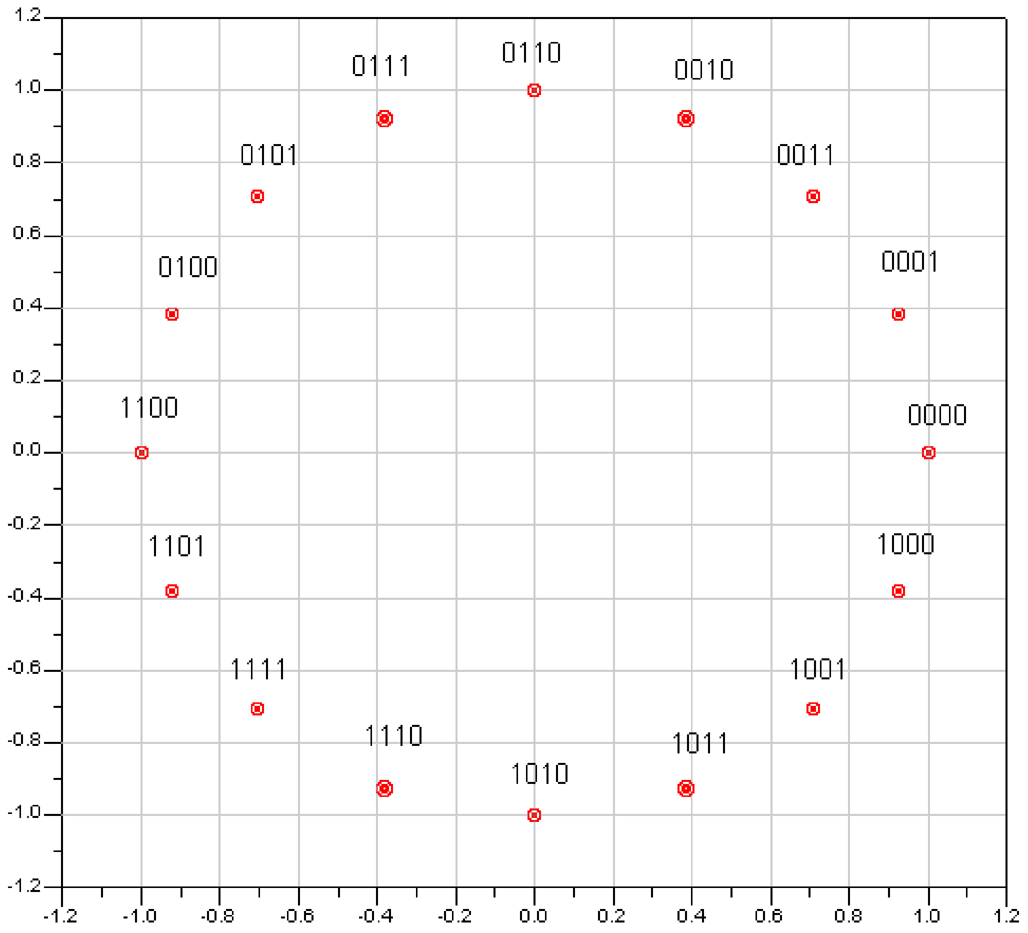




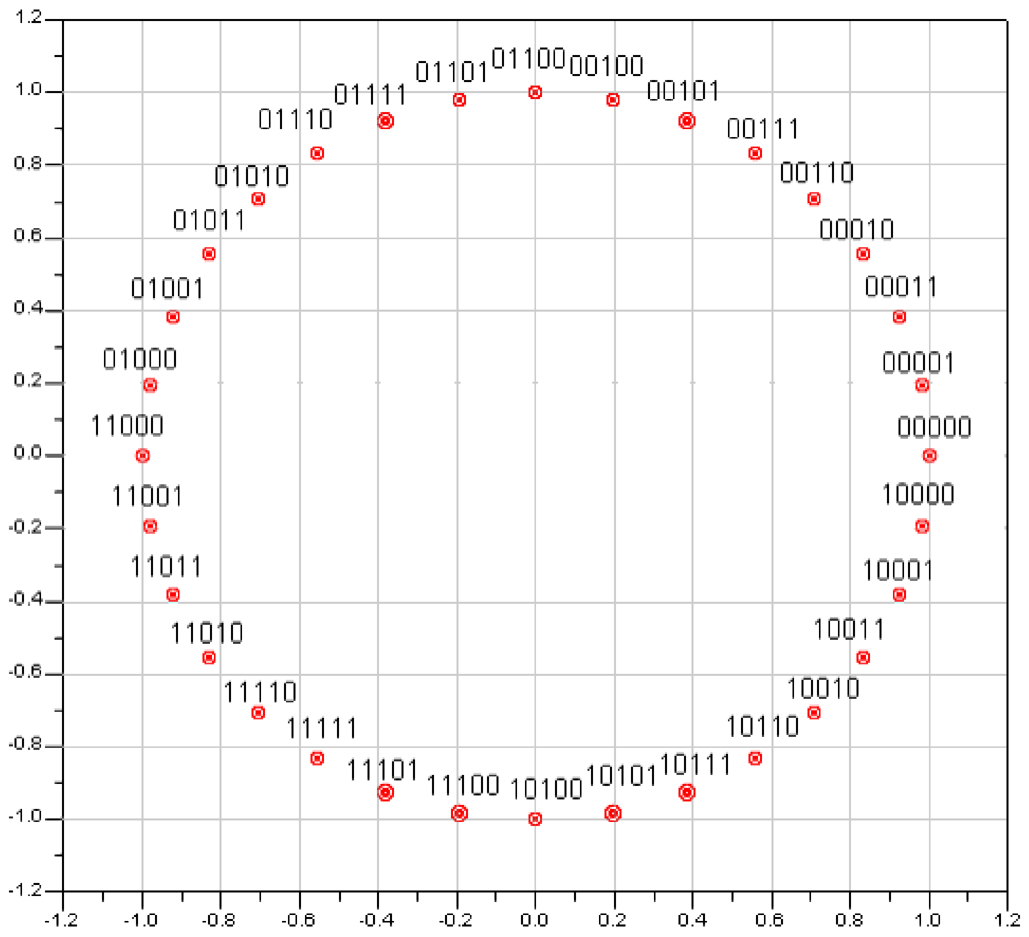
8PSK Modulation Using Gray Coding



16-PSK Modulation Using Gray Coding



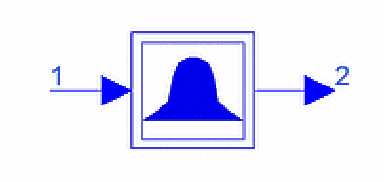
32-PSK Modulation Using Gray Coding



References

4. E. Agrell, J.Lassing, E. G. Strm, and T. Ottosson, "On the optimality of the binary reflected Gray code," *IEEE Transactions on Information Theory, vol. 50*, no. 12, pp. 3170-3182, Dec. 2004.
5. M. Jeruchim, P. Balaban and K. Shanmugan, *Simulation of Communication Systems*, Plenum Press, New York and London, 1992.

NoiseChannel



Description: Channel Modeling with Additive White Gaussian Noise
Library: Numeric, Communications
Class: SDFNoiseChannel

Parameters

Name	Description	Default	Unit	Type	Range
NoiseVariance	maximum settable value for noise variance	1.0		real	[0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real

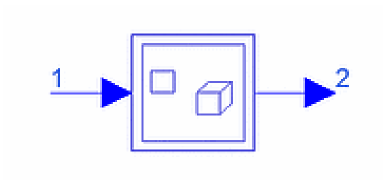
Pin Outputs

Pin	Name	Description	Signal Type
2	out	input signal plus Gaussian noise	real

Notes/Equations

1. NoiseChannel models a channel with additive white Gaussian noise.
   If x(t) is a band-limited input signal to a channel and y(t) is the corresponding output signal then, for the additive white Gaussian noise waveform channel, the real output is
   $y(t) = x(t) + n(t)$
   where n(t) is a sample function of the additive noise process.
2. See also, *AWGN_Channel* (numeric).

NonlinearDistortion



Description: Second and Third Harmonic Distortion
Library: Numeric, Communications
Class: SDFNonlinearDistortion

Parameters

Name	Description	Default	Unit	Type	Range
SecondHarmonic	proportion of second harmonic of input to add to original signal	0.0		real	(-∞, ∞)
ThirdHarmonic	proportion of third harmonic of input to add to original signal	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real

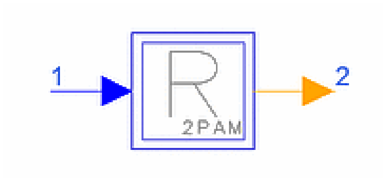
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output signal	real

Notes/Equations

1. With NonlinearDistortion, second- and third-order harmonic distortion is generated by squaring and cubing the input signal and adding the results in controlled proportions to the original signal.
   $output = input + SecondHarmonic \times (input)^{2} + ThirdHarmonic \times (input)^{3}$

PAM2Rec



Description: 2-Level Pulse Amplitude Modulation Input Signal Receiver
Library: Numeric, Communications
Class: SDFPAM2Rec

Pin Inputs

Pin	Name	Description	Signal Type
1	in	received PAM signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	bit that corresponds to the received PAM pulse	int

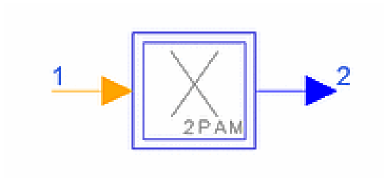
Notes/Equations

1. PAM2Rec receives a 2-level pulse amplitude modulation (PAM) signal and extracts the transmitted bits. It is assumed that the received PAM signal is a nonreturn-to-zero polar format with a symbol interval of 16. PAM2Rec will receive signals generated by the PAM2Xmit component.
2. Once the transmitted bits are extracted, these are descrambled before being sent to the output. The descrambling polynomial matches that of the PAM2Xmit component scrambler.
3. See also: *DeScrambler* (numeric), *DownSample* (numeric), *PAM2Xmit* (numeric), and *Scrambler* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 6.

PAM2Xmit



Description: 2-Level Pulse Amplitude Modulation Transmitter
Library: Numeric, Communications
Class: SDFPAM2Xmit

Parameters

Name	Description	Default	Unit	Type	Range
ExcessBW	excess bandwidth of the square root raised-cosine pulses used to transmit data	1.0		real	[0,1]
FilterLength	length of square root raised-cosine pulses used to transmit data	32		real	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input bits to be transmitted	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	square root raised-cosine pulses that correspond to the input bits	real

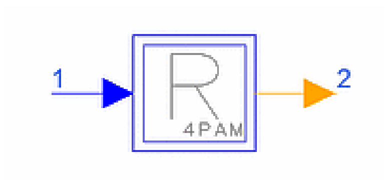
Notes/Equations

1. PAM2Xmit uses 2-level pulse amplitude modulation to convert the input bits into a transmission signal. The PAM signal is a nonreturn-to-zero polar format with square root raised-cosine pulses. The excess bandwidth and length of the square root raised-cosine pulses are set by the ExcessBW and FilterLength parameters. The PAM levels are +2 and −2; the symbol interval is 16; therefore, for each input bit received a 16-sample output pulse is produced.
   Note that the input bits are scrambled before transmitting. The bits must be descrambled after these are received.
2. See also: *DeScrambler* (numeric), *PAM2Rec* (numeric), *Scrambler* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 6.

PAM4Rec



Description: 4-Level Pulse Amplitude Modulation Input Signal Receiver
Library: Numeric, Communications
Class: SDFPAM4Rec

Pin Inputs

Pin	Name	Description	Signal Type
1	in	received PAM signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	bit that corresponds to received PAM pulse	int

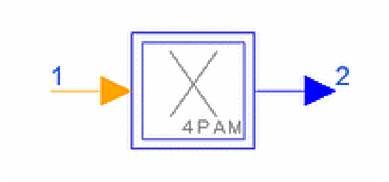
Notes/Equations

1. PAM4Rec receives a 4-level pulse amplitude modulation signal and extracts the transmitted bits. The four levels should be -3, -1, $+1$, and $+1$. It is assumed that the received PAM format has a symbol interval of 16. PAM4Rec will receive signals generated by PAM4Xmit.
   Once the transmitted bits are extracted, these are descrambled before being sent to the output. The descrambling polynomial matches the PAM4Xmit component scrambler.
2. See also: *DeScrambler* (numeric), *DownSample* (numeric), *PAM4Xmit* (numeric).

References

1. For more information about pulse amplitude modulation, see: S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 6.

PAM4Xmit



Description: 4-Level Pulse Amplitude Modulation Transmitter
Library: Numeric, Communications
Class: SDFPAM4Xmit

Parameters

Name	Description	Default	Unit	Type	Range
ExcessBW	excess bandwidth of square root raised-cosine pulses used to transmit data	1.0		real	[0,1]
FilterLength	length of square root raised-cosine pulses used to transmit data	32		real	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input bits to be transmitted	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	square root raised-cosine pulses that correspond to the input bits	real

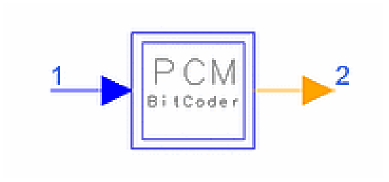
Notes/Equations

1. PAM4Xmit uses 4-level pulse amplitude modulation to convert pairs of input bits into a transmission signal. The input bits are first scrambled before transmitting. The bits must be descrambled after these are received.
   The PAM format used is a nonreturn-to-zero polar format with square root raised-cosine pulses. The excess bandwidth and length of the square root raised-cosine pulses are set by the ExcessBW and FilterLength parameters. The PAM levels are +3, +1, −1, and −3. The symbol interval is 16; therefore, for each two input bits received a 16-sample output pulse is produced.
2. See also: *DeScrambler* (numeric), *PAM4Rec* (numeric), *Scrambler* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 6.

PCM_BitCoder



Description: Pulse-Code Modulation Encoder
Library: Numeric, Communications
Class: SDFPCM_BitCoder

Pin Inputs

Pin	Name	Description	Signal Type
1	in	analog input signal with values from -4000 to 4000	real

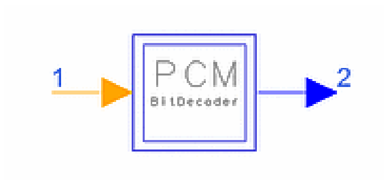
Pin Outputs

Pin	Name	Description	Signal Type
2	out	PCM encoded bits	int

Notes/Equations

1. PCM_BitCoder is a 64-kbits-per-second pulse-code modulation encoder. Each input value is companded and quantized to 8 bits that are then sent to the output.
2. The encoding follows the CCITT Recommendation G.711.
3. PCM_BitCoder works with *PCM_BitDecoder* (numeric), which performs the reverse operation.

PCM_BitDecoder



Description: Pulse-Code Modulation Decoder
Library: Numeric, Communications
Class: SDFPCM_BitDecoder

Pin Inputs

Pin	Name	Description	Signal Type
1	in	PCM encoded bits	int

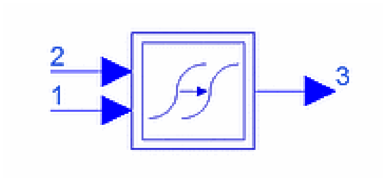
Pin Outputs

Pin	Name	Description	Signal Type
2	out	corresponding analog signal value	real

Notes/Equations

1. PCM_BitDecoder is a 64-kbits-per-second pulse-code modulation decoder. Each set of 8 input bits is mapped to its decoded analog value that is then sent to the output.
2. The decoding follows the CCITT Recommendation G.711.
3. PCM_BitDecoder works with the *PCM_BitCoder* (numeric) component, which performs the reverse operation.

PhaseShift



Description: Phase Shift Distortion
Library: Numeric, Communications
Class: SDFPhaseShift

Parameters

Name	Description	Default	Unit	Type	Range
HilbertFilterLength	Hilbert filter length	64		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real
2	shift	phase shift in radians	real

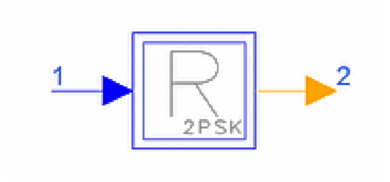
Pin Outputs

Pin	Name	Description	Signal Type
3	out	output signal	real

Notes/Equations

1. PhaseShift adds phase shift distortion found in channels such as telephone channels. The output is the input signal with the phase of the input signal shifted by the value of the shift input.
2. Very low- and very high-frequency signals (near the Nyquist frequency) will be distorted due to the Hilbert filter. This can be partially overcome by setting the HilbertFilterLength parameter for a longer, more accurate filter. The default Hilbert filter is acceptable for most applications.
3. See also, *FreqPhase* (numeric).

PSK2Rec



Description: Binary Phase-Shift Keying Demodulator
Library: Numeric, Communications
Class: SDFPSK2Rec

Parameters

Name	Description	Default	Unit	Type	Range
CarrierFrequency	cosine carier wave frequency	2000		real	(0, ∞)
SamplingRate	carrier wave sampling rate	8000		real	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	received binary phase-shift keyed transmission signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	binary wave of the received data (-N,+N)	real

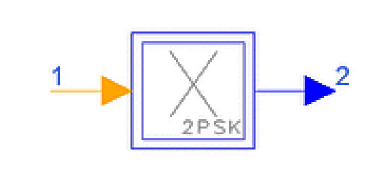
Notes/Equations

1. This component accepts a BPSK modulated wave and outputs the recovered binary data stream.
2. The input sequence is first demodulated by multiplication with a cosine wave sequence. The demodulated sequence is filtered with a square root of raised-cosine filter and scaled with an appropriate factor so that the output level of the downsampler that follows is independent of the filter length (which depends on the sampling and carrier frequencies given by the designer). Conversion to bits is done by downsampling, taking the sign of the downsampled values and mapping 1 and -1 to 1 and 0, respectively. Note that if a BPSK transmitter (PSK2Xmit) and receiver (PSK2Rec) are concatenated, the output bit stream will be delayed by one bit with respect to the input bit stream; this is due to the delay introduced by the filters.
3. See also, *PSK2Xmit* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 7.

PSK2Xmit



Description: Binary Phase-Shift Keying Modulator
Library: Numeric, Communications
Class: SDFPSK2Xmit

Parameters

Name	Description	Default	Unit	Type	Range
CarrierFrequency	cosine carrier wave frequency	2000		real	(0, ∞)
SamplingRate	carrier wave sampling rate	8000		real	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	binary wave (polar from) to be modulated	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	binary phase shift keyed transmission signal	real

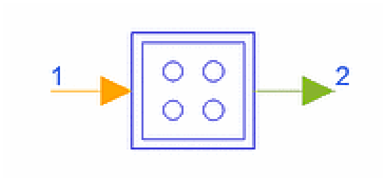
Notes/Equations

1. This component accepts a binary bit stream and outputs a BPSK modulated wave.
2. The input bit stream is first converted to an NRZ waveform that is then filtered by a square root of raised-cosine filter. The interpolation factor of the filter is chosen so that the rate at the output of the filter matches the sampling rate. The filtered sequence is scaled with an appropriate factor so that the amplitude level at the output of the transmitter is independent of the filter length (which depends on the sampling and carrier frequencies given by the designer). The sequence is then multiplied by a cosine wave resulting in a BPSK modulated wave.
3. See also, *PSK2Rec* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 7.

QAM4



Description: 4-State Quadrature Amplitude Modulator
Library: Numeric, Communications
Class: SDFQAM4

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input bit sequence	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	output symbol sequence	complex

Notes/Equations

1. QAM4 performs a 4-point quadrature amplitude modulation on the input bit stream, producing a complex output signal. The component consumes 2 bits from the input for each complex valued output it produces. Mapping of the 2 bits to the 4 points uses Gray encoding, that is:

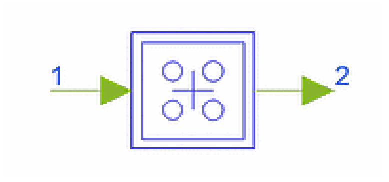
Input Bits	-->	Output Point
0, 1	-->	(−1, 1)
0, 0	-->	(1, 1)
1, 1	-->	(−1, −1)
1, 0	-->	(1, −1)

2. There are many ways to map sets of 2 bits into a 4-point grid; therefore, there are many different variations of 4QAM encoding. This component implements one of them.
3. See also, *QAM4Slicer* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, pages 318-322.

QAM4Slicer



Description: 4-State Quadrature Amplitude Modulator Slicer
Library: Numeric, Communications
Class: SDFQAM4Slicer

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	complex

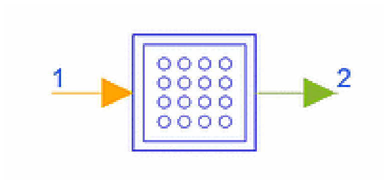
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output 4-QAM signal at exact grid points	complex

Notes/Equations

1. This component outputs the 4QAM grid point that is geometrically closest to the input point.
2. The quadrature amplitude modulation grid is assumed to be:
   (-1, 1) (1, 1)
   (-1, -1) (1, -1)
3. QAM4Slicer works with QAM4; refer to *QAM4* (numeric) for details of 4QAM encoding.

QAM16



Description: 16-State Quadrature Amplitude Modulator
Library: Numeric, Communications
Class: SDFQAM16

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input bit sequence	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	output symbol sequence	complex

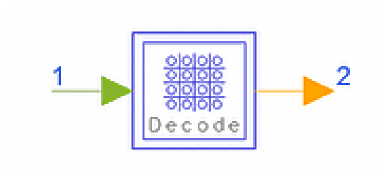
Notes/Equations

1. QAM16 performs a 16-point quadrature amplitude modulation on the input bit stream, producing a complex output signal. The component consumes 4 bits from the input for each complex valued output it produces. The first 2 bits are Gray and differentially encoded and are used to select the quadrant of the output point. The last 2 bits are used to select the point inside the quadrant selected by the first 2 bits. Mapping of the last 2 bits to the 4 points in each quadrant uses Gray encoding. Mapping is also invariant to phase rotations that are multiples of 90 degrees.
2. There are many ways to map sets of 4 bits into a 16-point grid; therefore, there are many different variations of 16QAM encoding. This component implements one of them.
3. See also: *QAM16Decode* (numeric) and *QAM16Slicer* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, pages 318-322.

QAM16Decode



Description: 16-State Quadrature Amplitude Modulator Decoder
Library: Numeric, Communications
Class: SDFQAM16Decode

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	complex

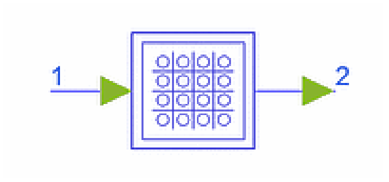
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output bit sequence	int

Notes/Equations

1. QAM16Decode decodes the 16QAM input signal into an output bit stream. It is assumed that the input 16QAM signal was encoded using the QAM16 component. For each value of the input, 4 bits are written at the output.
2. See also: *QAM16* (numeric) and *QAM16Slicer* (numeric).

QAM16Slicer



Description: 16-State Quadrature Amplitude Modulator Slicer
Library: Numeric, Communications
Class: SDFQAM16Slicer

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	complex

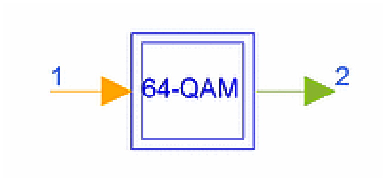
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output 16-QAM signal at exact grid points	complex

Notes/Equations

1. The component outputs the 16QAM grid point that is geometrically closest to the input point.
2. The quadrature amplitude modulation grid is assumed to be:
```
(-3,  3) (-1,  3) (1,  3) (3,  3)
(-3,  1) (-1,  1) (1,  1) (3,  1)
(-3, -1) (-1, -1) (1, -1) (3, -1)
(-3, -3) (-1, -3) (1, -3) (3, -3)
```
3. QAM16Slicer works with QAM16; refer to *QAM16* (numeric) for details of 16QAM encoding.

QAM64



Description: 64-State Quadrature Amplitude Modulator
Library: Numeric, Communications
Class: SDFQAM64

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input bit sequence	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	output symbol sequence	complex

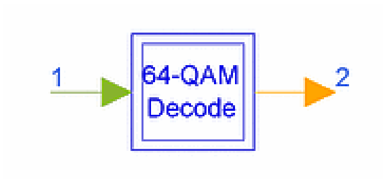
Notes/Equations

1. QAM64 performs a 64-point quadrature amplitude modulation on the input bit stream, producing a complex output signal. The component consumes 6 bits from the input for each complex valued output it produces. The first 2 bits are Gray and differentially encoded and used to select the quadrant of the output point. The last 4 bits are used to select the point inside the quadrant selected by the first 2 bits. Mapping of the last 4 bits to the 16 points in each quadrant uses Gray encoding. Mapping is also invariant to phase rotations that are multiples of 90 degrees.
2. There are many ways to map sets of 6 bits into a 64-point grid; therefore, there are many different variations of 64QAM encoding. This component implements one of them.
3. See also: *QAM64Decode* (numeric) and *QAM64Slicer* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, pages 318-322.

QAM64Decode



Description: 64-State Quadrature Amplitude Modulator Decoder
Library: Numeric, Communications
Class: SDFQAM64Decode

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	complex

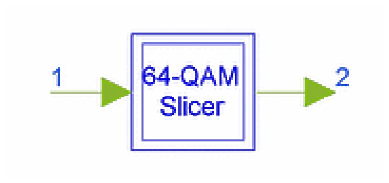
Pin Outputs

Pin	Name	Description	Signal Type
2	out	output bit sequence	int

Notes/Equations

1. QAM64Decode decodes the 64QAM input signal into an output bit stream. It is assumed that the input 64QAM signal was encoded using the QAM64 component. For each value at the input, 6 bits are written at the output.
2. See also: *QAM64* (numeric) and *QAM64Slicer* (numeric).

QAM64Slicer



Description: 64-State Quadrature Amplitude Modulator Slicer
Library: Numeric, Communications
Class: SDFQAM64Slicer

Pin Inputs

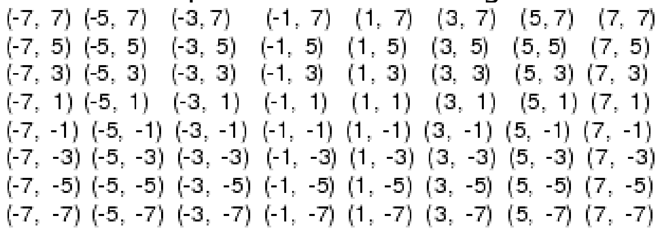
Pin	Name	Description	Signal Type
1	in	input signal	complex

Pin Outputs

Pin	Name	Description	Signal Type
2	out	output 64-QAM signal at exact grid points	complex

Notes/Equations

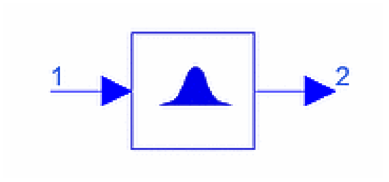
1. This component outputs the 64QAM grid point that is geometrically closest to the input point.
2. The quadrature amplitude modulation grid is assumed to be:



(-7, 7)	(-5, 7)	(-3, 7)	(-1, 7)	(1, 7)	(3, 7)	(5, 7)	(7, 7)
(-7, 5)	(-5, 5)	(-3, 5)	(-1, 5)	(1, 5)	(3, 5)	(5, 5)	(7, 5)
(-7, 3)	(-5, 3)	(-3, 3)	(-1, 3)	(1, 3)	(3, 3)	(5, 3)	(7, 3)
(-7, 1)	(-5, 1)	(-3, 1)	(-1, 1)	(1, 1)	(3, 1)	(5, 1)	(7, 1)
(-7, -1)	(-5, -1)	(-3, -1)	(-1, -1)	(1, -1)	(3, -1)	(5, -1)	(7, -1)
(-7, -3)	(-5, -3)	(-3, -3)	(-1, -3)	(1, -3)	(3, -3)	(5, -3)	(7, -3)
(-7, -5)	(-5, -5)	(-3, -5)	(-1, -5)	(1, -5)	(3, -5)	(5, -5)	(7, -5)
(-7, -7)	(-5, -7)	(-3, -7)	(-1, -7)	(1, -7)	(3, -7)	(5, -7)	(7, -7)

3. QAM64Slicer works with QAM64. Refer to *QAM64* (numeric) for details of 64QAM encoding.

RaisedCosine



Description: Raised-cosine filter
Library: Numeric, Communications
Class: SDFRaisedCosine
Derived From: FIR
C++ Code: See *doc/sp_items/SDFRaisedCosine.html* under your installation directory.

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
Decimation	decimation ratio	1	D		int	[1, ∞)
DecimationPhase	decimation phase	0			int	[0,Decimation-1]
Interpolation	interpolation ratio	16	I		int	[1, ∞)
Length	number of taps	64	L		int	[1, ∞)
SymbolInterval	distance from center to first zero crossing	16	T		int	[1, ∞)
ExcessBW	excess bandwidth	1.0	α		real	[0,1]
SquareRoot	square root raised-cosine pulse: NO, YES	NO			enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. RaisedCosine implements a finite-impulse response filter with a raised-cosine or square root raised-cosine frequency response. Excess bandwidth (also referred to as rolloff factor or alpha) is given by ExcessBW, symbol interval (in number of samples) of the application is given by SymbolInterval, length of filter (number of taps) is given by Length.
   This filter is derived from the FIR filter that uses an internal polyphase structure. This algorithm efficiently implements the rational sample rate changes with decimation and interpolation. For more information on multi-rate concepts, refer to FIR component documentation.
2. For the ordinary raised-cosine response, ideally the impulse response of the filter would be

$$h(n) = \left(\frac{\sin\left(\pi\frac{n}{T}\right)}{\pi\frac{n}{T}}\right)\left(\frac{\cos\left(\alpha\pi\frac{n}{T}\right)}{1-\left(2\alpha\frac{n}{T}\right)^2}\right)$$

However, this ideal pulse is centered at 0, but we can only implement causal filters. Therefore, the impulse response is actually

$g(n) = h(n-M)$

where

$M = \frac{L}{2}$ if L is even

$M = \frac{L-1}{2}$ if L is odd

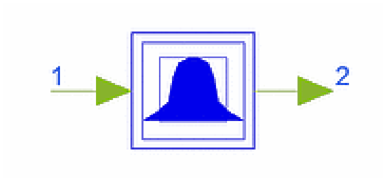
The impulse response is simply truncated outside this range, so the impulse response will generally not be symmetric if L is even because it will have one more sample to the left than to the right of center. Unless this extra sample is 0, the filter will not have linear phase if L is even. For the ordinary raised-cosine response, the distance (in number of samples) from the center to the first zero crossing is given by T.

3. The output sample rate is I times the input. The Interpolation default is set to 16 because this pulse is used in digital communication systems for the line coding of symbols, and upsampling is necessary. In this case, 16 outputs will be produced for each input. Typically, the value of Interpolation is the same as SymbolInterval.
4. When SquareRoot is selected for the raised-cosine filter with and without interpolation, some interesting facts can be observed:
   - The output of two-cascaded square root raised-cosine filter is approximately equal to the output of raised-cosine filter without square root when using the same input signal. In other words: h(n) is a raised-cosine filter and H(z) is a corresponding frequency response for h(n); h1(n) is a square-rooted raised-cosine filter and H1 (z) is a frequency response for h1(n). We should have h(n) = h1(n) × h1(n) or H(z) = H1(z)H1(z).
   - The output of the raised-cosine filter with interpolation rate I should equal the output of an UpSample component with its Factor parameter set to I followed by two cascaded square-root raised-cosine filters when using the same input signal.
   - The amplitude output value of square root raised-cosine filter should show results similar to the amplitude output value of square root raised-cosine filter with interpolation rate I when using the same input signal. However, it can be seen that the difference is more output amplitude data from the square root raised-cosine filter with interpolation rate I compared to square root raised-cosine filter without interpolation rate. This is because every two input-sampled data, I zeros are introduced during upsampling.
5. See also, *RaisedCosineCx* (numeric).

References

1. E. A. Lee and D. G. Messerchmitt, *Digital Communication*, Kluwer Academic Publishers, Boston, 1988.
2. I. Korn, *Digital Communications*, Van Nostrand Reinhold, New York, 1985.

RaisedCosineCx



Description: Complex Raised-Cosine Filter
Library: Numeric, Communications
Class: SDFRaisedCosineCx

Parameters

Name	Description	Default	Unit	Type	Range
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0,Decimation-\1]
Interpolation	interpolation ratio	16		real	[1, ∞)
Length	number of taps	64		int	[1, ∞)
SymbolInterval	distance from center to first zero crossing	16		int	[1, ∞)
ExcessBW	excess bandwidth, between 0 and 1	1.0		real	[0,1]
SquareRoot	square root raised-cosine pulse: NO, YES	NO		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	complex

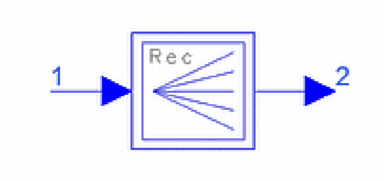
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	complex

Notes/Equations

1. RaisedCosineCx implements a pair of FIR filters with a raised-cosine or square root raised-cosine frequency response. The real part of the complex input goes through one filter to become the real part of the output signal. Similarly, the imaginary part of the input goes through the other filter to become the imaginary part of the output signal.
2. The excess bandwidth (also referred to as rolloff factor or alpha) for both filters is given by ExcessBW; the symbol interval (in number of samples) of the application is given by SymbolInterval; and the length of the filters (the number of taps) is given by Length. By default, this component upsamples by a factor of 16, so 16 outputs will be produced for each input unless the Interpolation parameter is changed.
3. For raised-cosine algorithm details, refer to the *RaisedCosine* (numeric) component.

RecSpread



Description: Spread Spectrum Receiver
Library: Numeric, Communications
Class: SDFRecSpread

Parameters

Name	Description	Default	Unit	Type	Range
PulseDuration	number of times to repeat each transmitted sample	1		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	received direct-sequence spread spectrum signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	received data	real

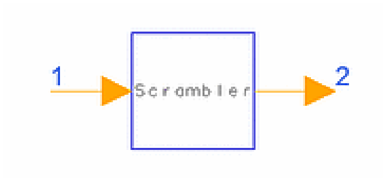
Notes/Equations

1. RecSpread is a direct-sequence spread spectrum receiver. The received signal is first downsampled to remove any signal repetition due to the PulseDuration. The received signal is then modulated with the same 31-bit pseudo-noise spreading code used in the XmitSpread component. The demodulated signal is then correlated and quantized to determine if the received signal is 1 or 0.
2. See also *DeSpreader* (numeric), *Spread* (numeric), and *XmitSpread* (numeric).

References

1. For more information about spread spectrum modulation, see: S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 9.

Scrambler



Description: Input bit sequence scrambler
Library: Numeric, Communications
Class: SDFScrambler
C++ Code: See *doc/sp_items/SDFScrambler.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Polynomial	generator polynomial for the shift register - decimal, octal, or hex integer	0440001		int	
ShiftReg	initial state of the shift register - decimal, octal, or hex integer	1		int	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input bit sequence (zero or nonzero)	int

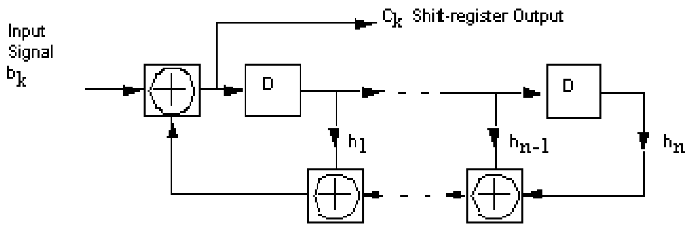
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output bit sequence (zero or one)	int

Notes/Equations

1. This component scrambles the input bit sequence using a feedback shift register, as shown in Feedback Shift Register. The taps of the feedback shift register are given by the Polynomial parameter, which should be a positive integer. The nth bit of this integer indicates whether the nth tap of the delay line is fed back. The low-order bit is called the 0th bit, and must be set. The next low-order bit indicates whether the output of the first delay should be fed back, and so on. The default Polynomial is an octal number defining the V.22bis scrambler.
2. In scramblers based on feedback shift registers, all the bits to be fed back are exclusive-ORed together (their parity is calculated), and the result is exclusive-ORed with the input bit. This result is produced at the output and shifted into the delay line. With proper choice of polynomial, the resulting output appears highly random even if the input is highly non-random (for example, all 0s or all 1s).

Feedback Shift Register



3. If the polynomial is a *primitive polynomial*, then the feedback shift register is a so-called *maximal length feedback shift register*. This means that with a constant input, the output will be sequence with period $2^N - 1$ where *N* is the order of the polynomial (the length of the shift register). This is the longest possible sequence. Moreover, within this period the sequence will appear to be white, in that a calculated autocorrelation will be very nearly an impulse. Therefore, the scrambler with a constant input can be very effectively used to generate a pseudo-random bit sequence.

   The maximal-length feedback shift register with constant input will pass through $2^N - 1$ states before returning to a state it has been in before. This is one short of the 2^N states that a register with *N* bits can take on. This one missing state, in fact, is a *lock-up* state, in that if the input is an appropriate constant, the scrambler will cease to produce random-looking output, and will output a constant. For example, if the input is all zeros, and the initial state of the scrambler is zero, then the outputs will be all zero, hardly random. This is easily avoided by initializing the scrambler to some non-0 state. That is why the default value for the ShiftReg parameter is set to 1.
4. The Polynomial parameter must be carefully chosen. It must represent a *primitive polynomial*, which is one that cannot be factored into two (nontrivial) polynomials with binary coefficients. For details, refer to [1].
5. The table below lists primitive polynomials (expressed as octal numbers so that these are easily translated into taps on shift register); these will result in maximal-length pseudo-random sequences if the input is constant and lockup is avoided.

Order	**Polynomial**	**Order**	**Polynomial**	**Order**	**Polynomial**
		11	04005	21	010000005
2	07	12	010123	22	020000003
3	013	13	020033	23	040000041
4	023	14	042103	24	0100000207
5	045	15	0100003	25	0200000011
6	0103	16	0210013	26	0400000107
7	0211	17	0400011	27	01000000047
8	0435	18	01000201	28	02000000011
9	01021	19	02000047	29	04000000005
10	02011	20	04000011	30	010040000007

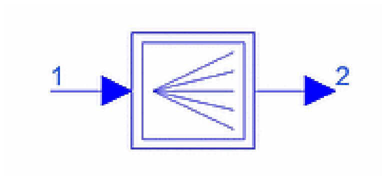
   The leading 0 in the polynomials indicates an octal number. Note also that reversing the order of the bits in any of these numbers will also result in a primitive polynomial. Therefore, the default value for the Polynomial parameter is 0440001 in octal, or "100 100 000 000 000 001" in binary. Reversing these bits we get "100 000 000 000 001 001" in binary, or 0400011 in octal. This latter number is listed in the table as the primitive polynomial of order 17. The order is the index of the highest-order non-0 bit in the polynomial, where the low-order bit has index 0.
   Because the polynomial and the feedback shift register are both implemented using type *int*, the order of the polynomial is limited by the size of the *int* data type. For simplicity and portability, the polynomial is also not allowed to be interpreted as a

negative integer, so the sign bit cannot be used. Therefore, if *int* is a 32-bit word, then the highest order polynomial allowed is 30 (recall that indexing for the order starts at 0, and we cannot use the sign bit). The primitive polynomials in the table are up to order 30 because of 32-bit integer machines.
Both the Polynomial and ShiftReg parameters can be set to a decimal, octal, or hex value. To enter an octal or hex value, prefix it with 0 or 0x, respectively. For example, in order to use the primitive polynomial of order 11, set Polynomial to 04005, 0x805, or 2053.
6. See also, *DeScrambler* (numeric).

References

1. Lee and Messerschmitt, *Digital Communication*, Second Edition, Kluwer Academic Publishers, 1994, pp. 595-603.

Spread



Description: Spread Spectrum Modulator
Library: Numeric, Communications
Class: SDFSpread

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	input signal modulated by a 31-bit pseudo-noise spreading code	real

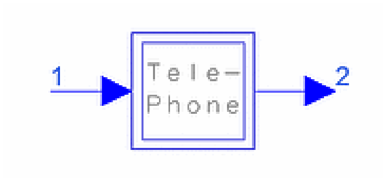
Notes/Equations

1. Spread is a direct-sequence spread spectrum modulator. Each input sample is modulated with a 31-bit pseudo-noise spreading code.
2. See also, *DeSpreader* (numeric) and *XmitSpread* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 9.

TelephoneChannel



Description: Telephone Channel Distortion Model
Library: Numeric, Communications
Class: SDFTelephoneChannel

Parameters

Name	Description	Default	Unit	Type	Range
LinearDistortionTaps	taps values of the FIR filter that models linear distortion	1.0		real array	
Noise	additive white Gaussian noise distortion gain	0		real	(-∞, ∞)
PhaseJitterFrequencyHz	frequency of the phase jitter distortion to add to signal, in Hertz	0.0		real	[0.0, ∞)
PhaseJitterAmplitudeDeg	phase jitter peak amplitude, in degrees	0.0		real	(-∞, ∞)
FrequencyOffsetHz	frequency offset distortion to add to the signal, in Hertz	0.0		real	[0.0, ∞)
SecondHarmonic	proportion of the second harmonic of the input that is added to the original signal	0.0		real	(-∞, ∞)
ThirdHarmonic	proportion of the third harmonic of the input that is added to the original signal	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	out	output signal	real

Notes/Equations

1. TelephoneChannel models the many types of distortion present in a telephone channel (such as amplitude distortion and phase distortion). The sampling rate of the channel is 8000 samples per second.
2. To model linear distortion, such as intersymbol interference, the input signal is passed through an FIR filter with the taps set by LinearDistortionTaps. Phase jitter and frequency offset distortions are then added to the signal.
   Phase jitter is a consequence of the sensitivity of oscillators used for carrier generation in single-sideband systems to fluctuations in power supply voltages. Whereas frequency offset is peculiar to telephone channels and channels with

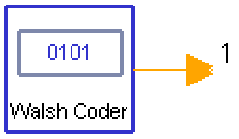
Doppler shift.

3. Nonlinear distortion is modeled by adding the second and third harmonics to the signal. Nonlinear distortion is due to imperfections in amplifiers and to tracking errors between A/D and D/A converters.
4. Gaussian noise with zero mean and a variance set by Noise is added. Primarily, there are four noise sources: quantization noise, thermal noise, impulse noise, and crosstalk.
5. See also: *AWGN_Channel* (numeric), *NoiseChannel* (numeric), and *NonlinearDistortion* (numeric).

References

1. E. A. Lee and D. G. Messerschmitt, *Digital Communication*, Second Edition, Kluwer Academic Publishers, 1994, pp. 595-603.

WalshCoder



Description: Walsh code generator
Library: Numeric, Communications
Class: SDFWalshCoder

Parameters

Name	Description	Default	Unit	Type	Range
Type	Walsh code type: Walsh, Hadamard, OVSF_3GPP	Walsh		enum	
Length	Code length	8		int	[1,8192] †
Index	Code index	0		int	[0,Length-1]

† The length used must be integer power of 2.

Pin Outputs

Pin	Name	Description	Signal Type
1	Out	Output	int

Notes/Equations

1. This component is used to generate variable-length Walsh codes. Each firing, 1 token is produced.
2. If Type = Walsh, the walsh codes are determined by:

$$h_{NK} = (-1)^{\sum_{i=0} r_i(n)k_i}$$

where
N is the index of the walsh code, [0, Length-1]
$N = n_{J-1}\, n_{J-2} \ldots n_1\, n_0$
K is the index of the chip in a walsh code, [0, Length-1]
$K = k_{J-1}\, k_{J-2} \ldots k_1\, k_0$
J = log2Length
$r_0(n) = n_{J-1}$
$r_1(n) = n_{J-1} + n_{J-2}$
$r_2(n) = n_{J-2} + n_{J-3}$

.
.
.
$r_{J-1}(n) = n_1 + n_0$

If Type = Hadamard, the walsh codes are determined by:

$$[H_2] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$[H_4] = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix}$$

.
.
.

$$[H_{2^m}] = \begin{bmatrix} H_{2^m} & H_{2^m} \\ H_{2^m} & -H_{2^m} \end{bmatrix}$$

If Type = OVSF_3GPP, the walsh codes are determined by:

$C_0(0) = 1$

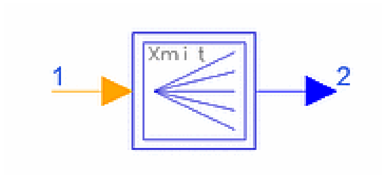
$$\begin{bmatrix} C_1(0) \\ C_1(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} C_{n+1}(0) \\ C_{n+1}(1) \\ C_{n+1}(2) \\ C_{n+1}(3) \\ C_{n+1}(2^{n+1}-2) \\ C_{n+1}(2^{n+1}-1) \end{bmatrix} = \begin{bmatrix} C_n(0) & C_n(0) \\ C_n(0) & -C_n(0) \\ C_n(1) & C_n(1) \\ C_n(1) & -C_n(1) \\ C_n(2^n-1) & C_n(2^n-1) \\ C_n(2^n-1) & \overline{C_n(2^n-1)} \end{bmatrix}$$

References

1. 3GPP Technical Specification TS 25.213 V3.0.0 "Spreading and modulation (FDD)," October 1999.

XmitSpread



Description: Spread Spectrum Transmitter
Library: Numeric, Communications
Class: SDFXmitSpread

Parameters

Name	Description	Default	Unit	Type	Range
PulseDuration	number of times to repeat each transmitted bit	1		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in	input signal to transmit	int

Pin Outputs

Pin	Name	Description	Signal Type
2	out	transmitted signal	int

Notes/Equations

1. XmitSpread is a direct-sequence spread spectrum transmitter. Each input sample to be transmitted is modulated with a 31-bit pseudo-noise spreading code.
2. The PulseDuration parameter determines how many times each transmitted sample is repeated. Every input sample will result in 31 × PulseDuration transmitted samples.
3. See also: *DeSpreader* (numeric), *RecSpread* (numeric), and *Spread* (numeric).

References

1. S. Hakin, *Digital Communications*, John Wiley & Sons, 1988, chapter 9.

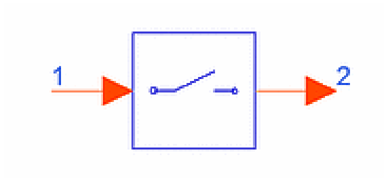
Numeric Control Components

- *ActivatePath* (numeric)
- *ActivatePath2* (numeric)
- *AsyncCommutator* (numeric)
- *AsyncDistributor* (numeric)
- *Bus* (numeric)
- *BusMerge2* (numeric)
- *BusMerge3* (numeric)
- *BusMerge4* (numeric)
- *BusMerge5* (numeric)
- *BusMerge6* (numeric)
- *BusMerge7* (numeric)
- *BusMerge8* (numeric)
- *BusMerge9* (numeric)
- *BusSplit2* (numeric)
- *BusSplit3* (numeric)
- *BusSplit4* (numeric)
- *BusSplit5* (numeric)
- *BusSplit6* (numeric)
- *BusSplit7* (numeric)
- *BusSplit8* (numeric)
- *BusSplit9* (numeric)
- *Chop* (numeric)
- *ChopVarOffset* (numeric)
- *Commutator* (numeric)
- *Commutator2* (numeric)
- *Commutator3* (numeric)
- *Commutator4* (numeric)
- *Delay* (numeric)
- *DeMux* (numeric)
- *DeMux2* (numeric)
- *Distributor* (numeric)
- *Distributor2* (numeric)
- *Distributor3* (numeric)
- *Distributor4* (numeric)
- *DownSample* (numeric)
- *DSampleWOffset* (numeric)
- *EnableUDSample* (numeric)
- *Fork* (numeric)
- *Fork2* (numeric)
- *Fork3* (numeric)
- *Fork4* (numeric)
- *Fork5* (numeric)
- *Fork6* (numeric)
- *Fork7* (numeric)
- *Fork8* (numeric)
- *Fork9* (numeric)
- *IfElse* (numeric)
- *InitDelay* (numeric)
- *Mux* (numeric)

- *Mux2* (numeric)
- *Repeat* (numeric)
- *Reverse* (numeric)
- *Trainer* (numeric)
- *Transpose* (numeric)
- *UpSample* (numeric)
- *VarDelay* (numeric)

The Numeric Control components library contains components that control signal flow in a data flow graph. These include signal bus merge, signal bus split, signal fork, signal distributor, signal commutator, and more. All of these components accept as inputs any signal class and output signals of the same class after the signal control operation is performed.

ActivatePath



Description: Activate or remove succeeding blocks
Library: Numeric, Control
Class: SDFActivatePath

Parameters

Name	Description	Default	Unit	Type	Range
Activate	"YES" to activate succeeding blocks: NO, YES	YES		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

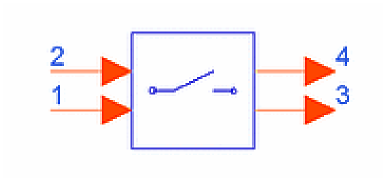
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. ActivatePath is used to activate or remove the succeeding blocks in a schematic design.
2. ActivatePath operates at the graph level. When the Activate parameter is set to NO, the succeeding block will be completely removed from the graph before the simulation starts.
3. The Activate parameter cannot be swept.
4. ActivatePath does not match impedances for timed signals.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

ActivatePath2



Description: Activate or remove succeeding blocks
Library: Numeric, Control
Class: SDFActivatePath2

Parameters

Name	Description	Default	Unit	Type	Range
Activate	"YES" to activate succeeding blocks: NO, YES	YES		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		multiple anytype
2	input2		multiple anytype

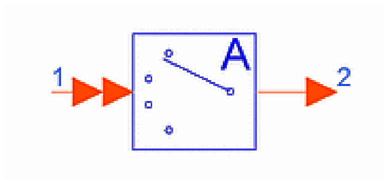
Pin Outputs

Pin	Name	Description	Signal Type
3	output1		multiple anytype
4	output2		multiple anytype

Notes/Equations

1. ActivatePath2 is used to activate or remove the succeeding blocks in a schematic design.
2. ActivatePath2 operates at the graph level. When the Activate parameter is set to NO, the succeeding block will be completely removed from the graph before the simulation starts.
3. When activated (Activate = YES), output1 is connected to input1, output2 is connected to input2.
4. The Activate parameter cannot be swept.
5. ActivatePath2 does not match impedances for timed signals.
6. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

AsyncCommutator



Description: Asynchronous Data Commutator
Library: Numeric, Control
Class: SDFAsyncCommutator
C++ Code: See *doc/sp_items/SDFAsyncCommutator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSizes	block sizes read from each input	1		int array	[1, ∞)†

† for each array element; number of elements in BlockSizes array must equal input bus width

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

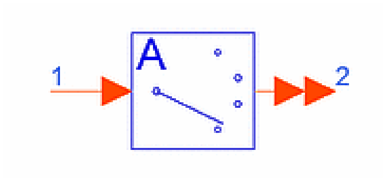
1. AsyncCommutator takes N input signal streams, where N is the input bus width, and asynchronously combines them into one output signal stream. It consumes B_i input samples from input#i (i = 1, ... , N), where B_i are the elements of the BlockSizes parameter. It produces $B_1 + B_2 + ... + B_N$ samples on the output. The first B_1 samples at the output come from the first input, the next B_2 samples come from the second input, and so on.
2. Example. Let's assume that three signals are connected to the input of AsyncCommutator:

input#1	a ramp with initial value 0.0 and step 1.0 (0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, ...)
input#2	a ramp with initial value -0.5 and step -0.5 (-0.5, -1.0, -1.5, -2.0, -2.5, -3.0, -3.5, -4.0, ...)
input#3	a constant signal with value 3.1 (3.1, 3.1, 3.1, 3.1, 3.1, 3.1, ...)

Let's also assume that the BlockSizes parameter is set to "2 3 2".
Then the output signal will be: 0.0, 1.0, -0.5, -1.0, -1.5, 3.1, 3.1, 2.0, 3.0, -2.0, -2.5, -3.5, 3.1, 3.1, 4.0, 5.0, -4.0, -4.5, -5.0, 3.1, 3.1, ...

3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
4. See also: *Commutator2* (numeric), *Commutator3* (numeric), *Commutator4* (numeric), *AsyncDistributor* (numeric), *Distributor2* (numeric), *Distributor3* (numeric), *Distributor4* (numeric).

AsyncDistributor



Description: Asynchronous Data Distributor
Library: Numeric, Control
Class: SDFAsyncDistributor
"C++ Code:* See *doc/sp_items/SDFAsyncDistributor.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSizes	block sizes written to each output	1		int array	[1, ∞)†

† for each array element; number of elements in BlockSizes array must equal output bus width.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

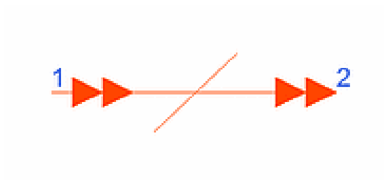
Notes/Equations

1. AsyncDistributor takes one input signal stream and asynchronously splits it into N output signal streams, where N is the output bus width. It consumes $B_1 + B_2 + ... + B_N$ samples from the input, where B_i (i = 1, ... , N) are the elements of the BlockSizes parameter. It produces B_i output samples on output#i (i = 1, ... , N). The samples on the first output are the first B_1 samples of the input, the samples on the second output are the next B_2 samples of the input, and so on.
2. Example. Let's assume that the input to the AsyncDistributor is a ramp signal with initial value 0 and step 1 (0, 1, 2, 3, 4, 5, ...). Let's also assume that the BlockSizes parameter is set to "1 4 2". Then the three output signals are:

output#1	0, 7, 14, 21, ...
output#2	1, 2, 3, 4, 8, 9, 10, 11, 15, 16, 17, 18, 22, 23, 24, 25, ...
output#3	5, 6, 12, 13, 19, 20, 26, 27, ...

3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
4. See also: *Distributor2* (numeric), *Distributor3* (numeric), *Distributor4* (numeric), *AsyncCommutator* (numeric), *Commutator2* (numeric), *Commutator3* (numeric), *Commutator4* (numeric).

Bus



Description: Bus Expander to specified bus width
Library: Numeric, Control
Class: HOFBus
Derived From: Nop

Parameters

Name	Description	Default	Type	Range
BusWidth	BusWidth	1	int	[2, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

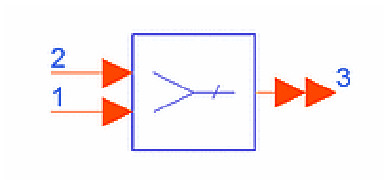
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. The Bus component is used between two multiports and expands the input bus to the output bus width specified.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

BusMerge2



Description: Merge 2 inputs to form a bus of width 2.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype

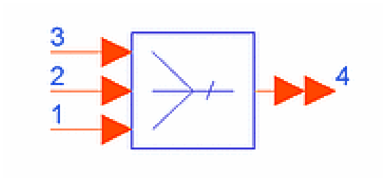
Pin Outputs

Pin	Name	Description	Signal Type
3	output		multiple anytype

Notes/Equations

1. The BusMerge2 component merges the top and bottom input busses into a single bus. If the input bus widths are M1 and M2 and the output bus width is N, then N = M1 + M2 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus. Both input signals must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how this component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open , BusMerge2_example*.

BusMerge3



Description: Merge 3 inputs to form a bus of width 3.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype

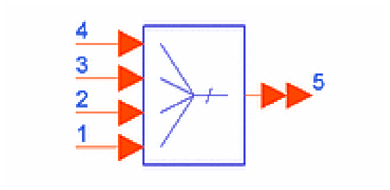
Pin Outputs

Pin	Name	Description	Signal Type
4	output		multiple anytype

Notes/Equations

1. BusMerge3 merges all 3 input busses into a single bus. If the input bus widths are M1, M2, and M3 and the output bus width is N, then N = M1 + M2 + M3 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on. All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how this component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open , BusMerge3_example*.

BusMerge4



Description: Merge 4 inputs to form a bus of width 4.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype

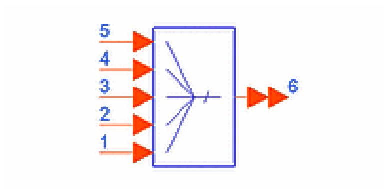
Pin Outputs

Pin	Name	Description	Signal Type
5	output		multiple anytype

Notes/Equations

1. BusMerge4 merges all 4 input busses into a single bus. If the input bus widths are M1, M2, M3, and M4 and the output bus width is N, then N = M1 + M2 + M3 + M4 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on. All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how a BusMerge component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusMerge2_example, BusMerge3_example*, or *BusMerge5_example*.

BusMerge5



Description: Merge 5 inputs to form a bus of width 5.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype
5	input#5		anytype

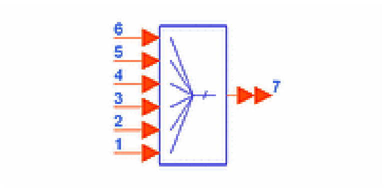
Pin Outputs

Pin	Name	Description	Signal Type
6	output		multiple anytype

Notes/Equations

1. BusMerge5 merges all 5 input busses into a single bus. If the input bus widths are M1, M2, M3, M4, and M5 and the output bus width is N, then N = M1 + M2 + M3 + M4 + M5 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on. All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how this component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusMerge5_example*.

BusMerge6



Description: Merge 6 inputs to form a bus of width 6.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype
5	input#5		anytype
6	input#6		anytype

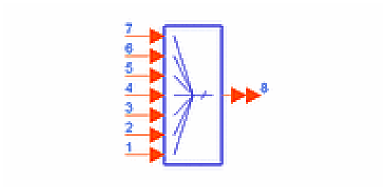
Pin Outputs

Pin	Name	Description	Signal Type
7	output		multiple anytype

Notes/Equations

1. BusMerge6 merges all 6 input busses into a single bus. If the input bus widths are M1, M2, ... , M6 and the output bus width is N, then N = M1 + M2 ... + M6 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on. All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how a BusMerge component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open*, *BusMerge2_example, BusMerge3_example*, or *BusMerge5_example*.

BusMerge7



Description: Merge 7 inputs to form a bus of width 7.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype
5	input#5		anytype
6	input#6		anytype
7	input#7		anytype

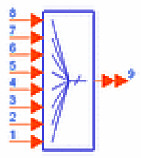
Pin Outputs

Pin	Name	Description	Signal Type
8	output		multiple anytype

Notes/Equations

1. BusMerge7 merges all 7 input busses into a single bus. If the input bus widths are M1, M2, ... , M7 and the output bus width is N, then N = M1 + M2 ... + M7 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on.
   All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how a BusMerge component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusMerge2_example, BusMerge3_example*, or *BusMerge5_example*.

BusMerge8



Description: Merge 8 inputs to form a bus of width 8.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype
5	input#5		anytype
6	input#6		anytype
7	input#7		anytype
8	input#8		anytype

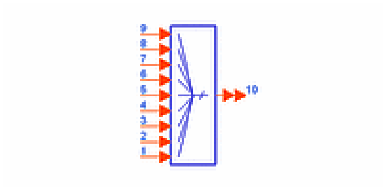
Pin Outputs

Pin	Name	Description	Signal Type
9	output		multiple anytype

Notes/Equations

1. BusMerge8 merges all 8 input busses into a single bus. If the input bus widths are M1, M2, ... , M8 and the output bus width is N, then N = M1 + M2 + ... + M8 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on.
   All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how a BusMerge component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open, BusMerge2_example, BusMerge3_example*, or *BusMerge5_example*.

BusMerge9



Description: Merge 9 inputs to form a bus of width 9.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype
5	input#5		anytype
6	input#6		anytype
7	input#7		anytype
8	input#8		anytype
9	input#9		anytype

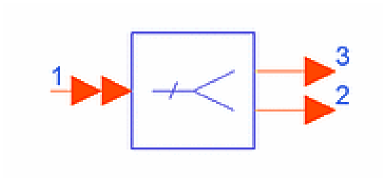
Pin Outputs

Pin	Name	Description	Signal Type
10	output		multiple anytype

Notes/Equations

1. BusMerge9 merges all 9 input busses into a single bus. If the input bus widths are M1, M2, ... , M9 and the output bus width is N, then N = M1 + M2 +, ... , + M9 is required. The first M1 outputs come from the first input bus, while the next M2 outputs come from the second input bus, and so on.
   All signal inputs must be of the same type.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. An example that shows how a BusMerge component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open*, *BusMerge2_example, BusMerge3_example*, or *BusMerge5_example*.

BusSplit2



Description: Split input bus to 2 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

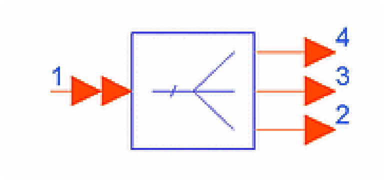
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype

Notes/Equations

1. BusSplit2 splits an input bus into two busses. If the input bus width is N, and the output bus widths are M1 and M2, then N = M1 + M2 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus.
2. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
3. BusSplit2 splits the constituent signals of the input bus. It produces 2 single signal outputs, both of the same type as the input.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how this component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusSplit2_example*.

BusSplit3



Description: Split input bus to 3 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

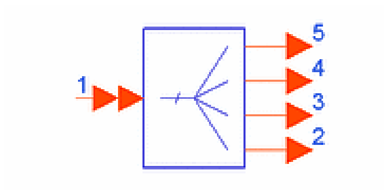
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype

Notes/Equations

1. BusSplit3 component splits an input bus into 3 busses. If the input bus width is N, and the output bus widths are M1, M2, and M3 then N = M1 + M2 + M3 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus and so on.
2. BusSplit3 splits the constituent signals of the input bus. It produces 3 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how this component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open, BusSplit3_example*.

BusSplit4



Description: Split input bus to 4 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

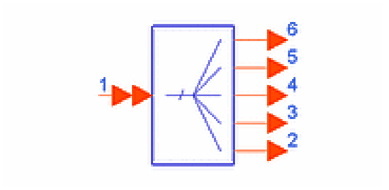
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype

Notes/Equations

1. BusSplit4 splits an input bus into 4 busses. If the input bus width is N, and the output bus widths are M1, M2, M3 and M4, then N = M1 + M2 + M3 + M4 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit4 splits the constituent signals of the input bus. It produces 4 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how a BusSplit component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open*, *BusSplit2_example, BusSplit3_example*, *BusSplit5_example*, or *BusSplit7_example*.

BusSplit5



Description: Split input bus to 5 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

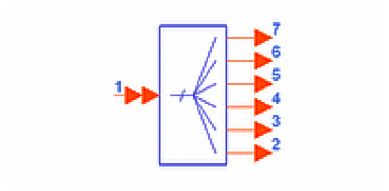
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype

Notes/Equations

1. BusSplit5 splits an input bus into 5 busses. If the input bus width is N, and the output bus widths are M1, M2, M3, M4, and M5, then N = M1 + M2 + M3 + M4 + M5 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit5 splits the constituent signals of the input bus. It produces 5 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how this component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusSplit5_example*.

BusSplit6



Description: Split input bus to 6 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

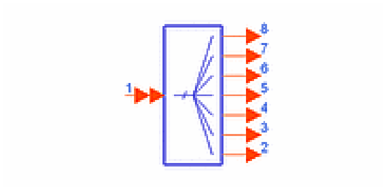
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype

Notes/Equations

1. BusSplit6 splits an input bus into 6 busses. If the input bus width is N, and the output bus widths are M1, M2, M3, M4, M5, and M6, then N = M1 + M2 + M3 + M4 + M5 + M6 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit6 splits the constituent signals of the input bus. It produces 6 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how a BusSplit component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open*, *BusSplit2_example, BusSplit3_example*, *BusSplit5_example*, or *BusSplit7_example*.

BusSplit7



Description: Split input bus to 7 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

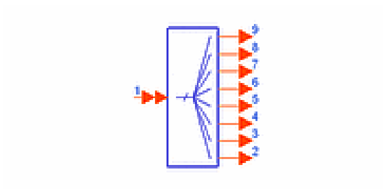
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype

Notes/Equations

1. BusSplit7 splits an input bus into 7 busses. If the input bus width is N, and the output bus widths are M1, M2, M3, M4, M5, M6, and M7 then N = M1 + M2 + M3 + M4 + M5 + M6 + M7 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit7 splits the constituent signals of the input bus. It produces 7 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how this component is used can be accessed from the ADS Main window: *File* > *Open* > *Example* > *PtolemyDocExamples* > *Numeric_Control_wrk*; from the Schematic window, choose *File* > *Open, BusSplit7_example*.

BusSplit8



Description: Split input bus to 8 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

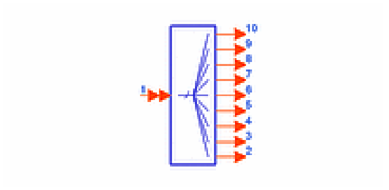
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype
9	output#8		anytype

Notes/Equations

1. BusSplit8 splits an input bus into 8 busses. If the input bus width is N, and the output bus widths are M1, M2, ... , M8 then N = M1 + M2 + ... + M8 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit8 splits the constituent signals of the input bus. It produces 8 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how a BusSplit component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusSplit2_example, BusSplit3_example*, *BusSplit5_example*, or *BusSplit7_example*.

BusSplit9



Description: Split input bus to 9 output buses.
Library: Numeric, Control
Class: HOFNop

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

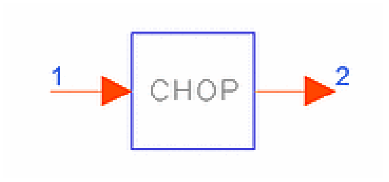
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype
9	output#8		anytype
10	output#9		anytype

Notes/Equations

1. BusSplit9 splits an input bus into 9 busses. If the input bus width is N, and the output bus widths are M1, M2, ... , M9 then N = M1 + M2 + ... + M9 is required. The first M1 inputs go to the first output bus, while the next M2 inputs go to the second output bus, and so on.
2. BusSplit9 splits the constituent signals of the input bus. It produces 9 single signal outputs, all of the same type as the input.
3. The input to the component is a bus, the bus on the lowest output pin always has a bus width of 1, and is not settable by the user.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
5. An example that shows how a BusSplit component is used can be accessed from the ADS Main window: *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, BusSplit2_example, BusSplit3_example*, *BusSplit5_example*, or *BusSplit7_example*.

Chop



Description: Chop input data into blocks
Library: Numeric, Control
Class: SDFChop
C++ Code: See *doc/sp_items/SDFChop.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
nRead	number of data items read	128		int	[1, ∞)
nWrite	number of data items written	64		int	[1, ∞)
Offset	start of output block relative to start of input block	0		int	(-∞, ∞)
UsePastInputs	use previously read inputs: NO, YES	YES		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. The Chop component reads a block of *nRead* samples from its input and produces a block of *nWrite* samples at its output. Depending on the parameter settings, the output block can have all or part of the samples in the input block, zeros could be appended or prepended, and even samples from the previously read input blocks can be reused.
2. The *Offset* parameter defines where in the output block of samples the first (oldest) input sample is output.
   - If *Offset* is ≤ 0, the first |*Offset*| input samples are discarded and the (|*Offset*| + 1)-th input sample is output as the first output sample (the *UsePastInputs* parameter is ignored)
   - If *Offset* > 0, the first input sample is output as the (*Offset* + 1)-th output sample. The first *Offset* output samples are:
     - 0, if *UsePastInputs* is set to NO
     - the last *Offset* samples from the previous blocks read, if *UsePastInputs* is set to YES
3. The following tables summarize the behavior of this component.
   If *nRead* ≥ *nWrite*

Case	***Offset***	***UsePastInputs***	***nWrite* ≤ *nRead* − |*Offset*|**	***Output***
1	(−∞, −*nRead*]	NO or YES	will always be FALSE	all zeros
2	(−*nRead*, 0]	NO or YES	TRUE	discard the first |*Offset*| input samples, output the next *nWrite* input samples
3	(−*nRead*, 0]	NO or YES	FALSE	discard the first |*Offset*| input samples, output the next (*nRead* − |*Offset*|) input samples followed by (*nWrite* − (*nRead* − |*Offset*|)) zeros
4	(0, *nWrite*)	NO	TRUE or FALSE	output *Offset* zeros followed by the first (*nWrite* − *Offset*) input samples
5	(0, *nWrite*)	YES	TRUE or FALSE	output the last *Offset* samples of the previously read input block followed by the first (*nWrite* − *Offset*) input samples
6	[*nWrite*, ∞)	NO	TRUE or FALSE	all zeros
7	[*nWrite*, ∞)	YES	TRUE or FALSE	output from the (*nRead* − *Offset* + 1)-th to (*nRead* − *Offset* + *nWrite*)-th samples of the previously read input block (for the first block the previous block is assumed to be all zeros)

If *nRead* < *nWrite*

Case	***Offset***	***UsePastInputs***	***nRead* ≤ *nWrite* − |*Offset*|**	***Output***
8	(−∞, −*nRead*]	NO or YES	TRUE or FALSE	all zeros
9	(−*nRead*, 0]	NO or YES	TRUE or FALSE	discard the first |*Offset*| input samples, output the next (*nRead* − |*Offset*|) input samples followed by (*nWrite* − (*nRead* − |*Offset*|)) zeros
10	(0, *nWrite*)	NO	TRUE	output *Offset* zeros followed by the *nRead* input samples followed by (*nWrite* − *nRead* − *Offset*) zeros
11	(0, *nWrite*)	NO	FALSE	output *Offset* zeros followed by the first (*nWrite* − *Offset*) input samples
12	(0, *nWrite*)	YES	TRUE	output the last *Offset* samples of the previously read input block followed by the *nRead* input samples followed by (*nWrite* − *nRead* − *Offset*) zeros
13	(0, *nWrite*)	YES	FALSE	output the last *Offset* samples of the previously read input block(s) followed by the first (*nWrite* − *Offset*) input samples
14	[*nWrite*, ∞)	NO	will always be FALSE	all zeros
15	[*nWrite*, ∞)	YES	will always be FALSE	output the last *nWrite* samples of the previously read input block(s) (for the first block the previous blocks are assumed to be all zeros)

4. Here are some examples. In all of these examples the input is assumed to be a ramp signal with initial value of 1 and step 1 (1, 2, 3, 4, 5, 6, ...).

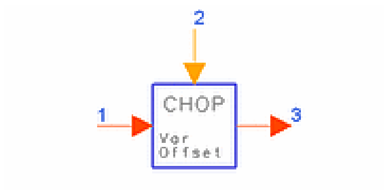
Case	nRead	nWrite	Offset	UsePastInputs	Output
1	10	5	-10 (or smaller)	NO or YES	0, 0, 0, 0, 0, 0, ...
2	10	3	-5	NO or YES	6, 7, 8, 16, 17, 18, 26, 27, 28, ...
3	10	5	-7	NO or YES	8, 9, 10, 0, 0, 18, 19, 20, 0, 0, 28, 29, 30, 0, 0, ...
4	10	5	2	NO	0, 0, 1, 2, 3, 0, 0, 11, 12, 13, 0, 0, 21, 22, 23, ...
5	10	5	2	YES	0, 0, 1, 2, 3, 9, 10, 11, 12, 13, 19, 20, 21, 22, 23, ...
6	10	5	5 (or bigger)	NO	0, 0, 0, 0, 0, 0, ...
7	10	5	5	YES	0, 0, 0, 0, 0, 6, 7, 8, 9, 10, 16, 16, 18, 19, 20, ...
7	10	5	7	YES	0, 0, 0, 0, 0, 4, 5, 6, 7, 8, 14, 15, 16, 17, 18, ...
8	5	10	-5 (or smaller)	NO or YES	0, 0, 0, 0, 0, 0, ...
9	5	10	-3	NO or YES	4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 9, 10, 0, 0, 0, 0, 0, 0, 0, 0, 14, 15, 0, 0, 0, 0, 0, 0, 0, 0, ...
10	5	10	3	N0	0, 0, 0, 1, 2, 3, 4, 5, 0, 0, 0, 0, 0, 6, 7, 8, 9, 10, 0, 0, 0, 0, 0, 11, 12, 13, 14, 15, 0, 0, ...
11	5	10	7	N0	0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 6, 7, 8, 0, 0, 0, 0, 0, 0, 0, 11, 12, 13, ...
12	5	10	3	YES	0, 0, 0, 1, 2, 3, 4, 5, 0, 0, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0, 8, 9, 10, 11, 12, 13, 14, 15, 0, 0, ...
13	5	10	7	YES	0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, ...
14	5	10	10 (or bigger)	NO	0, 0, 0, 0, 0, 0, ...
15	3	5	5	YES	0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 2, 3, 4, 5, 6, 5, 6, 7, 8, 9, ...

5. Common uses of the Chop component include:
   - Discard samples from the beginning of a block of data: *Offset* should be set to $-N$, where N is the number of samples that need to be discarded and *nWrite* should be set to *nRead* $- N$.
   - Discard samples from the end of a block of data: *Offset* should be set to 0 and *nWrite* should be set to *nRead* $- N$, where N is the number of samples that need to be discarded.
   - Discard samples from both the beginning and the end of a block of data: *Offset* should be set to $-N_b$, where N_b is the number of samples that need to be discarded from the beginning of the block and *nWrite* should be set to *nRead* $- N_b - N_e$, where N_e is the number of samples that need to be discarded from the end of the block.
   - Prepend zeros to a block of data: *Offset* should be set to N , where N is the number of zeros to be prepended, *UsePastInputs* should be set to NO, and *nWrite* should be set to *nRead* $+ N$.
   - Append zeros to a block of data: *Offset* should be set to 0 and *nWrite* should be set to *nRead* $+ N$, where N is the number of zeros to be appended.
   - Prepend and append zeros to a block of data: *Offset* should be set to N_p , where N_p is the number of zeros to be prepended, *UsePastInputs* should be set to NO, and *nWrite* should be set to *nRead* $+ N_p + N_a$, where N_a is the number of zeros to be appended.

- Break an input stream of samples in blocks of size N_b with N_o overlapping samples: *nRead* should be set to $N_b - N_o$, *nWrite* should be set to N_b, *Offset* should be set to N_o, and *UsePastInputs* should be set to YES.

6. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
7. See also: *ChopVarOffset* (numeric).

ChopVarOffset



Description: Chop input data into blocks with offset control
Library: Numeric, Control
Class: SDFChopVarOffset
Derived: From Chop
C++ Code: See *doc/sp_items/SDFChopVarOffset.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
nRead	number of data items read	128		int	[1, ∞)
nWrite	number of data items written	64		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype
2	offsetCntrl		int

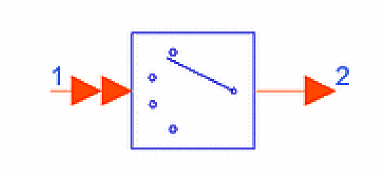
Pin Outputs

Pin	Name	Description	Signal Type
3	output		anytype

Notes/Equations

1. ChopVarOffset has the same functionality as the *Chop* (numeric) component except that the *Offset* parameter is determined at run time by a control input (*offsetCntrl*) and the *UsePastInputs* parameter is assumed to be NO.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. See also: *Chop* (numeric)

Commutator



Description: Synchronous Data Commutator
Library: Numeric, Control
Class: SDFCommutator
C++ Code: See *doc/sp_items/SDFCommutator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

Pin Outputs

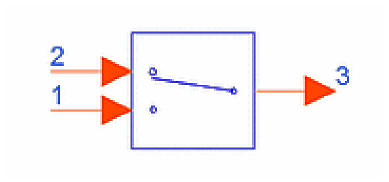
Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. This component takes N input streams and synchronously combines them into one output stream. It consumes B input data packets from each input (where B is BlockSize), and produces N × B data packets on the output.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Commutator2



Description: 2-Input Synchronous Data Commutator
Library: Numeric, Control
Class: SDFCommutator
C++ Code: See *doc/sp_items/SDFCommutator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype

Pin Outputs

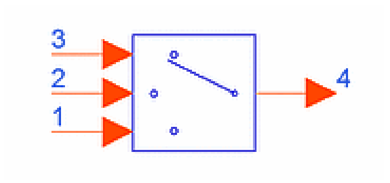
Pin	Name	Description	Signal Type
3	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. This component takes 2 input streams and synchronously combines them into one output stream. It accepts 2 single signals, both of the same type.
   It consumes B input data packets from each input (where B is BlockSize), and produces 2B data packets on the output. The first B data packets on the output come from the first input, the next B data packets from the next input.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Commutator3



Description: 3-Input Synchronous Data Commutator
Library: Numeric, Control
Class: SDFCommutator
C++ Code: See *doc/sp_items/SDFCommutator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype

Pin Outputs

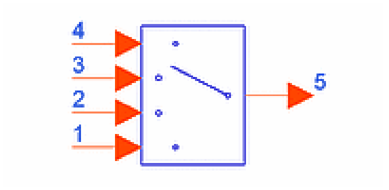
Pin	Name	Description	Signal Type
4	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. Commutator3 takes 3 input streams and synchronously combines them into one output stream. It accepts 3 single signals, all of the same type.
   It consumes B input data packets from each input (where B is BlockSize), and produces 3 B data packets on the output. The first B data packets on the output come from the first input, the next B data packets from the next input, and so on.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Commutator4



Description: 4-Input Synchronous Data Commutator
Library: Numeric, Control
Class: SDFCommutator
C++ Code: See *doc/sp_items/SDFCommutator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype

Pin Outputs

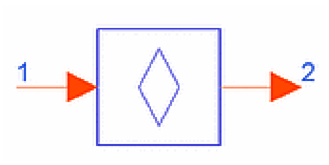
Pin	Name	Description	Signal Type
5	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. Commutator4 takes 4 input streams and synchronously combines them into one output stream. It accepts 4 single signals, all of the same type.
   It consumes B input data packets from each input (where B is BlockSize), and produces 4 B data packets on the output. The first B data packets on the output come from the first input, the next B data packets from the next input, and so on.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Delay



Description: Delay Component
Library: Numeric, Control
Class: HOFDelay
Derived From: Nop

Parameters

Name	Description	Default	Type	Range
N	N	1	int	[0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

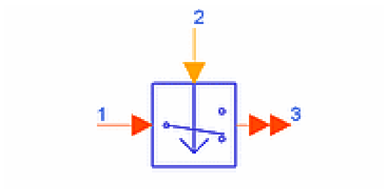
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. This component delays input tokens from output by N samples. The initial N output tokens have a null value.
2. For timed signals, use the DelayRF component.
3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
4. The N parameter cannot be swept.

DeMux



Description: Data demultiplexer
Library: Numeric, Control
Class: SDFDeMux
C++ Code: See *doc/sp_items/SDFDeMux.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of data items in a block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype
2	control		int

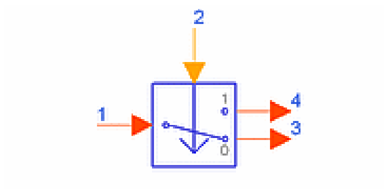
Pin Outputs

Pin	Name	Description	Signal Type
3	output		multiple anytype

Notes/Equations

1. DeMux demultiplexes one input onto any number of output streams. DeMux consumes B packets of data from the input, where B is the BlockSize. These B data packets are copied to exactly one output, determined by the control input. The other outputs get a zero of the appropriate type.
2. Integers 0 through N - 1 are accepted at the control input, where N is the number of outputs. If the control input is outside this range, all outputs get zero.
3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

DeMux2



Description: 2-Output Data Demultiplexer
Library: Numeric, Control
Class: SDFDeMux
C++ Code: See *doc/sp_items/SDFDeMux.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of data items in a block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype
2	control		int

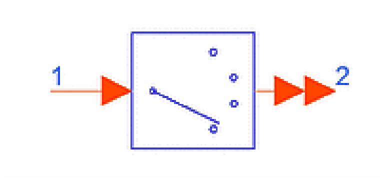
Pin Outputs

Pin	Name	Description	Signal Type
3	output#1		anytype
4	output#2		anytype

Notes/Equations

1. DeMux2 directs one input to either of two outputs based on the logic state (0 or 1) of the control input. DeMux2 produces 2 single signal outputs, all of the same type as the input.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Distributor



Description: Synchronous Data Distributor
Library: Numeric, Control
Class: SDFDistributor
C++ Code: See *doc/sp_items/SDFDistributor.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

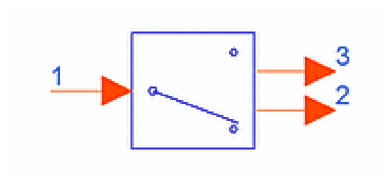
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. Distributor synchronously splits one input stream into N output streams. It consumes N × B input particles (where B = BlockSize) and sends the first B particles to the first output, the next B particles to the next output, and so on. It produces a single signal output of the same type as input.
   The number of output streams, N, is equal to the number of other component input pins connected to the Distributor output pin. For an ordered distribution of output streams to input pins, a BusSplit[2, ... , 9] component can be connected to the Distributor output pin and other component input pins connected to the BusSplit[2, ... , 9] component output pins.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Distributor2



Description: 2-Output Synchronous Data Distributor
Library: Numeric, Control
Class: SDFDistributor
C++ Code: See *doc/sp_items/SDFDistributor.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

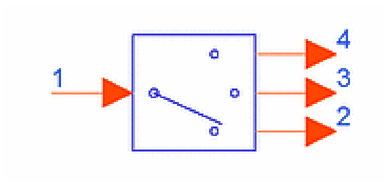
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype

Notes/Equations

1. Distributor2 synchronously splits one input stream into 2 output streams. It consumes 2 × B input particles (where B = BlockSize) and sends the first B particles to the first output and the next B particles to the second output. It produces 2 single signal outputs, both of the same type as the input.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Distributor3



Description: 3-Output Synchronous Data Distributor
Library: Numeric, Control
Class: SDFDistributor
C++ Code: See *doc/sp_items/SDFDistributor.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

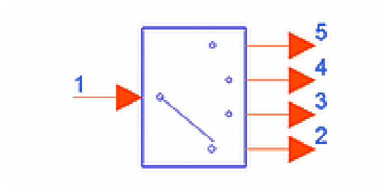
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype

Notes/Equations

1. Distributor3 synchronously splits one input stream into 3 output streams. It consumes 3 × B input particles (where B = BlockSize) and sends the first B particles to the first output, the second B particles to the second output, and the third B particles to the third output. It produces 3 single signal outputs, all of the same type as the input.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Distributor4



Description: 4-Output Synchronous Data Distributor
Library: Numeric, Control
Class: SDFDistributor
C++ Code: See *doc/sp_items/SDFDistributor.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	Number of particles in a block.	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

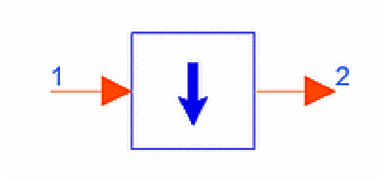
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype

Notes/Equations

1. Distributor4 synchronously splits one input into 4 output streams. It consumes 4 × B input particles (where B = BlockSize) and sends the first B particles to the first output, the second B particles to the second output, the third B particles to the third output, and the fourth B particles to the fourth output. It produces 4 single signal outputs, all of the same type as the input.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

DownSample



Description: Data Down Sampler
Library: Numeric, Control
Class: SDFDownSample
C++ Code: See *doc/sp_items/SDFDownSample.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Factor	downsample factor	2		int	[1, ∞)
Phase	downsample phase	0		int	[0,Factor-1]

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

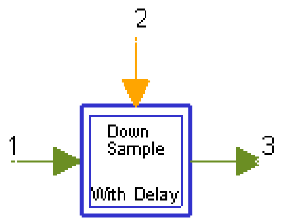
Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. Down-sampling is also referred to as *decimation*. This component reduces the sampling rate of its input signal by an integer Factor ratio. Decimation is performed by keeping one sample at the output for every Factor samples at the input.
2. This component does not have a built-in lowpass filter before decimation. To avoid aliasing, it may be necessary for the designer to ensure that the input signal bandwidth is appropriately limited by connecting a lowpass filter at the input.
3. Phase tells which sample to output: if Phase = 0, the most recent sample is the output; if Phase=Factor − 1 the oldest sample is the output. y[n] = x[Factor × (n + 1) − (Phase − 1)], where n is the output sample number, y is the output, and x is the input. (Note that *phase* has the opposite sense of the Phase parameter in the UpSample component, but the same sense as the Phase parameter in the FIR component.)
4. For timed signals, use the *DSampleRF* (timed) component.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

DSampleWOffset



Description: Down sample with detected delay
Library: Numeric, Control
Class: SDFDSampleWOffset

Parameters

Name	Description	Default	Unit	Type	Range
SampPerChip	Number of samples fer chip	8		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	Input	Input	anytype
2	Offset	Offset	int

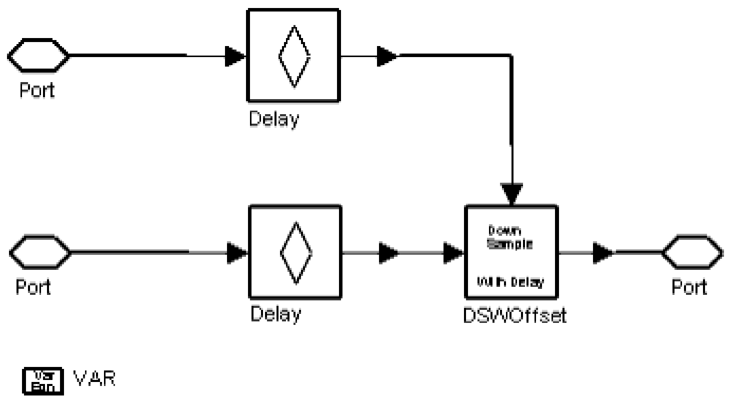
Pin Outputs

Pin	Name	Description	Signal Type
3	Output	Output	anytype

Notes/Equations

1. This model downsamples RF received data using RF channel delay information. The first input is the complex envelope data that will be downsampled; the second input is the RF channel delay detected by DelayEstimator. The downsampled complex envelope signal is output.
   The schematic for this subnetwork is shown in DSampleWOffset Schematic. Two Delay models are inserted in the subnet based on the RF feedback loop reqirement. The C++ model DSWOffset performs the downsample for the signal with an arbitrary delay.
2. This is a multirate component.
   If SampPerChip > 1, the component downsamples the signal with RF channel delay using SampPerChip as the downsample ratio.
   If SampPerChip = 1, the input signal is passed to the output with a delay and downsampling is not performed.
3. To downsample an RX signal with an arbitrary delay through an RF path, component sampling time is synchronized with the delayed TX signal start time.

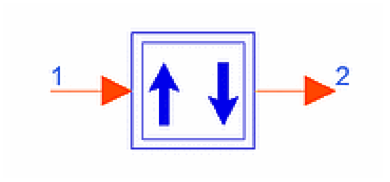
DSampleWOffset Schematic



References

1. M. Jeruchim, P. Balaban and K. Shanmugan, "Simulation of Communication System," Plenum Press, New York and London, 1992.

EnableUDSample



Description: Data Up/Down Sampler
Library: Numeric, Control
Class: SDFEnableUDSample

Parameters

Name	Description	Default	Unit	Type	Range
Enable	enable the up/down sampling: NO, YES	NO		enum	
USample	upsample ratio	1		int	[1, ∞)
DSample	downsample ratio	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	anytype

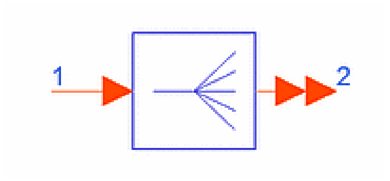
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	anytype

Notes/Equations

1. EnableUDSample can be used to resample the input signal at a new rate. Resampling occurs only when the Enable parameter is set to YES.
2. When USample is greater than 1 upsampling will occur. Upsampling is done as sample and hold (repeat input sample USample times).
   When DSample is greater than 1 downsampling will occur. The downsampling phase is DSample-1, that is, the first out of every DSample samples is selected and the subsequent DSample-1 samples are discarded.
   If USample is smaller than DSample loss of information may occur.
3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork
Derived From: Base

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

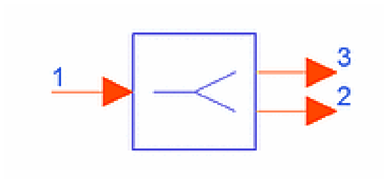
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. Fork is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems-for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork is typically used with numeric signals; one or more SplitterRF components should be used with timed signals.
   When a Fork is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork2



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

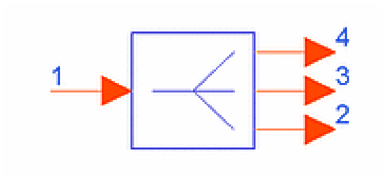
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype

Notes/Equations

1. Fork2 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork2 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork2 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork2 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems- for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork2 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork2 connects a single output port of a component to 2 input ports of other components. It has 2 single output ports rather than one multi-port output.
4. Fork2 is typically used with numeric signals; SplitterRF should be used with timed signals.
   When Fork2 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork3



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

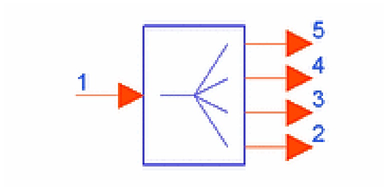
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype

Notes/Equations

1. Fork3 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork3 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork3 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork3 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems-for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork3 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork3 connects a single output port of a component to 3 input ports of other components. It has 3 single output ports rather than one multi-port output.
4. Fork3 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork3 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork4



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

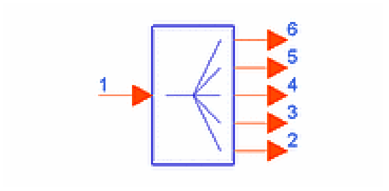
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype

Notes/Equations

1. Fork4 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork4 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork4 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork4 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems- for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork4 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork4 connects a single output port of a component to 4 input ports of other components. It has 4 single output ports rather than one multi-port output.
4. Fork4 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork4 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork5



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

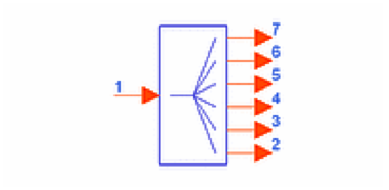
Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype

Notes/Equations

1. Fork5 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork5 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork5 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork5 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems-for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork5 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork5 connects a single output port of a component to 5 input ports of other components. It has 5 single output ports rather than one multi-port output.
4. Fork5 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork5 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork6



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

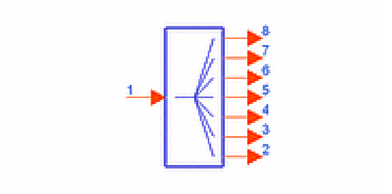
Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype

Notes/Equations

1. Fork6 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork6 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork6 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork6 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems- for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork6 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork6 connects a single output port of a component to 6 input ports of other components. It has 6 single output ports rather than one multi-port output.
4. Fork6 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork6 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.
5. For general information regarding numeric control components, refer to *Numeric*

Control Components (numeric).

Fork7



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

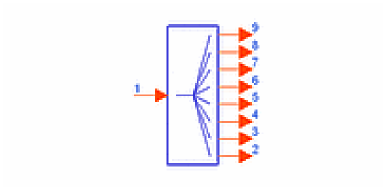
Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype

Notes/Equations

1. Fork7 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork7 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork7 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork7 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems-for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork7 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork7 connects a single output port of a component to 7 input ports of other components. It has 7 single output ports rather than one multi-port output.
4. Fork7 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork7 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.

5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork8



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype
9	output#8		anytype

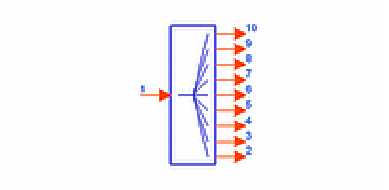
Notes/Equations

1. Fork8 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork8 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork8 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork8 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems- for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork8 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork8 connects a single output port of a component to 8 input ports of other components. It has 8 single output ports rather than one multi-port output.
4. Fork8 is typically used with numeric signals; SplitterRF components should be used with timed signals.
   When Fork8 is forced to connect with a timed signal, it assumes infinite equivalent

input resistance and zero equivalent output resistance.

5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Fork9



Description: Copy input particles to each output.
Library: Numeric, Control
Class: HOFFork

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

Pin Outputs

Pin	Name	Description	Signal Type
2	output#1		anytype
3	output#2		anytype
4	output#3		anytype
5	output#4		anytype
6	output#5		anytype
7	output#6		anytype
8	output#7		anytype
9	output#8		anytype
10	output#9		anytype

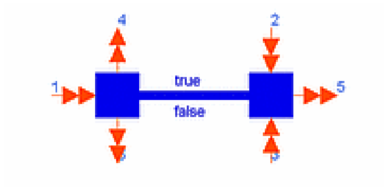
Notes/Equations

1. Fork9 is generally used to explicitly connect a single output port of a component to multiple input ports of other components. Fork9 cannot be used to connect a multi-port output of a component to multiple multi-port inputs of other components. For example, the input of the Fork9 component cannot be connected to a bus of width > 1.
2. In many data flow graphs, the explicit use of this component is optional. If not used, it will be automatically inserted when multiple inputs are connected to the same output in a schematic.
   Automatically inserted Fork9 components are not always desirable:
   - When multi-port inputs or outputs are used, auto-forking can cause problems- for example, two outputs and several inputs on the same net.
   - When there is a delay on one of the arcs, Fork9 must be explicitly inserted by the designer to avoid ambiguity about the location of the delay.
3. Fork9 connects a single output port of a component to 9 input ports of other components. It has 9 single output ports rather than one multi-port output.
4. Fork9 is typically used with numeric signals; SplitterRF components should be used with timed signals.

When Fork9 is forced to connect with a timed signal, it assumes infinite equivalent input resistance and zero equivalent output resistance.

5. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

IfElse



Description: Map one of two blocks
Library: Numeric, Control
Class: HOFIfElse
Derived From: Map

Parameters

Name	Description	Default	Type	Range
Condition	Select 'False' or 'True' path based on the Condition: False, True	True	enum	False or True

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input to the IfElse component	multiple anytype
2	true_mapoutput	Connect to the output pin, if any, of the design path that will be selected if Condition evaluates to TRUE	multiple anytype
3	false_mapoutput	Connect to the output pin, if any, of the design path that will be selected if Condition evaluates to FALSE	multiple anytype

Pin Outputs

Pin	Name	Description	Signal Type
4	true_mapinput	Connect to the input pin, if any, of the design path that will be selected if Condition evaluates to TRUE	multiple anytype
5	output	Output from the IfElse component	multiple anytype
6	false_mapinput	Connect to the input pin, if any, of the design path that will be selected if Condition evaluates to FALSE	multiple anytype

Notes/Equations

1. IfElse can be used to select one of the two components, the one in the true path or the one in the false path, and insert it in the signal flow path. If more than one component need to be connected to the true/false path these must be placed in a subnetwork and the subnetwork connected to the path.
The Condition parameter determines which path will be selected. If Condition is set to False, then the false path is selected; if Condition is set to True, then the true path is selected; if Condition is set to a variable or expression, the variable or expression must have a value of 0 or 1 (0 is treated as False and 1 is treated as True). Other values will result in a simulation error.
2. IfElse is similar to the Mux2 component with some differences as well as advantages

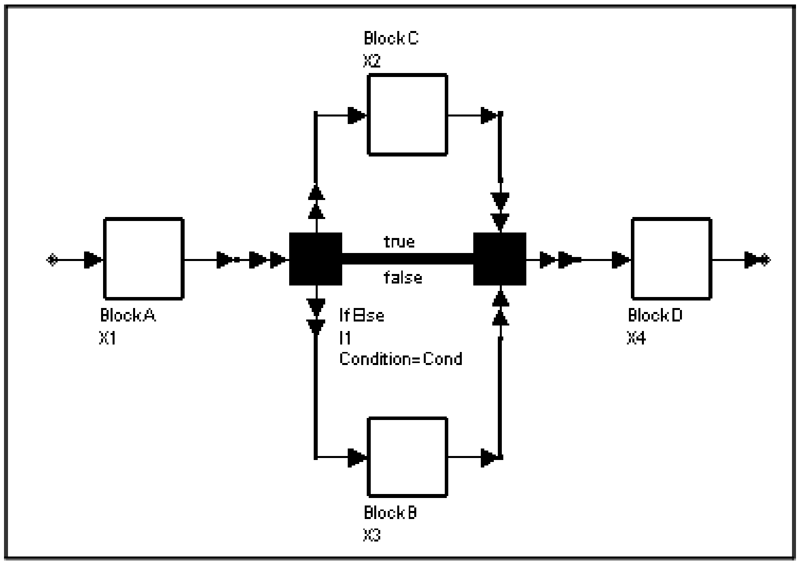
and disadvantages. Equivalent Schematics Using IfElse and Mux2 shows how IfElse and Mux2 can be used to generate equivalent schematics; these schematics will produce identical results assuming Condition is 0 or 1.

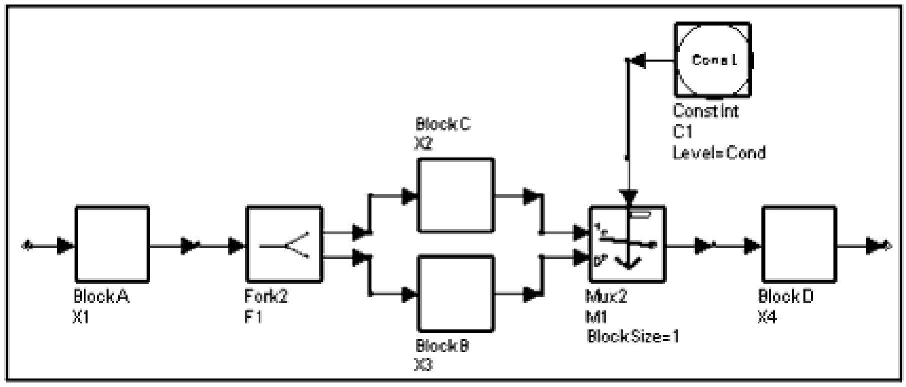
The important difference between IfElse and Mux2 is that IfElse operates at the graph level (which means that the component not selected by IfElse is completely removed from the graph before the simulation starts), whereas Mux2 operates at the signal level (which means that both input signals of Mux2 must be generated, then Mux2 selects one of them).

The advantage of operating at the graph level is that because one of the two components connected to the true or false path of IfElse is completely removed from the graph, only the selected one is simulated thus saving computing resources. On the other hand, the advantage of Mux2 is that the control signal that selects which of the two input signals will be selected can change during the simulation. In fact, this (a varying control signal) is the most typical use of Mux2. Having a constant control signal, as shown in Equivalent Schematics Using IfElse and Mux2, is not a typical use of Mux2 (the purpose of the example in Equivalent Schematics Using IfElse and Mux2 is to help explain similarities/differences between IfElse and Mux2 and not to provide a typical example for Mux2).

Another difference between the two schematics in Equivalent Schematics Using IfElse and Mux2 is that the Condition parameter of IfElse is not sweepable, whereas the Level parameter of the ConstInt component (although constant during the simulation) is sweepable.

Equivalent Schematics Using IfElse and Mux2



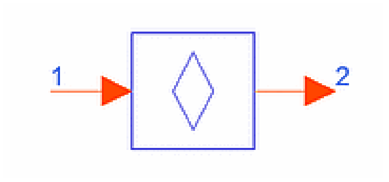


3. Although the Condition parameter of IfElse cannot be swept, the parameters of the components in the true or false path can be swept; for this, the expressions setting the values of these parameters must be enclosed in double quotes. For example, if a Repeat component is connected to the true or false path of IfElse and there is a swept variable called Rate, in order to use this variable to set the NumTimes parameter of Repeat the assignment should be done as NumTimes = "Rate" or NumTimes = "3 × Rate + 1".
   If more complicated expressions using functions such as sin(), log(), or sqrt() need to be used, then an intermediate variable must be defined. For the example described above, in order to set NumTimes to int(sqrt(Rate) + 3 × log(Rate)) an intermediate variable (RepeatFactor for example) must be defined in a VAR block as RepeatFactor = int(sqrt(Rate) + 3 × log(Rate)). Then the NumTimes parameter of Repeat must be set as NumTimes = "RepeatFactor".
   The above examples are exceptions to how expressions using swept variables are used to assign values to component parameters. These exceptions apply only to the parameters of the components connected to the true or false paths of IfElse.
4. IfElse is intended for use with numeric components. Timed components can be connected to the true or false path of IfElse but any series or shunt resistors connected outside IfElse that could form resistor networks with the resistors inside the timed components will not be correctly evaluated.
   Connecting Analog/RF subnetworks to the true or false path of IfElse is not supported. The simulator will not error out but the results of the simulation are not guaranteed to be correct.
5. To access examples that show how this component is used: from the Main window, choose *File > Open > Example > PtolemyDocExamples > Numeric_Control_wrk*; from the Schematic window, choose *File > Open, IfElse_Example1, IfElse_Example2*,

or *IfElse_Example3*. More examples showing the usage of IfElse are the *Bits* and the *TkConstellation* subnetworks (place these components in a schematic window and push into them).

6. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

InitDelay



Description: Initial Delay Component
Library: Numeric, Control
Class: HOFInitDelay
Derived From: Delay

Parameters

Name	Description	Default	Type	Range
N	N	1	int	[0, ∞)
InitialDelays	StringArray containing a list of intial delay tokens.	0	string array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

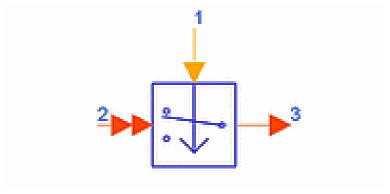
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. InitDelay delays input tokens from output by N sets of initial delay tokens.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).
3. The N parameter cannot be swept.

Mux



Description: Data multiplexer
Library: Numeric, Control
Class: SDFMux
C++ Code: See *doc/sp_items/SDFMux.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of data items in a block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	control		int
2	input		multiple anytype

Pin Outputs

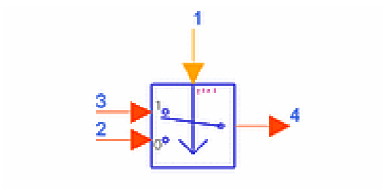
Pin	Name	Description	Signal Type
3	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. Mux multiplexes any number of inputs onto one output stream. At each firing, BlockSize data packets are consumed on each input, but only one of these blocks of data is copied to the output, as determined by the control input. Integers from 0 through N - 1 are accepted at the control input, where N is the number of inputs. If the control input is outside this range, an error is signaled.
   Use of a BusMerge component at input 2 of the Mux is recommended to ensure that the order of inputs is not ambiguous. When a BusMerge component is used, control inputs 0 through N-1 select inputs at pin 1 through N of the BusMerge, respectively.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Mux2



Description: 2-Input Data Multiplexer
Library: Numeric, Control
Class: SDFMux
C++ Code: See *doc/sp_items/SDFMux.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of data items in a block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	control		int
2	input#1		anytype
3	input#2		anytype

Pin Outputs

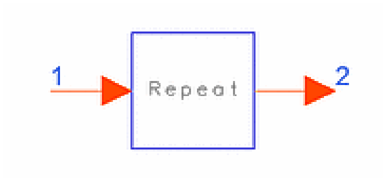
Pin	Name	Description	Signal Type
4	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. Mux2 multiplexes 2 inputs onto one output stream. At each firing, BlockSize data packets are consumed on each single signal input pin. Only one of these blocks of data is copied to the output; the one copied is determined by the control input. Integers 0 to 1 are accepted at the control input; 0 selects the input at pin 2; 1 selects the input at pin 3.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Repeat



Description: Data repeater
Library: Numeric, Control
Class: SDFRepeat
C++ Code: See *doc/sp_items/SDFRepeat.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
NumTimes	repetition factor	2		int	[1, ∞)
BlockSize	number of data items in a block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

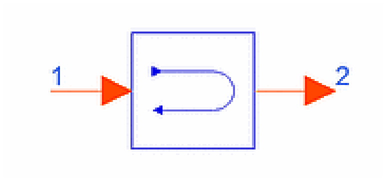
Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. Repeat repeats each input data packet the specified number of times (NumTimes) on the output. Note that this is a sample rate change, and therefore affects the number of invocations of downstream components.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Reverse



Description: Data reverser
Library: Numeric, Control
Class: SDFReverse
C++ Code: See *doc/sp_items/SDFReverse.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
N	number of data items read and written	64		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

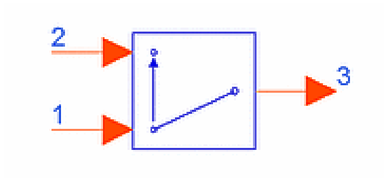
Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. On each execution, Reverse reads a block of N samples and writes the samples backwards.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Trainer



Description: Initial Sample Trainer
Library: Numeric, Control
Class: SDFTrainer
C++ Code: See *doc/sp_items/SDFTrainer.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
TrainLength	number of training samples to use	100		int	[0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	train		anytype
2	decision		anytype

Pin Outputs

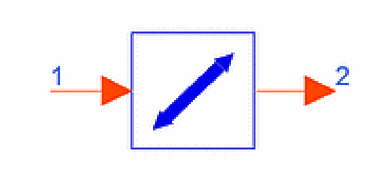
Pin	Name	Description	Signal Type
3	output		anytype

Notes/Equations

Note
Use of this component with timed signals having different characterization frequencies is not recommended and can lead to unexpected results.

1. Trainer passes the value of the train input to the output for the first TrainLength samples, then passes the decision input to the output. This component is designed for use with adaptive equalizers that require a training sequence at startup, but it can be used whenever one sequence is used during a startup phase, and another sequence after that.
2. During the startup phase, the decision inputs are discarded. After the startup phase, the train inputs are discarded.
3. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

Transpose



Description: Data transposer
Library: Numeric, Control
Class: SDFTranspose
C++ Code: See *doc/sp_items/SDFTranspose.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
SamplesInaRow	number of input samples constituting a row	8		int	[1, ∞)
NumberOfRows	number of rows in the input matrix	8		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

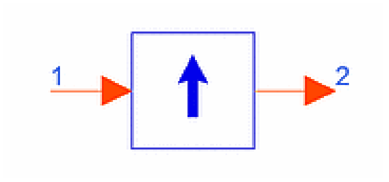
Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

1. Transpose transposes a rasterized matrix (one that is read as a sequence of data items, row by row, and written in the same form). The number of data items produced and consumed equals the product of SamplesInaRow and NumberOfRows.
2. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

UpSample



Description: Data Up Sampler
Library: Numeric, Control
Class: SDFUpSample
C++ Code: See *doc/sp_items/SDFUpSample.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Factor	number of samples produced	2		int	[1, ∞)
Phase	where to put the input in the output block	0		int	[0, Factor-1]
Fill	value to fill the output block	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype

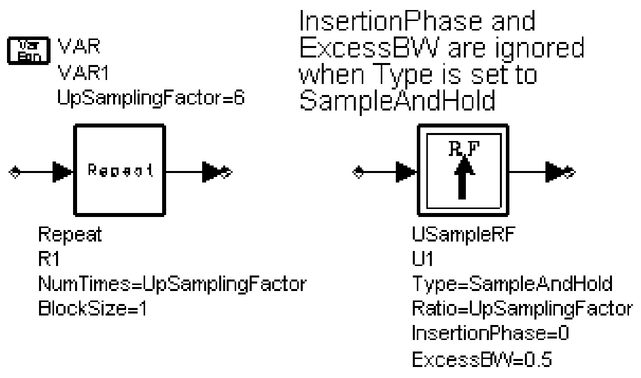
Pin Outputs

Pin	Name	Description	Signal Type
2	output		anytype

Notes/Equations

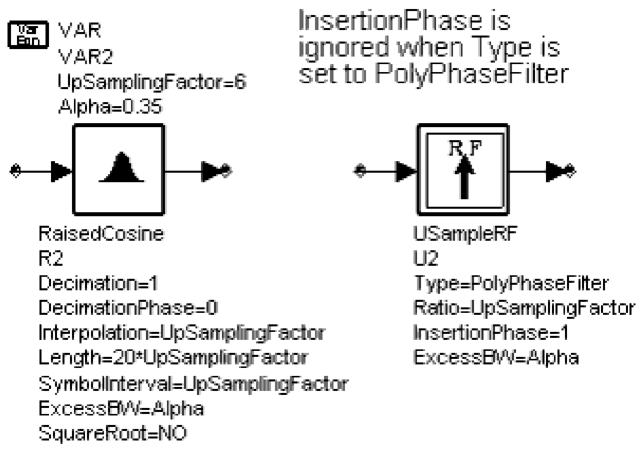
1. The Upsample component upsamples by a given Factor, giving inserted samples the value Fill. The Phase parameter tells where to put the sample in an output block. A Phase of 0 indicates to output the input sample first followed by the inserted samples. The maximum Phase is Factor − 1. $y[\text{Factor} \times n] = x[n + \text{Phase}]$, where n is the input sample number, y is the output, x is the input.
2. Although the Fill parameter is a floating-point (real) number, if the input is of some other type, such as complex, then Fill data will be obtained by casting Fill to the appropriate type.
3. For timed signals, use the USampleRF component.
   The USampleRF component has options for specifying how the inserted values will be generated: SampleAndHold, ZeroInsertion, PolyPhaseFilter, Linear.
   The UpSample component implements the ZeroInsertion option only, assuming Fill is set to 0. While UpSample cannot implement other USampleRF options, other components in the Numeric library can be used to implement them.
   - The SampleAndHold option can be implemented by the Repeat component. The NumTimes parameter of the Repeat component should be set to the upsampling factor and the BlockSize parameter should be set to 1. Equivalence of Repeat and SampleAndHold Option of USampleRF shows how to set the Repeat and the USampleRF components to get equivalent results.

Equivalence of Repeat and SampleAndHold Option of USampleRF



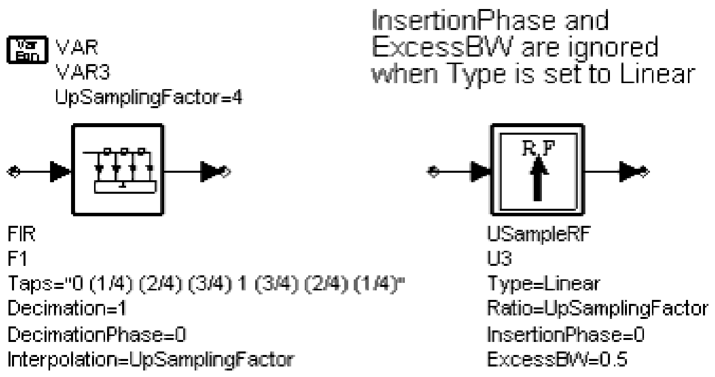
- The PolyPhaseFilter option can be implemented by the RaisedCosine component. Parameters of the RaisedCosine component should be set as follows: Decimation = 1, DecimationPhase = 0, Interpolation = N, Length = 20 × N, SymbolInterval = N, ExcessBW = a, and SquareRoot = 0 (where N is the USampleRF Ratio parameter value and a is the USampleRF ExcessBW parameter value). Equivalence of RaisedCosine and PolyPhaseFilter Option of USampleRF. shows how to set the RaisedCosine and the USampleRF components to get equivalent results.

Equivalence of RaisedCosine and PolyPhaseFilter Option of USampleRF.



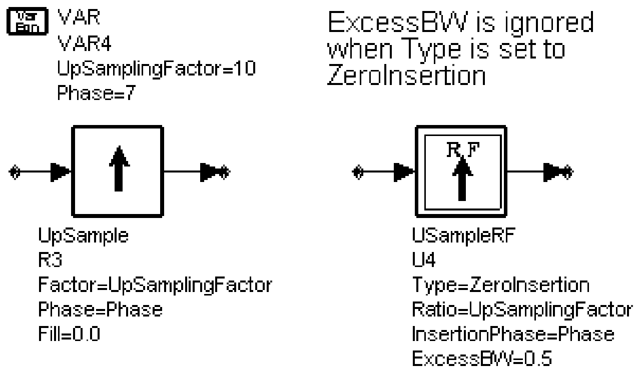
- The Linear option can be implemented by the FIR component. FIR parameters should be set as follows: Taps = "0 (1/N) (2/N) ... ((N-1)/N) 1 ((N-1)/N) ... (1/N)", Decimation = 1, DecimationPhase = 0, Interpolation = N (where N is the USampleRF Ratio parameter value). (Note that the open and close quotes in the Taps parameter value are required.) Equivalence of FIR and Linear Option of USampleRF shows how to set the FIR and the USampleRF components to get equivalent results.

Equivalence of FIR and Linear Option of USampleRF



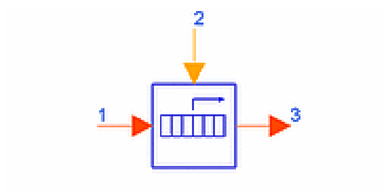
- For completeness, Equivalence of UpSample and ZeroInsertion Option of USampleRF shows the equivalance of UpSample and the ZeroInsertion option USampleRF.

Equivalence of UpSample and ZeroInsertion Option of USampleRF



4. For general information regarding numeric control components, refer to *Numeric Control Components* (numeric).

VarDelay



Description: Variable Delay
Library: Numeric, Control
Class: SDFVarDelay

Parameters

Name	Description	Default	Unit	Type	Range
MaxDelay	Maximum delay	10		int	[0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		anytype
2	control		int

Pin Outputs

Pin	Name	Description	Signal Type
3	output		anytype

Notes/Equations

This component implements a varying delay by delaying the input signal by as many samples as specified by the signal applied to the control pin. The maximum delay needs to be specified in the MaxDelay parameter.
The component uses an internal buffer of MaxDelay samples. The value at the control pin decides which value in the buffer is output. A control value of 0 or less outputs the most current sample in the buffer (the one just read). A control value of MaxDelay, or more, outputs the oldest sample in the buffer (the one read MaxDelay executions of the component ago). A control value of N (0 < N < MaxDelay) outputs the sample in the buffer that was read N executions of the component ago.

Numeric Fixed-Point DSP Components

- *AbsSyn* (numeric)
- *AccumSyn* (numeric)
- *AddRegSyn* (numeric)
- *AddSyn* (numeric)
- *And2Syn* (numeric)
- *AndSyn* (numeric)
- *BarShiftSyn* (numeric)
- *BitFillSyn* (numeric)
- *BPSKSyn* (numeric)
- *BufferSyn* (numeric)
- *Bus8MergeSyn* (numeric)
- *Bus8RipSyn* (numeric)
- *BusMergeSyn* (numeric)
- *BusRipSyn* (numeric)
- *CastSyn* (numeric)
- *CombFiltSyn* (numeric)
- *Comp6Syn* (numeric)
- *CompSyn* (numeric)
- *ConstSyn* (numeric)
- *CountCombSyn* (numeric)
- *CounterSyn* (numeric)
- *Div2ClockSyn* (numeric)
- *DPRamRegSyn* (numeric)
- *DPRamSyn* (numeric)
- *DPSKSyn* (numeric)
- *DualNCOSyn* (numeric)
- *FIRSyn* (numeric)
- *FixedGainSyn* (numeric)
- *FixToFloatSyn* (numeric)
- *FloatToFixSyn* (numeric)
- *FSMSyn* (numeric)
- *GainSyn* (numeric)
- *IntegratorSyn* (numeric)
- *LCounterSyn* (numeric)
- *MultRegSyn* (numeric)
- *MultSyn* (numeric)
- *Mux2Syn* (numeric)
- *Mux3Syn* (numeric)
- *Mux4Syn* (numeric)
- *MuxSyn* (numeric)
- *Nand2Syn* (numeric)
- *NCOSyn* (numeric)
- *Nor2Syn* (numeric)
- *NotSyn* (numeric)
- *OQPSKSyn* (numeric)
- *Or2Syn* (numeric)
- *OrSyn* (numeric)
- *PI4DQPSKSyn* (numeric)
- *PSK8Syn* (numeric)

- *QPSKSyn* (numeric)
- *RamRegSyn* (numeric)
- *RamSyn* (numeric)
- *RegSyn* (numeric)
- *RomRegSyn* (numeric)
- *RomSyn* (numeric)
- *SerialFIRSyn* (numeric)
- *ShiftRegPPSyn* (numeric)
- *ShiftRegPSSyn* (numeric)
- *ShiftRegSPSyn* (numeric)
- *SineCosineSyn* (numeric)
- *SinkRespSyn* (numeric)
- *SinkStimSyn* (numeric)
- *SubRegSyn* (numeric)
- *SymFIRSyn* (numeric)
- *Xor2Syn* (numeric)
- *XorSyn* (numeric)
- *ZeroInterpSyn* (numeric)

The numeric fixed-point DSP components provide digital signal processing functions on single data points of data that are fixed-point (fixed). These components do not accept any matrix class of signal.

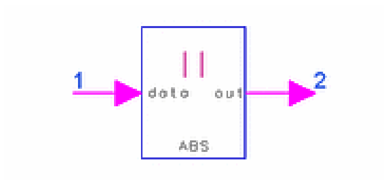
If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. The auto conversion from timed, complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30; a value of 0.5 would create one of precision of 1.31. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Fixed-point DSP components (such as registers, counters, shift registers) that have clock inputs have the following simulation behavior depending on whether clock inputs are connected or not. If clock is not connected, then each simulation step is taken as a positive clock edge; for example, if the data register RegSyn clock is not connected, then RegSyn simulates a a unit-step delay. If clock is connected, then the component will simulate according to the clock input state; for example, if the data register RegSyn clock is connected, then RegSyn simulates as a positive edge clock sensitive register.

Fixed-point DSP components (such as registers, counters, and shift registers) that have set inputs have the following simulation behavior depending on whether the set inputs are connected or not. If the set input is not connected, then the component is reset at the first simulation step. If the set input is connected, then the component will simulate according to the set input state.

For fixed-point DSP components that perform math operations (such as adders, subtractors, gain blocks, and filters), the ArithType parameter specifies the arithmetic type of the output signal and can be set to TWOS_COMPLEMENT or UN_SIGNED values. When the input fixed-point signal has an arithmetic type that is not the same as ArithType, the bit pattern representing the input number will be interpreted in the arithmetic defined by ArithType. This can lead to unexpected results; therefore, arithmetic types should not be mixed when performing math operations.

AbsSyn



Description: Absolute
Library: Numeric, Fixed-Point DSP
Class: SDFAbsSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

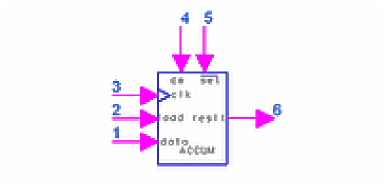
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. AbsSyn presents an output with the absolute value of the given data input.
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

AccumSyn



Description: Scaled by 1/2 Accumulator
Library: Numeric, Fixed-Point DSP
Class: SDFAccumSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input − Data input which is loaded by asserting Load input	fix
2	Load	Load input − loads Data into accumulator of accumulator	fix
3	Clock	Clock input − optional control pin	fix
4	CE	Clock enable input − optional control pin	fix
5	Set	Asynchronous set/reset input − optional control pin	fix

Pin Outputs

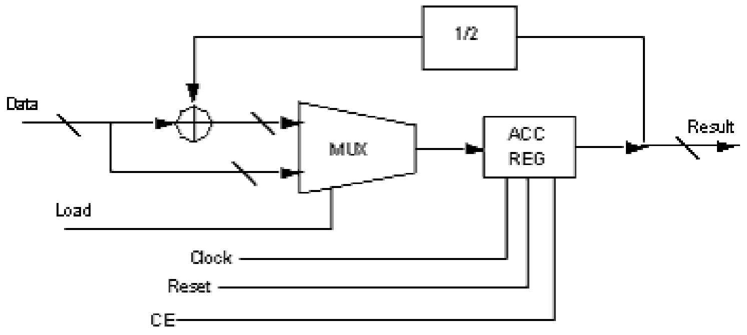
Pin	Name	Description	Signal Type
6	Result		fix

Notes/Equations

1. This model is a scale-by-half accumulator. Physically, the model can be viewed as an adder that adds the present input Data to one-half the value of the previous output of the adder. The delayed adder output feedback is achieved by using an internal data register that is clocked by the positive edge transitions of the Clock 1-bit. In discrete equation form, the equation defining the model is:

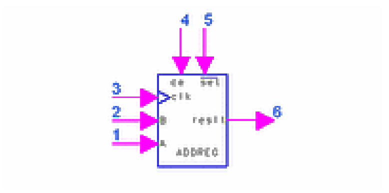
```
Result = Previous_Result/2 + Data
```

Internal Structure of Scale-by-Half Accumulator Model



2. The Clock input is optional:
   - if it is connected, the model will operate based on the positive edge transitions of the Clock input
   - if it is not connected, the model will operate as if every sample step of the simulator is a positive edge transition
3. Assertion of the Reset input by bringing it low (a value of 0) will clear the internal data register.
4. The (optional) CE input is the clock-enable control for the internal data register.
   - if it is connected and has a high value (a value of 1), the internal data register is enabled and will load its input onto a positive Clock edge
   - if it is not connected and low (a value of 0) the clock to the internal data register is disabled. The internal data register is always enabled when the CE input is not connected
5. The (optional) Load input is asserted by bring it high (a value of 1).
   - if it is asserted, the Data input is loaded into the internal data register
   - if it is unconnected, the Load is never asserted
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

AddRegSyn



Description: Registered Adder
Library: Numeric, Fixed-Point DSP
Class: SDFAddRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix
3	Clock	Clock input – optional control pin	fix
4	CE	Clock enable input – optional control pin	fix
5	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
6	Result		fix

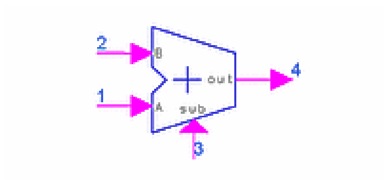
Notes/Equations

1. This model is a registered adder. It calculates the addition of its A and B data inputs (A+B) and registers its output Result such that it has the specified precision as set in the OutputPrecision parameter.
2. The Clock input is optional:
    - if it is connected, the model will operate based on the positive edge transitions of the Clock input
    - if it is not connected, the model will operate as if every sample step of the simulator is a positive edge transition
3. Assertion of the Reset input by bringing it low (a value of 0) will clear the output data register.
4. The (optional) CE input is the clock-enable control for the output data register:
    - if it is connected and has a high value (a value of 1), the output data register is

enabled and will load the addition result upon a positive Clock edge.
- if it is connected, and low (a value of 0) the clock to the output data register is disabled.
- if the CE input is not connected, the output data register is always enabled.

5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

AddSyn



Description: Adder/Subtractor
Library: Numeric, Fixed-Point DSP
Class: SDFAddSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum
AddSub	enumeration state: ADD, SUBTRACT	ADD	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix
3	Sub		fix

Pin Outputs

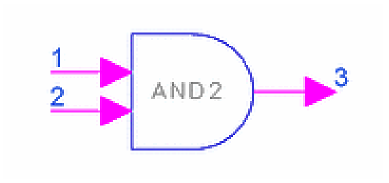
Pin	Name	Description	Signal Type
4	Result		fix

Notes/Equations

1. The add/sub control input pin is optional.
    - If the add/sub control input pin is not connected, the AddSub parameter is used to specify whether the adder adds or subtracts.
    - If the add/sub control input pin is connected: a zero value indicates add; a non-zero value indicates subtract. (The AddSub parameter is ignored in this case.)
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used to represent the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. When AddSub is used as an adder, out = A + B; when AddSub is used as a subtractor, out = A − B.
4. Bit alignment is automatic at the inputs so the two input values are added correctly. This is done by zero padding or sign extending the inputs such that their decimal points are aligned.

5. When the arithmetic type of an input to AddSyn is different from the ArithType parameter of AddSyn, then AddSyn interprets the input bit pattern in the arithmetic type specified by the ArithType parameter. For example, assume that the ArithType of AddSyn is TWOS_COMPLEMENT and that one of its inputs is 0.7 represented in unsigned arithmetic and 0.8 precision. The corresponding bit pattern is 10110011 (1 × 1/2 + 0 × 1/4 + 1 × 1/8 + 1 × 1/16 + 0 × 1/32 + 0 × 1/64 + 1 × 1/128 + 1 × 1/256 = 0.69921875).
   In two's complement this bit pattern represents a negative number since the first bit is 1. To get the magnitude of this number we first complement the bits to get 01001100 and then add 1 to get 01001101. Therefore, this bit pattern has a value of -(0 × 1/2 + 1 × 1/4 + 0 × 1/8 + 0 × 1/16 + 1 × 1/32 + 1 × 1/64 + 0 × 1/128 + 1 × 1/256 = 0.30078125), and this is the value that AddSyn will use.
   Thus, arithmetic types should not be mixed when adding or subtracting fixed-point numbers.
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

And2Syn



Description: 2-input AND
Library: Numeric, Fixed-Point DSP
Class: SDFAnd2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	Width of an input bus.	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

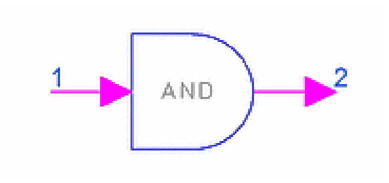
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. This model is a 2-input AND gate that takes a bitwise AND of inputs A and B (both of bitwidth Width) and outputs the results; that is, Result = A and B.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

AndSyn



Description: Bitwise AND
Library: Numeric, Fixed-Point DSP
Class: SDFAndSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of a bus segment within the input bus	8	int
Size	number of bus segments within the input bus	2	int

Pin Outputs

Pin	Name	Description	Signal Type
1	Data		fix

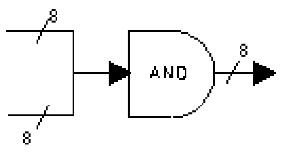
Pin Inputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. The input bus is composed of Size number of smaller bus segments. Each bus segment within the input bus is of bitwidth Width. AndSyn performs a bitwise AND of the bus segments resulting in the output Result of bitwidth Width. For example, Width = 8, Size = 2 means that the input bus is interpreted as having 2 bus segments, each of bitwidth 8. The output of AndSyn is the bitwise AND of the 2 bus segments, as illustrated below.

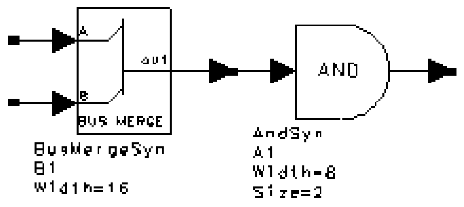
Width = 8, Size = 2



2. An example design where two 8-bit signals are ANDed together is shown below.

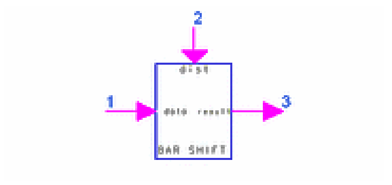
AndSyn Example Design

AndSyn Example Design



3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

BarShiftSyn



Description: Barrel Shifter
Library: Numeric, Fixed-Point DSP
Class: SDFBarShiftSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Mode	type of shifting: LOGICAL_SHIFT, ARITHMETIC_SHIFT, ROTATE_SHIFT	LOGICAL_SHIFT	enum
Direction	direction of shift in the barrel shifter: RIGHT_SHIFT, LEFT_SHIFT	LEFT_SHIFT	enum
NShift	number of bit positions to shift by	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Input data	fix
2	Dist	Dist control input for how many bits to shift by	fix

Pin Outputs

Pin	Name	Description	Signal Type
3	Result	Barrel shift result	fix

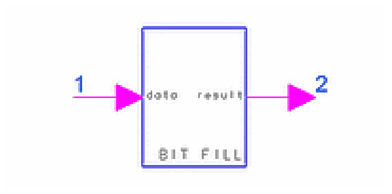
Notes/Equations

1. BarShiftSyn shifts the input bits by the amount specified by the control input Dist or (if Dist is not connected) by the integer parameter NShift. The output bit width, number of integer bits, and arithmetic type are set by the parameters of the barrel shifter.
    - Logical shifting to the right
      (Mode = LOGICAL_SHIFT, Direction = RIGHT_SHIFT)
      inserts zeros in the vacated most significant bits; logical shifting to the left
      (Mode = LOGICAL_SHIFT, Direction = LEFT_SHIFT)
      is the same as Arithmetic shifting to the left.
    - Arithmetic shifting to the right
      (Mode = ARITHMETIC_SHIFT, Direction = RIGHT_SHIFT)
      will sign extend the vacated most significant bits.
    - Rotate shifting to the right
      (Mode = ROTATE_SHIFT, Direction = RIGHT_SHIFT)

will shift the least significant bits into the vacated most significant bits.
Conversely, Rotate shifting to the left
(Mode = ROTATE_SHIFT, Direction = LEFT_SHIFT)
will shift the most significant bits into the vacated least significant bits.

2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. Direction of shifting is done assuming that the MSB is on the left and LSB is on the right. LEFT_SHIFT will shift towards the MSB. Conversely, RIGHT_SHIFT will shift towards the LSB.
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

BitFillSyn



Description: Bit Fill
Library: Numeric, Fixed-Point DSP
Class: SDFBitFillSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of output bus	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

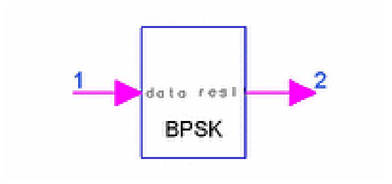
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. BitFillSyn takes the single bit input and copies it to an output bus of bitwidth Width. It replicates the single bit input value to the output bus.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

BPSKSyn



Description: BPSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFBPSKSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder output	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

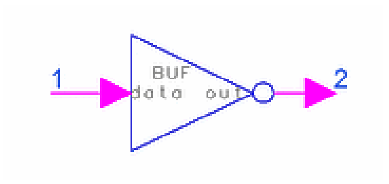
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. The output signal Result of the BPSK encoder is a twos-complement fixed-point number with 1 sign bit and (Width −1) fractional bits.
   An input bit value of 1 is mapped to the most positive-valued fixed-point number that can be represented by 1 sign bit and (Width-1) fractional bits. Conversely, an input bit value of 0 is mapped to the next-to-most negative-valued fixed-point number that can be represented by 1 sign bit and (Width-1) fractional bits. This ensures that the positive and negative valued outputs of the model have the same magnitude.
   For example, with Width = 8, mapping will be done in the following manner:
   - input bit value of 1 will be mapped to 01111111
   - input bit value of 0 will be mapped to 10000001
2. For general information regarding numeric fixed-point DSP functions, refer to the *Numeric Fixed-Point DSP Components* (numeric).

BufferSyn



Description: Buffer
Library: Numeric, Fixed-Point DSP
Class: SDFBufferSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	number of bits in input	16	int
InvMask	bit mask pattern	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

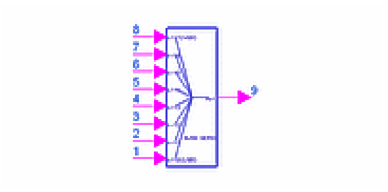
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. BufferSyn inverts the bits within the input bus based on the InvMask parameter; 1 in a bit position in InvMask will invert the corresponding bit in the input bus.
   InvMask can be specified in hex (0x prefix), octal (0 prefix),
   binary (0b prefix), or decimal (without a 0 prefix). For example, if Width = 2:
   - to invert both inputs bits, specify: InvMask = 0x3 (hex), InvMask = 03 (octal), InvMask = 0b11 (binary), InvMask = 3 (decimal).
   - to invert the LSB of the two input bits, specify: InvMask = 0x1 (hex), InvMask = 01 (octal), InvMask = 0b01 (binary), InvMask = 1 (decimal).
2. For general information regarding numeric fixed-point DSP functions, refer to the *Numeric Fixed-Point DSP Components* (numeric).

Bus8MergeSyn



Description: 8-Bit-to-Bus Merge
Library: Numeric, Fixed-Point DSP
Class: SDFBus8MergeSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	number of bits in output bus		int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data0		fix
2	Data1		fix
3	Data2		fix
4	Data3		fix
5	Data4		fix
6	Data5		fix
7	Data6		fix
8	Data7		fix

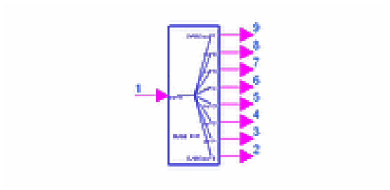
Pin Outputs

Pin	Name	Description	Signal Type
9	Output		fix

Notes/Equations

1. Bus8MergeSyn merges its eight 1-bit inputs into a bus.
2. The most significant bit in the output bus is taken from the 1-bit Data7 input pin; the next most significant bit is taken from the 1-bit Data6, and so on.
3. Width parameter specifies the size of the output bus. Input pins must be connected to the appropriate Width. For example: if Width = 1, Data7 is connected; if Width = 5, input pins Data7, Data6, Data5, Data4, and Data3 must all be connected.
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Bus8RipSyn



Description: Bus-to-8-Bit Ripper
Library: Numeric, Fixed-Point DSP
Class: SDFBus8RipSyn
Derived From: SDFHPFix

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

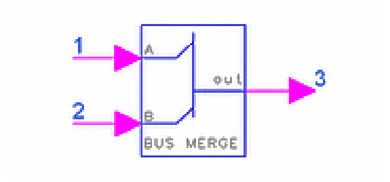
Pin Outputs

Pin	Name	Description	Signal Type
2	Output0		fix
3	Output1		fix
4	Output2		fix
5	Output3		fix
6	Output4		fix
7	Output5		fix
8	Output6		fix
9	Output7		fix

Notes/Equations

1. Bus8RipSyn rips out the highest byte in the data input bus and outputs them as 1-bit outputs.
2. The most significant bit in the data input bus is output on the pin marked Output7; correspondingly, the least significant bit in the data input bus is output on the pin marked Output0.
3. For general information regarding numeric fixed-point DSP functions, refer to the *Numeric Fixed-Point DSP Components* (numeric).

BusMergeSyn



Description: Bus Merge
Library: Numeric, Fixed-Point DSP
Class: SDFBusMergeSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bitwidth of output	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

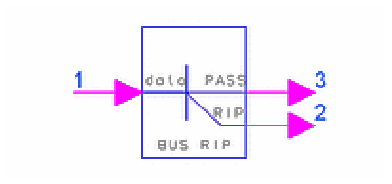
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. BusMergeSyn merges the two input buses A and B into a larger, merged bus.
   In the merged bus, A will be located in the MSB portion, while B will be located in the LSB portion.
2. The output bitwidth is specified by Width and must be equal to the sum of the two input bitwidths.
3. The output arithmetic type is always unsigned, Width number of integer bits, 0 fractional bits.
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

BusRipSyn



Description: Bus Ripper
Library: Numeric, Fixed-Point DSP
Class: SDFBusRipSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Offset	how far to right of MSB (Sign bit for TWOS_COMPLEMENT) to take ripped bit_vector	0	int
RipPrecision	precision of ripped-out segment of input bus	2.6	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

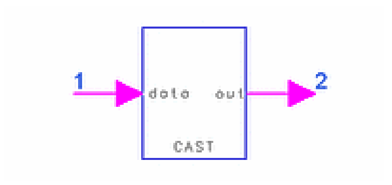
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix
3	PassThru		fix

Notes/Equations

1. BusRipSyn rips out a smaller contiguous bit vector (Fix) from the input bit vector (Fix).
2. The arithmetic type of the RIP output is the same as ArithType.
3. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

CastSyn



Description: Cast
Library: Numeric, Fixed-Point DSP
Class: SDFCastSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

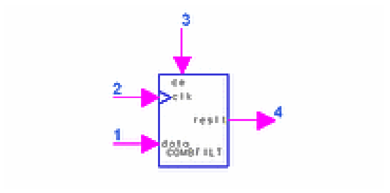
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. CastSyn copies the bits within the input bus to the output bus. It does not alter the input bits, but only changes the precision and arithmetic type associated with the input bits. The total number of output bits should be the same as the input.
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

CombFiltSyn



Description: Comb Filter
Library: Numeric, Fixed-Point DSP
Class: SDFCombFiltSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
PipeStages	Depth of pipeline, must be > 0.	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Clock	Clock input – optional control pin	fix
3	CE	Clock enable input – optional control pin	fix

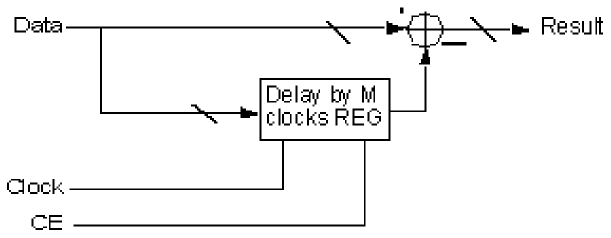
Pin Outputs

Pin	Name	Description	Signal Type
4	Result	Comb Filter output	fix

Notes/Equations

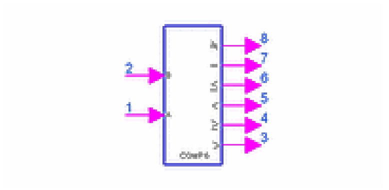
1. This model implements the transfer function of (1-z $^{-M}$) which comprises the comb section of a comb filter, where M = PipeStages. In other words, a delayed version of the input data value (PipeStages clocks previously) is subtracted from the present input data value. In discrete equation form, it can be represented as:
   ```
   Result = Data - Data(Delayed_by_M_clocks)
   ```

   Internal Structure of Comb Section Model



2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Comp6Syn



Description: Compare with 6 Outputs
Library: Numeric, Fixed-Point DSP
Class: SDFComp6Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

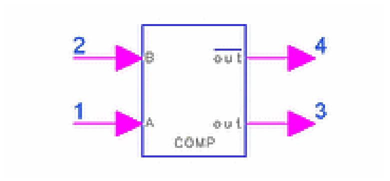
Pin Outputs

Pin	Name	Description	Signal Type
3	GT		fix
4	GE		fix
5	LT		fix
6	LE		fix
7	EQ		fix
8	NE		fix

Notes/Equations

1. Comp6Syn compares the value as represented by the two inputs and tests for six conditions. If a condition is TRUE, the output result is a 1, else 0.
2. Comparison modes are: A ≠ B, A = B, A ≤ B, A < B, A ≥ B, A > B.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

CompSyn



Description: Compare
Library: Numeric, Fixed-Point DSP
Class: SDFCompSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Mode	condition to be tested: EQUAL, LESS_OR_EQUAL, GREATER_OR_EQUAL	EQUAL	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

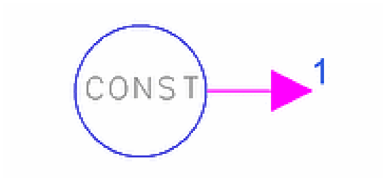
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix
4	ResultB		fix

Notes/Equations

1. CompSyn compares the value as represented by the two inputs and tests for the condition specified by Mode. If the condition is TRUE, the output *out* will go HIGH and the output $\overline{out}$ will go LOW.
2. Comparison modes are: A = B, A ≤ B, A ≥ B.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

ConstSyn



Description: Constant
Library: Numeric, Fixed-Point DSP
Class: SDFConstSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
ConstValue	constant value of device, specified as a real value	1.0	real

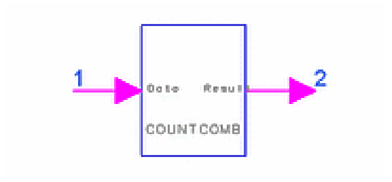
Pin Outputs

Pin	Name	Description	Signal Type
1	Result		fix

Notes/Equations

1. ConstValue is converted to the precision and type specified by OutputPrecision and ArithType.
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

CountCombSyn



Description: Counter Combinational Logic
Library: Numeric, Fixed-Point DSP
Class: SDFCountCombSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of counter	8	int
CounterType	type of counter: JOHNSON_CTR, LFSR_CTR, GRAY_CTR	JOHNSON_CTR	enum
LFSR_Poly	LFSR polynomial to be used in LFSR counter	0xff	string

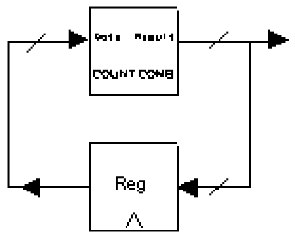
Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

Pin Outputs

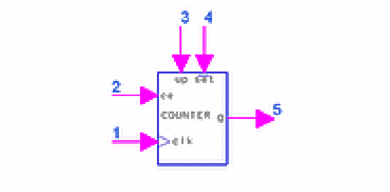
Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. CountCombSyn models the combinational logic portion of a Johnson, LFSR (linear feedback shift register), or Gray counter. Usage is illustrated.


2. LFSR_Poly sets the LFSR polynomial to be used when CounterType = LFSR_CTR. It is specified as a hex string; for example, LFSR_Poly = 0xFE.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

CounterSyn



Description: Binary Counter
Library: Numeric, Fixed-Point DSP
Class: SDFCounterSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of binary counter	16	int
ValueS	value of counter when Set is asserted (low)	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Clock	Clock signal – if connected, counter is positive edge triggered on clock transitions.	fix
2	CE	Clock Enable signal – if connected and asserted (high) enables counter when asserted (high).	fix
3	Up	Up/Down control signal – if connected and asserted (high) counter counts up.	fix
4	Set	Set/Reset control signal – if connected and asserted (low) counter resets	fix

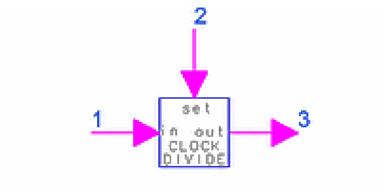
Pin Outputs

Pin	Name	Description	Signal Type
5	Q	Counter output signal – parallel data.	fix

Notes/Equations

1. The Binary Counter is positive-edge clock triggered when the CE pin is asserted (high).
2. The control pins are optional-these do not have to be connected.
3. ValueS can be specified in hex (0x prefix), octal (0 prefix), binary (0b prefix), or decimal (without a 0 prefix).
   For example, to specify a ValueS of decimal value 31, set
   ValueS = 31 (decimal), ValueS = 0x1F (hex), ValueS = 037 (octal), or ValueS = 0b11111 (binary).
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Div2ClockSyn



Description: Power-of-2 Clock Divider
Library: Numeric, Fixed-Point DSP
Class: SDFDiv2ClockSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
DivideBy	Value to divide input Clock by.: TWO, FOUR, EIGHT, SIXTEEN	TWO	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	InClock	Clock input	fix
2	Set	Asynchronous set/reset input – optional control pin	fix

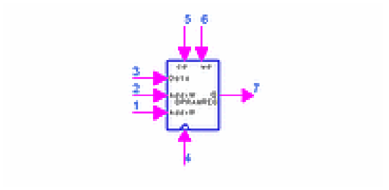
Pin Outputs

Pin	Name	Description	Signal Type
3	DivClock	Clock output	fix

Notes/Equations

1. This model is a divide-by-power-of-2 clock divider; options are to divide by 2, 4, 8, or 16.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

DPRamRegSyn



Description: Registered Dual-Port RAM
Library: Numeric, Fixed-Point DSP
Class: SDFDPRamRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Depth	Number of words in RAM	16	int
ramFile	File containing initial RAM values		filename
ramFileFormat	Format of RAM init file.: REAL, HEX	HEX	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	AddrR	input read address	fix
2	AddrW	input write address	fix
3	Data	input data	fix
4	Clock	Clock input – optional control pin	fix
5	CE	Clock enable input – optional control pin	fix
6	WE	write enable input: if low, then the input data is written to the RAM location specified by AddrW.	fix

Pin Outputs

Pin	Name	Description	Signal Type
7	Q	output data	fix

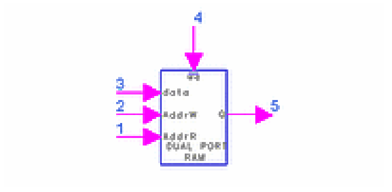
Notes/Equations

1. This model implements a dual-port RAM with a registered output. Given an input address in AddrW (write address), and data in Data, the model will write the input Data into an internal array if WE is asserted by a low value. If WE is not asserted, then the model will not write Data into the address location as specified in AddrW. The input address in AddrR (read address) is used to read out the data in the dual-port RAM model, which is sent to the output Q.
2. The output of the dual-port RAM is registered with a positive edge Clock input. The clock enable CE control input is optional:

- if it is not connected, the model is always enabled
- if it is connected, it is enabled by a high value in CE

3. The initial values in the dual-port RAM can be defined in the (optional) file as specified in the ramFile parameter. The format of the file is specified by the ramFileFormat parameter; the initial values can be specified as REAL or HEX. The address of each initial data read into the model is the same as the line number of the corresponding data read from the initialization file.
   The initial values are specified as a column of values as in the following examples.
   - if ramFileFormat = REAL, which specifies that the RAM initialization file contains real values, an example of such a file would be:
     0.98
     0.24
     0.12
     .
     .
     .
     From this example, the model will interpret the first line as address 0 with data equal to the fixed-point value corresponding to 0.98, and so on. Note that the model will convert the real values to its fixed-point representation using the specified precision in the OutputPrecision parameter, and arithmetic type as specified in the ArithType parameter.
   - if ramFileFormat = HEX, an example of such a file would be:
     0x7f
     0x06
     0x08
     .
     .
     .
     From this example, the model will interpret the first line as address 0 with data equal to 0x7f, and so on.
4. The Depth parameter specifies the number of words in the dual-port RAM.
5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

DPRamSyn



Description: Dual-Port RAM
Library: Numeric, Fixed-Point DSP
Class: SDFDPRamSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Depth	size of (number of words in) RAM	16	int
ramFile	name of file containing initial RAM values (optional) (represented in hex data format in file)		filename

Pin Inputs

Pin	Name	Description	Signal Type
1	AddrR	input read address	fix
2	AddrW	input write address	fix
3	Data	input data	fix
4	WE	write enable input: if low, then the input data is written to the RAM location specified by AddrW.	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	Q	output data	fix

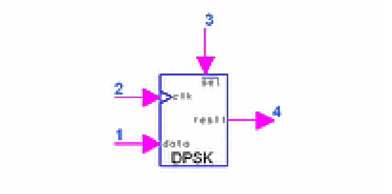
Notes/Equations

1. DPRamSyn models a dual-port RAM. Data in the RAM can be initialized by specifying the file name in the ramFile parameter.
2. The path name for ramFile can be specified in several ways: one is to just specify the file name, for example ramFile = *foo*, which is assumed to be located within the current workspace data directory; another is by specifying the absolute path, as in ramFile = */usr/user_name/foo*; or, the environmental variables can also be used to set the file path name, for example ramFile = *$ENV_FOO/foo*, where *ENV_FOO* is an environmental variable.
3. The bitwidths and arithmetic type of the output data are defined by the device parameters. The size of the RAM is specified by the Depth parameter. An example file format is:

```
0x01
0xff
0xca
.
.
.
```
and so on.

4. The data format in the file is assumed to be right-justified.
5. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

DPSKSyn



Description: Differential BPSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFDPSKSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder outputs	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix
2	Clock	Clock input – optional control pin	fix
3	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
4	Result		fix

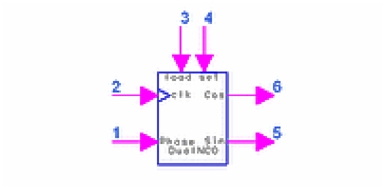
Notes/Equations

1. The output signal Result of the DPSK encoder is a twos-complement fixed-point number with 1 sign bit and (Width-1) fractional bits.
   The 1-bit input data is clocked (positive edge triggered) into a 2-deep FIFO buffer. Values of the 2-deep FIFO buffer are XORed together to get the differential output bit result. A resulting bit value of 1 (after the XOR operation on the 2 data bits in the FIFO buffer) is mapped to the most positive-valued fixed-point number that can be represented by 1 sign bit and (Width-1) fractional bits. Conversely, a resulting bit value of 0 is mapped to the next-to-most negative-valued fixed-point number that can be represented by 1 sign bit and (Width-1) fractional bits.
   This ensures that the positive and negative valued outputs of the model have the same magnitude.
   Assertion of the Set input (a low value, i.e. 0) will clear the values of the FIFO buffers.
   For example, with Width = 8, with an input bit sequence of 0 1 (with 0 being older, and 1 being the most recent), and assuming that initially the encoder is reset, the following will result:
   - first input bit 0 will result in the XOR output of 0 ^{0} = 0, which maps to 10000001
   - second input bit 1 will result in the XOR output of 1 ^{0} = 1, which maps to

```
01111111
```

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

DualNCOSyn



Description: Dual Channel Numerically Controlled Oscillator
Library: Numeric, Fixed-Point DSP
Class: SDFDualNCOSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
SetType	Mode for Set/Reset control input.: ASYNCHRONOUS, SYNCHRONOUS, SET_PIN_NOTUSED	ASYNCHRONOUS	enum
OutWidth	Output width of NCO.	10	int
PhaseAccWidth	Width of phase accumulator in NCO.	16	int
PhaseWidth	Number of bits used from phase accumulator for sine/cosine table.	8	int
PhaseIncrWidth	Width of phase increment input.	10	int

Pin Inputs

Pin	Name	Description	Signal Type
1	PhaseIncr		fix
2	Clock	Clock input – optional control pin	fix
3	Load	Load control input – optional control pin	fix
4	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	SineOut		fix
6	CosineOut		fix

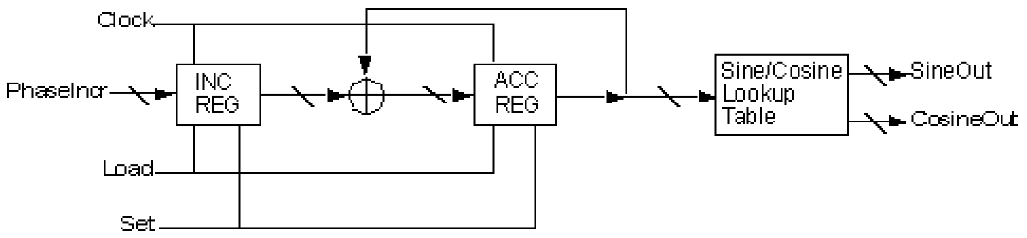
Notes/Equations

1. This model implements a dual-output numerically controlled oscillator (NCO). Given a phase increment PhaseIncr input value, it outputs sine and cosine fixed-point signals (1 sign bit, (OutWidth-1) fractional bits twos-complement) with a frequency proportional to the value of the PhaseIncr input.
2. When the Load input is asserted by bringing it high (a value of 1), the PhaseIncr input data is loaded into an internal phase increment register in the NCO model. The input phase increment value in PhaseIncr is interpreted within the model as an unsigned fixed-point number (with PhaseIncrWidth integer bits, and no fractional bits).
3. The model contains a phase accumulator (of bitwidth PhaseAccWidth) that adds the value in the phase increment register to the previous phase accumulator value. The

result of the phase accumulator (actually the most significant PhaseWidth bits of the phase accumulator) is used as an index to a sine/cosine look-up table that outputs the sine and cosine values corresponding to the current phase accumulator value. The output sine and cosine signals SineOut, CosineOut are represented as twos-complement, 1-sign bit, (OutWidth −1) fractional bits, fixed-point numbers.

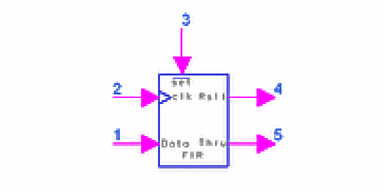
4. Assertion of the Reset input by bringing it low (a value of 0) will clear the NCO phase increment register and the phase accumulator.

Internal Structure of Dual-Output NCO Model



5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

FIRSyn



Description: General Finite Impulse Response (FIR) Filter
Library: Numeric, Fixed-Point DSP
Class: SDFFIRSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
NumOfTaps	Number of taps in FIR filter.	1	int
CoefPrecision	Precision of the coefficients in the coefficient file.	2.14	precision
DataPrecision	Precision of the DataFeedThru output (used in cascading FIR filters).	2.14	precision
CoefFile	File containing FIR coefficient values.		filename
CoefFileFormat	Format of FIR Coefficients file.: REAL, HEX	HEX	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	DataIn	Data input	fix
2	Clock	Clock input – optional control pin	fix
3	Set	Asynchronous set/reset input – optional control pin	fix

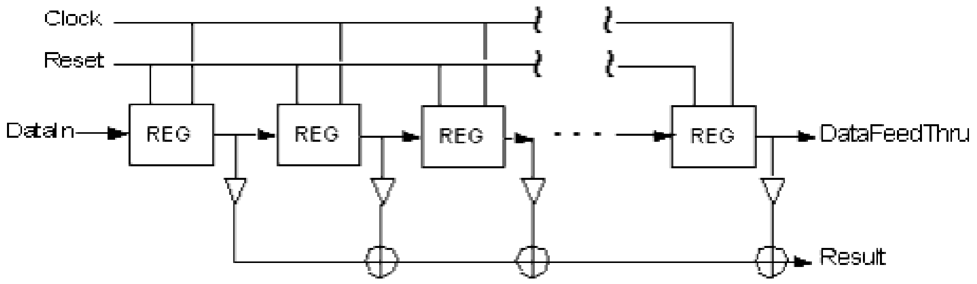
Pin Outputs

Pin	Name	Description	Signal Type
4	Result	FIR result output (with precision OutputPrecision)	fix
5	DataFeedThru	Data output (with precision DataPrecision = precision of DataIn input)	fix

Notes/Equations

1. This model is a FIR (finite impulse response) filter model. It implements a general parallel FIR structure and retains full precision internally when computing filter output values. The only quantization done is at the Result output.

Internal Structure of FIR Model

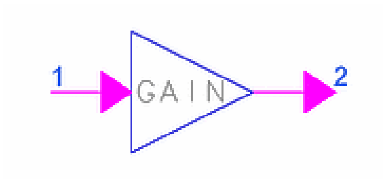


2. The Result output of the FIR model is the final result of the FIR filtering done within the model, and quantized to the precision specified by OutputPrecision.
3. Data from the DataIn input is clocked into the internal data registers of the FIR model upon the positive edge transitions of the Clock input if the Clock pin is connected. If the Clock pin is not connected, data is shifted into the internal data registers at every sample step in the simulator.
4. The 1-bit Reset input pin is asserted by bring it low (value of 1), which will clear all internal data registers.
5. The DataFeedThru output of the FIR model is the output of the oldest data in the internal data registers.
   The designer can use this output to feed the next stage of a FIR filter model in order to create a cascade of FIR filter models. By cascading sections of FIR cores, the designer can build a larger order FIR filter than the maximum for just one FIR core.
6. The filter tap coefficients of the FIR filter are defined in the file as specified in the CoefFile parameter. The format of the file is specified by the CoefFileFormat parameter; tap coefficients can be specified as REAL or HEX values. The tap coefficients are specified as a column of values in the file. The 0th tap filter coefficient is the value on the first line of the filter tap coefficient file, the 1th tap filter coefficient corresponds to the value on the second line, the 2th tap filter coefficient corresponds to the value on the third line, and so on.
   Consider the following examples:
   - If CoefFileFormat = REAL, which specifies that the filter tap coefficient file contains real values for the filter tap coefficients, an example of such a file would be:
     0.98
     0.24
     0.12
     0.05
     -0.13
     0.21
     .
     .
     .
   - If CoefFileFormat = HEX which specifies that the filter tap coefficient file contains hex values for the filter tap coefficients, an example of such a file would be:
     0x7f
     0x06
     0x02
     0x8f
     0x07
     0x08
     .
     .

.

7. The NumOfTaps parameter specifies the number of tap coefficients to be read from the file specified by CoefFile.
    - If NumOfTaps is assigned a value that is less than the taps value provided in CoefFile, only the first NumOfTaps coefficients will be picked from the file.
    - If NumOfTaps is greater than the taps provided, the rest of the taps will be padded with 0.
8. The CoefPrecision parameter specifies the precision of the filter tap coefficients, that is, the number of integer bits (including the sign bit) and the number of fractional bits to be used to represent the filter tap coefficients.
9. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

FixedGainSyn



Description: Fixed Gain
Library: Numeric, Fixed-Point DSP
Class: SDFFixedGainSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum
Gain	Gain of device specified as a real value.	1.0	real
GainPrecision	Precision of the gain parameter.	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

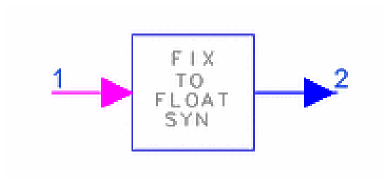
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. FixedGainSyn models a gain block that multiplies the input value by the specified Gain (quantized by GainPrecision) and outputs the result at the specified OutputPrecision.
2. OutputPrecision specifies the fixed-point precision format of the output: if OutputPrecision = 1.15, 1 bit is used to represent the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

FixToFloatSyn



Description: Fixed-Point to Floating-Point
Library: Numeric, Fixed-Point DSP
Class: SDFFixToFloatSyn
Derived From: SDFHPFix

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Input fix type	fix

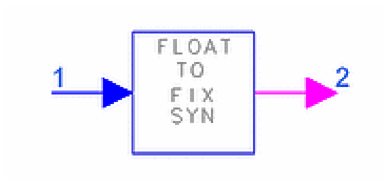
Pin Outputs

Pin	Name	Description	Signal Type
2	Result	Output float type	real

Notes/Equations

1. FixToFloatSyn converts a fixed-point input to a floating-point (real) output.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

FloatToFixSyn



Description: Floating-Point to Fixed-Point
Library: Numeric, Fixed-Point DSP
Class: SDFFloatToFixSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Input float type	real

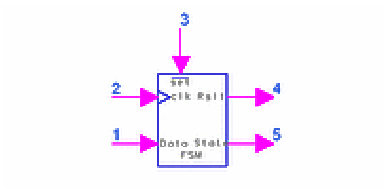
Pin Outputs

Pin	Name	Description	Signal Type
2	Result	Output fix type	fix

Notes/Equations

1. FloatToFixSyn converts a floating-point (real) input to a fixed-point output. It quantizes by rounding and it saturates upon overflow.
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

FSMSyn



Description: Mealy Finite State Machine (FSM)
Library: Numeric, Fixed-Point DSP
Class: SDFFSMSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
fsmFile	File containing Mealy FSM definition	user_defined.fsm	filename
InputWidth	Bit width of data input of Mealy FSM	1	int
StateWidth	Bit width of state register of Mealy FSM	1	int
OutputWidth	Bit width of output of Mealy FSM	1	int
fsmFileFormat	Format of Mealy FSM definition file: HEX, OCTAL, DECIMAL	HEX	enum
Depth	Number of row entries in FSM definition file	1	int
ResetStateVal	Reset State Value	0	int
DefaultStateVal	Default State Value	0	int
DefaultOutVal	Default Output Value	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix
2	Clock	Clock input – optional control pin	fix
3	Reset	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
4	Result		fix
5	OutState		fix

Notes/Equations

1. This model implements a Mealy finite state machine. The state transitions and output values of the Mealy FSM are defined in the file specified in the fsmFile parameter. The format of the entries within the Mealy FSM definition file can be hex (0x01FE, for example), octal (016, for example), or decimal (230, for example).
   Each line in the file contains the following entries separated by at least a space: the first entry is the input data value; the second entry is the present state value; the third entry is the next state value; the final entry is the output value. Thus, each line

in the FSM definition file should look like:

input_data present_state next_state output

Consider the example of a Mealy FSM definition file entries:

0x01 0x00 0x01 0x1

0x00 0x00 0x00 0x0

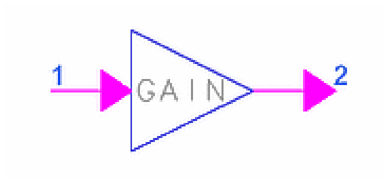
0x01 0x01 0x02 0x0

0x00 0x01 0x01 0x1

- The first line in the example file specifies that given an input of 0x01, and a present state of 0x00, the next state of the FSM will be 0x01, and the output is 0x1.
- The second line specifies that given an input of 0x00, and a present state of 0x00, the next state of the FSM will be 0x00, and the output is 0x0.
- The third line specifies that given an input of 0x01, and a present state of 0x01, the next state of the FSM will be 0x02, and the output is 0x0. It should be clear how the definition file is interpreted by the model from this example.
  Any input and state combinations that are not covered by the Mealy FSM definition file will be covered by the default state and output values as specified in model parameters DefaultStateVal and DefaultOutVal.
  The state of the Mealy FSM can be initialized to a known reset state by asserting the Reset input (by giving it a low value of 0) which will set the state of the Mealy FSM to the value specified in model parameter ResetStateVal.
  The values for parameters DefaultStateVal, DefaultOutVal, and ResetStateVal can be specified in decimal form (for example, DefaultOutVal = 15), or in hex form (for example, DefaultStateVal = 0x001).

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

GainSyn



Description: Gain
Library: Numeric, Fixed-Point DSP
Class: SDFGainSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum
Gain	gain of device specified as a real value. It is converted to the precision of GainPrecision of ArithType arithmetic	1.0	real
GainPrecision	precision of gain in bits and precision of accumulation. When the gain value extends outside of the precision, the overflow type is called	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

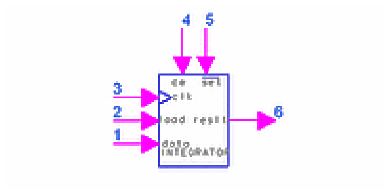
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. GainSyn models a gain block that multiplies the input value by the specified Gain (quantized by GainPrecision) and outputs the result at the specified OutputPrecision.
2. OutputPrecision specifies the fixed-point precision format of the output: if OutputPrecision = 1.15, 1 bit is used to represent the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

IntegratorSyn



Description: Integrator
Library: Numeric, Fixed-Point DSP
Class: SDFIntegratorSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input – Data input which is loaded by asserting Load input	fix
2	Load	Load input – loads Data into accumulator of integrator	fix
3	Clock	Clock input – optional control pin	fix
4	CE	Clock enable input – optional control pin	fix
5	Set	Asynchronous set/reset input – optional control pin	fix

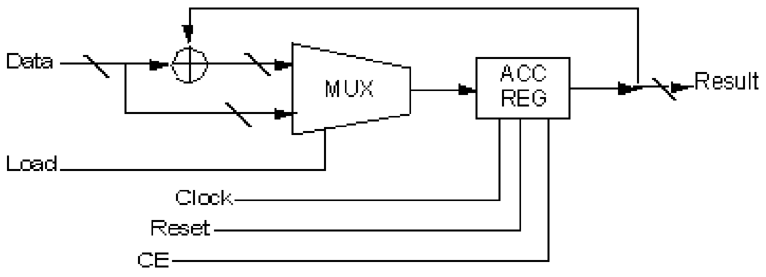
Pin Outputs

Pin	Name	Description	Signal Type
6	Result		fix

Notes/Equations

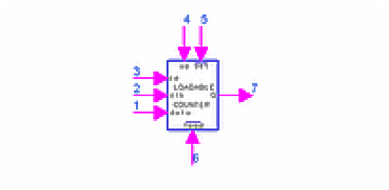
1. This model is a first order integrator. It has a transfer function of $(1- z^{-1})^{-1}$ where z^{-1} refers to a unit Clock delay. Physically, the model can be viewed as an adder that adds the present input Data to the previous output of the adder. The delayed adder output feedback is achieved by using an internal data register that is clocked by the positive edge transitions of the Clock 1-bit. In discrete equation form, the equation defining the model is:
   Result = Previous_Result + Data

   Internal Structure of Integrator Model



2. The Clock input is optional.
    - if it is connected, the model will operate based on the positive edge transitions of the Clock input.
    - if it is not connected, the model will operate as if every sample step of the simulator is a positive edge transition.
3. Assertion of the Reset input by bringing it low (a value of 0) will clear the internal data register.
4. The (optional) CE input is the clock-enable control for the internal data register.
    - if it is connected and has a high value (a value of 1), then the internal data register is enabled and will load its input upon a positive Clock edge.
    - if it is not connected, and low (a value of 0) then the clock to the internal data register is disabled. The internal data register is always enabled when the CE input is not connected.
5. The (optional) Load input is asserted by bring it high (a value of 1).
    - if it is asserted, the Data input is loaded into the internal data register.
    - if it is unconnected, the Load is never asserted.
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

LCounterSyn



Description: Loadable Binary Counter
Library: Numeric, Fixed-Point DSP
Class: SDFLCounterSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of binary counter	16	int
ValueS	value to which the counter is set when Set is asserted (high)	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Input data signal	fix
2	Clock	Clock signal – if connected, counter is positive edge triggered on clock transitions.	fix
3	CE	Clock Enable signal – if connected and asserted (high) enables counter when asserted (high).	fix
4	Up	Up/Down control signal – if connected and asserted (high) counter counts up.	fix
5	Set	Set/Reset control signal – if connected and asserted (low) counter resets	fix
6	Load	Load control signal – if connected and asserted (low) counter loads Data input.	fix

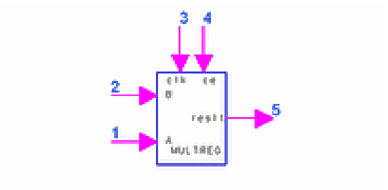
Pin Outputs

Pin	Name	Description	Signal Type
7	Q	Counter output signal – parallel data.	fix

Notes/Equations

1. LCounterSyn is positive-edge clock triggered when the count enabled pin is asserted (high).
2. The control pins are optional-these do not have to be connected.
3. ValueS can be specified in hex (0x prefix), octal (0 prefix), binary (0b prefix), or decimal (without a 0 prefix). For example, to specify a ValueS of decimal value 31, set ValueS = 31 (decimal), ValueS = 0x1F (hex), ValueS = 037 (octal), or ValueS = 0b11111 (binary).
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

MultRegSyn



Description: Registered Multiplier
Library: Numeric, Fixed-Point DSP
Class: SDFMultRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum
Latency	Latency in clock cycles for multiplier result.	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A	input A	fix
2	B	input B	fix
3	Clock	Clock input − optional control pin	fix
4	CE	Clock enable input − optional control pin	fix

Pin Outputs

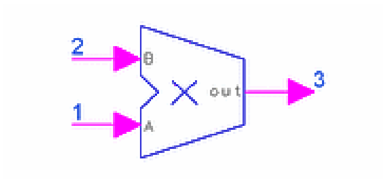
Pin	Name	Description	Signal Type
5	Result	Registered multiplier output	fix

Notes/Equations

1. This model is a registered adder. It calculates the multiplication of its A and B data inputs (A × B) and registers its output Result such that it has the specified precision as set in the OutputPrecision parameter.
2. The Clock input is optional:
   - if it is connected, the model will operate based on the positive edge transitions of the Clock input
   - if it is not connected, the model will operate as if every sample step of the simulator is a positive edge transition.
3. Assertion of the Reset input by bringing it low (a value of 0) will clear the output data register.

4. The (optional) CE input is the clock-enable control for the output data register.
    - if it is connected and has a high value (a value of 1), the output data register is enabled and will load the addition result upon a positive Clock edge.
    - if it is connected and low (a value of 0) the clock to the output data register is disabled.
    - if the CE input is not connected, the output data register is always enabled.
5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

MultSyn



Description: Multiplier
Library: Numeric, Fixed-Point DSP
Class: SDFMultSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

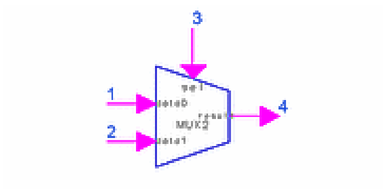
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. MultSyn multiplies two data inputs.
2. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Mux2Syn



Description: 2-input Multiplexer
Library: Numeric, Fixed-Point DSP
Class: SDFMux2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	Width of an input bus.	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data0		fix
2	Data1		fix
3	Sel		fix

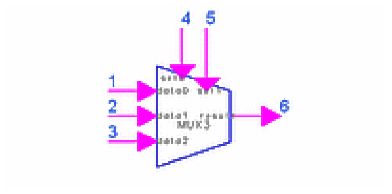
Pin Outputs

Pin	Name	Description	Signal Type
4	Result		fix

Notes/Equations

1. This model is a 2-input multiplexer. It selects input Data0 or Data1 depending on the value of its Sel input. If the Sel input value is 0 (low value), Data0 is assigned to its output Result; if the Sel input value is 1 (high value), Data1 is assigned to its output Result.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Mux3Syn



Description: 3-input Multiplexer
Library: Numeric, Fixed-Point DSP
Class: SDFMux3Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	Width of an input bus.	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data0		fix
2	Data1		fix
3	Data2		fix
4	Sel0		fix
5	Sel1		fix

Pin Outputs

Pin	Name	Description	Signal Type
6	Result		fix

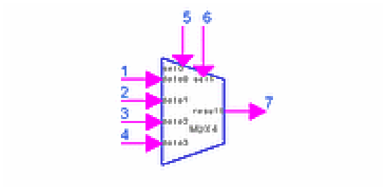
Notes/Equations

1. This model is a 3-input multiplexer. It selects one of 3 inputs Data0, or Data1 or Data2 depending on the value of its Sel0 and Sel1 inputs given in *Data Selection*,

Sel1	Sel0	Result
0	0	Data0
0	1	Data1
1	0	Data2
1	1	invalid input

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Mux4Syn



Description: 4-input Multiplexer
Library: Numeric, Fixed-Point DSP
Class: SDFMux4Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	Width of an input bus.	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data0		fix
2	Data1		fix
3	Data2		fix
4	Data3		fix
5	Sel0		fix
6	Sel1		fix

Pin Outputs

Pin	Name	Description	Signal Type
7	Result		fix

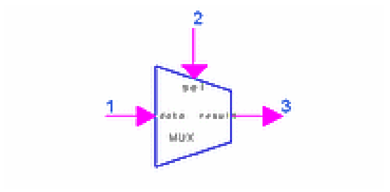
Notes/Equations

1. This model is a 4-input MUX; it selects input Data0, Data1, Data2, or Data3 based on the values of inputs Sel0 and Sel1 given in *Data Selection*.

Sel1	Sel0	Result
0	0	Data0
0	1	Data1
1	0	Data2
1	1	Data3

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

MuxSyn



Description: Mux
Library: Numeric, Fixed-Point DSP
Class: SDFMuxSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within the input bus	8	int
Size	number of bus segments within the input bus	2	int
WidthS	bit width of select control input	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix
2	Sel		fix

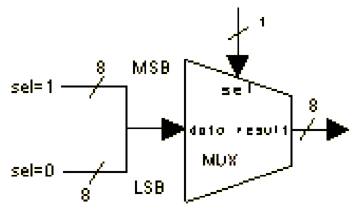
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

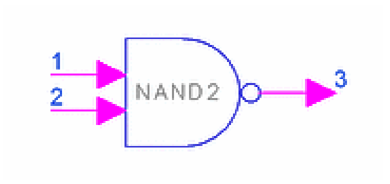
1. The input bus is composed of Size number of smaller bus segments. Each bus segment within the input bus is of bitwidth Width. MuxSyn selects one of the Size bus segments and outputs it as result. The sel input is used to control which bus segment is selected. A value of 0 in sel will select the least significant bus segment; a value of 1 will select the next-to-least-significant bus segment, and so on.

Width = 8, Size = 2, WidthS = 1



2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Nand2Syn



Description: 2-input NAND
Library: Numeric, Fixed-Point DSP
Class: SDFNand2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within the input bus	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

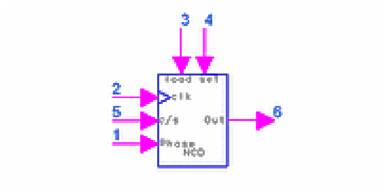
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. This model is a 2-input NAND gate, which takes a bitwise NAND of inputs A and B (both of bitwidth Width) and outputs the results, that is, Result = A NAND B.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

NCOSyn



Description: Numerically Controlled Oscillator
Library: Numeric, Fixed-Point DSP
Class: SDFNCOSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
SetType	Mode for Set/Reset control input.: ASYNCHRONOUS, SYNCHRONOUS, SET_PIN_NOTUSED	ASYNCHRONOUS	enum
OutWidth	Output width of NCO.	10	int
PhaseAccWidth	Width of phase accumulator in NCO.	16	int
PhaseWidth	Number of bits used from phase accumulator for sine/cosine table.	8	int
PhaseIncrWidth	Width of phase increment input.	10	int

Pin Inputs

Pin	Name	Description	Signal Type
1	PhaseIncr		fix
2	Clock	Clock input − optional control pin	fix
3	Load	Load control input − optional control pin	fix
4	Set	Asynchronous set/reset input − optional control pin	fix
5	SineOrCosine	SineOrCosine − controls whether sine or cosine is output	fix

Pin Outputs

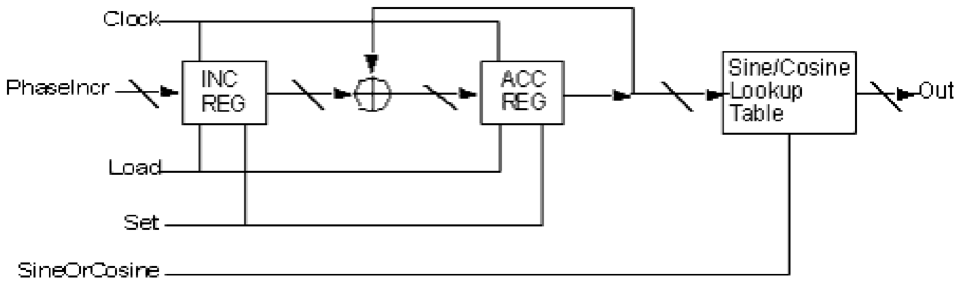
Pin	Name	Description	Signal Type
6	Out		fix

Notes/Equations

1. This model implements an Numerically Controlled Oscillator (NCO). Given a phase increment PhaseIncr input value, it outputs a sine or cosine fixed-point signal (1 sign bit, (OutWidth-1) fractional bits twos-complement) with a frequency proportional to the value of the PhaseIncr input.
   When the Load input is asserted by bring it high (a value of 1), the PhaseIncr input data is loaded into an internal phase increment register in the NCO model. The input phase increment value in PhaseIncr is interpreted within the model as an unsigned fixed-point number (with PhaseIncrWidth integer bits, and no fractional bits).
   The model contains a phase accumulator (of bitwidth PhaseAccWidth) which adds the value in the phase increment register to the previous phase accumulator value. The

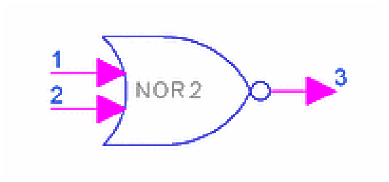
result of the phase accumulator (actually the most significant PhaseWidth bits of the phase accumulator) is used as an index to a sine/cosine look-up table that outputs a sine or cosine value corresponding to the current phase accumulator value.

Internal Structure of NCO model



2. The output sine or cosine signal in Out is represented as a twos-complement, 1-sign bit, (OutWidth-1) fractional bits, fixed-point number.
3. The 1-bit control input SineOrCosine is optional. It is used to determine whether a sine or cosine signal is evaluated by the model.
   - if the SineOrCosine pin is not connected, the default output of the model is a sine signal.
   - if the SineOrCosine pin is connected: a low value (corresponding to 0) will cause the model to output a cosine signal; conversely, a high value (corresponding to 1) will cause the model to output a sine signal.
4. Assertion of the Reset input by bringing it low (a value of 0) will clear the NCO phase increment register and the phase accumulator.
5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Nor2Syn



Description: 2-input NOR
Library: Numeric, Fixed-Point DSP
Class: SDFNor2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within the input bus	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

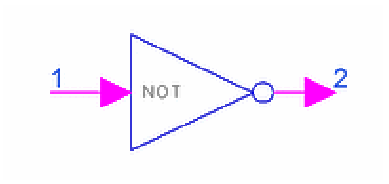
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. This model is a 2-input NOR gate. It takes a bitwise NOR of inputs A and B, (both of bitwidth Width) and outputs the results, that is, Result = A NOR B.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

NotSyn



Description: NOT
Library: Numeric, Fixed-Point DSP
Class: SDFNotSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within the input bus	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

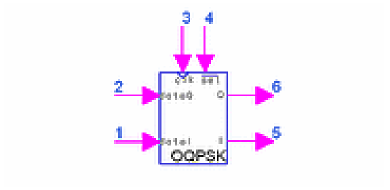
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

1. This model is a NOT gate. It takes a bitwise NOT of input Data and outputs the results, that is, Result = NOT(Data).
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

OQPSKSyn



Description: Offset QPSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFOQPSKSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder outputs	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	DataI		fix
2	DataQ		fix
3	Clock		fix
4	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	Iout		fix
6	Qout		fix

Notes/Equations

1. The output signals of the OQPSK encoder are 2 twos-complement fixed-point numbers with 1 sign bit and (Width −1) Iout and Qout fractional bits.
   The In-phase data input DataI is clocked into an internal register (in the model we will call *dataireg*) on the positive Clock edge, while the Quadrature-phase data input DataQ is clocked into its internal register (in the model we will call *dataqreg*) on the negative Clock edge (that is, a half symbol time later).
   Assertion of the Set input (a low value, that is, 0) will clear the values of the internal data registers.
   For each *dataireg* or *dataqreg* bit value of 1, a mapping to the fixed-point number (represented by a 1 sign bit and (Width −1) fractional bits) closest to the negative value of the square root of 1/2 (that is, −0.7071067811..) is done. Conversely, for each *dataireg* or *dataqreg* bit value of 0, a mapping to the fixed point number (represented by a 1 sign bit and (Width −1) fractional bits) closest to the square root of 1/2 (that is, +0.7071067811..) is done.
   For example, with Width = 8, mapping will be done in the following manner.

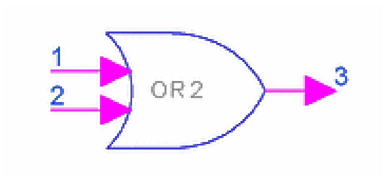
dataireg	**dataqreg**	**-->**	**Output Iout**	**Output Qout**
0	0	-->	01011011	01011011
0	1	-->	01011011	10100101
1	0	-->	10100101	01011011
1	1	-->	10100101	10100101

Note that, with 1 sign bit and 7 fractional bits twos-complement:

- 01011011 corresponds to 0.7109375
- 10100101 corresponds to −0.7109375

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Or2Syn



Description: 2-input OR
Library: Numeric, Fixed-Point DSP
Class: SDFOr2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within the input bus	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

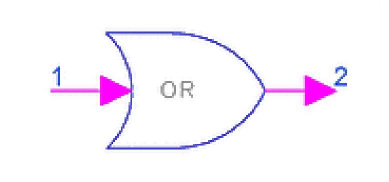
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. This model is a 2-input OR gate. It takes a bitwise OR of its inputs A and B (both of bitwidth Width) and outputs the results, that is, Result = A OR B.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

OrSyn



Description: Bitwise OR
Library: Numeric, Fixed-Point DSP
Class: SDFOrSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of bus segment within input bus	8	int
Size	number of bus segments within input bus	2	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

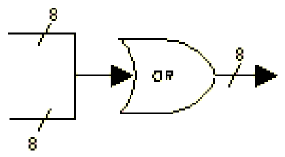
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

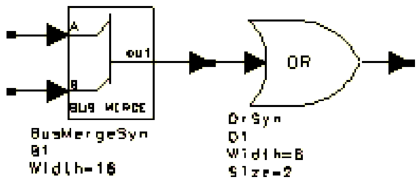
1. The input bus is composed of Size number of smaller bus segments. Each bus segment within the input bus is of bitwidth Width. OrSyn performs a bitwise OR of the bus segments resulting in the output result of bitwidth Width. For example, if Width = 8, Size = 2 means that the input bus is interpreted as having 2 bus segments, each of bitwidth 8. The output of OrSyn is the bitwise OR of the 2 bus segments, as illustrated below.

Width = 8, Size = 2



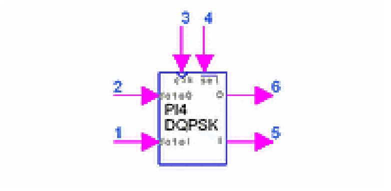
2. An example design where two 8-bit signals are ORed together is shown below.

OrSyn Example Design



3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

PI4DQPSKSyn



Description: Pi/4 DQPSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFPI4DQPSKSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder outputs	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	DataI		fix
2	DataQ		fix
3	Clock		fix
4	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	Iout		fix
6	Qout		fix

Notes/Equations

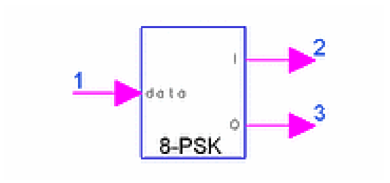
1. The 2 output signals of the /4-DQPSK encoder are twos-complement fixed- point numbers with 1 sign bit and (Width-1) fractional bits Iout and Qout.
   In-phase and Quadrature-phase data inputs DataI, DataQ are clocked into internal registers on the positive Clock edge. Outputs Iout and Qout are rotated in phase increments that are multiples of /4 (that is, multiples of 45 degrees) depending on the values of DataI and DataQ. Phase rotations are specified in *Phase Rotations*.

Input DataI	Input DataQ	Rotate (Iout, Qout) by
0	0	+Π/4 (+45 deg)
0	1	−Π/4 (−45 deg)
1	0	+3Π/4 (+135 deg)
1	1	−3Π/4 (-135 deg)

   Assertion of the Set input (a low value, i.e. 0) will clear the values of the internal registers of the model and the outputs (Iout, Qout) are set to the fixed point numbers closest to the value of (sqrt(1/2), sqrt(1/2)), where sqrt(1/2) denotes the square root of 1/2 (as close as can be represented by 1 sign bit and (Width-1) fractional bits in twos-complement).

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

PSK8Syn



Description: 8-PSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFPSK8Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder outputs	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

Pin Outputs

Pin	Name	Description	Signal Type
2	Iout		fix
3	Qout		fix

Notes/Equations

1. Output signals of the 8PSK encoder are 2 twos-complement fixed-point numbers with 1 sign bit and (Width-1) fractional bits, Iout and Qout. The 3-bit input Data is mapped to the Iout and Qout outputs according to the Data Mapping table below.

Data Mapping

Input Data	Iout (real-value)	Qout (real-value)
000	value closest to +sqrt(1/2)	value closest to +sqrt(1/2)
001	0.0	1.0 - 2 $^{-(\text{Width}-1)}$
010	-sqrt(1/2)	+sqrt(1/2)
011	-1.0 + 2 $^{-(\text{Width}-1)}$	0.0
100	-sqrt(1/2)	-sqrt(1/2)
101	0.0	-1.0 + 2 $^{-(\text{Width}-1)}$
110	+sqrt(1/2)	-sqrt(1/2)
111	1.0 - 2 $^{-(\text{Width}-1)}$	0.0

For example, with Width = 8, mapping will be done in the following manner:

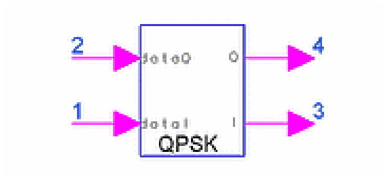
Input Data	**Iout (twos-compliment binary)**	**Qout (twos-compliment binary)**
000	01011011	01011011
001	00000000	01111111
010	10100101	01011011
011	10000001	00000000
100	10100101	10100101
101	00000000	10000001
110	01011011	10100101
111	01111111	00000000

Note that, with 1 sign bit and 7 fractional bits twos-complement:

- 01011011 corresponds to 0.7109375
- 10100101 corresponds to -0.7109375
- 01111111 corresponds to 1.0- 2 $^{-7}$
- 10000001 corresponds to -1.0 + 2 $^{-7}$

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

QPSKSyn



Description: QPSK Encoder
Library: Numeric, Fixed-Point DSP
Class: SDFQPSKSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	bit width of encoder outputs	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	DataI		fix
2	DataQ		fix

Pin Outputs

Pin	Name	Description	Signal Type
3	Iout		fix
4	Qout		fix

Notes/Equations

1. The output signals of the QPSK encoder are 2 twos-complement fixed-point numbers with 1 sign bit and (Width −1) fractional bits, Iout and Qout.
   For each DataI or DataQ input bit value of 1, a mapping to the fixed-point number (represented by a 1 sign bit and (Width −1) fractional bits) closest to the negative value of the square root of 1/2 (that is, −0.7071067811..) is done. Conversely, for each DataI or DataQ input bit value of 0 a mapping to the fixed point number (represented by a 1 sign bit and (Width-1) fractional bits) closest to the square root of 1/2 (that is, +0.7071067811..) is done.
   For example, with Width = 8, mapping will be done as in the table below.

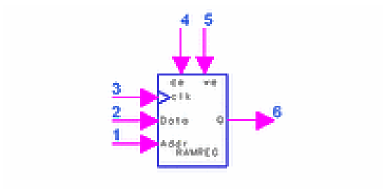
Input DataI	Input DataQ	-->	Output Iout	Output Qout
0	0	-->	01011011	01011011
0	1	-->	01011011	10100101
1	0	-->	10100101	01011011
1	1	-->	10100101	10100101

   Note that, with 1 sign bit and 7 fractional bits twos-complement:

   - 01011011 corresponds to 0.7109375
   - 10100101 corresponds to −0.7109375

2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

RamRegSyn



Description: Registered Random-Access-Memory (RAM)
Library: Numeric, Fixed-Point DSP
Class: SDFRamRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Depth	Number of words in RAM.	16	int
ramFile	File containing initial RAM values.		filename
ramFileFormat	Format of RAM init file.: REAL, HEX	HEX	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	Addr	input address	fix
2	Data	input data	fix
3	Clock	Clock input – optional control pin	fix
4	CE	Clock enable input – optional control pin	fix
5	WE	write enable input: if low then the input Data is	fix

Pin Outputs

Pin	Name	Description	Signal Type
6	Q	output data	fix

Notes/Equations

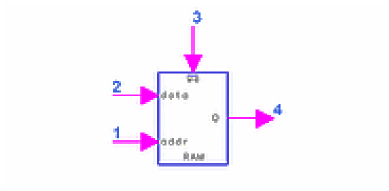
1. This model implements a RAM with a registered output.
   Given an input address in Addr, and data in Data, the model will write the input Data into an internal array if WE is asserted by a low value; if WE is not asserted, the model will put data addressed by Addr onto its output Q.
2. The output of the RAM is registered with a positive edge Clock input.
   The clock enable CE control input is optional:
   - if it is not connected, the model is always enabled
   - if it is connected, it is enabled by a high value in CE.
     The initial values in the RAM can be defined in the (optional) file as specified in the ramFile parameter. The format of the file is specified by the ramFileFormat

parameter; initialization values can be specified as REAL or HEX. The address of each initial data read into the model is the same as the line number of the corresponding data read from the initialization file.
The initial values are specified as a column of values as in the following examples.

- If ramFileFormat = REAL which specifies that the RAM initialization file contains real values, then an example of such a file would be:
  0.98
  0.24
  0.12
  .
  .
  .
  From this example, the model will interpret the first line as address 0 with data equal to the fixed-point value corresponding to 0.98, and so on. Note that the model will convert the real values to its fixed-point representation using the specified precision in the OutputPrecision parameter, and arithmetic type as specified in the ArithType parameter.
- If ramFileFormat = HEX, then an example of such a file would be:
  0x7f
  0x06
  0x08
  .
  .
  .
  From this example, the model will interpret the first line as address 0 with data equal to 0x7f, and so on.

3. The Depth parameter specifies the number of words in the RAM.
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

RamSyn



Description: RAM
Library: Numeric, Fixed-Point DSP
Class: SDFRamSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
Depth	size of (number of words in) RAM	16	int
ramFile	name of file containing initial RAM values (optional) (represented in hex data format in file)		filename

Pin Inputs

Pin	Name	Description	Signal Type
1	Addr	input address	fix
2	Data	input data	fix
3	WE	write enable input: if low then the input Data is	fix

Pin Outputs

Pin	Name	Description	Signal Type
4	Q	output data	fix

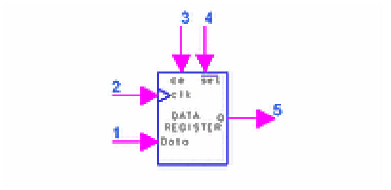
Notes/Equations

1. RamSyn models the RAM. Data in the RAM can be initialized by specifying the file name in the ramFile parameter.
2. The path name for ramFile can be specified in several ways: one is to just specify the file name, for example ramFile = *foo*, which is assumed to be located within the current workspace data directory; another is to specify the absolute path, as in ramFile = */usr/user_name/foo*; or, the environmental variables can also be used to set the file path name, for example ramFile = *$ENV_FOO/foo*, where *ENV_FOO* is an environmental variable.
3. The bitwidths and arithmetic type of the output data are defined by the device parameters. The size of the RAM is specified by the Depth parameter. An example file format is:
   0x01
   0xff

0xca
.
.
.
and so on.

4. The data format in the file is assumed to be right-justified.
5. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

RegSyn



Description: Data Register
Library: Numeric, Fixed-Point DSP
Class: SDFRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
ValueS	value loaded into the register when the Set control pin is asserted	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Clock	Clock input - optional control pin	fix
3	CE	Clock enable input - optional control pin	fix
4	Set	Synchronous set/reset input - optional control pin	fix

Pin Outputs

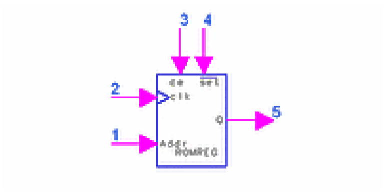
Pin	Name	Description	Signal Type
5	Q	Register data output	fix

Notes/Equations

1. RegSyn is positive-edge triggered and latches the input data upon detecting the positive edge.
2. The control pins are optional; if these are not connected, the defaults will be:
   - Clock not connected, the device reverts to a unit-delay register.
   - CE connected and high, the input data is latched by the register upon a positive clock edge.
   - CE connected and low (it holds a value of 0), the register output stays the same and the input data is not latched.
   - CE not connected, the clock is enabled by default and the input data is latched by the register upon a positive clock edge.
   - Set connected and low, the register output is set to the value specified by the parameter ValueS.
   - Set connected and high, the register output is not set to ValueS.
   - Set not connected, the register output is never set to ValueS.

3. OutputPrecision specifies the fixed-point precision format of the output. For example, if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.
4. ValueS can be specified in hex (0x prefix), octal (0 prefix), binary (0b prefix), or decimal (without a 0 prefix).
   For example, to specify a ValueS of decimal value 31, set
   ValueS = 31 (decimal), ValueS = 0x1F (hex), ValueS = 037 (octal), or ValueS = 0b11111 (binary).
5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

RomRegSyn



Description: Registered Read-Only-Memory (ROM)
Library: Numeric, Fixed-Point DSP
Class: SDFRomRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
romFile	Filename containing ROM data.		filename
romFileFormat	Format of ROM init file.: REAL, HEX	HEX	enum
Depth	Number of words in ROM.	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Addr		fix
2	Clock	Clock input – optional control pin	fix
3	CE	Clock enable input – optional control pin	fix
4	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	Q		fix

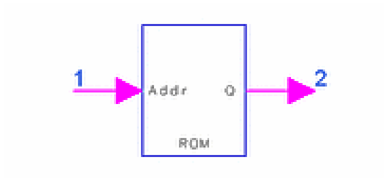
Notes/Equations

1. This model implements a ROM with a registered output. Given an input address in Addr, the model will put the data addressed by Addr onto output Q.
2. The output of the ROM is registered with a positive edge Clock input.
   The clock enable CE control input is optional.
   - if it is not connected, the model is always enabled
   - if it is connected, it is enabled by a high value in CE.
3. The initial values in the ROM can be defined in the file specified in the romFile parameter. The format of the file is specified by the romFileFormat parameter; data can be specified as REAL or HEX values. The address of each data value read into the model is the same as the line number of the corresponding data read from the file.
4. The values are specified as a column of values as in the following examples.
   If romFileFormat = REAL which specifies that the ROM file contains real values, then

an example of such a file would be:
0.98
0.24
0.12
.
.
.
From the above file example, the model will interpret the first line as address 0 with data equal to the fixed point value corresponding to 0.98, etc. Note that the model will convert the real values to its fixed point representation using the specified precision in the OutputPrecision parameter, and arithmetic type as specified in the ArithType parameter.
If romFileFormat = HEX, then an example of such a file would be:
0x7f
0x06
0x08
.
.
.
From the above file example, the model will interpret the first line as address 0 with data equal to 0x7f, and so on.

5. The Depth parameter specifies the number of words in the ROM.
6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

RomSyn



Description: ROM
Library: Numeric, Fixed-Point DSP
Class: SDFRomSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
romFile	name of file containing ROM values (represented in hex data format in file)		filename
Depth	size of (number of words in) ROM	1	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Addr		fix

Pin Outputs

Pin	Name	Description	Signal Type
2	Q		fix

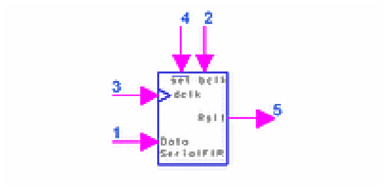
Notes/Equations

1. RomSyn reads the specified file of ASCII hex values and stores them in a linear array to model the ROM.
2. The path name for romFile can be specified in several ways: one is to just specify the file name, for example romFile = *foo*, which is assumed to be located within the current workspace data directory; another is to specify the absolute path, as in romFile = */usr/user_name/foo*; or, the environmental variables can also be used to set the file path name, for example romFile = *$ENV_FOO/foo*, where *ENV_FOO* is an environmental variable.
3. The input address value is used as an index into the array. An example file format:
   0x0ff0a
   0x0bcd9
   .
   .
   .
   and so on.
4. The data format in the file is assumed to be right-justified.
5. OutputPrecision specifies the fixed-point precision format of the output. For example,

if OutputPrecision = 1.15, 1 bit is used for representing the integer part of the output, and 15 bits are used to represent the fractional portion of the output.

6. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

SerialFIRSyn



Description: Serial Finite Impulse Response (FIR) Filter
Library: Numeric, Fixed-Point DSP
Class: SDFSerialFIRSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
NumOfTaps	Number of taps in FIR filter.	6	int
CoefPrecision	Precision of the coefficients in the coefficient file.	2.14	precision
DataPrecision	Precision of the input data.	2.14	precision
CoefFile	File containing FIR coefficient values.		filename
CoefFileFormat	Format of FIR Coefficients file.: REAL, HEX	HEX	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	DataIn	Data input	fix
2	BitClock	Bit Clock input – Bit-rate clock	fix
3	DataClock	Data Clock input – input sample rate clock	fix
4	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
5	Result	FIR result output (with precision OutputPrecision)	fix

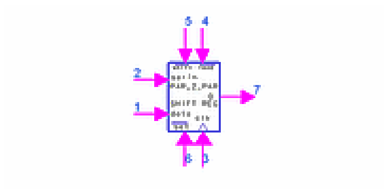
Notes/Equations

1. This model is a bit-serial finite impulse response (FIR) filter model. It implements a bit-serial FIR structure and retains full precision internally when calculating filter output values.
   The only quantization done is at the Result output of the model.
2. The Result output of the bit-serial FIR model is the final result of the FIR filtering done within the model, and quantized to the precision specified by OutputPrecision.
3. Data from DataIn input is clocked into the internal data registers of the bit-serial FIR model upon the positive edge transitions of the DataClock input.
4. The BitClock input is used to simulate the bit-serial nature of the FIR filter; it clocks

the result of the FIR filter into a FIFO buffer of depth equal to the total number of bits in DataIn (as specified by the DataPrecision parameter). If the total number of bits in DataPrecision is equal to W, there is a delay equal to W BitClock positive edges before the FIR filter output is sent to Result.

5. The 1-bit Reset input pin is asserted by bring it low (a value of 1), which will clear all the internal data registers.
6. The filter tap coefficients of the bit-serial FIR filter is defined in the file specified in the CoefFile parameter. The format of the file is specified by the CoefFileFormat parameter; tap coefficients can be specified as REAL or HEX values. The tap coefficients are specified as a column of values in the file. The 0th tap filter coefficient is the value on the first line of the filter tap coefficient file; the 1th tap filter coefficient corresponds to the value on the second line; the 2th tap filter coefficient corresponds to the value on the third line, and so on.
   Consider the following examples.
   - If CoefFileFormat = REAL, which specifies that the filter tap coefficient file contains real values for the filter tap coefficients, an example of such a file would be:
     0.98
     0.24
     0.12
     0.05
     -0.13
     0.21
     .
     .
     .
   - If CoefFileFormat = HEX, which specifies that the filter tap coefficient file contains hex values for the filter tap coefficients, an example of such a file would be:
     0x7f
     0x06
     0x02
     0x8f
     0x07
     0x08
     .
     .
     .
7. The NumTaps parameter specifies the number of tap coefficients to be read from the file specified by CoefFile.
8. The CoefPrecision parameter specifies the precision of the filter tap coefficients, that is, the number of integer bits (including the sign bit) and the number of fractional bits to be used to represent the filter tap coefficients.
9. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

ShiftRegPPSyn



Description: Parallel In/Parallel Out Shift Register
Library: Numeric, Fixed-Point DSP
Class: SDFShiftRegPPSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	number of bits in internal state of shift register	16	int
Dir	direction of bit shift: RIGHT, LEFT	LEFT	enum
ValueS	value loaded into the register when the Set control pin is asserted	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Serin	Serial bit input	fix
3	Clock	Clock input – optional control pin	fix
4	Load	Load control input – optional control pin	fix
5	Shift	Shift control input – optional control pin	fix
6	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

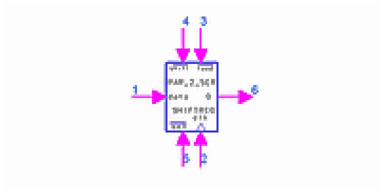
Pin	Name	Description	Signal Type
7	Q	Shifted data output	fix

Notes/Equations

1. ShiftRegPPSyn (Parallel_In/Parallel_Out) clock is positive-edge triggered and shifts the internal register data upon detecting the positive edge.
2. Direction of shifting is done assuming that the MSB is on the left and the LSB is on the right. For example, if Dir = LEFT, then shifting is done toward the MSB; conversely, if Dir = RIGHT, then shifting is done toward the LSB.
3. ValueS can be specified in hex (0x prefix), octal (0 prefix), binary (0b prefix), or decimal (without a 0 prefix). For example, to specify a ValueS of decimal value 31, set ValueS = 31 (decimal), ValueS = 0x1F (hex), ValueS = 037 (octal), or ValueS = 0b11111 (binary).
4. For general information regarding numeric fixed-point DSP functions, refer to

Numeric Fixed-Point DSP Components (numeric).

ShiftRegPSSyn



Description: Parallel In/Serial Out Shift Register
Library: Numeric, Fixed-Point DSP
Class: SDFShiftRegPSSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	number of bits in internal state of shift register	16	int
Dir	direction of bit shift: RIGHT, LEFT	LEFT	enum
ValueS	value loaded into the register when the Set control pin is asserted	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Clock	Clock input – optional control pin	fix
3	Load	Load control input – optional control pin	fix
4	Shift	Shift control input – optional control pin	fix
5	Set	Asynchronous set/reset input – optional control pin	fix

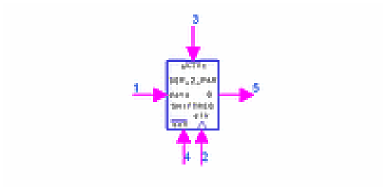
Pin Outputs

Pin	Name	Description	Signal Type
6	Q	Shifted data output	fix

Notes/Equations

1. ShiftRegPSSyn (Parallel_In/Serial_Out) clock is positive-edge triggered and shifts the internal register data upon detecting the positive edge.
2. Direction of shifting is done assuming that the MSB is on the left and the LSB is on the right. For example, if Dir = LEFT, then shifting is done toward the MSB; conversely, if Dir = RIGHT, then shifting is done toward the LSB.
3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

ShiftRegSPSyn



Description: Serial In/Parallel Out Shift Register
Library: Numeric, Fixed-Point DSP
Class: SDFShiftRegSPSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	number of bits in internal state of shift register	16	int
Dir	direction of bit shift: RIGHT, LEFT	LEFT	enum
ValueS	value loaded into the register when the set control pin is asserted	0	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Clock	Clock input – optional control pin	fix
3	Shift	Shift control input – optional control pin	fix
4	Set	Asynchronous set/reset input – optional control pin	fix

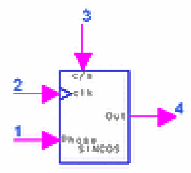
Pin Outputs

Pin	Name	Description	Signal Type
5	Q	Shifted data output	fix

Notes/Equations

1. ShiftRegSPSyn (Serial_In/Parallel_Out) clock is positive-edge triggered and shifts the internal register data upon detecting the positive edge.
2. Direction of shifting is done assuming that the MSB is on the left and the LSB is on the right. For example, if Dir = LEFT, then shifting is done toward the MSB; conversely, if Dir = RIGHT, then shifting is done toward the LSB.
3. ValueS can be specified in hex (0x prefix), octal (0 prefix), binary (0b prefix), or decimal (without a 0 prefix). For example, to specify a ValueS of decimal value 31, set ValueS = 31 (decimal), ValueS = 0x1F (hex), ValueS = 037 (octal), or ValueS = 0b11111 (binary).
4. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

SineCosineSyn



Description: Sine/Cosine Look-up Table
Library: Numeric, Fixed-Point DSP
Class: SDFSineCosineSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutWidth	Output width of NCO.	10	int
PhaseInWidth	Width of PhaseIn input.	10	int

Pin Inputs

Pin	Name	Description	Signal Type
1	PhaseIn	Phase input – unsigned	fix
2	Clock	Clock input – optional control pin	fix
3	SineOrCosine	SineOrCosine – controls whether sine or cosine is output	fix

Pin Outputs

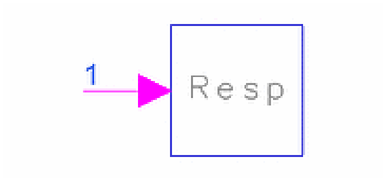
Pin	Name	Description	Signal Type
4	Out		fix

Notes/Equations

1. This model implements a sine or cosine look-up table; given an input phase value, it outputs a fixed point value (1 sign bit, (OutWidth-1) fractional bits twos-complement) corresponding to the Sine or Cosine of the phase.
2. The (optional) 1-bit control input SineOrCosine determines whether a sine or cosine value is evaluated by the model.
   - If the SineOrCosine pin is un-connected (in other words, unused) then the default output of the model is a sine value.
   - If the SineOrCosine pin is connected, then a low value (corresponding to 0) will cause the model to output a cosine value, and, conversely a high value (corresponding to 1) will cause the model to output a sine value.
3. The input phase value in PhaseIn is interpreted within the model as an unsigned fixed point number (with PhaseInWidth integer bits, and no fractional bits) and the value of sine(2π × PhaseIn/(2 $^{\text{PhaseInWidth}}$)) or cosine(2π × PhaseIn/(2 $^{\text{PhaseInWidth}}$)) is evaluated, and output. The output value in Out is represented as a twos-complement, 1-sign bit, (OutWidth-1) fractional bits, fixed point number.
   For general information regarding numeric fixed-point DSP functions, refer to

4. *Numeric Fixed-Point DSP Components* (numeric).

SinkRespSyn



Description: Response Sink
Library: Numeric, Fixed-Point DSP
Class: SDFSinkStimSyn

Parameters

Name	Description	Default	Type
Start	sample number at which to start recording	DefaultNumericStart	int
Stop	sample number at which to stop recording	DefaultNumericStop	int

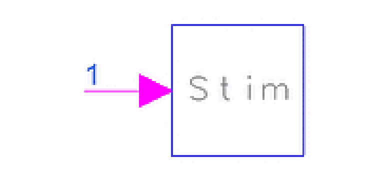
Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	fix

Notes/Equations

1. SinkRespSyn collects Fix data for test vector responses.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

SinkStimSyn



Description: Stimulus Sink
Library: Numeric, Fixed-Point DSP
Class: SDFSinkStimSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Start	sample number at which to start recording	DefaultNumericStart	int
Stop	sample number at which to stop recording	DefaultNumericStop	int

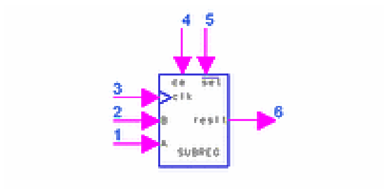
Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	fix

Notes/Equations

1. SinkStimSyn collects Fix data for test vector stimulus.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

SubRegSyn



Description: Registered Subtracter
Library: Numeric, Fixed-Point DSP
Class: SDFSubRegSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
OvflowType	overflow characteristic for device: WRAPPED, SATURATE	WRAPPED	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix
3	Clock	Clock input – optional control pin	fix
4	CE	Clock enable input – optional control pin	fix
5	Set	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
6	Result		fix

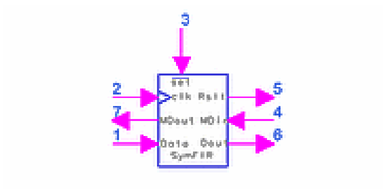
Notes/Equations

1. This model is a registered subtracter. It calculates the subtraction of its A and B data inputs (A-B) and registers its output Result such that it has the specified precision as set in the OutputPrecision parameter.
2. The Clock input is optional.
   - if it is connected, the model will operate based on the positive edge transitions of the Clock input.
   - if it is not connected, the model will operate as if every sample step of the simulator is a positive edge transition.
3. Assertion of the Reset input by bringing it low (a value of 0) will clear the output data register.
4. The (optional) CE input is the clock-enable control for the output data register.
   - if it is connected and has a high value (a value of 1), the output data register is

enabled and will load the addition result upon a positive Clock edge.
- if it is connected, and low (a value of 0), the clock to the output data register is disabled.
- if the CE input is not connected, the output data register is always enabled.

5. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

SymFIRSyn



Description: Symmetric Finite Impulse Response (FIR) Filter
Library: Numeric, Fixed-Point DSP
Class: SDFSymFIRSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
OutputPrecision	precision of the output in bits	2.14	precision
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
NumOfTaps	Number of taps in FIR filter.	2	int
CoefPrecision	Precision of the coefficients in the coefficient file.	2.14	precision
DataPrecision	Precision of the MidDataOut output (used in cascading FIR filters).	2.14	precision
CoefFile	File containing FIR coefficient values.		filename
CoefFileFormat	Format of FIR Coefficients file.: REAL, HEX	HEX	enum
CascadeMode	Use filter in cascade mode? NO, YES	NO	enum
SymmetricMode	Is filter symmetric or anti-symmetric? SYMMETRIC, ANTI_SYMMETRIC	SYMMETRIC	enum

Pin Inputs

Pin	Name	Description	Signal Type
1	DataIn	Data input	fix
2	Clock	Clock input – optional control pin	fix
3	Set	Asynchronous set/reset input – optional control pin	fix
4	MidDataIn	Mid point data input (optional) (with precision = precision of DataIn input)	fix

Pin Outputs

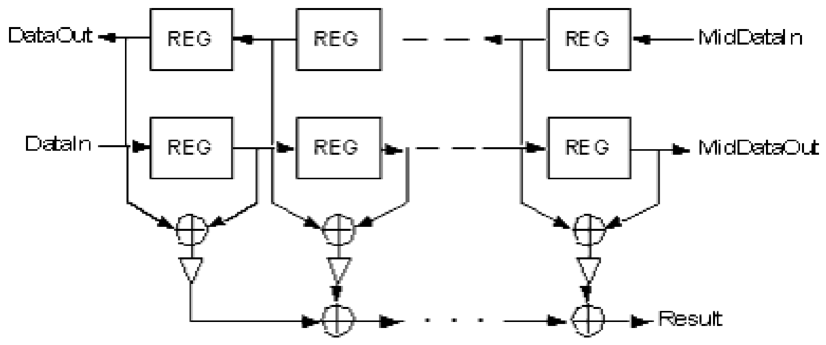
Pin	Name	Description	Signal Type
5	Result	FIR result output (with precision OutputPrecision)	fix
6	DataOut	End point data output (with precision DataPrecision = precision of DataIn input)	fix
7	MidDataOut	Mid point data output (with precision DataPrecision = precision of DataIn input)	fix

Notes/Equations

1. This model is a symmetric FIR (finite impulse response) filter model. It implements a

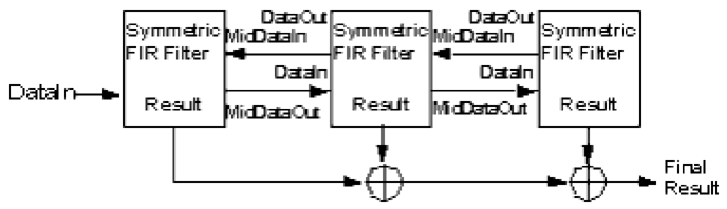
general parallel FIR structure with symmetric filter tap coefficients. It retains full precision internally when calculating filter output values. The only quantization done is at the Result output of the model.

Internal Structure of Symmetric FIR Model



2. Data from DataIn input is clocked into the internal data registers of the FIR model upon the positive edge transitions of the Clock input if the Clock pin is connected. If the Clock pin is not connected, then data is shifted into the internal data registers at every sample step in the simulator.
3. The (optional) input MidDataIn and outputs MidDataOut and DataOut are used when cascading several symmetric FIR models. Cascading may be desirable in the case where there is a limit on the FIR filter order per Symmetric FIR model, which is the case in the Xilinx CORE Generator Symmetric FIR filter that is limited, at most, to 20 filter taps per Symmetric FIR core.
   The parameter CascadeMode should be set to YES if the model is to be cascaded to feed another Symmetric FIR model or NO if it does not feed into another Symmetric FIR model. If CascadeMode is set to NO for no cascading, then (internally within the model), the MidDataIn input takes its input data from the MidDataOut output of the model.
   Cascading of several Symmetric FIR filter models is illustrated below.

Cascading of Several Symmetric FIR Filter Models



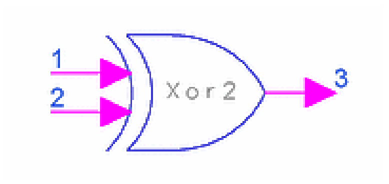
4. The parameter SymmetricMode is used to select whether the FIR filter coefficients are symmetric or anti-symmetric.
5. The Result output of the symmetric FIR model is the final result of the FIR filtering done within the model, and quantized to the precision specified by OutputPrecision.
6. The 1-bit Reset input pin is asserted by bring it low (i.e., value of 1), which will clear all the internal data registers.
7. Since the filter is symmetric or anti-symmetric, only the first half of the filter tap coefficients need to be defined in the filter definition file.
   The filter tap coefficients of the FIR filter is defined in the file as specified in the

CoefFile parameter. The format of the file is specified by the CoefFileFormat parameter; the tap coefficients can be specified as real or hex values. The tap coefficients are specified as a column of values in the file. The 0th tap filter coefficient is the value on the first line of the filter tap coefficient file, the 1th tap filter coefficient corresponds to the value on the second line, the 2th tap filter coefficient corresponds to the value on the third line, and so on.
Consider the following examples.

- If CoefFileFormat = REAL, which specifies that the filter tap coefficient file contains real values for the filter tap coefficients, then an example of such a file would be:
  0.98
  0.24
  0.12
  0.05
  -0.13
  0.21
  .
  .
  .
- if CoefFileFormat = HEX, which specifies that the filter tap coefficient file contains hex values for the filter tap coefficients, then an example of such a file would be:
  0x7f
  0x06
  0x02
  0x8f
  0x07
  0x08
  .
  .
  .

8. The NumTaps parameter specifies the number of tap coefficients to be read from the file specified by CoefFile.
9. The CoefPrecision parameter specifies the precision of the filter tap coefficients, that is, the number of integer bits (including the sign bit) and the number of fractional bits to be used to represent the filter tap coefficients.
10. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

Xor2Syn



Description: 2-input XOR
Library: Numeric, Fixed-Point DSP
Class: SDFXor2Syn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	Width of an input bus.	8	int

Pin Inputs

Pin	Name	Description	Signal Type
1	A		fix
2	B		fix

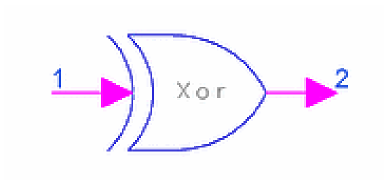
Pin Outputs

Pin	Name	Description	Signal Type
3	Result		fix

Notes/Equations

1. This model is a 2-input XOR gate. It takes a bitwise XOR of inputs A and B (both of bitwidth Width) and outputs the results, that is, Result = A XOR B.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

XorSyn



Description: Bitwise XOR
Library: Numeric, Fixed-Point DSP
Class: SDFXorSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
Width	size of a bus segment within the input bus	8	int
Size	number of bus segments within the input bus	2	int

Pin Inputs

Pin	Name	Description	Signal Type
1	Data		fix

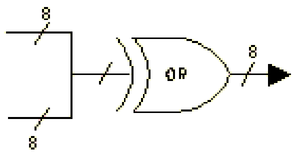
Pin Outputs

Pin	Name	Description	Signal Type
2	Result		fix

Notes/Equations

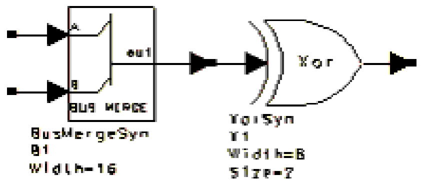
1. The input bus is composed of Size number of smaller bus segments. Each bus segment within the input bus is of bitwidth Width. XorSyn performs a bitwise XOR of the bus segments resulting in the output Result of bitwidth Width. For example, Width = 8, Size = 2 means that the input bus is interpreted as having 2 bus segments, each of bitwidth 8. The output of XorSyn is the bitwise XOR of the 2 bus segments, as illustrated in the figure below.

Width = 8, Size = 2



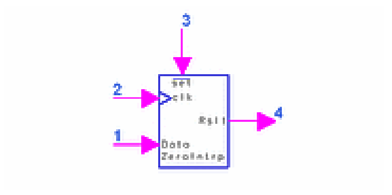
2. An example design where two 8-bit signals are XORed together is shown below.

XorSyn Example



3. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

ZeroInterpSyn



Description: Zero insertion interpolator
Library: Numeric, Fixed-Point DSP
Class: SDFZeroInterpSyn
Derived From: SDFHPFix

Parameters

Name	Description	Default	Type
ArithType	arithmetic type of output: TWOS_COMPLEMENT, UN_SIGNED	TWOS_COMPLEMENT	enum
UpSampleRatio	Up-sample ratio	2	int
DataPrecision	Precision of output data – its bitwidth must equal input data bitwidth	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	Data	Data input	fix
2	Clock	Clock input	fix
3	Reset	Asynchronous set/reset input – optional control pin	fix

Pin Outputs

Pin	Name	Description	Signal Type
4	Result	Clock output	fix

Notes/Equations

1. This model is a data interpolator. It performs an upsampling of the input data by inserting extra zeros (UpSampleRatio -1 zeros) for each input data. For example, given an input value of 0x1F, with UpSampleRatio equal to 2 (meaning this model is upsampling by 2), the output Result will give the values 0x1F, 0.
2. For general information regarding numeric fixed-point DSP functions, refer to *Numeric Fixed-Point DSP Components* (numeric).

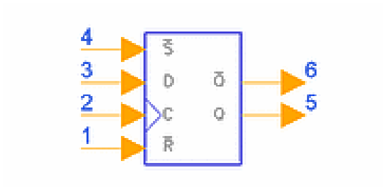
Numeric Logic Components

- *DFF* (numeric)
- *DivByN* (numeric)
- *JKFF* (numeric)
- *LFSR* (numeric)
- *Logic* (numeric)
- *LogicAND* (numeric)
- *LogicAND2* (numeric)
- *LogicInverter* (numeric)
- *LogicLatch* (numeric)
- *LogicNAND* (numeric)
- *LogicNAND2* (numeric)
- *LogicNOR* (numeric)
- *LogicNOR2* (numeric)
- *LogicOR* (numeric)
- *LogicOR2* (numeric)
- *LogicXNOR* (numeric)
- *LogicXNOR2* (numeric)
- *LogicXOR* (numeric)
- *LogicXOR2* (numeric)
- *Multiple* (numeric)
- *Test* (numeric)
- *TestEQ* (numeric)
- *TestGE* (numeric)
- *TestGT* (numeric)
- *TestLE* (numeric)
- *TestLT* (numeric)
- *TestNE* (numeric)

The Numeric Logic component library contains operators on Boolean valued integer signals (values are either 0 or 1) or double precision floating-point (real) signals. Each component produces Boolean integer values. Positive logic is used: low (or false) = 0, high (or true) = 1.

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

DFF



Description: D-Type Binary Data Flip-Flop (Edge Triggered)
Library: Numeric, Logic
Class: SDFDFF
Derived From: baseOmniSysNumericStar

Pin Inputs

Pin	Name	Description	Signal Type
1	R	clear input	int
2	C	clock input	int
3	D	D input	int
4	S	preset input	int

Pin Outputs

Pin	Name	Description	Signal Type
5	Q	Q output	int
6	NQ	inverted Q output	int

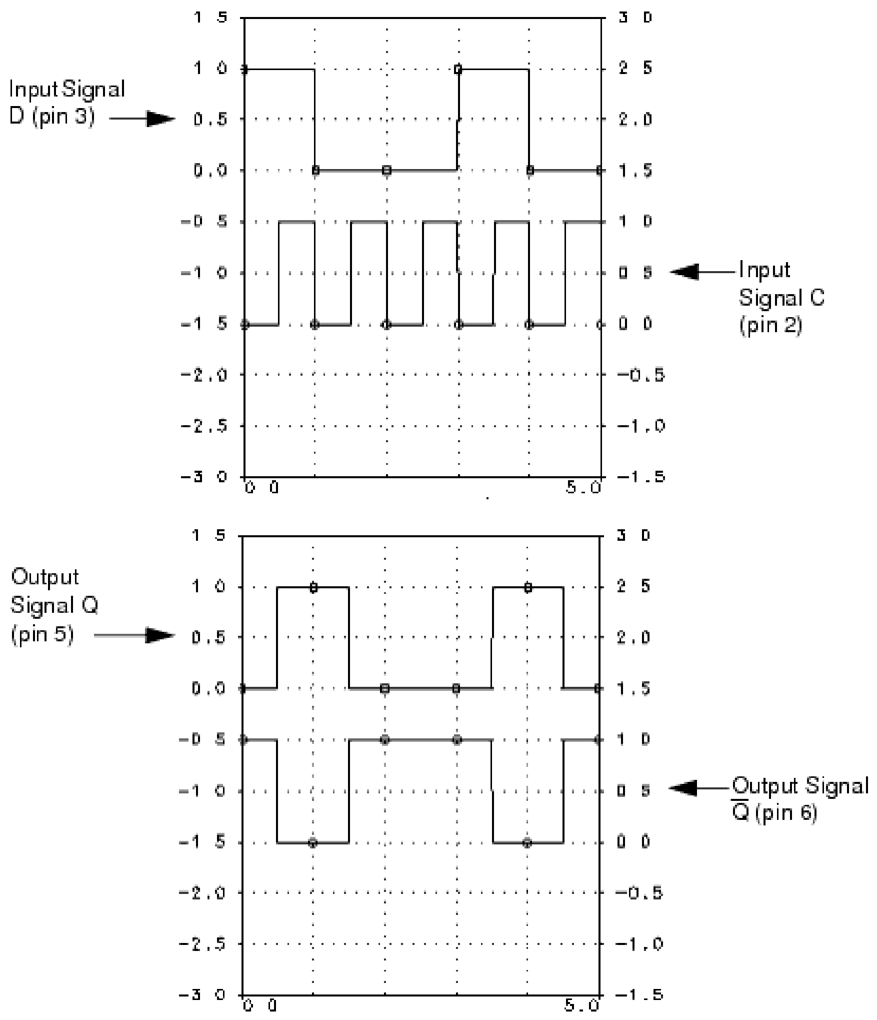
Notes/Equations

1. Function table

Inputs				**Outputs**	
R (Pin 1)	**C (Pin 2)**	**D (Pin 3**	**S (Pin 4)**	**Q (Pin 5)**	**NQ (Pin 6)**
H	x	x	L	H	L
L	x	x	H	L	H
L	x	x	L	H	H
H	UP	H	H	H	L
H	UP	L	H	L	H
H	L	x	H	Q0	NQ0
whereS = input preset, active with logic low levelR = input clear, active with logic low levelC = input clock, active with low to high transitionx = don't care stateL = logic low level. Input: < 0.5; Output: 0.0H = logic high level. Input: > 0.5; Output: 1.0UP = low-to-high transitionQ0 = previous Q stateNQ = inverted Q stateNQ0 = previous inverted Q state					

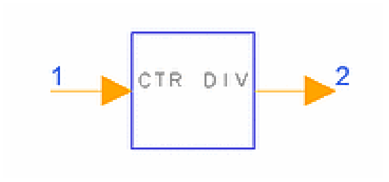
2. At the first sample, the outputs Q and NQ are equal to L and H, respectively.
3. Input, output and clock signal values of the DFF component, with S (pin 4) and R (pin 1) both tied to a high logic level, are shown below.

DFF Input, Output, and Clock Signal Values



4. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

DivByN



Description: Binary Data Divide-By-N Counter
Library: Numeric, Logic
Class: SDFDivByN
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
N	divide-by factor	1		int	[1, ∞)
N0	initial counter value	0		int	[0, N)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	int

Pin Outputs

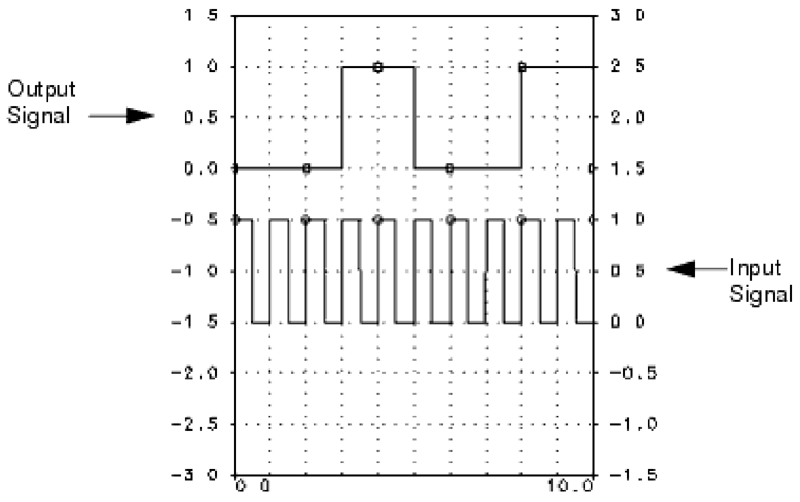
Pin	Name	Description	Signal Type
2	output	output signal	int

Notes/Equations

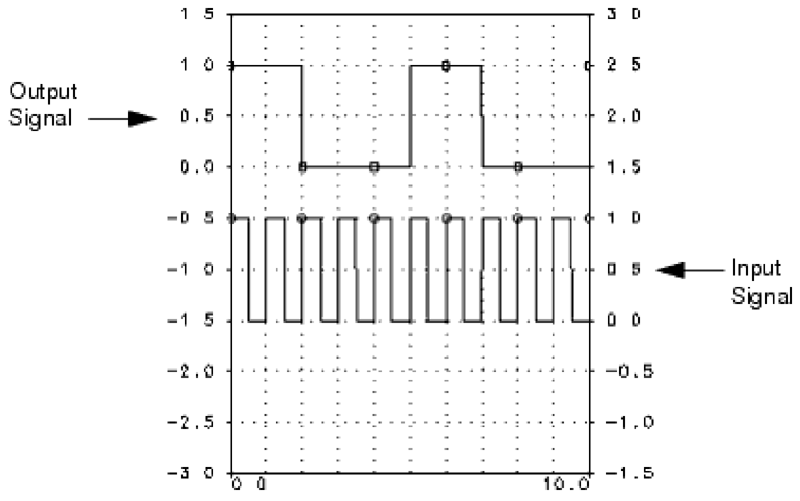
1. DivByN is a model of a positive edge-triggered, modulo N down counter. The input to the component is a clock signal; the output is a signal that is high or low, depending on whether the current counter value is greater or less than floor (N/2). (Note that the counter value itself is not available as an output.)
   Let M(k) denote the counter value after the _k_th positive clock edge. Then
   M(0) = N0
   M(k) = (M(k − 1) − 1) modulo N, k ≥ 1
   $$V_2(t) = \begin{cases} 0 \text{ if } M(k) \geq floor\left(\frac{N}{2}\right) \\ 1 \text{ if } M(k) < floor\left(\frac{N}{2}\right) \end{cases}$$
2. The input and output signal values of the DivByN component, parameters N = 5 and N0 = 4, are shown in DivByN Input and Output Signal Values, N = 5 and N0 = 4. Note that the initial counter value is 4, and therefore the output is low (because it is ≥ 2 (floor(5/2)). When the first input positive edge occurs, the counter is decremented to 3 and the output is still low. At the second input positive edge, the counter is decremented to 2 and the output is low. At the third positive edge the

counter is decremented to 1, which makes the output high (because it is < 2 (floor(N/2))). Similarly, at the fourth positive edge, the counter decrements to 0 and the output is high. At the fifth positive edge, the counter decrements to negative and is therefore reset to 4 and the output is low.
The input and output signal values of the DivByN component, parameters N = 5 and N0 = 1, are shown in DivByN Input and Output Signal Values, N = 5 and N0 = 1.
Note that the initial counter value is 1, and therefore the output is high (because it is < 2 (floor(5/2)). At the first input positive edge, the counter is decremented to 0, which means that the output is still high. At the second positive edge the counter is reset to 4 and the output is low.

DivByN Input and Output Signal Values, N = 5 and N0 = 4

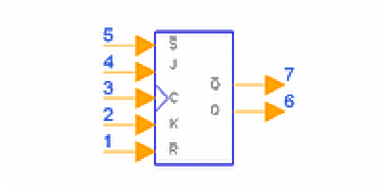


DivByN Input and Output Signal Values, N = 5 and N0 = 1



3. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

JKFF



Description: Binary Data J-K Type Flip-Flop
Library: Numeric, Logic
Class: SDFJKFF
Derived From: baseOmniSysNumericStar

Pin Inputs

Pin	Name	Description	Signal Type
1	R	clear input	int
2	K	K input	int
3	C	clock input	int
4	J	J input	int
5	S	preset input	int

Pin Outputs

Pin	Name	Description	Signal Type
6	Q	Q output	int
7	NQ	inverted Q output	int

Notes/Equations

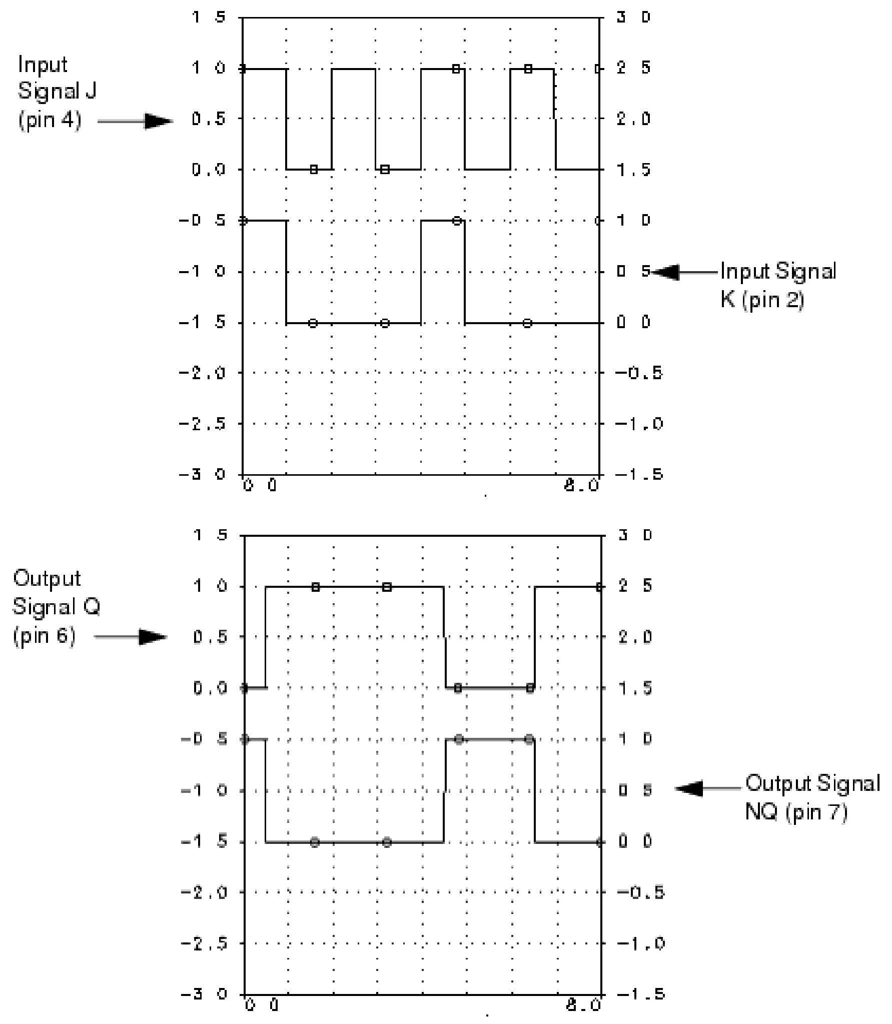
1. Function table:

Input					Output	
R (pin 1)	**K (pin 2)**	**C (pin 3)**	**J (pin 4)**	**S (pin 5)**	**Q (pin 6)**	**NQ (pin 7)**
L	x	x	x	L	H	H
H	x	x	x	L	H	L
L	x	x	x	H	L	H
H	L	UP	L	H	Q0	NQ0
H	L	UP	H	H	H	L
H	H	UP	L	H	L	H
H	H	UP	H	H	TOGGLE	
whereC = input clock, active with low to high transitionS = input preset, active with logic low levelR = input clear, active with logic low levelx = don't care stateL = logic low level; Inputs < 0.5; Outputs 0.0H = logic high level; Inputs ≥ 0.5; Outputs 1.0UP = low-to-high transitionQ0 = previous Q stateNQ = inverted Q stateNQ0 = previous inverted Q state						

2. At the first sample, the outputs Q and NQ are equal to L and H, respectively.
3. The input and output signal values of the JKFF component, with S (pin 5) and R (pin

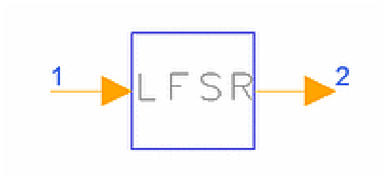
1) both tied to a high logic level, are shown in JKFF Input and Output Signal Values. The clock signal (not shown) is set such that it is first active (low to high transition occurs) at the first 0.5 grid unit and has a period of 1 grid unit.

JKFF Input and Output Signal Values



4. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LFSR



Description: Linear feedback shift register
Library: Numeric, Logic
Class: SDFLFSR
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Seed	initial value loaded into the shift register	1		int array	
FeedbackList	tap positions for non-zero feedback coefficients	7 3 2 1		int array	

Pin Inputs

Pin	Name	Description	Signal Type
1	clock	clock signal	int

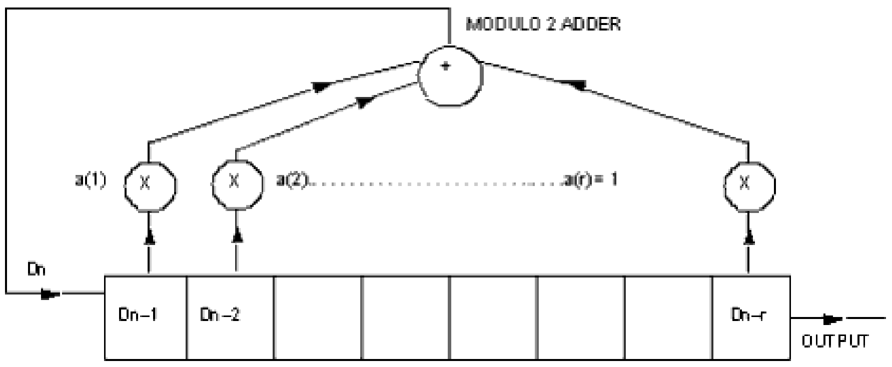
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	int

Notes/Equations

1. The linear feedback shift register component can be used to generate PN sequences with user-defined recurrence relations. The input to the LFSR is a clock signal. A new bit value is generated at the output every time the input signal transitions from 0 to 1. The diagram below illustrates an LFSR model.

LFSR Model



Data is shifted to the right in the shift register. The length of the shift register is r. The numbers a(1), a(2), ... , a(r) are the binary feedback coefficients specified by FeedbackList.
The shift register length r is defined by the largest value in FeedbackList. For example, a FeedbackList of 7 3 2 1 results in a shift register length of 7; the maximum value allowed in FeedbackList is 31, which results in a maximum shift register length of 31.
The initial contents of the shift register are specified by the value of Seed. The maximum meaningful value for Seed is $2^r - 1$ for a specific FeedbackList. The maximum Seed value allowed is $2^{31} - 1$.
The following equations describe the operation of LFSR.

$$D(n) = \left[\sum_{k=1}^{r} a(k)D(n-k)\right] \bmod 2 \quad \text{for } n \geq 1$$

where
$D(0) = Seed_2(0)$

$D(-1) = Seed_2(1)$

.
.
.
$D(1-r) = Seed_2(r-1)$

and

$$Seed = \sum_{k \geq o} Seed_2(k)2^k$$

where
$Seed_2(k)$ Œ{0,1} for $0 \leq k < r$

Example: Let Seed = 2, and r = 7
Then
$Seed_2(0) = 0$

$Seed_2(1) = 1$

$Seed_2(2) = 0$

.
.
.
$Seed_2(6) = 0$

Therefore,
$D(0) = Seed_2(0) = 0$

$D(-1) = Seed_2(1) = 1$

$D(-2) = Seed_2(2) = 0$

.
.
.
$D(-6) = Seed_2(6) = 0$

2. The binary feedback coefficients are specified by FeedbackList, which is a list of

feedback coefficients. The coefficients are specified by listing the locations where the feedback coefficients equal 1. For example, the recurrence relation
$D(n) = (D(n - 7) + D(n - 3) + D(n - 2) + D(n - 1))\bmod 2$
is specified by the list [7, 3, 2, 1].
The table below includes an extensive list of feedback coefficients for linear feedback shift registers showing one or more alternate feedback connections for a given number of stages.

Feedback Connections for Linear m-Sequences

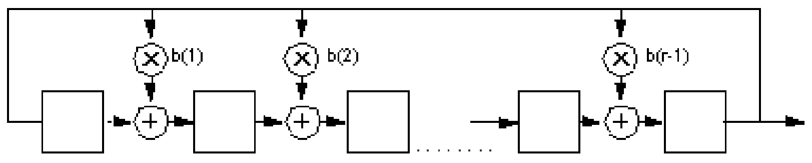
Number of Stages	Code Length	Maximal Taps
2 [a]	3	[2, 1]
3 [a]	7	[3, 1]
4	15	[4, 1]
5 [a]	31	[5, 2] [5, 4, 3, 2] [5, 4, 2, 1]
6	63	[6, 1] [6, 5, 2, 1,] [6, 5, 3, 2,]
7 [a]	127	[7, 1] [7, 3] [7, 3, 2, 1,] [7, 4, 3, 2,] [7, 6, 4, 2] [7, 6, 3, 1] [7, 6, 5, 2] [7, 6, 5, 4, 2, 1] [7, 5, 4, 3, 2, 1]
8	255	[8, 4, 3, 2] [8, 6, 5, 3] [8, 6, 5, 2] [8, 5, 3, 1] [8, 6, 5, 1] [8, 7, 6, 1] [8, 7, 6, 5, 2, 1] [8, 6, 4, 3, 2, 1]
9	511	[9, 4] [9, 6, 4, 3] [9, 8, 5, 4] [9, 8, 4, 1] [9, 5, 3, 2] [9, 8, 6, 5] [9, 8, 7, 2] [9, 6, 5, 4, 2] [9, 7, 6, 4, 3, 1] [9, 8, 7, 6, 5, 3]
10	1023	[10, 3] [10, 8, 3, 2] [10, 4, 3, 1] [10, 8, 5, 1] [10, 8, 5, 4] [10, 9, 4, 1] [10, 8, 4, 3] [10, 5, 3, 2] [10, 5, 2, 1] [10, 9, 4, 2]
11	2047	[11, 2] [11, 8, 5, 2] [11, 7, 3, 2] [11, 5, 3, 5] [11, 10, 3, 2] [11, 6, 5, 1] [11, 5, 3, 1] [11, 9, 4, 1] [11, 8, 6, 2] [11, 9, 8, 3]
12	4095	[12, 6, 4, 1] [12, 9, 3, 2] [12, 11, 10, 5, 2, 1] [12, 11, 6, 4, 2, 1] [12, 11, 9, 7, 6, 5] [12, 11, 9, 5, 3, 1] [12, 11, 9, 8, 7, 4] [12, 11, 9, 7, 6, 5] [12, 9, 8, 3, 2, 1] [12, 10, 9, 8, 6, 2]
13 [a]	8191	[13, 4, 3, 1] [13, 10, 9, 7, 5, 4] [13, 11, 8, 7, 4, 1] [13, 12, 8, 7, 6, 5] [13, 9, 8, 7, 5, 1] [13, 12, 6, 5, 4, 3] [13, 12, 11, 9, 5, 3] [13, 12, 11, 5, 2, 1] [13, 12, 9, 8, 4, 2] [13, 8, 7, 4, 3, 2]
14	16,383	[14, 12, 2, 1] [14, 13, 4, 2] [14, 13, 11, 9] [14, 10, 6, 1] [14, 11, 6, 1] [14, 12, 11, 1] [14, 6, 4, 2] [14, 11, 9, 6, 5, 2] [14, 13, 6, 5, 3, 1] [14, 13, 12, 8, 4, 1] [14, 8, 7, 6, 4, 2] [14, 10, 6, 5, 4, 1] [14, 13, 12, 7, 6, 3] [14, 13, 11, 10, 8, 3]
15	32,767	[15, 1] [15, 4] [15, 13, 10, 9] [15, 13, 10, 1] [15, 14, 9, 2] [15, 9, 4, 1] [15, 12, 3, 1] [15, 10, 5, 4] [15, 10, 5, 4, 3, 2] [15, 11, 7, 6, 2, 1] [15, 7, 6, 3, 2, 1][15, 10, 9, 8, 5, 3] [15, 12, 5, 4, 3, 2] [15, 10, 9, 7, 5, 3] [15, 13, 12, 10] [15, 13, 10, 2] [15, 12, 9, 1] [15, 14, 12, 2] [15, 13, 9, 6] [15, 7, 4, 1] [15, 13, 7, 4]
16	65,535	[16, 12, 3, 1] [16, 12, 9, 6] [16, 9, 4, 3] [16, 12, 7, 2] [16, 10, 7, 6] [16, 15, 7, 2] [16, 9, 5, 2] [16, 13, 9, 6] [16, 15, 4, 2] [16, 15, 9, 4]
17 [a]	131,071	[17, 3] [17, 3, 2] [17, 7, 4, 3] [17, 16, 3, 1] [17, 12, 6, 3, 2, 1] [17, 8, 7, 6, 4, 3] [17, 11, 8, 6, 4, 2] [17, 9, 8, 6, 4, 1] [17, 16, 14, 10, 3, 2] [17, 12, 11, 8, 5, 2]

18	262,143	[18, 7] [18, 10, 7, 5] [18, 13, 11, 9, 8, 7, 6, 3] [18, 17, 16, 15, 10, 9, 8, 7] [18, 15, 12, 11, 9, 8, 7, 6]
19 [a]	524,287	[19, 5, 2, 1] [19, 13, 8, 5, 4, 3] [19, 12, 10, 9, 7, 3] [19, 17, 15, 14, 13, 12, 6, 1] [19, 17, 15, 14, 13, 9, 8, 4, 2, 1] [19, 16, 13, 11, 19, 9, 4, 1] [19, 9, 8, 7, 6, 3] [19, 16, 15, 13, 12, 9, 5, 4, 2, 1] [19, 18, 15, 14, 11, 10, 8, 5, 3, 2] [19, 18, 17, 16, 12, 7, 6, 5, 3, 1]
20	1, 048,575	[20, 3] [20, 9, 5, 3] [20, 19, 4, 3] [20, 11, 8, 6, 3, 2] [20, 17, 14, 10, 7, 4, 3, 2]
21	2,097,151	[21, 2] [21, 14, 7, 2] [21, 13, 5, 2] [21, 14, 7, 6, 3, 2] [21, 8, 7, 4, 3, 2] [21, 10, 6, 4, 3, 2] [21, 15, 10, 9, 5, 4, 3, 2] [21, 14, 12, 7, 6, 4, 3, 2] [21, 20, 19, 18, 5, 4, 3, 2]
22	4,194,303	[22,1] [22, 9, 5, 1] [22, 20, 18, 16,6, 4, 2, 1] [22, 19, 16, 13, 10, 7, 4, 1] [22, 17, 9, 7, 2, 1] [22, 17, 13, 12, 8, 7, 2, 1] [22, 14, 13, 12, 7, 3, 2, 1]
23	8,388,607	[23, 5] [23, 17, 11, 5] [23, 5, 4, 1] [23, 12, 5, 4] [23, 21, 7, 5] [23, 16, 13, 6, 5, 3] [23, 11, 10, 7, 6, 5] [23, 15, 10, 9, 7, 5, 4, 3] [23, 17, 11, 9, 8, 5, 4, 1] [23, 18, 16, 13, 11, 8, 5, 2]
24	16,777,215	[24, 7, 2] [24, 4, 3, 1] [24, 22, 20, 18, 16, 14, 11, 9, 8, 7, 5, 4] [24, 21, 19, 18, 17, 16, 15, 14, 13, 10, 9, 5, 4, 1]
25	33,554, 431	[25, 3] [25, 3, 2, 1] [25, 20, 5, 3] [25, 12, 5, 4] [25, 17, 10, 3, 2, 1] [25, 23, 21, 19, 9, 7, 5, 3] [25, 18, 12, 11, 6, 5, 4] [25, 20, 16, 11, 5, 3, 2, 1] [25, 12, 11, 8, 7, 6, 4, 3]
26	67,108,863	[26, 6, 2, 1] [26, 22, 21, 16, 12, 11, 10, 8, 5, 4, 3, 1]
27	134,217,727	[27, 5, 2, 1] [27, 18, 11, 10, 9, 5, 4, 3]
28	268,435,455	[28, 3] [28, 13, 11, 9, 5, 3] [28, 22, 11, 10, 4, 3] [28, 24, 20, 16, 12, 8, 4, 3, 2, 1]
29	536,870,911	[29, 2] [29, 20, 11, 2] [29, 13, 7, 2] [29, 21, 5, 2] [29, 26, 5, 2] [29, 19, 16, 6, 3, 2] [29, 18, 14, 6, 3, 2]
30	1,073,741,823	[30, 23, 2, 1] [30, 6, 4, 1] [30, 24, 20, 16, 14, 13, 11, 7, 2, 1]
31 [a]	2,147,483,647	[31, 29, 21, 17] [31, 28, 19, 15] [31, 3] [31, 3, 2, 1] [31, 13, 8, 3] [31, 21, 12, 3, 2, 1] [31, 20, 18, 7, 5, 3] [31, 30, 29, 25] [31, 28, 24, 10] [31, 20, 15, 5, 4, 3] [31, 16, 8, 4, 3, 2]
32	4,294,967,295	[32, 22, 2, 1] [32, 7, 5, 3, 2, 1] [32, 28, 19, 18, 16, 14, 11, 10, 9, 6, 5, 1]
33	8,589,934,591	[33, 13] [33, 22, 13, 11] [33, 26, 14, 10] [33, 6, 4, 1] [33, 22, 16, 13, 11, 8]
34	17,179,869,183	[34,27,2,1]
35	34,359,738,367	[35,33]
36	68,719,476,735	[36,25]
37	137,438,953,471	[37,5,4,3,2,1]
38	274,877,906,943	[38,6,5,1]
39	549,755,813,887	[39,35]
40	1,099,511,627,776	[40,38,21,19]
41	2,199,023,255,551	[41,38]
42	4,398,046,511,103	[42,41,20,19]
43	8,796,093,022,207	[43,42,38,37]
44	17,592,186,044,415	[44,43,18,17]
45	35,184,372,088,831	[45,44,42,41]
46	70,368,744,177,663	[46,45,26,25]

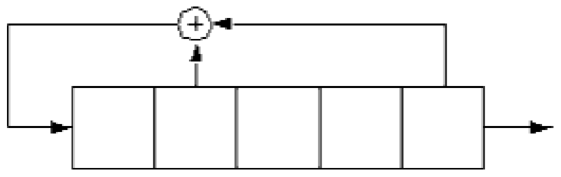
47	140,737,488,355,327	[47,42]
48	281,474,976,710,656	[48,47,21,20]
49	562,949,953,421,312	[49,40]
50	1,125,899,906,84,2623	[50,49,24,23]
51	2,251,799,813,685,248	[51,50,36,35]
52	4,503,599,627,370,496	[52,49]
53	9,007,199,254,740,991	[53,52,38,37]
54	18,014,398,509,481,983	[54,53,18,17]
55	36,028,797,018,963,967	[55,31]
56	72,057,594,037,927,935	[56,55,35,34]
57	144,115,188,075,855,871	[57,50]
58	288,230,376,151,711,743	[58,39]
59	576,460,752,303,423,488	[59,58,38,37]
60	1,152,921,504,606,846,975	[60,59]
61	2,305,843,009,213,693,951	[61, 5, 2, 1]
62	4,611,686,018,427,387,903	[62,61,6,5]
63	9,223,372,036,854,775,807	[63,62]] [[33, 13]
64	18,446,744,073,709,551,615	[64,63,61,60]

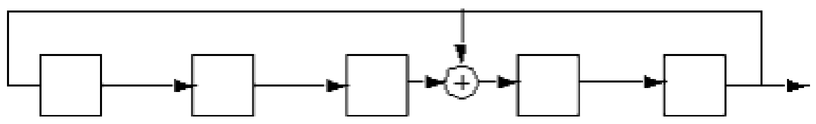
3. An alternative implementation of the LFSR is shown in Alternative Implementation of LFSR. In order to get the same output sequence from the two implementations the following relationships should hold between a(i) and b(i):
b(i) = a(r − i), i = 1, 2, ... , r − 1.
Implementation of 5-Stage LFSR illustrates implementation for a shift register of length 5 and FeedbackList = "2 5".
The sequence of the LFSR states in both implementations and the output (rightmost bit of the state) is shown in LFSR States. The initial state was assumed to be 10000. Although the shift register in the two implementations does not go through the same sequence of states, the output sequence is the same for both. It is also worth noting that if the initial state is different from 10000, the output sequences may not be exactly the same but a shifted version of each other.

Alternative Implementation of LFSR

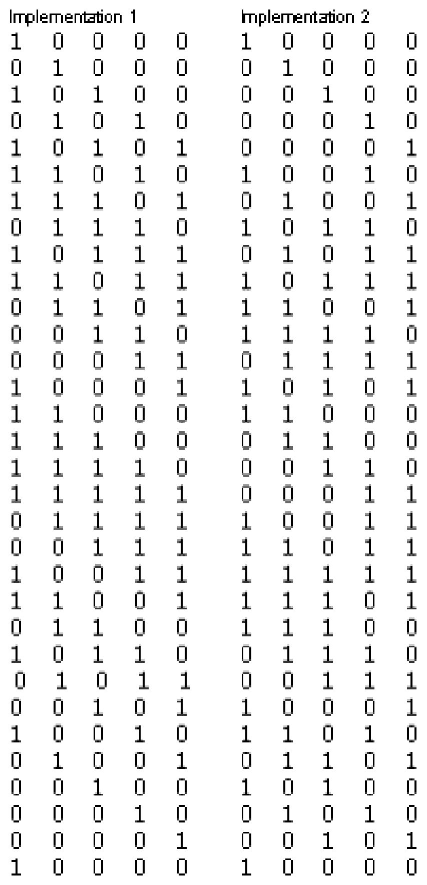


Implementation of 5-Stage LFSR


Implementation 1


Implementation 2

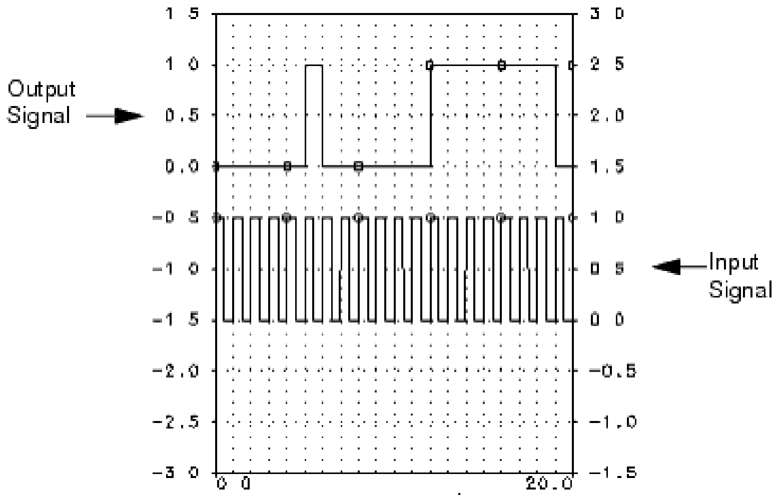
LFSR States



Implementation 1					Implementation 2				
1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	0	0	1	0
1	0	1	0	1	0	0	0	0	1
1	1	0	1	0	1	0	0	1	0
1	1	1	0	1	0	1	0	0	1
0	1	1	1	0	1	0	1	1	0
1	0	1	1	1	0	1	0	1	1
1	1	0	1	1	1	0	1	1	1
0	1	1	0	1	1	1	0	0	1
0	0	1	1	0	1	1	1	1	0
0	0	0	1	1	0	1	1	1	1
1	0	0	0	1	1	0	1	0	1
1	1	0	0	0	1	1	0	0	0
1	1	1	0	0	0	1	1	0	0
1	1	1	1	0	0	0	1	1	0
1	1	1	1	1	0	0	0	1	1
0	1	1	1	1	1	0	0	1	1
0	0	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	0	1
0	1	1	0	0	1	1	1	0	0
1	0	1	1	0	0	1	1	1	0
0	1	0	1	1	0	0	1	1	1
0	0	1	0	1	1	0	0	0	1
1	0	0	1	0	1	1	0	1	0
0	1	0	0	1	0	1	1	0	1
0	0	1	0	0	1	0	1	0	0
0	0	0	1	0	0	1	0	1	0
0	0	0	0	1	0	0	1	0	1
1	0	0	0	0	1	0	0	0	0

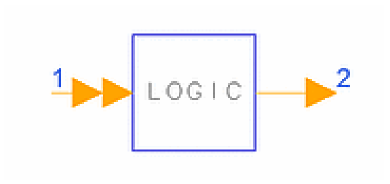
4. Input and output signal voltages of the LFSR component are shown below.

LFSR Input and Output Signal Voltages



5. This component has been upgraded in ADS2005A. In earlier versions of ADS (before ADS2005A), the maximum code length was 2,147,483,647 and the number of stages was less than 32. Starting with ADS2005A, the maximum code length is 18,446,744,073,709,551,615 and the maximum number of stages is 64.
   For the Seed parameter, designers can now specify a binary sequence to set the initial signal stages up to 64 bits. (Before ADS2005A, Seed was specified by an integer number that limited this component to support code lengths less than 32.)
6. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

Logic



Description: test logic
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Logic	test logic: NOT, AND, NAND, OR, NOR, XOR, XNOR	AND		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

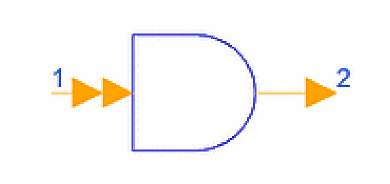
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. Logic applies a logical operation to all inputs. The inputs are integers interpreted as Boolean values.
2. The NOT operation requires only one input.
3. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicAND



Description: Multiple input logical AND function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

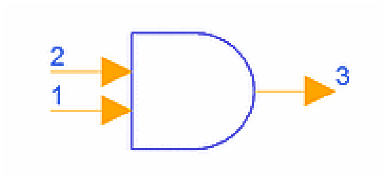
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicAND applies the AND logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicAND2



Description: 2-Input Logical AND Function
Library: Numeric, Logic
Class: SDFLogicAND2
C++ Code

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

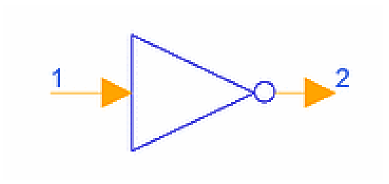
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicAND2 applies the AND logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicInverter



Description: Logic inverter
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

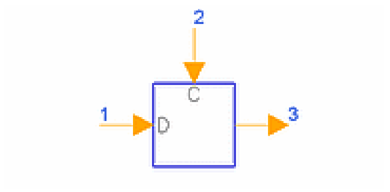
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicInverter applies the logic inversion operation on the input.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicLatch



Description: Logic Latch
Library: Numeric, Logic
Class: SDFLogicLatch
Derived From: baseOmniSysNumericStar

Pin Inputs

Pin	Name	Description	Signal Type
1	data	input data	int
2	clock	clock signal	int

Pin Outputs

Pin	Name	Description	Signal Type
3	output	output data	int

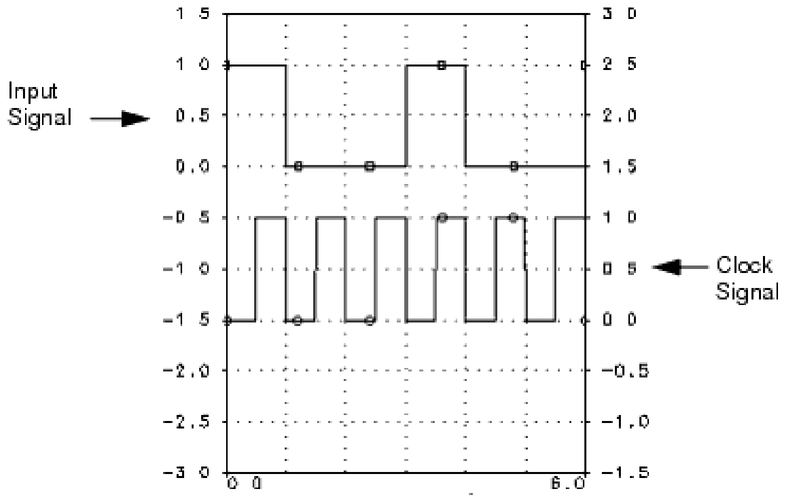
Notes/Equations

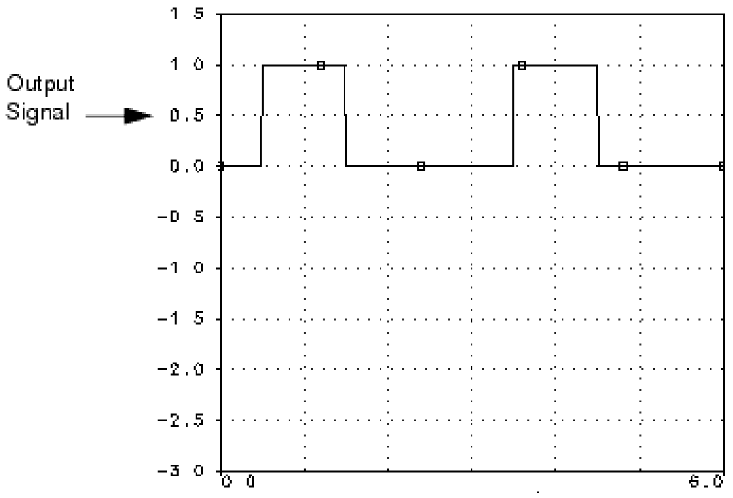
1. Function table:

Input		Output
Data (pin 1)	**Clock (pin 2)**	**Q (pin 3)**
L	H	L
H	H	H
x	L	Q0
whereClock = input clock, active with logic high levelx = don't care stateL = logic low level; Inputs < 0.5; Outputs 0.0H = logic high level; Inputs > 0.5; Outputs 1.0Q0 = previous Q state		

   Initially, at the first sample, the output Q is equal to L.
2. This component is clock level sensitive. If the designer prefers a clock edge-triggered latch, the DFF component can be used with S = R = H.
3. The input, clock, and output signal voltages of the LogicLatch component are shown below.

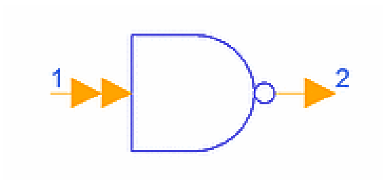
LogicLatch Input and Output Signal Values





4. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicNAND



Description: Multiple input logical NAND function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

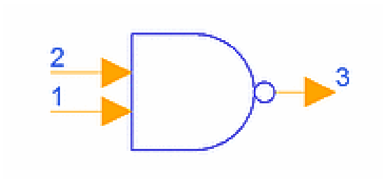
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicNAND applies the NAND logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicNAND2



Description: 2-Input Logical NAND Function
Library: Numeric, Logic
Class: SDFLogicNAND2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

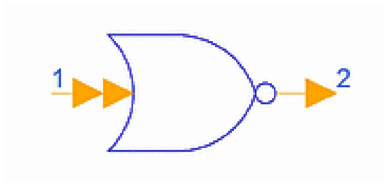
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicNAND2 applies the NAND logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicNOR



Description: Multiple input logical NOR function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

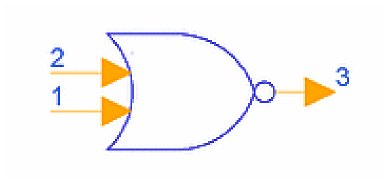
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicNOR applies the NOR logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicNOR2



Description: 2-Input Logical NOR Function
Library: Numeric, Logic
Class: SDFLogicNOR2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

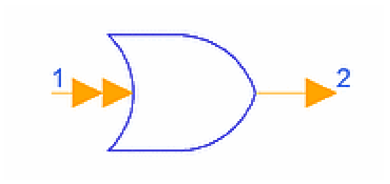
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicNOR2 applies the NOR logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicOR



Description: Multiple input logical OR function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

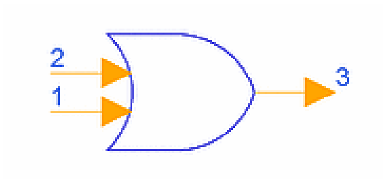
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicOR applies the OR logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicOR2



Description: 2-Input Logical OR Function
Library: Numeric, Logic
Class: SDFLogicOR2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

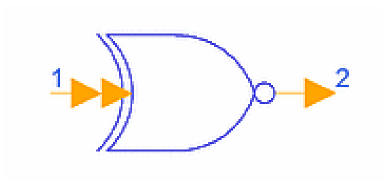
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicOR2 applies the OR logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicXNOR



Description: Multiple input logical XNOR function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

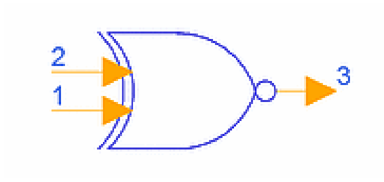
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicXNOR applies the XNOR logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicXNOR2



Description: 2-Input Logical XNOR Function
Library: Numeric, Logic
Class: SDFLogicXNOR2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

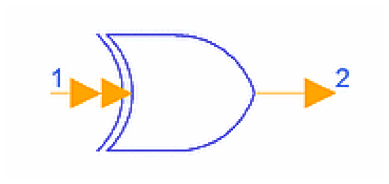
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicXNOR2 applies the XNOR logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicXOR



Description: Multiple input logical XOR function
Library: Numeric, Logic
Class: SDFLogic
C++ Code: See *doc/sp_items/SDFLogic.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input logic values.	multiple int

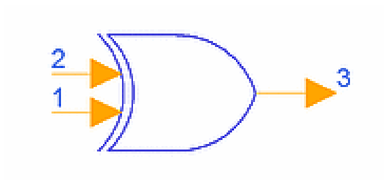
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Result of the logic test, with FALSE equal to zero and TRUE equal to a\nnon-zero integer (not necessarily 1).	int

Notes/Equations

1. LogicXOR applies the XOR logical operation to all inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

LogicXOR2



Description: 2-Input Logical XOR Function
Library: Numeric, Logic
Class: SDFLogicXOR2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

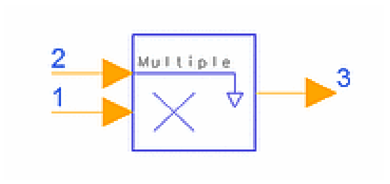
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. LogicXOR2 applies the XOR logical operation to both inputs.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

Multiple



Description: Multiple Test
Library: Numeric, Logic
Class: SDFMultiple
C++ Code: See *doc/sp_items/SDFMultiple.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	signal	Is this a multiple of the other input?	int
2	test	Reference input (must be positive)	int

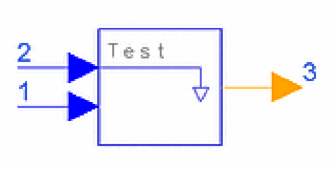
Pin Outputs

Pin	Name	Description	Signal Type
3	mult	Equals 1 if signal is a multiple of test	int

Notes/Equations

1. Multiple outputs a logic high if the signal is an integer multiple of test; otherwise output is a logic low.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

Test



Description: Comparison test
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
Condition	test condition: EQ, NE, GT, GE, LT, LE	EQ	enum	
Tolerance	finite-precision parameter for EQ and NE conditions only	0.0	real	(-∞, ∞)
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False	enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

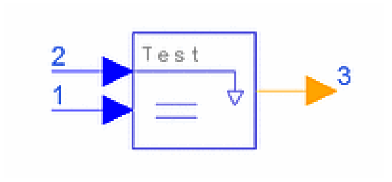
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. For EQ condition, Test outputs 1 if the following expression is satisfied (otherwise output is 0):
   test − signal|≥ Tolerance
   For NE condition, Test outputs 1 if the following expression is satisfied (otherwise output is 0):
   test − signal|< Tolerance
   For GT, GE, LT, or LE condition, Test outputs 1 if the following expression is satisfied (otherwise output is 0):
   (test) condition (signal)
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestEQ



Description: Comparision test (equal to)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Tolerance	finite-precision parameter for EQ and NE conditions only	0.0		real	(-∞, ∞)
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

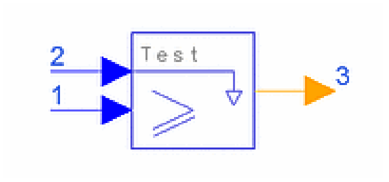
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestEQ outputs 1 if the following expression is satisfied (otherwise output is 0):
   $|test - signal| \leq Tolerance$
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestGE



Description: Comparision test (greater than or equal to)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

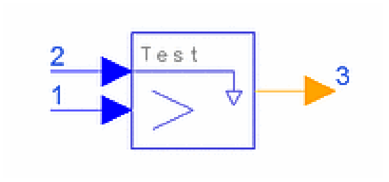
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestGE outputs 1 if the following expression is satisfied (otherwise output is 0): (signal) GE (test)
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestGT



Description: Comparision test (greater than)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

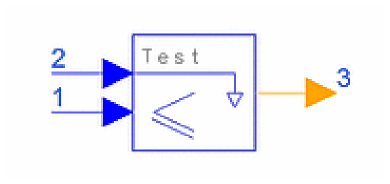
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestGT outputs 1 if the expression
   (signal) GT (test)
   is satisfied; otherwise output is 0.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestLE



Description: Comparision test (less than or equal to)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

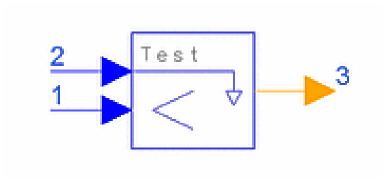
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestLE outputs 1 if the expression
   (signal) LE (test)
   is satisfied; otherwise output is 0.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestLT



Description: Comparision test (less than)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

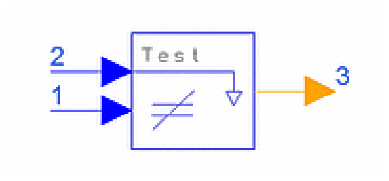
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestLT outputs 1 if the expression
   (signal) LT (test)
   is satisfied; otherwise output is 0.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

TestNE



Description: Comparision test (not equal to)
Library: Numeric, Logic
Class: SDFTest
C++ Code: See *doc/sp_items/SDFTest.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Tolerance	finite-precision parameter for EQ and NE conditions only	0.0		real	(-∞, ∞)
CrossingsOnly	if True, output is True only when the test result toggles: False, True	False		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	Signal	Signal to compare against the test (left hand side)	real
2	Test	Comparison test	real

Pin Outputs

Pin	Name	Description	Signal Type
3	output	Result of the test	int

Notes/Equations

1. TestNE outputs 1 if the expression test − signal|> Tolerance is satisfied; otherwise, output is 0.
2. For general information regarding numeric logic component signals, refer to *Numeric Logic Components* (numeric).

Numeric Math Components

- *Abs* (numeric)
- *Add* (numeric)
- *Add2* (numeric)
- *AddCx* (numeric)
- *AddCx2* (numeric)
- *AddFix* (numeric)
- *AddFix2* (numeric)
- *AddInt* (numeric)
- *AddInt2* (numeric)
- *Average* (numeric)
- *AverageCx* (numeric)
- *AverageCxWOffset* (numeric)
- *Cos* (numeric)
- *DB* (numeric)
- *DivByInt* (numeric)
- *Exp* (numeric)
- *Floor* (numeric)
- *Gain* (numeric)
- *GainCx* (numeric)
- *GainFix* (numeric)
- *GainInt* (numeric)
- *Integrate* (numeric)
- *Ln* (numeric)
- *Math* (numeric)
- *MathCx* (numeric)
- *MaxMin* (numeric)
- *Modulo* (numeric)
- *ModuloInt* (numeric)
- *Mpy* (numeric)
- *Mpy2* (numeric)
- *MpyCx* (numeric)
- *MpyCx2* (numeric)
- *MpyFix* (numeric)
- *MpyFix2* (numeric)
- *MpyInt* (numeric)
- *MpyInt2* (numeric)
- *Reciprocal* (numeric)
- *SDC1* (numeric)
- *SDC2* (numeric)
- *SDC3* (numeric)
- *SDC4* (numeric)
- *SDCCx1* (numeric)
- *SDCCx2* (numeric)
- *SDCCx3* (numeric)
- *SDCCx4* (numeric)
- *Sgn* (numeric)
- *Sin* (numeric)
- *Sinc* (numeric)
- *Sqrt* (numeric)

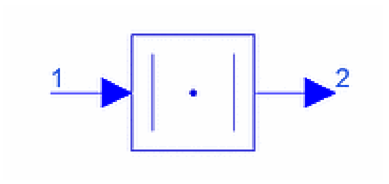
- *Sub* (numeric)
- *SubCx* (numeric)
- *SubFix* (numeric)
- *SubInt* (numeric)
- *Trig* (numeric)
- *TrigCx* (numeric)
- *Variance* (numeric)

The Numeric Math components library contains integer, double precision floating-point (real), fixed-point (fixed), and complex mathematical scalar operators. Each component accepts a specific class of signal and outputs a resultant signal. (These components do not accept any matrix class of signal.)

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. The auto conversion from timed, complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30, and a value of 0.5 would create one of precision of 1.31. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components operate with fixed-point numbers. These components use one or more parameters that define the characteristics of the fixed-point processing. These parameters include: OverflowHandler, OutputPrecision, RoundFix, ReportOverflow, and others. For details on the use of these parameters for fixed-point components refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. The arithmetic used by these components is two's complement. Therefore, all precision values must specify at least one bit to the left of the decimal point (used as sign bit).

Abs



Description: Absolute Value
Library: Numeric, Math
Class: SDFAbs
C++ Code: See *doc/sp_items/SDFAbs.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

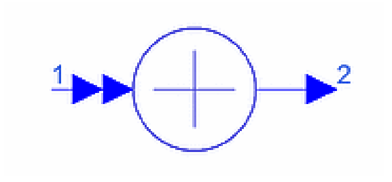
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Abs outputs the absolute value of the input as a floating-point (real) value.
   $y(n) = |x(n)|$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*
2. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

Add



Description: Multiple Input Adder
Library: Numeric, Math
Class: SDFAdd
C++ Code: See *doc/sp_items/SDFAdd.html* under your installation directory.

Pin Inputs

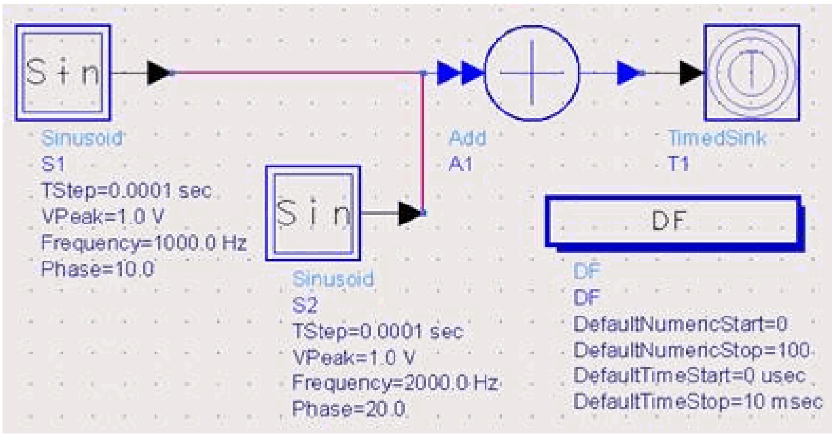
Pin	Name	Description	Signal Type
1	input		multiple real

Pin Outputs

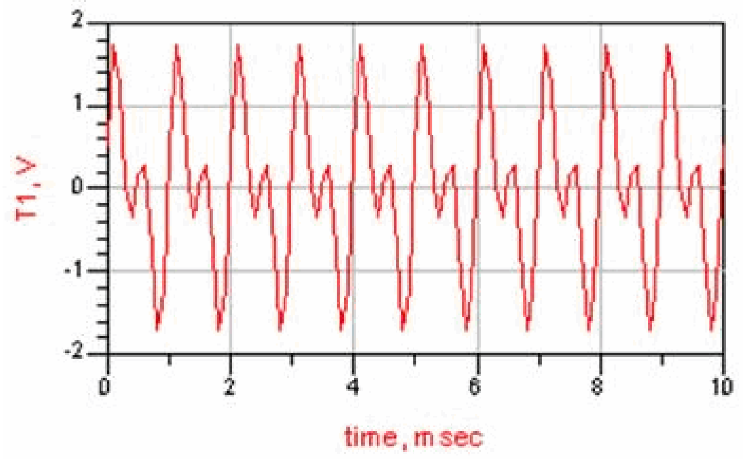
Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Add outputs the sum of inputs as a floating-point (real) value.
2. Two source outputs connected to the Add input as shown in the add schematic below:

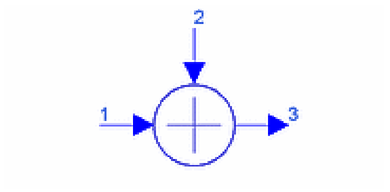


3. The Add output is shown in the add plot below:.



4. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

Add2



Description: 2-Input Adder
Library: Numeric, Math
Class: SDFAdd2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		real
2	input2		real

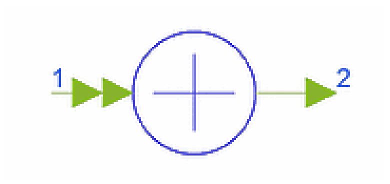
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real

Notes

1. Add2 outputs the sum of the two inputs as a floating-point (real) value.
2. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

AddCx



Description: Complex Multiple Input Adder
Library: Numeric, Math
Class: SDFAddCx
C++ Code: See *doc/sp_items/SDFAddCx.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple complex

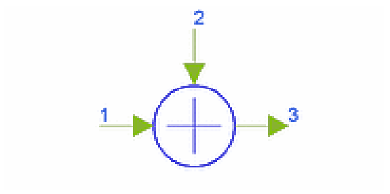
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. AddCx outputs the sum of inputs as a complex value.
2. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

AddCx2



Description: 2-Input Complex Adder
Library: Numeric, Math
Class: SDFAddCx2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		complex
2	input2		complex

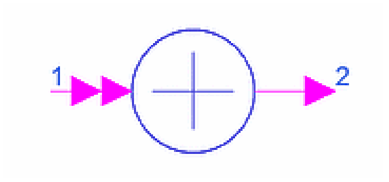
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex

Notes

1. AddCx2 outputs the sum of the two inputs as a complex value.
2. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

AddFix



Description: Fixed-Point Multiple Input Adder
Library: Numeric, Math
Class: SDFAddFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFAddFix.html* under your installation directory.

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple fix

Pin Outputs

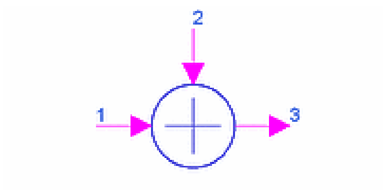
Pin	Name	Description	Signal Type
2	output		fix

Notes/Equations

1. AddFix outputs the sum of inputs as a fixed-point value.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

4. For general information regarding numeric math component signals, refer to *Numeric Math Components* (numeric).

AddFix2



Description: 2-Input Fixed-Point Adder
Library: Numeric, Math
Class: SDFAddFix2

Parameters

Name	Description	Default	Unit	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero saturate, warning	wrapped		enum	
ReportOverflow	simulation overflow error report: DONT REPORT, REPORT	REPORT		enum	
RoundFix	fixed-point computations, assignments, and data type conversions: TRUNCATE, ROUND	TRUNCATE		enum	
UseArrivingPrecision	use precision of arriving data: NO, YES	NO		enum	
InputPrecision	precision of input(used only if UseArrivingPrecision is set to NO)	2.14		precision	
OutputPrecision	precision of output accumulation	2.14		precision	

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		fix
2	input2		fix

Pin Outputs

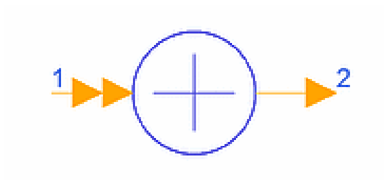
Pin	Name	Description	Signal Type
3	output		fix

Notes/Equations

1. AddFix2 outputs the sum of the two inputs as a fixed-point value with precision specified by OutputPrecision.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example,

when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

AddInt



Description: Integer Multiple Input Adder
Library: Numeric, Math
Class: SDFAddInt
C++ Code: See *doc/sp_items/SDFAddInt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple int

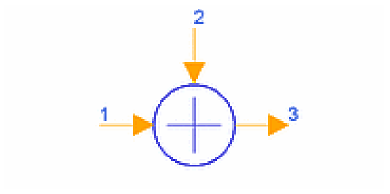
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. AddInt outputs the sum of inputs as an integer value.

AddInt2



Description: 2-Input Integer Adder
Library: Numeric, Math
Class: SDFAddInt2

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		int
2	input2		int

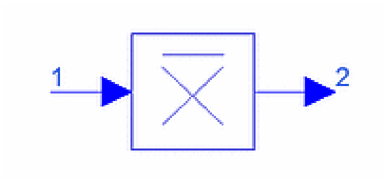
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes

1. AddInt2 outputs the sum of the two inputs as an integer value.

Average



Description: Averager
Library: Numeric, Math
Class: SDFAverage
C++ Code: See *doc/sp_items/SDFAverage.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
NumInputsToAverage	number of input data items to average	8		int	[1, ∞)
BlockSize	input blocks of this size will be averaged to produce an output block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

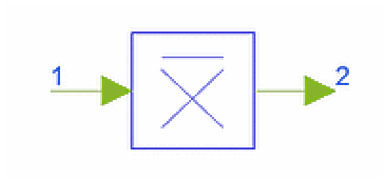
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Average calculates the output floating-point (real) average for a specified number of input samples or blocks of input samples. Blocks of successive input samples are treated as vectors and produce a block of output values.

AverageCx



Description: Complex averager
Library: Numeric, Math
Class: SDFAverageCx
C++ Code: See *doc/sp_items/SDFAverageCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
NumInputsToAverage	number of input data items to average	8		int	[1, ∞)
BlockSize	input blocks of this size will be averaged to produce an output block	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex

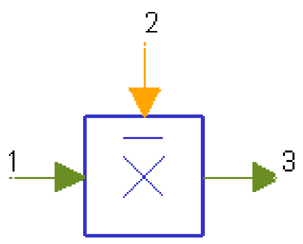
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. AverageCx calculates the output complex average for a specified number of input samples or blocks of complex input samples. Blocks of successive input samples are treated as vectors and produce a block of output values.

AverageCxWOffset



Description: Average Complex data with detected delay information
Library: Numeric, Math
Class: SDFAverageCxWOffset

Parameters

Name	Description	Default	Unit	Type	Range
NumSymToAverage	Number of symbols to average	256		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	Input	Input	complex
2	Offset	Offset	int

Pin Outputs

Pin	Name	Description	Signal Type
3	Output	Output	complex

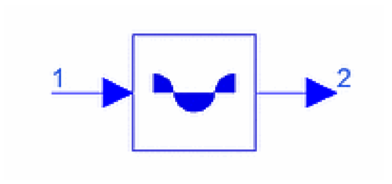
Notes/Equations

1. This component averages the RF received data using detected RF channel delay information.
2. This is a single-rate component. Each firing, one input token is consumed for both Input pin 1 and Offset pin 2, and one output token is produced.
3. Averaging is performed on pin 1 input data using pin 2 detected delay information. The output is the averaged complex signal envelope.
   For example, a DelayEstimator component can be used with AverageCxWOffset and the detected delay sent from DelayEstimator; at the AverageCxWOffset output, the average value is held constant for each NumSymToAverage sample.

References

1. M. Jeruchim, P. Balaban and K. Shanmugan, "Simulation of Communication System," Plenum Press, New York and London, 1992.

Cos



Description: Cosine Function
Library: Numeric, Math
Class: SDFCos
C++ Code: See *doc/sp_items/SDFCos.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

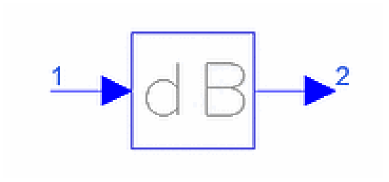
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Cos calculates the cosine of its input, which is assumed to be an angle in radians.
   $y(n) = \cos(x(n))$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

DB



Description: dB value
Library: Numeric, Math
Class: SDFDB
C++ Code: See *doc/sp_items/SDFDB.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Min	minimum output value	-100		real	(-∞, ∞)
Type	type of input signal measurement: Power as 10*log(input), Amplitude as 20*log(input)	Amplitude as 20*log(input)		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

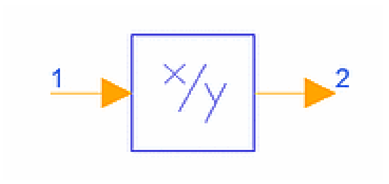
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. DB converts the input value to floating-point (real) dB scale. Zero and negative values are assigned the Min value.
2. If the input signal is a power measurement set Type to Power; if the input signal is an amplitude measurement set Type to Amplitude.
   If Type = Power as 10log(input):
   $$y(n) = \begin{cases} 10\log_{10} x(n) & \text{if } 10\log_{10} x(n) \geq \text{Min} \\ \text{Min} & \text{otherwise} \end{cases}$$
   If Type = Power as 20log(input):
   $$y(n) = \begin{cases} 20\log_{10} x(n) & \text{if } 20\log_{10} x(n) \geq \text{Min} \\ \text{Min} & \text{otherwise} \end{cases}$$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

DivByInt



Description: Integer division
Library: Numeric, Math
Class: SDFDivByInt
C++ Code: See *doc/sp_items/SDFDivByInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Divisor	integer divisor	2		int	(-∞, 0) or (0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

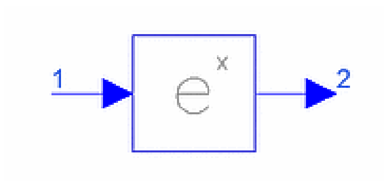
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. DivByInt calculates the integer output equal to the integer input divided by the integer Divisor. Truncated integer division is used.

Exp



Description: Exponential Function
Library: Numeric, Math
Class: SDFExp
C++ Code: See *doc/sp_items/SDFExp.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

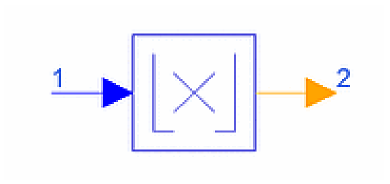
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Exp calculates the floating-point (real) exponential function (base e) of the input.
   $y(n) = e^{x(n)}$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*
2. The input value must be ≤ ln (maximum double-precision floating-point (real) value) to avoid overflow.

Floor



Description: Floor Function
Library: Numeric, Math
Class: SDFFloor
C++ Code: See *doc/sp_items/SDFFloor.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

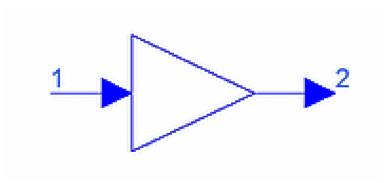
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. Floor outputs the integer floor of the input.
   $y(n) = \text{floor}\,(x(n))$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

Gain



Description: gain value
Library: Numeric, Math
Class: SDFGain
C++ Code: See *doc/sp_items/SDFGain.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Gain	gain value	1.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

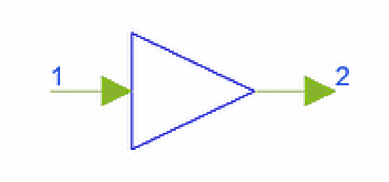
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Gain calculates the floating-point (real) output equal to the input multiplied by Gain.

GainCx



Description: Complex gain
Library: Numeric, Math
Class: SDFGainCx
C++ Code: See *doc/sp_items/SDFGainCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Gain	gain value	1		complex	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex

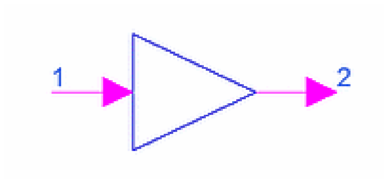
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. GainCx calculates the complex output equal to the input multiplied by the complex Gain.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

GainFix



Description: Fixed-Point Gain
Library: Numeric, Math
Class: SDFGainFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFGainFix.html* under your installation directory.

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
Gain	gain value	1.0	fix
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix

Pin Outputs

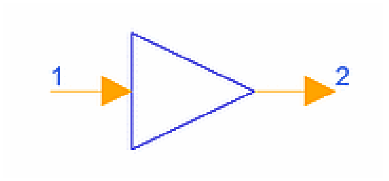
Pin	Name	Description	Signal Type
2	output		fix

Notes/Equations

1. GainFix calculates the fixed-point output equal to the input multiplied by Gain.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a

precision of 1.0 and the output value will be forced to 0.

GainInt



Description: Integer gain
Library: Numeric, Math
Class: SDFGainInt
C++ Code: See *doc/sp_items/SDFGainInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Gain	gain value	1		int	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

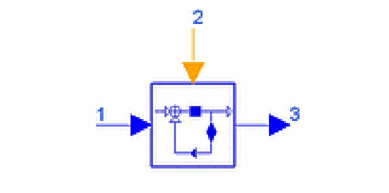
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. GainInt calculates the integer output equal to the input multiplied by the integer Gain.

Integrate



Description: Integrator
Library: Numeric, Math
Class: SDFIntegrate
C++ Code: See *doc/sp_items/SDFIntegrate.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
FeedbackGain	gain on feedback path	1.0		real	(-∞, ∞)
Top	upper limit	0.0		real	(-∞, ∞)
Bottom	lower limit	0.0		real	(-∞, ∞)
Saturate	perform saturation: NO, YES	YES		enum	
State	an internal state	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	data		real
2	reset		int

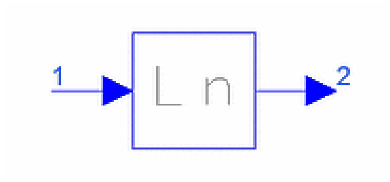
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real

Notes/Equations

1. Integrate calculates the output floating-point (real) summation for a specified number of input samples or blocks of input samples. Blocks of successive input samples are treated as vectors and produce a block of output values.
2. Integrate is an integrator with leakage, limits, and reset. With the default parameters, input samples are simply accumulated, and the running sum is the output. To prevent any resetting in the middle of a run, connect a Const source with value 0 to the reset input. Otherwise, whenever a non-zero is received on this input, the accumulated sum is reset to the current input (that is, no feedback).
3. Limits are controlled by Top and Bottom. If Top ≤ Bottom, no limiting is performed; otherwise, the output is kept between Top and Bottom.
   If Saturate = YES, saturation is performed. If Saturate = NO, wraparound is performed. Limiting is performed before output.
4. Leakage is controlled by the FeedbackGain state. The output is the data input plus FeedbackGain × State, where State is the previous output.

Ln



Description: Natural Log
Library: Numeric, Math
Class: SDFLn
C++ Code: See *doc/sp_items/SDFLn.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

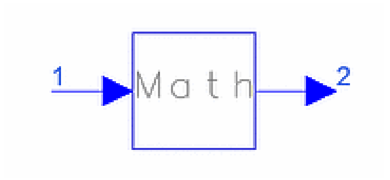
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Ln outputs the floating-point (real) natural logarithm of the input.
   $y(n) = \ln x(n)$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*
2. The input must be > 0.

Math



Description: Math Function
Library: Numeric, Math
Class: SDFMath

Parameters

Name	Description	Default	Unit	Type	Range
Type	mathematical function: Abs, Ceil, Exp, Floor, Ln, Log10, Pow10, Recip, Round, Sqr, Sqrt	Abs		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

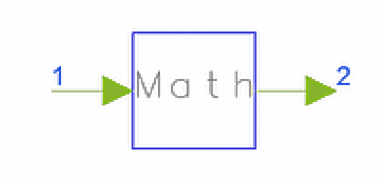
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Math performs the floating-point (real) mathematical functions:
   $y(n) = f(x(n))$
   where:
   $y(n)$ is the output for sample n
   $x(n)$ is the input for sample n
   and where $f(\ )$ is any function that can be selected from the Type parameter.
2. If Type = Abs, then $y(n) = |x(n)|$
   If Type = Ceil, then $y(n) = \lceil x(n) \rceil$, where $x(n) \le \lceil x(n) \rceil < x(n) + 1$
   If Type = Exp, then $y(n) = e^{x(n)}$
   If Type = Floor, then $y(n) = \lfloor x(n) \rfloor$, where $x(n) - 1 < \lfloor x(n) \rfloor \le x(n)$
   If Type = Ln, then $y(n) = \ln(x(n))$
   If Type = Log10, then $y(n) = \log_{10}(x(n))$
   If Type = Pow10, then $y(n) = 10^{x(n)}$
   If Type = Recip, then $y(n) = 1 / x(n)$
   If Type = Round, then $y(n)$ = closest integer to $x(n)$ (numbers at the same distance from two integers map away from 0; for example, 2.5 maps to 3 and −2.5 maps to −3)
   If Type = Sqr, then $y(n) = x(n)^2$
   If Type = Sqrt, then $y(n) = \sqrt{x(n)}$

MathCx



Description: Complex Math Function
Library: Numeric, Math
Class: SDFMathCx
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Type	mathematical function: Abs, Ceil, Exp, Floor, Ln, Log10, Pow10, Recip, Round, Sqr, Sqrt, Conj	Abs		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	complex

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	complex

Notes/Equations

1. MathCx performs the complex mathematical functions:
   y(n) = f(x(n))
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*
   and where *f( )* is any function that can be selected from the Type parameter.
2. If Type = Abs, then *y(n)* = | *x(n)* | = $\sqrt{Re\{x(n)\}^2 + Im\{x(n)\}^2}$
   If Type = Ceil, then *y(n)* = $\lceil Re\{x(n)\} \rceil + j \times \lceil Im\{x(n)\} \rceil$ (see Ceil function of Math component)
   If Type = Exp, then *y(n)* = $e^{x(n)} = e^{Re\{x(n)\}}$ (cos(Im{*x(n)*}) + j sin(Im{*x(n)*}))
   If Type = Floor, then *y(n)* = $\lfloor Re\{x(n)\} \rfloor + j \times \lfloor Im\{x(n)\} \rfloor$ (see Floor function of Math component)
   If Type = Ln, then *y(n)* = $\ln(x(n)) = \ln(|x(n)|) + j \times \angle x(n)$, where $\angle x(n)$ is the phase of *x(n)* in radians.
   If Type = Log10, then *y(n)* = $\log_{10}$ (x(n)) = ln(*x(n)*) / ln(10).
   If Type = Pow10, then *y(n)* = $10^{x(n)} = e^{x(n) \ln(10)}$
   If Type = Recip, then *y(n)* = 1 / *x(n)* = (*Re*{*x* (*n*)} − *j Im* {*x*(*n*)}) / | *x(n)* | ^{2}

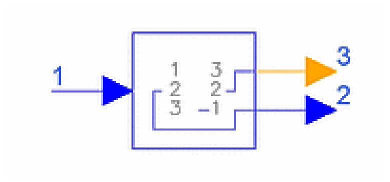
If Type = Round, then y(n) = *Round*(*Re*{*x*(*n*)}) + *j Round* (*Im*{*x*(*n*)}) (see Round function of Math component)

If Type = Sqr, then *y(n)* = *x(n)* ^{2}

If Type = Sqrt, then *y(n)* = $\sqrt{x(n)} = \sqrt{|x(n)|} \times e^{j \times 0.5 \times \angle x(n)}$, where $\angle x(n)$ is the phase of *x(n)* in radians.

If Type = Conj, then *y(n)* = $Re\{x(n)\} - j \;\; Im\{x(n)\}$

MaxMin



Description: Maximum or minimum value
Library: Numeric, Math
Class: SDFMaxMin
C++ Code: See *doc/sp_items/SDFMaxMin.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
N	default samples	10		int	[0, ∞)
MaxOrMin	output value: min, max	max		enum	
Compare	compare input value or magnitude: valueIn, magnitudeIn	valueIn		enum	
OutputType	output value or magnitude: valueOut, magnitudeOut	valueOut		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

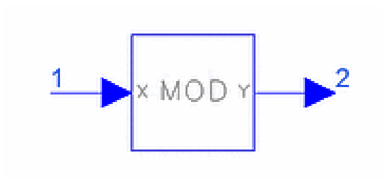
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real
3	index		int

Notes/Equations

1. MaxMin finds the minimum or minimum value or magnitude of a fixed number of data values on the input.
   Use MaxMin to operate over multiple data streams by preceding it with a Commutator and set the N state accordingly.
2. If Compare = valueIn, the input with the maximum or minimum value is located; if Compare = magnitudeIn, the input with the maximum or minimum magnitude is located.
3. If OutputType = magnitudeOut, the magnitude of the result is written to the output; if OutputType = valueOut, the result itself is written to the output. Returns maximum value among N input samples. The index of the output is also provided (count starts at 0).

Modulo



Description: Floating-point modulo
Library: Numeric, Math
Class: SDFModulo
C++ Code: See *doc/sp_items/SDFModulo.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Modulo	modulo value	1.0		real	(-∞, 0) or (0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

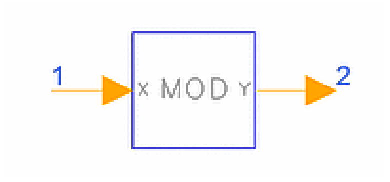
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Modulo outputs the floating-point (real) remainder with the same sign as input after dividing the input by the Modulo parameter.
   $y(n) = \mathrm{fmod}\, x(n)$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

ModuloInt



Description: Integer modulo
Library: Numeric, Math
Class: SDFModuloInt
C++ Code: See *doc/sp_items/SDFModuloInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Modulo	modulo value	10		int	(-∞, 0) or (0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	int

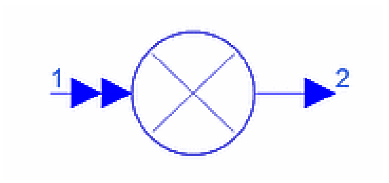
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	int

Notes/Equations

1. ModuloInt outputs the integer remainder with the same sign as input after dividing the input by the integer Modulo parameter.
   $y(n) = \mathrm{mod}\, x(n)$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

Mpy



Description: Multiple Input Multiplier
Library: Numeric, Math
Class: SDFMpy
C++ Code: See *doc/sp_items/SDFMpy.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple real

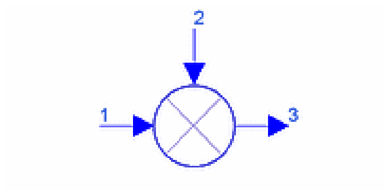
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Mpy outputs the product of inputs as a floating-point (real) value.

Mpy2



Description: 2-Input Multiplier
Library: Numeric, Math
Class: SDFMpy
C++ Code: See *doc/sp_items/SDFMpy.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		real
2	input#2		real

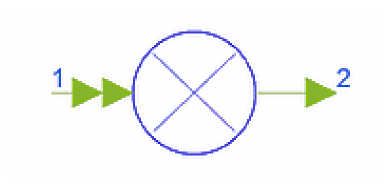
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real

Notes/Equations

1. Mpy2 outputs the product of the two inputs as a floating-point (real) value.

MpyCx



Description: Complex Multiple Input Multiplier
Library: Numeric, Math
Class: SDFMpyCx
C++ Code: See *doc/sp_items/SDFMpyCx.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple complex

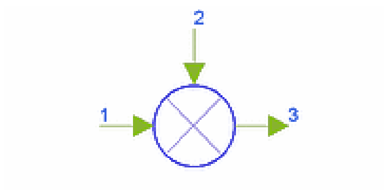
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. MpyCx outputs the product of the complex inputs as a complex value.

MpyCx2



Description: 2-Input Complex Multiplier
Library: Numeric, Math
Class: SDFMpyCx
C++ Code: See *doc/sp_items/SDFMpyCx.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		complex
2	input#2		complex

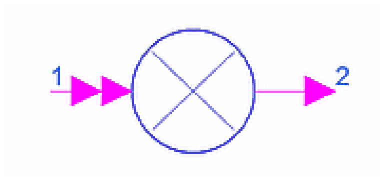
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex

Notes/Equations

1. MpyCx2 outputs the product of two inputs as a complex value.

MpyFix



Description: Fixed-Point Multiple Input Multiplier
Library: Numeric, Math
Class: SDFMpyFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFMpyFix.html* under your installation directory.

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple fix

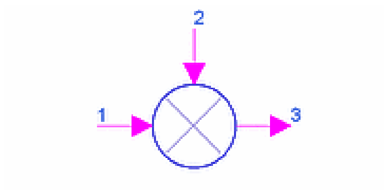
Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix

Notes/Equations

1. MpyFix outputs the product of the inputs as a fixed-point value.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

MpyFix2



Description: 2-Input Fixed-Point Multiplier
Library: Numeric, Math
Class: SDFMpyFix
C++ Code: See *doc/sp_items/SDFMpyFix.html* under your installation directory.

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		fix
2	input#2		fix

Pin Outputs

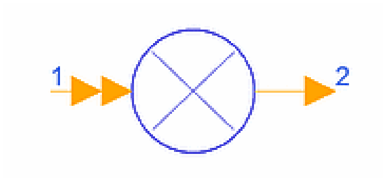
Pin	Name	Description	Signal Type
3	output		fix

Notes/Equations

1. MpyFix2 outputs the product of the two inputs as a fixed-point value.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a

precision of 1.0 and the output value will be forced to 0.

MpyInt



Description: Integer Multiple Input Multiplier
Library: Numeric, Math
Class: SDFMpyInt
C++ Code: See *doc/sp_items/SDFMpyInt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple int

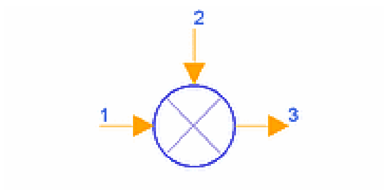
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. MpyInt outputs the product of the inputs as an integer value.

MpyInt2



Description: 2-Input Integer Multiplier
Library: Numeric, Math
Class: SDFMpyInt
C++ Code: See *doc/sp_items/SDFMpyInt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		int
2	input#2		int

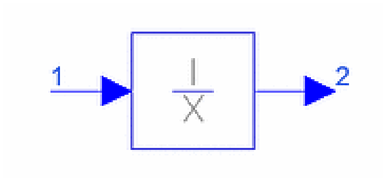
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. MpyInt2 outputs the product of two inputs as an integer value.

Reciprocal



Description: Reciprocal function
Library: Numeric, Math
Class: SDFReciprocal
C++ Code: See *doc/sp_items/SDFReciprocal.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
MagLimit	magnitude limit; non-zero limits the output magnitude	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

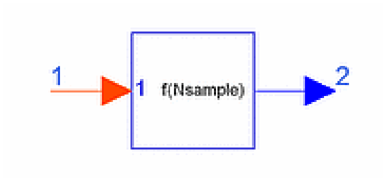
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Reciprocal calculates the reciprocal of the input, with an optional magnitude limit.
   If MagLimit = 0
   $$y(n) = \frac{1}{x(n)}$$
   If MagLimit ≠ 0 and input = 0
   y(n) = MagLimit
   If MagLimit ≠ 0 and input ≠ 0
   $$y(n) = \begin{cases} \text{MagLimit} & \text{if } \frac{1}{x(n)} > \text{MagLimit} \\ -\text{MagLimit} & \text{if } \frac{1}{x(n)} < -\text{MagLimit} \\ \frac{1}{x(n)} & \text{otherwise} \end{cases}$$
   where:
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

SDC1



Description: 1-Input Symbolic Defined Component
Library: Numeric, Math
Class: SDFSDC

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0	real

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype

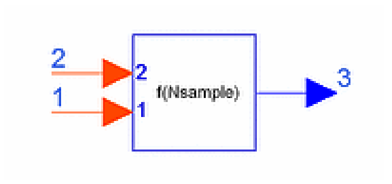
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Numeric output signal	real

Notes/Equations

1. This component generates numeric data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDC2



Description: 2-Input Symbolic Defined Component
Library: Numeric, Math
Class: SDFSDC

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0	real

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype

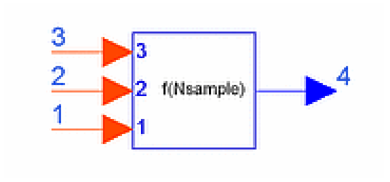
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Numeric output signal	real

Notes/Equations

1. This component generates numeric data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDC3



Description: 3-Input Symbolic Defined Component
Library: Numeric, Math
Class: SDFSDC

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0	real

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype

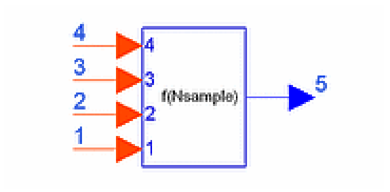
Pin Outputs

Pin	Name	Description	Signal Type
4	output	Numeric output signal	real

Notes/Equations

1. This component generates numeric data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDC4



Description: 4-Input Symbolic Defined Component
Library: Numeric, Math
Class: SDFSDC

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0	real

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype

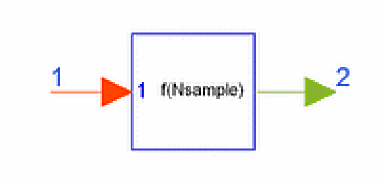
Pin Outputs

Pin	Name	Description	Signal Type
5	output	Numeric output signal	real

Notes/Equations

1. This component generates numeric data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDCCx1



Description: 1-Input Symbolic Defined Component with Complex Output
Library: Numeric, Math
Class: SDFSDCCx

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0+j*0.0	complex

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype

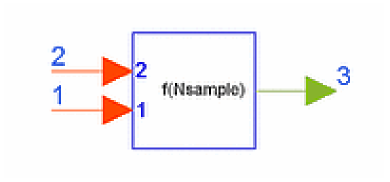
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Numeric output signal	complex

Notes/Equations

1. This component generates complex data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDCCx2



Description: 2-Input Symbolic Defined Component with Complex Output
Library: Numeric, Math
Class: SDFSDCCx

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0+j*0.0	complex

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype

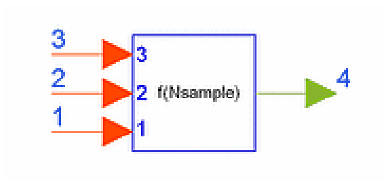
Pin Outputs

Pin	Name	Description	Signal Type
3	output	Numeric output signal	complex

Notes/Equations

1. This component generates complex data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDCCx3



Description: 3-Input Symbolic Defined Component with Complex Output
Library: Numeric, Math
Class: SDFSDCCx

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0+j*0.0	complex

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype

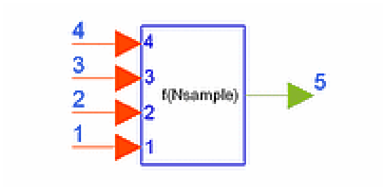
Pin Outputs

Pin	Name	Description	Signal Type
4	output	Numeric output signal	complex

Notes/Equations

1. This component generates complex data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

SDCCx4



Description: 4-Input Symbolic Defined Component with Complex Output
Library: Numeric, Math
Class: SDFSDCCx

Parameters

Name	Description	Default	Type
Expression	Expression, function of inputs	0.0+j*0.0	complex

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		anytype
2	input#2		anytype
3	input#3		anytype
4	input#4		anytype

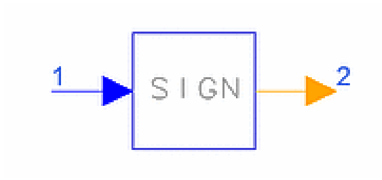
Pin Outputs

Pin	Name	Description	Signal Type
5	output	Numeric output signal	complex

Notes/Equations

1. This component generates complex data output that is evaluated using an expression based on input data. Expression can be any valid expression, following the syntax used for writing expressions on a VAR block.
2. Input data is specified by predefined variables _v1, _v2, etc. where 1 and 2 is the port number. The Expression can also be dependent on predefined variable, *Nsample*, which is incremented for each firing of this component determined by the schedule.

Sgn



Description: Signum Function
Library: Numeric, Math
Class: SDFSgn
C++ Code: See *doc/sp_items/SDFSgn.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

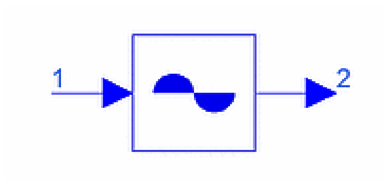
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. Sgn calculates the signum of the input.
   $y(n)$ = sign of $x(n)$
   where
   $y(n)$ is the output for sample n
   $x(n)$ is the input for sample n
2. The output is 1 if $x \geq 0$. The output is -1 if $x < 0$.

Sin



Description: Sine Function
Library: Numeric, Math
Class: SDFSin
C++ Code: See *doc/sp_items/SDFSin.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

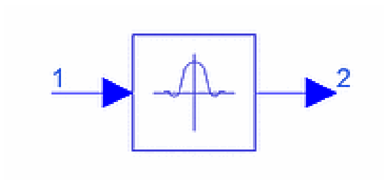
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Sin calculates the sine of its input, which is assumed to be an angle in radians.
   y(n)) = sin *(x(n))*
   where
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*

Sinc



Description: Sinc Function
Library: Numeric, Math
Class: SDFSinc
C++ Code: See *doc/sp_items/SDFSinc.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input	The input x to the sinc function.	real

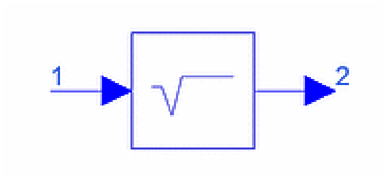
Pin Outputs

Pin	Name	Description	Signal Type
2	output	The output of the sinc function.	real

Notes/Equations

1. Sinc calculates the floating-point (real) sinc of its input given in radians. The sinc function is defined as sin(x)/x, with value 1.0 when x = 0.

Sqrt



Description: Square Root Function
Library: Numeric, Math
Class: SDFSqrt
C++ Code: See *doc/sp_items/SDFSqrt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

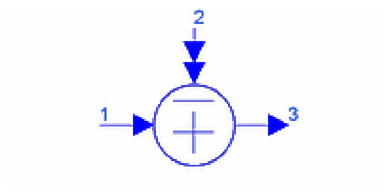
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Sqrt calculates the floating-point (real) square root of the input.
   $y(n) = \sqrt{x(n)}$
   where
   y(n) is the output for sample *n*
   x(n) is the input for sample *n*
2. The input value must be ≥ 0.

Sub



Description: Multiple Input Subtractor
Library: Numeric, Math
Class: SDFSub
C++ Code: See *doc/sp_items/SDFSub.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		real
2	neg		multiple real

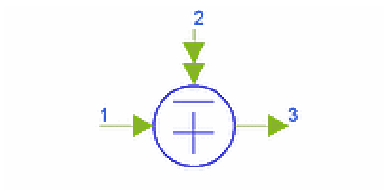
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real

Notes/Equations

1. Sub outputs input1 minus all input2 values as a floating-point (real) value.

SubCx



Description: Complex Multiple Input Subtractor
Library: Numeric, Math
Class: SDFSubCx
C++ Code: See *doc/sp_items/SDFSubCx.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		complex
2	neg		multiple complex

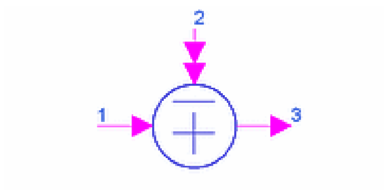
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex

Notes/Equations

1. SubCx outputs input1 minus all input2 values as a complex value.

SubFix



Description: Fixed-Point Multiple Input Subtractor
Library: Numeric, Math
Class: SDFSubFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFSubFix.html* under your installation directory.

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		fix
2	neg		multiple fix

Pin Outputs

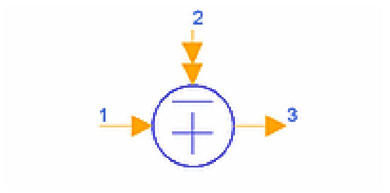
Pin	Name	Description	Signal Type
3	output		fix

Notes/Equations

1. SubFix outputs input1 minus all input2 values as a fixed-point value.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example,

when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

SubInt



Description: Integer Multiple Input Subtractor
Library: Numeric, Math
Class: SDFSubInt
C++ Code: See *doc/sp_items/SDFSubInt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		int
2	neg		multiple int

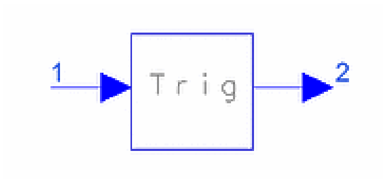
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int

Notes/Equations

1. SubInt outputs input1 minus all input2 values as an integer value.

Trig



Description: Trigonometric function
Library: Numeric, Math
Class: SDFTrig
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Type	function: Sin, Cos, Tan, Cot, Asin, Acos, Atan, Acot, Sinh, Cosh, Tanh, Coth, Asinh, Acosh, Atanh, Acoth	Sin		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

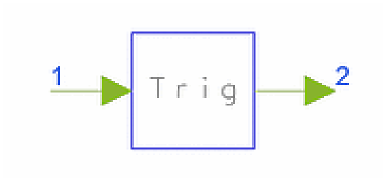
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Trig performs the floating-point (real) trigonometric functions:
   $v_2(t) = f(v_1(t))$
   where *f*( ) is any of the functions that can be selected from the Type parameter.
2. All angles are in radians.

TrigCx



Description: Complex trigonometric function
Library: Numeric, Math
Class: SDFTrigCx
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Type	function: Sin, Cos, Tan, Cot, Asin, Acos, Atan, Acot, Sinh, Cosh, Tanh, Coth, Asinh, Acosh, Atanh, Acoth	Sin		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	complex

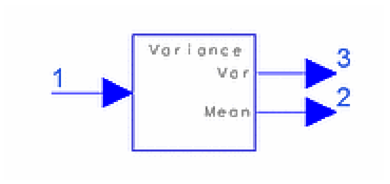
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	complex

Notes/Equations

1. This component performs the complex trigonometric functions:
   $v_2(t) = f(v_1(t))$
   where *f( )* is any of the functions that can be selected from the Type parameter.
2. All angles are in radians.

Variance



Description: Variance function
Library: Numeric, Math
Class: SDFVariance

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of inputs to process between each mean and variance estimate	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	in		real

Pin Outputs

Pin	Name	Description	Signal Type
2	mean		real
3	variance		real

Notes/Equations

1. Variance calculates a running floating-point (real) estimate of the mean and variance of the inputs.

Numeric Matrix Components

- *Abs M* (numeric)
- *Add2 M* (numeric)
- *AddCx2 M* (numeric)
- *AddCx M* (numeric)
- *AddFix2 M* (numeric)
- *AddFix M* (numeric)
- *AddInt2 M* (numeric)
- *AddInt M* (numeric)
- *Add M* (numeric)
- *AvgSqrErr M* (numeric)
- *Conjugate M* (numeric)
- *Delay M* (numeric)
- *GainCx M* (numeric)
- *GainFix M* (numeric)
- *GainInt M* (numeric)
- *Gain M* (numeric)
- *Hermitian M* (numeric)
- *InverseCx M* (numeric)
- *InverseFix M* (numeric)
- *InverseInt M* (numeric)
- *Inverse M* (numeric)
- *Kalman M* (numeric)
- *MpyCx M* (numeric)
- *MpyFix M* (numeric)
- *MpyInt M* (numeric)
- *Mpy M* (numeric)
- *MpyScalarCx M* (numeric)
- *MpyScalarFix M* (numeric)
- *MpyScalarInt M* (numeric)
- *MpyScalar M* (numeric)
- *MxCom M* (numeric)
- *MxDecom M* (numeric)
- *PackCx M* (numeric)
- *PackFix M* (numeric)
- *PackInt M* (numeric)
- *Pack M* (numeric)
- *SampleMean M* (numeric)
- *SubCx M* (numeric)
- *SubFix M* (numeric)
- *SubInt M* (numeric)
- *Sub M* (numeric)
- *SubMxCx M* (numeric)
- *SubMxFix M* (numeric)
- *SubMxInt M* (numeric)
- *SubMx M* (numeric)
- *SVD M* (numeric)
- *TableCx M* (numeric)
- *TableInt M* (numeric)
- *Table M* (numeric)

- *ToeplitzCx M* (numeric)
- *ToeplitzFix M* (numeric)
- *ToeplitzInt M* (numeric)
- *Toeplitz M* (numeric)
- *TransposeCx M* (numeric)
- *TransposeFix M* (numeric)
- *TransposeInt M* (numeric)
- *Transpose M* (numeric)
- *UnPkCx M* (numeric)
- *UnPkFix M* (numeric)
- *UnPkInt M* (numeric)
- *UnPk M* (numeric)

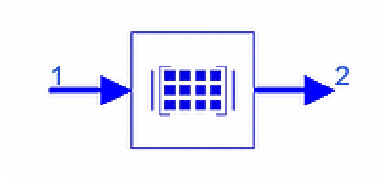
Numeric matrix components provide basic matrix data processing functions such as matrix addition, multiplication, inversion and more and operate on matrix data sets that are integer, double precision floating-point (real)), fixed-point (fixed), or complex values. Each component accepts a specific class of signal and outputs a resultant signal. (These components do not accept any scalar class of signal.)

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. The auto conversion from complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30; a value of 0.5 creates one with a precision of 1.31. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components accept parameter values that are arrays of data. The syntax for referencing arrays of data as parameter values includes an explicit list of values, a reference to a file that contains those values, or a combination of explicit values along with file references. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components operate with fixed-point numbers. These components use one or more parameters that define the characteristics of the fixed-point processing. These parameters include: OverflowHandler, OutputPrecision, RoundFix, ReportOverflow, and others. For details on the use of these parameters for fixed-point components refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. The arithmetic used by these components is two's complement. Therefore, all precision values must specify at least one bit to the left of the decimal point (used as sign bit).

Abs_M



Description: Absolute Value Matrix
Library: Numeric, Matrix
Class: SDFAbs_M
Derived From: MatrixBase
C++ Code: See *doc/sp_items/SDFAbs_M.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

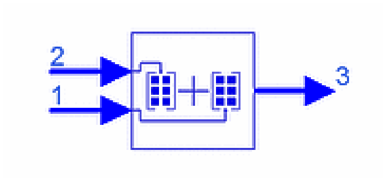
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Abs_M outputs a matrix composed of the absolute value of each entry of the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Add2_M



Description: 2-Input Matrix Adder
Library: Numeric, Matrix
Class: SDFAdd_M
C++ Code: See *doc/sp_items/SDFAdd_M.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		real matrix
2	input#2		real matrix

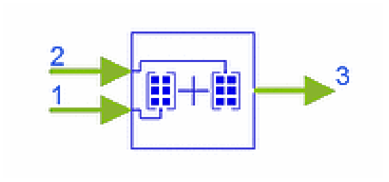
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real matrix

Notes

1. Add2 adds the two inputs and outputs the resulting matrix. The two input matrix signals must have the same matrix row and column values, otherwise an error will be reported.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddCx2_M



Description: 2-Input Complex Matrix Adder
Library: Numeric, Matrix
Class: SDFAddCx_M

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		complex matrix
2	input#2		complex matrix

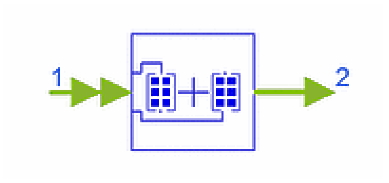
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex matrix

Notes

1. AddCx2_M adds the two inputs and outputs the resulting matrix. The two input matrix signals must have the same matrix row and column values, otherwise an error will be reported.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddCx_M



Description: Complex Matrix Adder
Library: Numeric, Matrix
Class: SDFAddCx_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple complex matrix

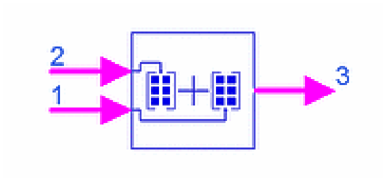
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. AddCx_M adds all input matrices and outputs the resulting matrix.
2. All input matrices must be of the same dimensions.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddFix2_M



Description: 2-Input Fixed-Point Matrix Adder
Library: Numeric, Matrix
Class: SDFAddFix_M

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		fix matrix
2	input#2		fix matrix

Pin Outputs

Pin	Name	Description	Signal Type
3	output		fix matrix

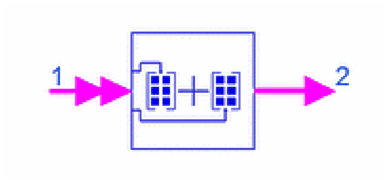
Notes/Equations

1. AddFix2_ M adds the two inputs and outputs the resulting matrix with precision specified by OutputPrecision. The two input matrix signals must have the same matrix row and column values, otherwise an error will be reported.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example,

when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddFix_M



Description: Fixed Matrix Adder
Library: Numeric, Matrix
Class: SDFAddFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple fix matrix

Pin Outputs

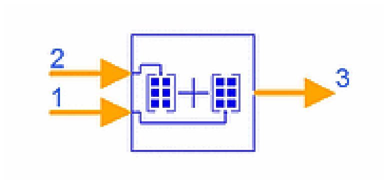
Pin	Name	Description	Signal Type
2	output		fix matrix

Notes/Equations

1. AddFix_M adds all input matrices and outputs the resulting matrix. If the result of the sum for any entry in the matrix cannot be fit into the precision of the output, overflow occurs and is handled by OverflowHandler.
2. All input matrices must be of the same dimensions.
3. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

4. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
5. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddInt2_M



Description: 2-Input Integer Matrix Adder
Library: Numeric, Matrix
Class: SDFAddInt_M

Pin Inputs

Pin	Name	Description	Signal Type
1	input#1		int matrix
2	input#2		int matrix

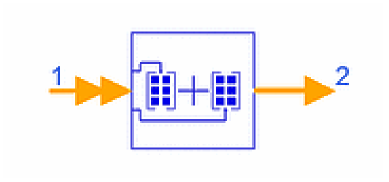
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int matrix

Notes/Equations

1. AddInt2_M adds the two inputs and outputs the resulting matrix. The two input matrix signals must have the same matrix row and column values, otherwise an error will be reported.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AddInt_M



Description: Integer Matrix Adder
Library: Numeric, Matrix
Class: SDFAddInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple int matrix

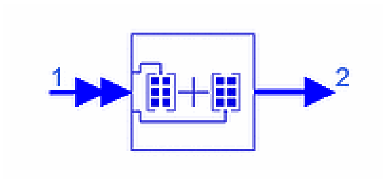
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. AddInt_M adds all input matrices and outputs the resulting matrix. All input matrices must be of the same dimensions.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Add_M



Description: Matrix Adder
Library: Numeric, Matrix
Class: SDFAdd_M
Derived From: MatrixBase
C++ Code: See *doc/sp_items/SDFAdd_M.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple real matrix

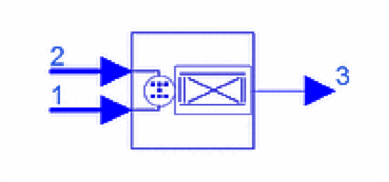
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Add_M adds all input matrices together and outputs the resulting matrix. All input matrices must be of the same dimensions.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

AvgSqrErr_M



Description: Average Mean Squared Error Matrix
Library: Numeric, Matrix
Class: SDFAvgSqrErr_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumInputs	number of input matrices to average	8	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input1		real matrix
2	input2		real matrix

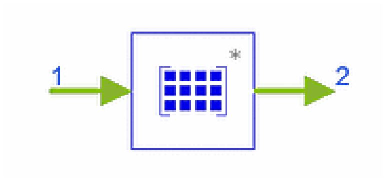
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real

Notes/Equations

1. AvgSqrErr_M computes the average mean squared error over a set of input matrix pairs. The squared error between each corresponding element of a pair of input matrices (input1 and input2) is computed and the errors from each element are summed together. The sums are then averaged over the number of input matrix pairs. NumInputs gives the number of consecutive input matrix pairs that are averaged.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Conjugate_M



Description: Conjugate Matrix
Library: Numeric, Matrix
Class: SDFConjugate_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

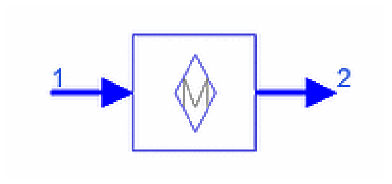
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. Conjugate_M outputs the conjugate of the input matrix. Each element of the output matrix is the complex conjugate of the corresponding input matrix element.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Delay_M



Description: Matrix Delay Component
Library: Numeric, Matrix
Class: HOFDelay_M
Derived From: Delay

Parameters

Name	Description	Default	Type	Range
N	N	1	int	[0, ∞)
NumRows	number of rows in initial matrix	2	int	[1, ∞)
NumCols	number of columns in initial matrix	2	int	[1, ∞)
InitialMatrixContents	contents of CustomMatrix	1 0 0 1	string	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		multiple anytype

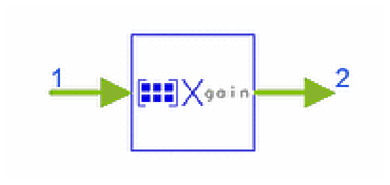
Pin Outputs

Pin	Name	Description	Signal Type
2	output		multiple anytype

Notes/Equations

1. Delay_M adds N initial matrices to the output signal.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).
3. The parameters N, NumRows, and NumCols cannot be swept.

GainCx_M



Description: Complex Gain Matrix
Library: Numeric, Matrix
Class: SDFGainCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type
Gain	gain to be multiplied with each entry of the input matrix	1	complex

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

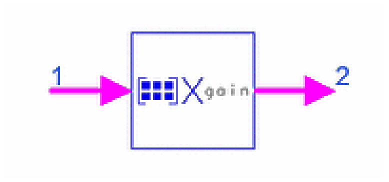
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. GainCx_M multiplies a complex matrix by a scalar complex gain value given by the Gain parameter.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

GainFix_M



Description: Fixed-Point Gain Matrix
Library: Numeric, Matrix
Class: SDFGainFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
Gain	gain to be multiplied with each input matrix entry	1.0	fix
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix

Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix matrix

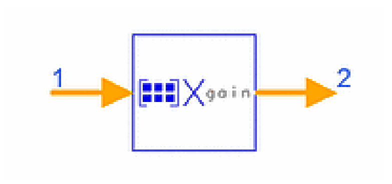
Notes/Equations

1. GainFix_M multiplies a fixed-point matrix by a fixed-point scalar given by the Gain parameter.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example,

when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

GainInt_M



Description: Integer Gain Matrix
Library: Numeric, Matrix
Class: SDFGainInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
Gain	gain to be multiplied with each input matrix entry	1	int	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix

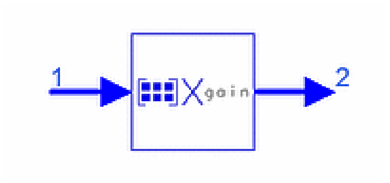
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. GainInt_M multiplies an integer matrix by a scalar integer given by the Gain parameter.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Gain_M



Description: Gain Matrix
Library: Numeric, Matrix
Class: SDFGain_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
Gain	gain to be multiplied with each entry of the input matrix	1.0	real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

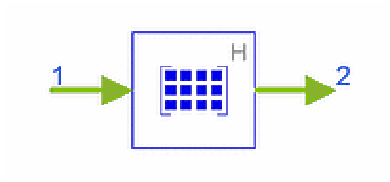
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Gain_M multiplies a floating-point (real) matrix by a scalar gain value given by the Gain parameter.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Hermitian_M



Description: Hermitian Matrix
Library: Numeric, Matrix
Class: SDFHermitian_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

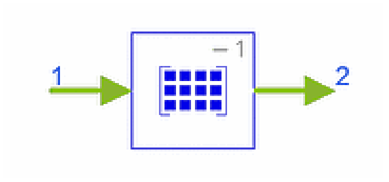
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. Hermitian_M performs a Hermitian transpose (conjugate transpose) on the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

InverseCx_M



Description: Complex Inverse Matrix
Library: Numeric, Matrix
Class: SDFInverseCx_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

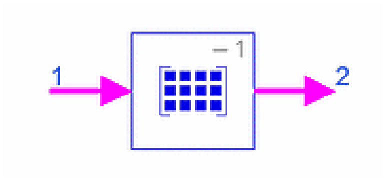
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. The complex output matrix is the inverse of the complex input matrix.
   $$[input] \times [output] = [IdentityMatrix]$$
2. The input matrix must be square.
3. For information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

InverseFix_M



Description: Fixed-Point Inverse Matrix
Library: Numeric, Matrix
Class: SDFInverseFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix

Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix matrix

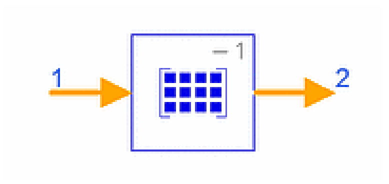
Notes/Equations

1. The fixed-point output matrix is the inverse of the fixed-point input matrix.
$$[input] \times [output] = [IdentityMatrix]$$
2. The input matrix must be square.
3. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
4. If UseArrivingPrecision = YES, then components that send a NULL particle on their

first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.

5. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

InverseInt_M



Description: Integer Inverse Matrix
Library: Numeric, Matrix
Class: SDFInverseInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix

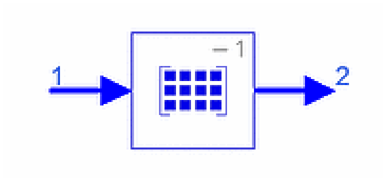
Pin Inputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. The integer output matrix is the inverse of the input matrix. (Due to integer arithmetic limitations, the output may not be the exact inverse of the input.)
$$[input] \times [output] \cong [IdentityMatrix]$$
2. The input matrix must be square.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Inverse_M



Description: Inverse Matrix
Library: Numeric, Matrix
Class: SDFInverse_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

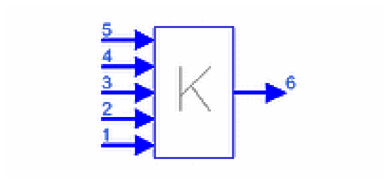
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. The output matrix is the inverse of the input matrix.
$$[input] \times [output] = [IdentityMatrix]$$
2. The input matrix must be square.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Kalman_M



Description: Kalman Filter Matrix
Library: Numeric, Matrix
Class: SDFKalman_M
Derived From: MatrixBase
C++ Code: See *doc/sp_items/SDFKalman_M.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
StateDimension	number of elements in state vector	5	int	[1, ∞)
InputDimension	number of elements in observation vector	1	int	[1, ∞)
InitialState	initial value of state vector	0.0 [5]	real array	
InitialCorrMatrix	initial value of correlation matrix of error	.1 0 [5] .1 0 [5] .1 0 [5] .1 0 [5] .1	real array	
InitialStateTransitionMatrix	state transition matrix at time 0. PHI(1,0)	1 0 [5] 1 0 [5] 1 0 [5] 1 0 [5] 1	real array	
InitialProcessNoiseCorrMatrix	correlation matrix of process noise vector at time 0. Q(0)	0.0 [25]	real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix
2	StateTransitionMatrixAtTimeN		real matrix
3	MeasurementMatrixAtTimeN		real matrix
4	ProcessNoiseCorrMatrixAtTimeN		real matrix
5	MeasurementNoiseCorrMatrixAtTimeN		real matrix

Pin Outputs

Pin	Name	Description	Signal Type
6	output		real matrix

Notes/Equations

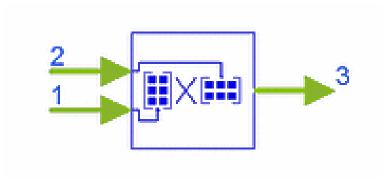
1. Kalman_M implements a Kalman filter using the one-step prediction algorithm. The initial values for the state transition, correlation, process noise correlation matrices, and state vector are given as parameters.
2. Inputs are the current values of the state transition, process noise correlation, measurement noise correlation, and measurement matrices, and the observation vector.

3. The single output is the state vector.
4. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

References

1. R.E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME, J. Basic Eng*., Ser 82D, pp. 35-45, March 1960.
2. S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1986.

MpyCx_M



Description: Complex Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in initial matrix	2	int	[1, ∞)
NumCols	number of columns in initial matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	Ainput		complex matrix
2	Binput		complex matrix

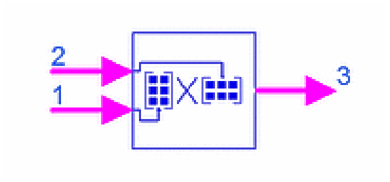
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex matrix

Notes/Equations

1. MpyCx_M multiplies the complex input matrices and outputs the resulting matrix.
2. The output matrix will have same number of rows as the Ainput and the same number of columns as the Binput.
$$[output] = [Ainput] \times [Binput]$$
3. The number of columns in the Ainput matrix must match the number of rows in the Binput matrix.
4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyFix_M



Description: Fixed-Point Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	Ainput		fix matrix
2	Binput		fix matrix

Pin Outputs

Pin	Name	Description	Signal Type
3	output		fix matrix

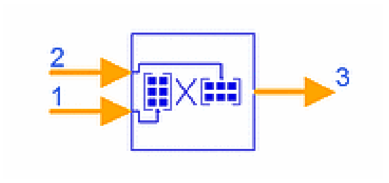
Notes/Equations

1. MpyFix_M multiplies the input matrices and outputs the resulting fixed-point matrix. If the result of the multiplication for any entry in the matrix cannot be fit into the precision of the output, overflow occurs and is handled by OverflowHandler.
2. The output matrix will have same number of rows as the Ainput and the same number of columns as the Binput.
$$\left[output\right] = \left[Ainput\right] \times \left[Binput\right]$$
3. The number of columns in the Ainput matrix must match the number of rows in the Binput matrix.
4. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will

be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy)in the *ADS Ptolemy Simulation* (ptolemy) documentation.

5. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
6. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyInt_M



Description: Integer Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	Ainput		int matrix
2	Binput		int matrix

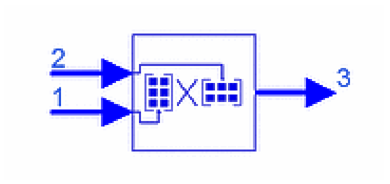
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int matrix

Notes/Equations

1. MpyInt_M multiplies the input matrices and outputs the resulting matrix.
2. The output matrix will have same number of rows as the Ainput and the same number of columns as the Binput.
   $$\left[output\right] = \left[Ainput\right] \times \left[Binput\right]$$
3. The number of columns in the Ainput matrix must match the number of rows in the Binput matrix.
4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Mpy_M



Description: Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpy_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	Ainput		real matrix
2	Binput		real matrix

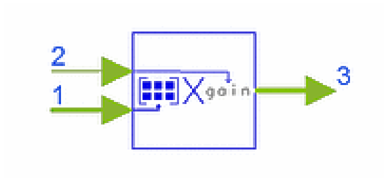
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real matrix

Notes/Equations

1. Mpy_M multiplies the input matrices and outputs the resulting matrix.
2. The output matrix will have same number of rows as the Ainput and the same number of columns as the Binput.
$$[output] = [Ainput] \times [Binput]$$
3. The number of columns in the Ainput matrix must match the number of rows in the Binput matrix.
4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyScalarCx_M



Description: Matrix and Complex Scalar Multiplier
Library: Numeric, Matrix
Class: SDFMpyScalarCx_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix
2	gain	Input gain to be multiplied with the input matrix	complex

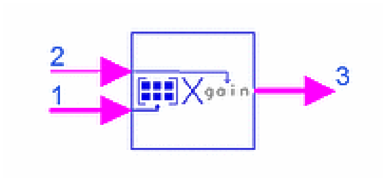
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex matrix

Notes/Equations

1. MpyScalarCx_M multiplies a complex matrix by a scalar complex input value.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyScalarFix_M



Description: Scalar and Fixed-Point Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyScalarFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix
2	gain	Input gain to be multiplied with the input matrix	fix

Pin Outputs

Pin	Name	Description	Signal Type
3	output		fix matrix

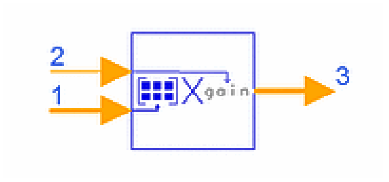
Notes/Equations

1. MpyScalarFix_M multiplies a fixed-point matrix by a scalar fixed-point input value.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a

precision of 1.0 and the output value will be forced to 0.

4. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyScalarInt_M



Description: Scalar and Integer Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyScalarInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix
2	gain	Input gain to be multiplied with the input matrix	int

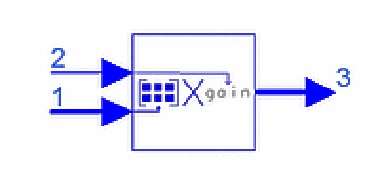
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int matrix

Notes/Equations

1. MpyScalarCx_M multiplies an integer matrix by a scalar integer input value.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MpyScalar_M



Description: Scalar and Matrix Multiplier
Library: Numeric, Matrix
Class: SDFMpyScalar_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix
2	gain	Input gain to be multiplied with the input matrix	real

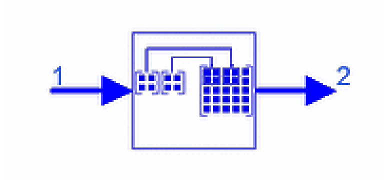
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real matrix

Notes/Equations

1. MpyScalar_M multiplies a floating-point (real) matrix by a scalar input value.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MxCom_M



Description: Composed Matrix
Library: Numeric, Matrix
Class: SDFMxCom_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
OutputNumRows	number of rows for output matrix	100	int	[InputNumRows, ∞)†
OutputNumColumns	number of columns for output matrix	100	int	[InputNumColumns, ∞)††
InputNumRows	number of rows for input matrix	4	int	[1, ∞)
InputNumColumns	number of columns for input matrix	4	int	[1, ∞)

† must be an integer multiple of InputNumRows
†† must be an integer multiple of InputNumColumns

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

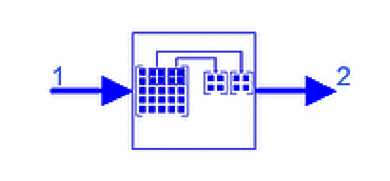
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Each output matrix is composed from the input submatrices. The output matrix is filled with input submatrices in rasterized order; that is, the top of the output matrix is filled first, from left to right, with the first input matrices.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

MxDecom_M



Description: Decomposed Matrix
Library: Numeric, Matrix
Class: SDFMxDecom_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
StartRow	starting row in input matrix to generate output matrices (first row is 1)	1	int	[1, ∞)
StartCol	starting column in input matrix to generate output matrices (first column is 1; therefore, the upper left corner of the matrix is (1,1)	1	int	[1, ∞)
InputNumRows	number of rows for input matrix	100	int	[OutputNumRows, ∞)†
InputNumCols	number of columns from input matrix to use to generate the output matrices.	100	int	[OutputNumCols, ∞)††
OutputNumRows	number of rows for output matrix	4	int	[1, ∞)
OutputNumCols	number of columns for output matrix	4	int	[1, ∞)

† must be an integer multiple of OutputNumRows
†† must be an integer multiple of OutputNumCols

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input matrix to be decomposed into the output submatrices.	real matrix

Pin Outputs

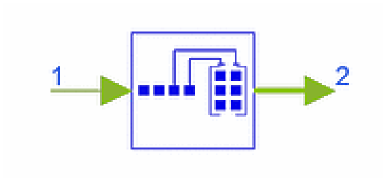
Pin	Name	Description	Signal Type
2	output	Output matrices with dimensions OutputNumRows*OutputNumCols.	real matrix

Notes/Equations

1. All or part of the input matrix is decomposed into a sequence of output submatrices. The part of input matrix to be decomposed is specified by StartRow, StartCol, InputNumRows, and InputNumColumns. The dimensions of each output submatrix are specified by the OutputNumRows and OutputNumColumns.

2. For each input matrix, the number of output matrices is:
$$\frac{InputNumRows}{OutputNumRows} \times \frac{InputNumColumns}{OutputNumColumns}$$
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

PackCx_M



Description: Pack Complex Matrix
Library: Numeric, Matrix
Class: SDFPackCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in output matrix	2	int	[1, ∞)
NumCols	number of columns in output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex

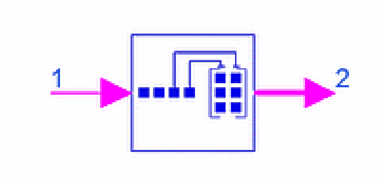
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. PackCx_M constructs a complex output matrix from scalar input values. Inputs are entered into the matrix in rasterized order; for example, for an M×N matrix, the first row is filled from left to right using the first N input values.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

PackFix_M



Description: Pack Fixed-Point Matrix
Library: Numeric, Matrix
Class: SDFPackFix_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in output matrix	2	int	[1, ∞)
NumCols	number of columns in output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix

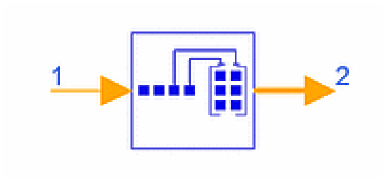
Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix matrix

Notes/Equations

1. PackFix_M constructs a fixed-point output matrix from scalar input values. Inputs are entered into the matrix in rasterized order; for example, for an M×N matrix, the first row is filled from left to right using the first N input values.
2. There are no fixed-point parameters for this component because fixed-point arithmetic is not performed.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

PackInt_M



Description: Pack Integer Matrix
Library: Numeric, Matrix
Class: SDFPackInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in output matrix	2	int	[1, ∞)
NumCols	number of columns in output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

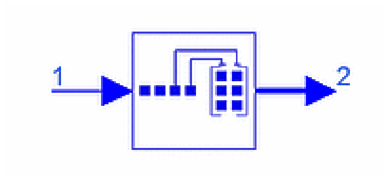
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. PackInt_M constructs an integer output matrix from scalar input values. Inputs are entered into the matrix in rasterized order; for example, for an M×N matrix, the first row is filled from left to right using the first N input values.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Pack_M



Description: Pack Matrix
Library: Numeric, Matrix
Class: SDFPack_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in output matrix	2	int	[1, ∞)
NumCols	number of columns in output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

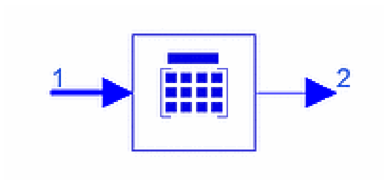
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Pack_M constructs a complex output matrix from scalar input values. Inputs are entered into the matrix in rasterized order; for example, for an M×N matrix, the first row is filled from left to right using the first N input values.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SampleMean_M



Description: Mean Value Matrix
Library: Numeric, Matrix
Class: SDFSampleMean_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

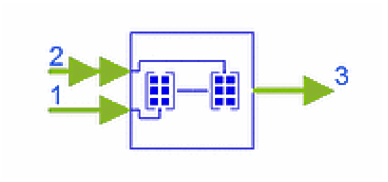
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. SampleMean_M finds the average value of the elements of the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubCx_M



Description: Complex Subtraction
Library: Numeric, Matrix
Class: SDFSubCx_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		complex matrix
2	neg		multiple complex matrix

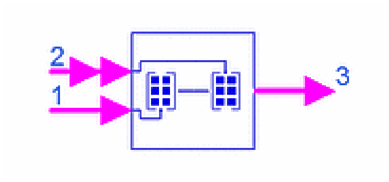
Pin Outputs

Pin	Name	Description	Signal Type
3	output		complex matrix

Notes/Equations

1. SubCx_M outputs the pos input matrix minus all of the neg inputs.
2. All input matrices must be of the same dimensions.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubFix_M



Description: Fixed Subtraction
Library: Numeric, Matrix
Class: SDFSubFix_M
Derived From: SDFFix

h5 Parameters

Name	Description	Default	Type
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision
OutputPrecision	precision of output in bits and accumulation	2.14	precision

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		fix matrix
2	neg		multiple fix matrix

Pin Outputs

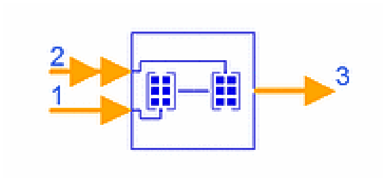
Pin	Name	Description	Signal Type
3	output		fix matrix

Notes/Equations

1. SubFix_M outputs the pos input matrix minus the neg inputs.
2. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. All input matrices must be of the same dimensions.

4. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
5. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubInt_M



Description: Integer Subtraction
Library: Numeric, Matrix
Class: SDFSubInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		int matrix
2	neg		multiple int matrix

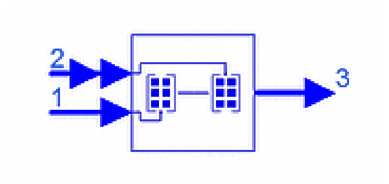
Pin Outputs

Pin	Name	Description	Signal Type
3	output		int matrix

Notes/Equations

1. SubInt_M outputs the pos input matrix minus all of the neg inputs.
2. All input matrices must be of the same dimensions.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Sub_M



Description: Subtraction
Library: Numeric, Matrix
Class: SDFSub_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	pos		real matrix
2	neg		multiple real matrix

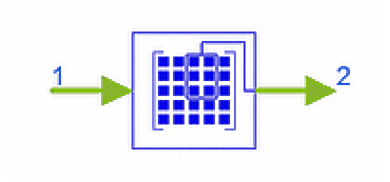
Pin Outputs

Pin	Name	Description	Signal Type
3	output		real matrix

Notes/Equations

1. Sub_M outputs the pos input matrix minus all neg inputs.
2. All input matrices must be of the same dimensions.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubMxCx_M



Description: Complex Submatrix
Library: Numeric, Matrix
Class: SDFSubMxCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
StartRow	starting row in the submatrix within the input matrix. The first (top) row in a matrix is 1.	1	int	[1, ∞)
StartCol	starting column in the submatrix within the input matrix. The first (left) column in a matrix is 1; therefore, the upper left corner of the matrix is (1,1).	1	int	[1, ∞)
NumRows	number of rows for submatrix	1	int	[1, ∞)
NumCols	number of columns for submatrix	1	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

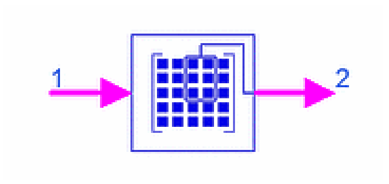
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. Output matrix is a submatrix of the input matrix. The parameters specify the size and position of the output submatrix from within the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubMxFix_M



Description: Fixed Submatrix
Library: Numeric, Matrix
Class: SDFSubMxFix_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
StartRow	starting row in the submatrix within the input matrix. The first (top) row in a matrix is 1.	1	int	[1, ∞)
StartCol	starting column in the submatrix within the input matrix. The first (left) column in a matrix is 1; therefore, the upper left corner of the matrix is (1,1).	1	int	[1, ∞)
NumRows	number of rows for submatrix	2	int	[1, ∞)
NumCols	number of columns for submatrix	1	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix

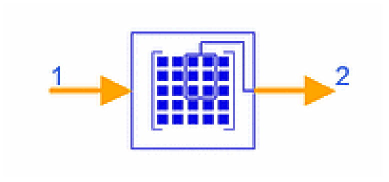
Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix matrix

Notes/Equations

1. Output matrix is a submatrix of the input matrix. The parameters specify the size and position of the output submatrix from within the input matrix.
2. There are no fixed-point parameters because fixed-point arithmetic is not performed. The output precision is the same as the input precision.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubMxInt_M



Description: Integer Submatrix
Library: Numeric, Matrix
Class: SDFSubMxInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
StartRow	starting row in the submatrix within the input matrix. The first (top) row in a matrix is 1.	1	int	[1, ∞)
StartCol	starting column in the submatrix within the input matrix. The first (left) column in a matrix is 1; therefore, the upper left corner of the matrix is (1,1).	1	int	[1, ∞)
NumRows	number of rows for submatrix	1	int	[1, ∞)
NumCols	number of columns for submatrix	1	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix

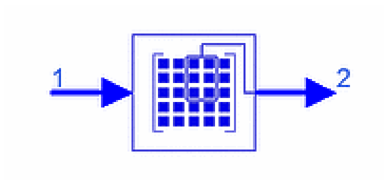
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. The output matrix is a submatrix of the input matrix. The parameters specify the size and position of the output submatrix from within the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SubMx_M



Description: Submatrix
Library: Numeric, Matrix
Class: SDFSubMx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
StartRow	starting row in the submatrix within the input matrix. The first (top) row in a matrix is 1.	1	int	[1, ∞)
StartCol	starting column in the submatrix within the input matrix. The first (left) column in a matrix is 1; therefore, the upper left corner of the matrix is (1,1).	1	int	[1, ∞)
NumRows	number of rows for submatrix	2	int	[1, ∞)
NumCols	number of columns for submatrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

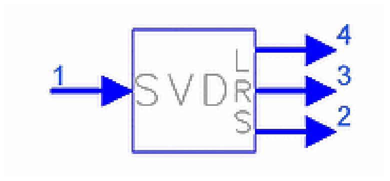
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Output matrix is a submatrix of the input matrix. The parameters specify the size and position of the output submatrix from within the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

SVD_M



Description: Singular Value Decomposition of a Toeplitz Matrix
Library: Numeric, Matrix
Class: SDFSVD_M
Derived From: MatrixBase
C++ Code: See *doc/sp_items/SDFSVD_M.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
Threshold	threshold for similarities; algorithm assumes values below Threshold have reached zero	0.00000000000000001	real	(-∞, ∞)
MaxIterations	maximum iterations for SVD convergence	30	int	[1, ∞)
GenerateLeft	matrix generation of left singular vectors: Do not Generate Left Singular Vectors, Generate Left Singular Vectors	Generate Left Singular Vectors	enum	
GenerateRight	matrix generation of right singular vectors: Do not Generate Right Singular Vectors, Generate Right Singular Vectors	Generate Right Singular Vectors	enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input stream.	real matrix

Pin Outputs

Pin	Name	Description	Signal Type
2	svals	The singular values of input - The diagonal of "W".	real matrix
3	rsvec	Right singular vectors of input - "V".	real matrix
4	lsvec	Left singular vectors of input - "W".	real matrix

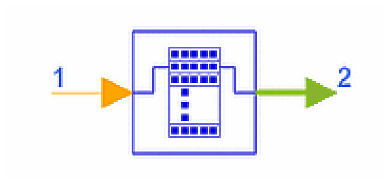
Notes/Equations

1. SVD_M computes the singular-value decomposition (SVD) of an input Toeplitz matrix *A* by decomposing *A* into $A = UWV'$, where *U* and *V* are orthogonal matrices and V' represents the transpose of *V*.
2. The input matrix must be a Toeplitz matrix. The output S is the diagonal of the matrix *W*, the output L is the matrix *U*, and the output R is the matrix *V*. If the input matrix is of size M rows by N columns, the output S will be of size N × 1, output L will be of size M × N, and output R will be of size N × N.
3. The MaxIterations parameter allows the designer to control the number of iterations that the SVD algorithm will be allowed to run before stopping. Normally, the SVD algorithm will converge before this number of iterations is reached but this parameter

is provided to prevent non-convergent matrices from causing the component to run too long.

4. The execution time of SVD_M may be reduced by using the GenerateLeft and GenerateRight parameters to specify that the matrices of the left and right singular vectors not be generated. The vector of singular values (the S output) is always generated.
5. S. Haykin, *Modern Filters*, pp. 333-335, Macmillan Publishing Company, New York, 1989.
6. See Also: *Toeplitz_M* (numeric)
7. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

TableCx_M



Description: Complex Lookup Table Matrix
Library: Numeric, Matrix
Class: SDFTableCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows for each matrix in the table	1	int	[1, ∞)
NumCols	number of columns for each matrix in the table	1	int	[1, ∞)
ComplexTable	table containing matrices. Each matrix with dimensions NumRows x NumCols is given in row major ordering.	1.0+j 1.0-j (-1.0+j) (-1.0-j)	complex array	†

† ComplexTable number of elements must be an integer multiple of the output matrix size (NumRows <B> NumCols)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	the index for table lookup. The first matrix is index "0"	int

Pin Outputs

Pin	Name	Description	Signal Type
2	output	the matrix in the table corresponding to the index.	complex matrix

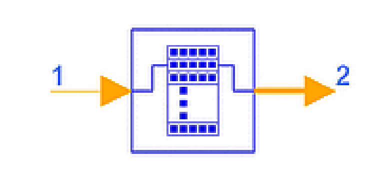
Notes/Equations

1. TableCx_M implements a matrix lookup table indexed by an integer-valued input. The output will be a the matrix corresponding to the index input. The input must be from 0 to $N - 1$, inclusive, where N is the number of matrices in the table. ComplexTable specifies the entries of matrices in the table.
   Entries of each matrix in the table should be given in row major ordering. Therefore, the upper left corner entry of the first matrix is the first value in the table, and the first NumCols items in the table parameter make up the first row of the first matrix in the table.
   An error occurs if the index input value is out of bounds.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the the

ADS Ptolemy Simulation (ptolemy) documentation.

3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

TableInt_M



Description: Integer Lookup Table Matrix
Library: Numeric, Matrix
Class: SDFTableInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows for each matrix in the table	1	int	[1, ∞)
NumCols	number of columns for each matrix in the table	2	int	[1, ∞)
IntTable	table containing matrices. Each matrix with dimensions NumRows x NumCols is given in row major ordering.	1 1 1 -1 -1 1 -1 -1	int array	†

† IntTable number of elements must be an integer multiple of the output matrix size (NumRows <B> NumCols)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	the index for table lookup. The first matrix is index "0"	int

Pin Outputs

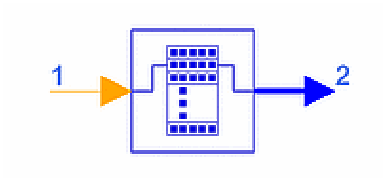
Pin	Name	Description	Signal Type
2	output	the matrix in the table corresponding to the index.	int matrix

Notes/Equations

1. TableInt_M implements a matrix lookup table indexed by an integer-valued input. The output will be a the matrix corresponding to the index input. The input must be from 0 to $N - 1$, inclusive, where N is the number of matrices in the table. IntTable specifies the entries of matrices in the table.
2. The entries of each matrix in the table should be given in row major ordering. Therefore, the upper left corner entry of the first matrix is the first value in the table, and the first NumCols items in the table parameter make up the first row of the first matrix in the table.
3. An error occurs if the index input value is out of bounds.
4. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. For general information regarding numeric matrix component signals, refer to

Numeric Matrix Components (numeric).

Table_M



Description: Lookup Table Matrix
Library: Numeric, Matrix
Class: SDFTable_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows for each matrix in the table	2	int	[1, ∞)
NumCols	number of columns for each matrix in the table	2	int	[1, ∞)
FloatTable	table containing matrices. Each matrix with dimensions NumRows x NumCols is given in row major ordering.	0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0	real array	†

† FloatTable number of elements must be an integer multiple of the output matrix size (NumRows <B> NumCols)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	the index for table lookup. The first matrix is index "0"	int

Pin Outputs

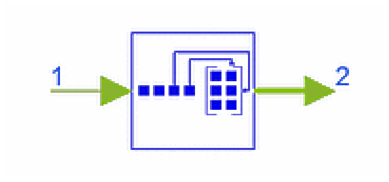
Pin	Name	Description	Signal Type
2	output	the matrix in the table corresponding to the index.	real matrix

Notes/Equations

1. Table_M implements a matrix lookup table indexed by an integer-valued input. The output will be the matrix corresponding to the index input. The input must be from 0 to $N - 1$, inclusive, where N is the number of matrices in the table. FloatTable specifies the entries of matrices in the table.
2. Entries of each matrix in the table should be given in row major ordering. Therefore, the upper left corner entry of the first matrix is the first value in the table, and the first NumCols items in the table parameter make up the first row of the first matrix in the table.
3. An error occurs if the index input value is out of bounds.
4. For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. For general information regarding numeric matrix component signals, refer to

Numeric Matrix Components (numeric).

ToeplitzCx_M



Description: Complex Toeplitz Matrix
Library: Numeric, Matrix
Class: SDFToeplitzCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in the output matrix	2	int	[1, ∞)
NumCols	number of columns in the output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input stream.	complex

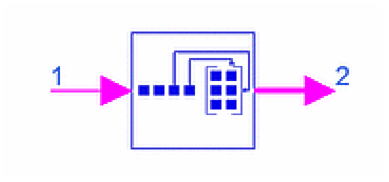
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Data matrix X.	complex matrix

Notes/Equations

1. ToeplitzCx_M builds a rectangular Toeplitz matrix from the input scalar values.
2. ToeplitzCx_M generates an output matrix *X*, with dimensions NumRows × NumCols, from an input stream of NumRows + NumCols − 1 particles. The output matrix is a Toeplitz matrix such that
   the first row is
   $$\left[x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(0)\right]$$
   the second row is
   $$\left[x(M)\ \ x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(1)\right]$$
   and so forth until the last row, which is
   $$\left[x(N-1)\ \ x(N-2)\ \ \ldots\ \ x(N-M)\right]$$
   where NumRows = *N* − *M* + 1 and NumCols = *M* and conversely, *M* = NumCols and *N* = NumRows + NumCols − 1.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

ToeplitzFix_M



Description: Fixed Toeplitz Matrix
Library: Numeric, Matrix
Class: SDFToeplitzFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
UseArrivingPrecision	use precision of arriving matrices: NO, YES	NO	enum	
InputPrecision	precision of input matrix elements, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision	
OutputPrecision	precision of output in bits and accumulation	2.14	precision	
NumRows	number of rows in the output matrix	2	int	[1, ∞)
NumCols	number of columns in the output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input stream.	fix

Pin Outputs

Pin	Name	Description	Signal Type
2	output	the data matrix X.	fix matrix

Notes/Equations

1. ToeplitzFix_M builds a rectangular Toeplitz matrix from the input scalar values.
2. This component generates an output matrix *X*, with dimensions NumRows × NumCols, from an input stream of NumRows + NumCols − 1 particles. The output matrix is a Toeplitz matrix such that
   the first row is
   $$\left[x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(0)\right]$$
   the second row is

$$\left[x(M)\ \ x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(1)\right]$$

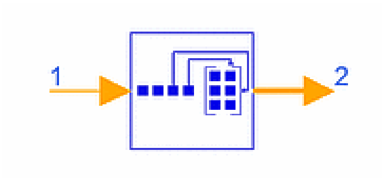
and so forth until the last row, which is

$$\left[x(N-1)\ \ x(N-2)\ \ \ldots\ \ x(N-M)\right]$$

where NumRows = $N - M$ +1 and NumCols = M and conversely, M = NumCols and N = NumRows + NumCols − 1.

3. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
4. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
5. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

ToeplitzInt_M



Description: Integer Toeplitz Matrix
Library: Numeric, Matrix
Class: SDFToeplitzInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in the output matrix	2	int	[1, ∞)
NumCols	number of columns in the output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input stream.	int

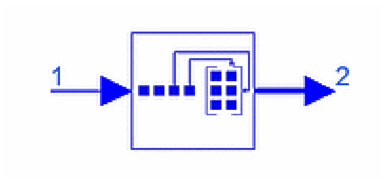
Pin Outputs

Pin	Name	Description	Signal Type
2	output	the data matrix X.	int matrix

Notes/Equations

1. ToeplitzInt_M builds a rectangular Toeplitz matrix from input scalar values.
2. This component generates an output matrix *X*, with dimensions NumRows × NumCols, from an input stream of NumRows + NumCols − 1 particles. The output matrix is a Toeplitz matrix such that
   the first row is
   $$\left[x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(0)\right]$$
   the second row is
   $$\left[x(M)\ \ x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(1)\right]$$
   and so forth until the last row, which is
   $$\left[x(N-1)\ \ x(N-2)\ \ \ldots\ \ x(N-M)\right]$$
   where NumRows = *N* − *M* + 1 and NumCols = *M* and conversely *M* = NumCols and *N* = NumRows + NumCols − 1.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Toeplitz_M



Description: Toeplitz Matrix
Library: Numeric, Matrix
Class: SDFToeplitz_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in the output matrix	2	int	[1, ∞)
NumCols	number of columns in the output matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input stream.	real

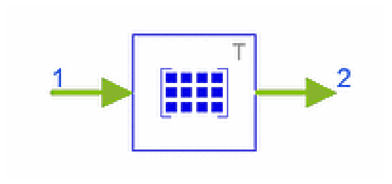
Pin Outputs

Pin	Name	Description	Signal Type
2	output	the data matrix X.	real matrix

Notes/Equations

1. Toeplitz_M builds a rectangular Toeplitz matrix from the input scalar values.
2. This component generates an output matrix *X*, with dimensions NumRows × NumCols, from an input stream of NumRows+NumCols− 1particles. The output matrix is a Toeplitz matrix such that
   the first row is
   $$\left[x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(0)\right]$$
   the second row is
   $$\left[x(M)\ \ x(M-1)\ \ x(M-2)\ \ \ldots\ \ x(1)\right]$$
   and so forth until the last row, which is
   $$\left[x(N-1)\ \ x(N-2)\ \ \ldots\ \ x(N-M)\right]$$
   where NumRows = *N* − *M* +1 and NumCols = *M* and conversely *M* = NumCols and *N* = NumRows + NumCols − 1.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

TransposeCx_M



Description: Complex Transpose Matrix
Library: Numeric, Matrix
Class: SDFTransposeCx_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

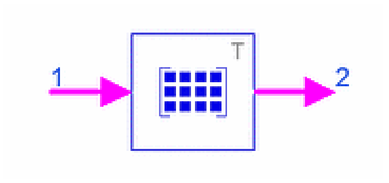
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex matrix

Notes/Equations

1. TransposeCx_M outputs the transpose of the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

TransposeFix_M



Description: Fixed Transpose Matrix
Library: Numeric, Matrix
Class: SDFTransposeFix_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix

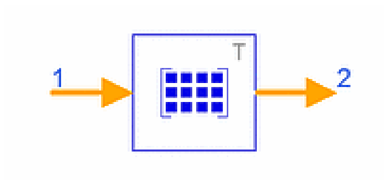
Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix matrix

Notes/Equations

1. TransposeFix_M outputs the transpose of the input matrix.
2. There are no fixed-point parameters for this component because fixed-point arithmetic is not performed.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

TransposeInt_M



Description: Integer Transpose Matrix
Library: Numeric, Matrix
Class: SDFTransposeInt_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix

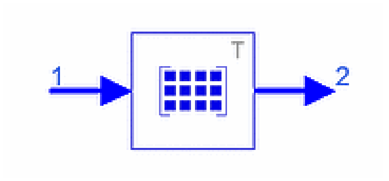
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int matrix

Notes/Equations

1. TransposeInt_M outputs the transpose of the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Transpose_M



Description: Transpose Matrix
Library: Numeric, Matrix
Class: SDFTranspose_M
Derived From: MatrixBase

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

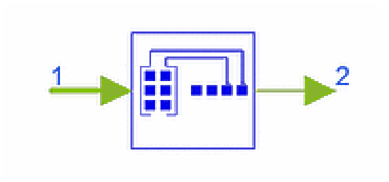
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real matrix

Notes/Equations

1. Transpose_M outputs the transpose of the input matrix.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

UnPkCx_M



Description: Unpack Complex Matrix
Library: Numeric, Matrix
Class: SDFUnPkCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in input matrix	2	int	[1, ∞)
NumCols	number of columns in input matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex matrix

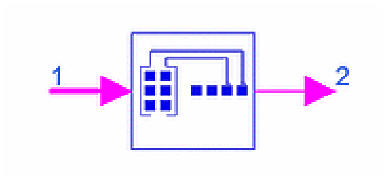
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. The scalar outputs are each of the elements of the input matrix. The elements are sent to the output row-by-row, top-to-bottom. Top row entries are sent first (left to right) followed by the next row down, and so on.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

UnPkFix_M



Description: Unpack Fixed Matrix
Library: Numeric, Matrix
Class: SDFUnPkFix_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in input matrix	2	int	[1, ∞)
NumCols	number of columns in input matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		fix matrix

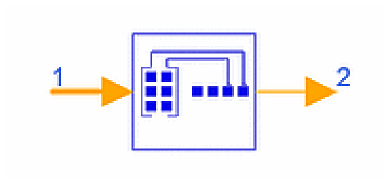
Pin Outputs

Pin	Name	Description	Signal Type
2	output		fix

Notes/Equations

1. The scalar outputs are each of the elements of the input matrix. The elements are sent to the output row-by-row, top-to-bottom. Top row entries are sent first (left to right) followed by the next row down, and so on.
2. There are no fixed-point parameters for this component because fixed-point arithmetic is not performed.
3. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

UnPkInt_M



Description: Unpack Integer Matrix
Library: Numeric, Matrix
Class: SDFUnPkInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in the input matrix	2	int	[1, ∞)
NumCols	number of columns in the input matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int matrix

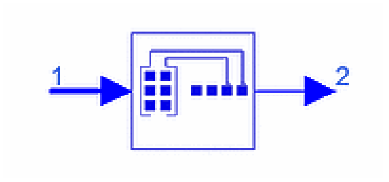
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. The scalar outputs are each of the elements of the input matrix. The elements are sent to the output row-by-row, top-to-bottom. Top row entries are sent first (left to right) followed by the next row down, and so on.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

UnPk_M



Description: Unpack Matrix
Library: Numeric, Matrix
Class: SDFUnPk_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
NumRows	number of rows in input matrix	2	int	[1, ∞)
NumCols	number of columns in input matrix	2	int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real matrix

Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. The scalar outputs are each of the elements of the input matrix. The elements are sent to the output row-by-row, top-to-bottom. Top row entries are sent first (left to right) followed by the next row down, and so on.
2. For general information regarding numeric matrix component signals, refer to *Numeric Matrix Components* (numeric).

Numeric Signal Processing Components

- *Autocor* (numeric)
- *Biquad* (numeric)
- *BiquadCascade* (numeric)
- *BlockAllPole* (numeric)
- *BlockFIR* (numeric)
- *BlockLattice* (numeric)
- *BlockRLattice* (numeric)
- *Burg* (numeric)
- *ConvolCx* (numeric)
- *Convolve* (numeric)
- *CrossCorr* (numeric)
- *DelayEstimator* (numeric)
- *DTFT* (numeric)
- *FFT Cx* (numeric)
- *FIR* (numeric)
- *FIR Cx* (numeric)
- *FIR Fix* (numeric)
- *Hilbert* (numeric)
- *IIR* (numeric)
- *IIR Cx* (numeric)
- *IIR Fix* (numeric)
- *Lattice* (numeric)
- *LevDur* (numeric)
- *LMS* (numeric)
- *LMS Cx* (numeric)
- *LMS Leak* (numeric)
- *LMS OscDet* (numeric)
- *PattMatch* (numeric)
- *RLattice* (numeric)
- *SlidWinAvg* (numeric)

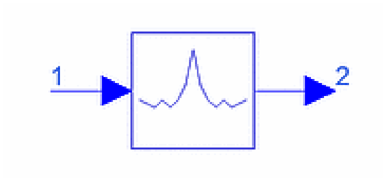
The numeric signal processing components provide basic signal processing functions on single data points or arrays of data that are integer, double precision floating-point (real), fixed-point (fixed), or complex values. Each component accepts a specific class of signal and outputs a resultant signal. (These components do not accept any matrix class of signal.)

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. The auto conversion from timed, complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30; a value of 0.5 would create one of precision of 1.31. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components accept parameter values that are arrays of data. The syntax for referencing arrays of data as parameter values includes an explicit list of values, a reference to a file that contains those values, or a combination of explicit values along with file references. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components operate with fixed-point numbers. These components use one or more parameters that define the characteristics of the fixed-point processing. These parameters include: OverflowHandler, OutputPrecision, RoundFix, ReportOverflow, and others. For details on the use of these parameters for fixed-point components refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. The arithmetic used by these components is two's complement. Therefore, all precision values must specify at least one bit to the left of the decimal point (used as sign bit).

Autocor



Description: Autocorrelation estimator
Library: Numeric, Signal Processing
Class: SDFAutocor
C++ Code: See *doc/sp_items/SDFAutocor.html* under your installation directory.

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
NoInputsToAvg	number of input samples to average	256	N		int	(NoLags, ∞)
NoLags	number of lags to output	64	L		int	(0, ∞)
Unbiased	autocorrelation estimate bias: NO, YES	YES			enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Autocor estimates the autocorrelation function of the input signal. Every time the component fires it reads N samples from its input and outputs 2×L values to its output.
   The output values represent the values of the input signal's autocorrelation function
   $\hat{r}_x(k)$
   ,
   evaluated for k = −L + 1, ... , L

   (!numeric-09-02-03.gif! is output first and $\hat{r}_x(L)$ is output last).
   The 2 × L values written to the output make the output almost symmetrical (discard the last sample to get a perfectly symmetric output).
2. Both unbiased and biased estimates are supported.
   - If Unbiased=YES, the autocorrelation estimate is

$$\hat{r}_x(k) = \begin{cases} \dfrac{1}{N-|k|} \displaystyle\sum_{n=0}^{N-1-|k|} x(n)x(n+|k|), & (-L+1) \le k \le L \\ 0, & \text{otherwise} \end{cases}$$

The unbiased estimate does not guarantee a positive definite sequence, so a power spectral estimate based on this autocorrelation estimate may have negative components.

- If Unbiased = NO, the autocorrelation estimate is

$$\hat{r}_x(k) = \begin{cases} \dfrac{1}{N} \displaystyle\sum_{n=0}^{N-1-|k|} x(n)x(n+|k|), & (-L+1) \le k \le L \\ 0, & \text{otherwise} \end{cases}$$

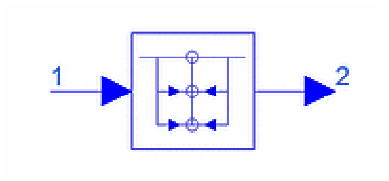
This estimate is biased because the outermost lags have fewer than *N* terms in the summation, yet the summation is still normalized by *N*.

3. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

Biquad



Description: Biquad IIR Filter
Library: Numeric, Signal Processing
Class: SDFBiquad
C++ Code: See *doc/sp_items/SDFBiquad.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
D1	first-order denominator coefficient	-1.1430		real	(-∞, ∞)
D2	second-order denominator coefficient	0.41280		real	(-∞, ∞)
N0	zeroth-order numerator coefficient	0.067455		real	(-∞, ∞)
N1	first-order numerator coefficient	0.135		real	(-∞, ∞)
N2	second-order numerator coefficient	0.067455		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Biquad is a 2-pole, 2-zero digital IIR filter (a biquad). This IIR filter has a Z-domain transfer function of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{N_0 + N_1 z^{-1} + N_2 z^{-2}}{1 + D_1 z^{-1} + D_2 z^{-2}}$$

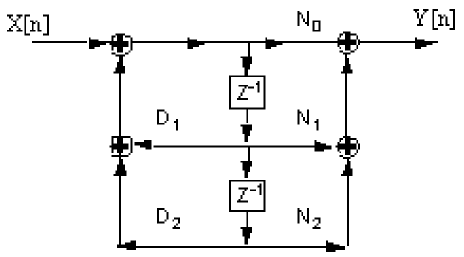
(8-1)
The default is a Butterworth filter with a cutoff 0.1 times sampling frequency.
2. The transfer function in Eq. (8-1) results in the following second order difference equation.

$$y(n) = N_0 x(n) + N_1 x(n-1) + N_2 x(n-2) - D_1 y(n-1) - D_2 y(n-2)$$

where
y(n) is the output for sample *n*
x(n) is the input for sample *n*
3. The transfer function in Eq. (8-1) is a linear time invariant system and can be

rearranged to yield difference equation in direct form II as shown in Yield Difference Equation in Direct Form II.
Indeed, it is the minimum number of delay elements required to implement a system with transfer function given by Eq. (8-1). An implementation with the minimum number of delay elements is also referred to as a canonic form implementation.

Yield Difference Equation in Direct Form II

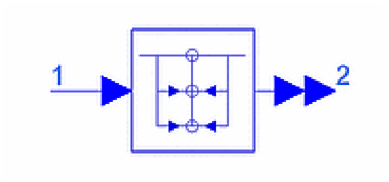


4. See also: *IIR* (numeric), *IIR_Cx* (numeric), *IIR_Fix* (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

BiquadCascade



Description: IIR filter with cascaded biquad IIR sections
Library: Numeric, Signal Processing
Class: SDFBiquadCascade

Parameters

Name	Description	Default	Unit	Type	Range
Taps	sets of six biquad coefficients	0.067455 0.135 0.067455 1.0 -1.143 0.4128		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	The outputs from each of the biquads in the cascade,\n starting with the output from last.	multiple real

Notes/Equations

1. BiquadCascade is a cascade of 2-pole, 2-zero digital IIR filter (a biquad). This IIR filter has a Z-domain transfer function of

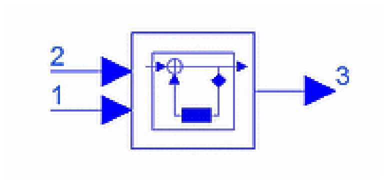
$$H(z) = \Pi\frac{Yi(z)}{Xi(z)} = \Pi\frac{N_{0i} + N_{1i}z^{-1} + N_{2i}z^{-2}}{D_{0i} + D_{1i}z^{-1} + D_{2i}z^{-2}}$$

2. Each biquad section is defined by six coefficients in order: N_{0i} N_{1i} N_{2i} D_{0i} D_{1i} D_{2i} .
3. The multi-output pin contains each of the outputs of the cascade, starting with the output from the last.
4. See also: *Biquad* (numeric), *IIR* (numeric), *IIR_Cx* (numeric), *IIR_Fix* (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

BlockAllPole



Description: All-Pole Filter for Data Blocks
Library: Numeric, Signal Processing
Class: SDFBlockAllPole
C++ Code: See *doc/sp_items/SDFBlockAllPole.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of inputs that use each coefficient set	128		int	(0, ∞)
Order	number of new coefficients to read each time	16		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	coefs	Coefficients of the denominator polynomial	real

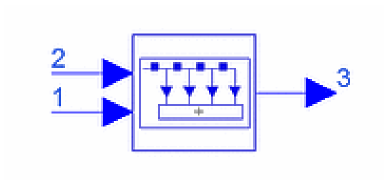
Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real

Notes/Equations

1. BlockAllPole implements an all-pole filter with coefficients that are periodically updated from the outside. For each set of coefficients, a block of input samples is processed, all in one firing.
2. The BlockSize parameter tells how often the updates occur. This integer parameter specifies how many input samples are to be processed using each set of coefficients. The Order parameter tells how many coefficients there are.
3. The transfer function of the filter is
$$H(z) = \frac{1}{1 - d_1 z^{-1} - d_2 z^{-2} - \ldots - d_M z^{-M}}$$
where the *d* values are the externally specified coefficients and *M* is the value of the Order parameter.
4. Decimation or interpolation is not supported.
5. See also: *IIR* (numeric), *IIR_Cx* (numeric), *IIR_Fix* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

BlockFIR



Description: FIR filter for data blocks
Library: Numeric, Signal Processing
Class: SDFBlockFIR
C++ Code: See *doc/sp_items/SDFBlockFIR.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of inputs that use each coefficient set	128		int	(0, ∞)
Order	number of new coefficients to read each time	16		int	(0, ∞)
Decimation	decimation ratio	1		int	(0, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
Interpolation	interpolation ratio	1		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	coefs		real

Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real

Notes/Equations

1. BlockFIR implements an FIR filter with coefficients that are periodically updated from the outside. For each set of coefficients, a block of input samples is processed, all in one firing.
   The BlockSize parameter tells how often updates occur. This integer parameter specifies how many input samples are to be processed using each set of coefficients. The Order parameter tells the number of coefficients.
2. This filter efficiently implements rational sample rate changes. When the Decimation ratio is ≥1 the filter behaves as if it were followed by a DownSample component; when the Interpolation ratio is set, the filter behaves as if it were preceded by an UpSample component. However, the implementation is much more efficient than it would be using UpSample and DownSample. A polyphase structure is used internally, avoiding unnecessary use of memory and multiplication by 0. Arbitrary sample-rate conversions by rational factors can be accomplished this way.
3. The DecimationPhase parameter is somewhat subtle. It is equivalent to the Phase parameter of the DownSample component. When decimating, samples are

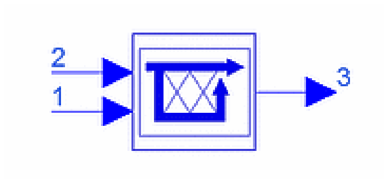
conceptually discarded (although a polyphase structure does not actually compute the discarded samples). To decimate by a factor of three, one of every three outputs is selected. The DecimationPhase parameter determines which of these is selected. When DecimationPhase is 0 (default) the most recent samples are the ones selected.

4. When designing a multirate filter, avoid aliasing. One may assume that the filter sample rate is the product of the Interpolation parameter and the input sample rate. Equivalently, one may use the product of the Decimation parameter and the output sample rate.
5. See also: *IIR* (numeric), *IIR_Cx* (numeric), *IIR_Fix* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling," *Handbook of Digital Signal Processing*, Academic Press, 1987.

BlockLattice



Description: Forward Lattice Filter for Data Blocks
Library: Numeric, Signal Processing
Class: SDFBlockLattice
C++ Code: See *doc/sp_items/SDFBlockLattice.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of inputs that use each coefficient set	128		int	(0, ∞)
Order	number of new coefficients to read each time	16		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	coefs		real

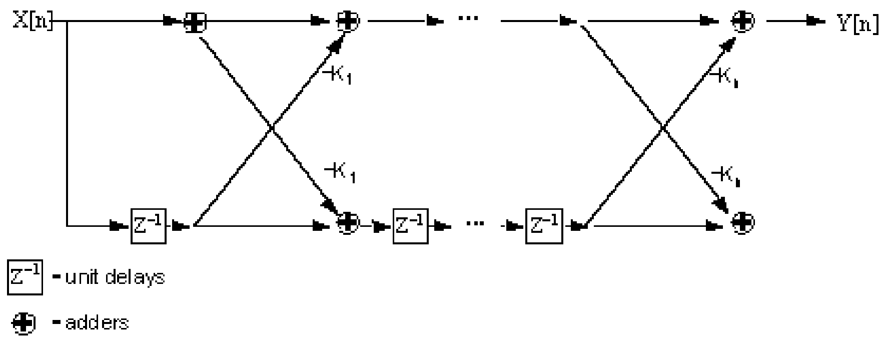
Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real

Notes/Equations

1. BlockLattice implements a forward lattice filter with coefficients that are periodically updated from the outside. For each set of coefficients, a block of input samples is processed, all in one firing.
   The BlockSize parameter tells how often the updates occur. This parameter specifies how many input samples are to be processed using each set of coefficients. The Order parameter tells the number of coefficients.
2. The structure of this filter is shown below. The reflection (PARCOR) coefficients should be specified left to right, K_1 to K_n, as shown.

BlockLattice Filter Structure

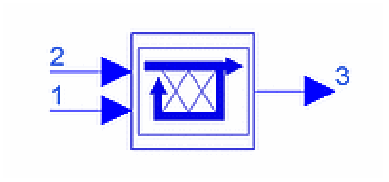


3. The definition of reflection coefficients varies in the literature. The reflection coefficients in [2] and [3] are the negative of the ones used by BlockLattice, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
   The signs of the coefficients used in BlockLattice are appropriate for values given by the LevDur and Burg components.
4. See also: *BlockRLattice* (numeric), *Lattice* (numeric), *RLattice* (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Prediction: A Tutorial Review," *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

BlockRLattice



Description: Recursive Lattice Filter for Data Blocks
Library: Numeric, Signal Processing
Class: SDFBlockRLattice
C++ Code: See *doc/sp_items/SDFBlockRLattice.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
BlockSize	number of inputs that use each coefficient set	128		int	(0, ∞)
Order	number of new coefficients to read each time	16		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	coefs		real

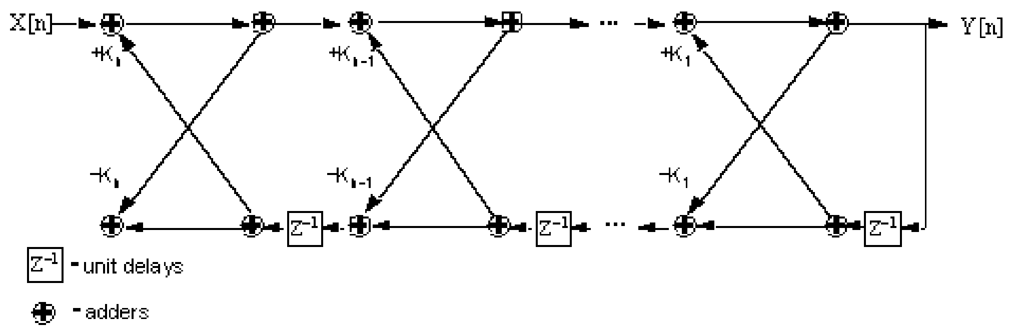
Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real

Notes/Equations

1. BlockRLattice implements a block recursive lattice filter with coefficients that are periodically updated from the outside. For each set of coefficients, a block of input samples is processed, all in one firing.
   The BlockSize parameter tells how often the updates occur. This parameter specifies how many input samples are to be processed using each set of coefficients. The Order parameter tells the number of coefficients.
2. The filter structure is shown below. The reflection (or PARCOR) coefficients should be entered from K_1 to K_n, where K_1 through K_n are specified as shown.

BlockRLattice Filter Structure

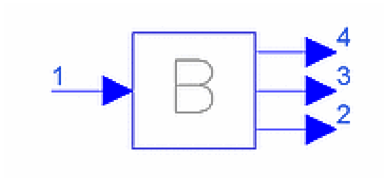


3. The definition of reflection coefficients varies in the literature. The reflection coefficients in [2] and [3] are the negative of the ones used by BlockRLattice, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
   The signs of the coefficients used in BlockRLattice are appropriate for values given by the LevDur and Burg components.
4. See also: *BlockLattice* (numeric), *Lattice* (numeric), *RLattice* (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Linear Prediction: A Tutorial Review," *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

Burg



Description: Linear predictor coefficients estimator
Library: Numeric, Signal Processing
Class: SDFBurg
C++ Code: See *doc/sp_items/SDFBurg.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Order	order of the regression (also number of coefficients to generate)	8		int	(0, ∞)
NumInputs	number of inputs used to generate each set of coefficients	64		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	Input random process.	real

Pin Outputs

Pin	Name	Description	Signal Type
2	lp	AR coefficients output.	real
3	refl	Lattice predictor coefficients output.	real
4	errPower	Prediction error power.	real

Notes/Equations

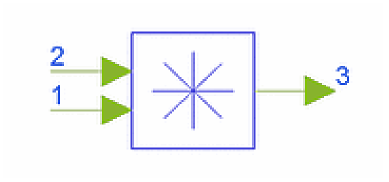
1. Burg uses Burg's algorithm to estimate the linear predictor coefficients of an input random process. The number of inputs looked at is given by the NumInputs parameter and the order of the autoregressive (AR) model is given by the Order parameter. Order specifies how many outputs appear on the lp and refl output portholes.
   These outputs are, respectively, the autoregressive (AR) parameters (also called the linear predictor parameters), and the reflection coefficients. The autoregressive (AR) coefficients are the estimated coefficients of the all-pole filter that could have produced the observations (input data) given a white noise input.
2. The definition of reflection coefficients varies in the literature. The reflection coefficients in [2] and [3] are the negative of the ones generated by Burg, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
3. The errPower output is the power of the prediction error as a function of the model order. There are Order+1 output samples, and the first sample corresponds to the prediction error of a 0th order predictor. This is simply an estimate of the input signal power.
4. See also: *BlockAllPole* (numeric), *BlockLattice* (numeric), *BlockRLattice* (numeric),

LevDur (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

ConvolCx



Description: Complex causal convolution
Library: Numeric, Signal Processing
Class: SDFConvolCx
C++ Code: See *doc/sp_items/SDFConvolCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
TruncationDepth	maximum number of terms in convolution sum	256		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	inA		complex
2	inB		complex

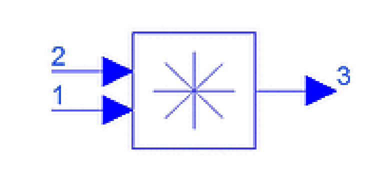
Pin Outputs

Pin	Name	Description	Signal Type
3	out		complex

Notes/Equations

1. ConvolCx convolves two complex causal finite sequences. Set TruncationDepth larger than the number of output samples of interest; if it is smaller, you will get unexpected results after TruncationDepth samples.
2. If one input has finite length and does not change over time, whereas the other input can be arbitrarily long, use the *FIR_Cx* (numeric) component. Set the Taps parameter of the FIR_Cx component to the values of the finite length sequence. For example, if the finite length sequence is (1.5,3.1), (2.8,1.2), (−1.9,0.4), set Taps to "(1.5,3.1) (2.8,1.2) (−1.9,0.4)".
3. See also: *Convolve* (numeric).
4. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

Convolve



Description: Causal Convolution
Library: Numeric, Signal Processing
Class: SDFConvolve
C++ Code: See *doc/sp_items/SDFConvolve.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
TruncationDepth	maximum number of terms in convolution sum	256		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	inA		real
2	inB		real

Pin Outputs

Pin	Name	Description	Signal Type
3	out		real

Notes/Equations

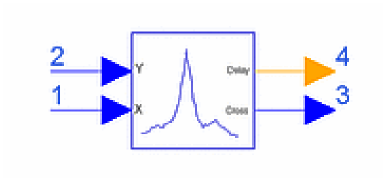
1. Convolve convolves two causal finite sequences. Set TruncationDepth larger than the number of output samples of interest; if it is smaller, you will get unexpected results after TruncationDepth samples.
2. If one input has finite length and does not change over time, whereas the other input can be arbitrarily long, use the *FIR* (numeric) component. Set the Taps parameter of the FIR component to the values of the finite length sequence. For example, if the finite length sequence is 1.5, 3.1, 2.8, 1.2, −1.9, 0.4, set Taps to "1.5 3.1 2.8 1.2 −1.9 0.4".
   If one input has finite length and changes over time, whereas the other input can be arbitrarily long, use the BlockFIR component. BlockFIR allows filtering of a signal in fixed size blocks where each input block is filtered with a different set of coefficients.
3. See also: *ConvolCx* (numeric).
4. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall:

Englewood Cliffs, NJ, 1989.

CrossCorr



Description: Cross-correlation
Library: Numeric, Signal Processing
Class: SDFCrossCorr
C++ Code: See *doc/sp_items/SDFCrossCorr.html* under your installation directory.

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
NoInputsToAvg	number of input samples to average	256	N		int	(NoLags, ∞)
NoLags	number of lags to output	64	L		int	(0, ∞)
Unbiased	autocorrelation estimate bias: NO, YES	YES			enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real
2	input2	second input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
3	output	output signal	real
4	delay	delay of input2 with respect to input1	int

Notes/Equations

1. CrossCorr estimates the cross-correlation function of its two inputs. Every time the component fires it reads N samples from each of its two inputs.
   The number of values written on the output pin is 2×L. These values represent the values of the cross-correlation function
   $\hat{r}_{xy}(k)$
   ,
   evaluated for k = −L + 1, ..., L

   (!numeric-09-12-25.gif! is output first and $\hat{r}_{xy}(L)$ is output last).
   One sample per firing is written on delay pin 4; it represents the estimated delay (in number of samples) of the second input signal with respect to the first input signal (negative values mean that the signal at pin 1 is delayed with respect to the signal at pin 2).
2. Both unbiased and biased estimates are supported.
   - If Unbiased = YES, the autocorrelation estimate is

$$\hat{r_{xy}}(k) = \begin{cases} \dfrac{1}{N-|k|} \displaystyle\sum_{n=0}^{N-1-k} x(n) \cdot y(n+k), & 0 \le k \le L \\ \dfrac{1}{N-|k|} \displaystyle\sum_{n=0}^{N-1-|k|} y(n) \cdot x(n+|k|), & -L < k < 0 \\ 0, & \text{otherwise} \end{cases}$$

- If Unbiased = NO, the cross-correlation estimate is

$$\hat{r_{xy}}(k) = \begin{cases} \dfrac{1}{N} \displaystyle\sum_{n=0}^{N-1-k} x(n) \cdot y(n+k), & 0 \le k \le L \\ \dfrac{1}{N} \displaystyle\sum_{n=0}^{N-1-|k|} y(n) \cdot x(n+|k|), & -L < k < 0 \\ 0, & \text{otherwise} \end{cases}$$

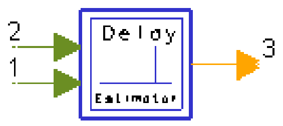
This estimate is biased because the outermost lags have fewer than *N* terms in the summation, and yet the summation is still normalized by *N*.

3. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

DelayEstimator



Description: Delay Estimate
Library: Numeric, Signal Processing
Class: SDFDelayEstimator
C++ Code: See *doc/sp_items/SDFDelayEstimator.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
MaxSampleDelay	Maximum delay estimate samples	100		int	[0, Tstop]

Pin Inputs

Pin	Name	Description	Signal Type
1	Ref	Reference input	complex
2	Test	Test input	complex

Pin Outputs

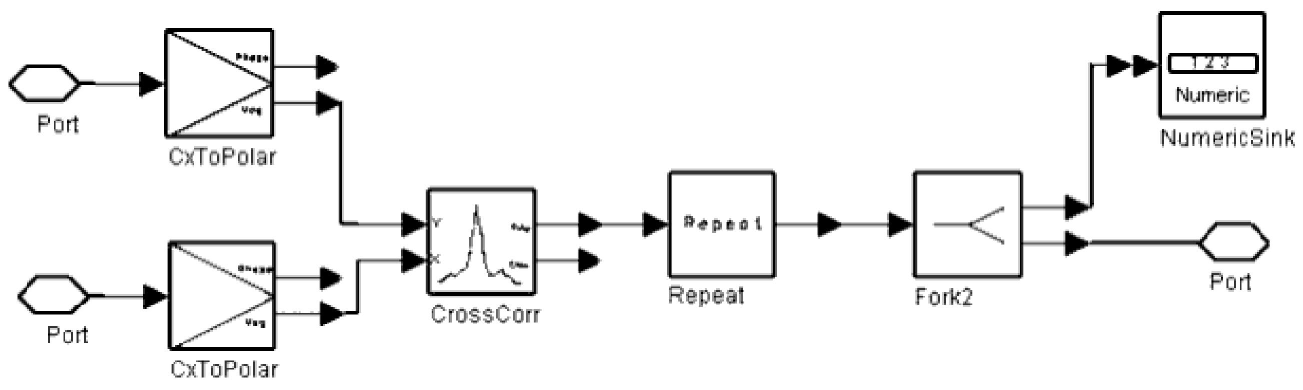
Pin	Name	Description	Signal Type
3	Delay	Delay estimate	int

Notes/Equations

1. This component is used to estimate the delay between two different nodes in an RF subsystem. When simulating multirate PLL systems, it is important to determine the RF subsystem delay.
   The structure of this component is shown in DelayEstimator Structure.
2. This is a single-rate component. Each firing, one input token is consumed for both Ref pin 1 and Test pin 2 and one output token is produced.
   Pin 1 must be connected to a reference signal and pin 2 must be connected to a test signal. The estimated sample delay for the test signal relative to the reference signal will be output.
3. The basic principle for detecting the delay is to perform a cross-correlation for two signals in different nodes.
   Two input complex signals are converted to an I,Q signal by two CxToPolar components, then sent to CrossCorr for performing a cross-correlation to detect the delay between the input signals. To make a single-rate component, the estimated delay is repeated by using a Repeat component then output.
4. The MaxSampleDelay parameter is the upper bound for sample delay estimation; the delay estimate is based on MaxSampleDelay number of input samples.

DelayEstimator Structure

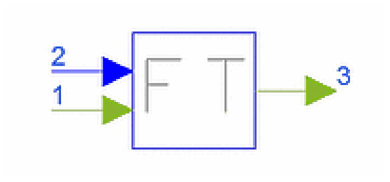
DelayEstimator Structure



References

1. M. Jeruchim, P. Balaban and K. Shanmugan, "Simulation of Communication System," Plenum Press, New York and London, 1992.

DTFT



Description: Discrete-time Fourier transform
Library: Numeric, Signal Processing
Class: SDFDTFT

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
Length	length of input signal	8	L		int	(0, ∞)
NumberOfSamples	number of transform samples to output	128	N		int	(0, ∞)
TimeBetweenSamples	time between input samples (T)	1.0	T		real	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signal	Signal to be transformed.	complex
2	omega	Frequency values at which to sample the transform.	real

Pin Outputs

Pin	Name	Description	Signal Type
3	dtft	The samples of the transform.	complex

Notes/Equations

1. DTFT calculates the discrete-time Fourier transform (DTFT) of the sequence applied at its signal input at each of the frequency points specified on the omega input. Every time the component fires it reads L samples from its signal input and N samples from its omega input and writes N samples to its output.
2. The DTFT of a sequence x[n] is a continuous function of ω defined by

$$X(j\omega) = \sum_{n=-\infty}^{\infty} x[n] \times e^{-j\omega n}$$

If sequence x[n] is obtained by sampling a continuous time signal $x_c(t)$ at intervals of T_s, that is $x[n] = x_c(nTs)$, and if $X_c(f)$, the continuous-time Fourier transform of $x_c(t)$, equals 0 for f > 1/(2T), then X(jω) and $X_c(f)$ have the following relationship:

$$X_c(f) = T \times X(j \times 2\pi f T) = T \times \sum_{n=-\infty}^{\infty} x[n] \times e^{-j2\pi f T n}$$

, for f < 1 / (2T).

3. The DTFT component can calculate *X*(jω) at arbitrary values of ω for sequences *x*[*n*] of finite length. Let the L values on the signal input be *x*[0], *x*[1], ... , *x*[L − 1] and the N values on the omega input be ω[0], ω[1], ... , ω[N − 1]. Then the N values at the output are:

   $$X(j\omega[i]) = \sum_{n=0}^{L-1} x[n] e^{-j\omega[i]nT}$$

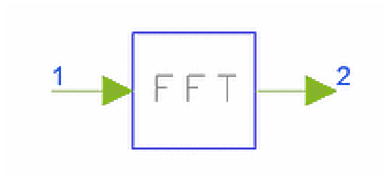
   , i = 0, 1, ... , N − 1.

   where *T* is the time between samples (TimeBetweenSamples). Notice that in this last formula the exponent of e has the extra term T compared to the formula defining the DTFT. Therefore, to calculate the Fourier transform of the corresponding continuous time signal xc(t) at the frequencies f_i , *i* = 0, 1, ... , N, generate the values $\omega_i = 2\pi f_i$ and apply them at the omega input. And, scale the output by T. The values f_i do not need to span the entire frequency range of the signal or be equally spaced.
4. To access the example that shows how this component is used: from the Main window, choose **File > Open > Example > PtolemyDocExamples > Numeric_Signal_Processing_wrk**; from the Schematic window, choose **File > Open Design, DTFT_example**.
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

FFT_Cx



Description: Complex fast Fourier transform
Library: Numeric, Signal Processing
Class: SDFFFT_Cx
C++ Code: See *doc/sp_items/SDFFFT_Cx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Order	base 2 of the transform size	8		int	[0, ∞)
Size	number of input samples to read	256		int	$[1, 2^{Order}]$
Direction	direction of transform: Inverse, Forward	Forward		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		complex

Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

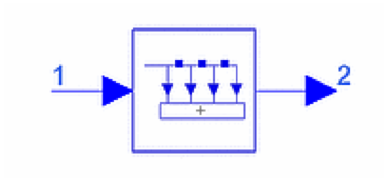
Notes/Equations

1. FFT algorithms are based on the fundamental principle of decomposing the computation of the discrete Fourier transform of a sequence of length N into successively smaller DFT. Many different algorithms are generated based on the decomposing principle, all with comparable improvements in computational speed.
2. FFT_Cx calculates the DFT of a complex input using the fast Fourier transform (FFT) algorithm. FFT_Cx reads Size (default 256) complex samples, zero pads the data if necessary, then takes an FFT of length 2^{Order} where Size $\le 2^{Order}$.
The default value of Order is 8. Direction specifies a forward or inverse FFT. A single firing of FFT_Cx consumes Size inputs and produces 2^{Order} outputs.
3. See also: *DTFT* (numeric).
4. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

FIR



Description: FIR filter
Library: Numeric, Signal Processing
Class: SDFFIR
C++ Code: See *doc/sp_items/SDFFIR.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Taps	filter tap values	-.040609 -.001628 .17853 .37665 .37665 .17853 -.001628 -.040609		real array	
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
Interpolation	interpolation ratio	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. FIR implements a finite-impulse response filter with multirate capability. The default tap coefficients correspond to an eighth-order, equiripple, linear-phase, lowpass filter. The cutoff frequency is approximately one-third of the Nyquist frequency.
2. The filter coefficients are specified by the Taps parameter. The filter coefficients may be specified directly or these may be read from a file. To load filter coefficients from a file, replace the default coefficients with the string *<filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow the FIR filter to work as expected regardless of the directory where the simulation process actually runs). For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. This filter efficiently implements rational sample rate changes. When the Decimation ratio is ≥ 1, the filter behaves exactly as if it were followed by a DownSample component; similarly, when the Interpolation ratio is set, the filter behaves as if it

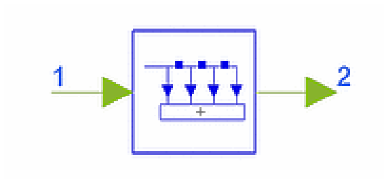
were preceded by an UpSample component. However, the implementation is much more efficient than it would be using UpSample and DownSample. A polyphase structure is used internally, avoiding unnecessary use of memory and unnecessary multiplication by 0. Arbitrary sample-rate conversions by rational factors can be accomplished this way.

4. The DecimationPhase parameter is somewhat subtle. It is equivalent to the Phase parameter of the DownSample component. When decimating, samples are conceptually discarded (although a polyphase structure does not actually compute the discarded samples). For example, to decimate by a factor of 3, one of every 3 outputs is selected. The DecimationPhase parameter determines which of these is selected. If DecimationPhase is 0 (default), the most recent samples are selected.
5. When designing a multirate filter, avoid accidentally introducing aliasing. One may assume that the filter sample rate is the product of the Interpolation parameter and the input sample rate. Equivalently, one may use the product of the Decimation parameter and the output sample rate.
6. See also: *FIR_Cx* (numeric), *FIR_Fix* (numeric).
7. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling," *Handbook of Digital Signal Processing*, Academic Press, 1987.
2. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.
3. P. P. Vaidyanathan, "Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial," *Proc. of the IEEE*, vol. 78, no. 1, pp. 56-93, Jan. 1990.

FIR_Cx



Description: Complex FIR filter
Library: Numeric, Signal Processing
Class: SDFFIR_Cx
C++ Code: See *doc/sp_items/SDFFIR_Cx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Taps	filter tap values	(-.040609,0.0) (-.001628,0.0) (.17853,0.0) (.37665,0.0)(.37665,0.0) (.17853,0.0) (-.001628,0.0) (-.040609,0.0)		complex array	
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
Interpolation	interpolation ratio	1		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		complex

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		complex

Notes/Equations

1. The FIR_Cx component implements a complex-valued finite-impulse response filter with multirate capability. The default tap coefficients correspond to an eighth-order, equiripple, linear-phase, lowpass filter. The cutoff frequency is approximately one-third of the Nyquist frequency.
2. The filter coefficients are specified by the Taps parameter. The real and imaginary parts should be enclosed in parenthesis, for example (0.1,0.3). The filter coefficients may be specified directly or these may be read from a file. To load filter coefficients from a file, replace the default coefficients with the string <*filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow the FIR filter to work as expected regardless of the directory where the simulation process actually runs).
3. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

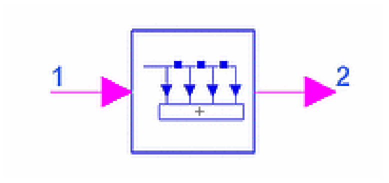
For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

4. This filter efficiently implements rational sample rate changes. When the Decimation ratio is ≥1, the filter behaves exactly as if it were followed by a DownSample component; similarly, when the Interpolation ratio is set, the filter behaves as if it were preceded by an UpSample component. However, the implementation is much more efficient than it would be using UpSample and DownSample. A polyphase structure is used internally, avoiding unnecessary use of memory and unnecessary multiplication by 0. Arbitrary sample-rate conversions by rational factors can be accomplished this way.
5. The DecimationPhase parameter is somewhat subtle. It is equivalent to the Phase parameter of the DownSample component. When decimating, samples are conceptually discarded (although a polyphase structure does not actually compute the discarded samples). For example, to decimate by a factor of 3, one of every 3 outputs is selected. The DecimationPhase parameter determines which of these is selected. If DecimationPhase is 0 (default), the most recent samples are selected.
6. When designing a multirate filter, avoid accidentally introducing aliasing. One may assume that the filter sample rate is the product of the Interpolation parameter and the input sample rate. Equivalently, one may use the product of the Decimation parameter and the output sample rate.
7. See also: *FIR* (numeric), *FIR_Fix* (numeric).
8. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling," *Handbook of Digital Signal Processing*, Academic Press, 1987.

FIR_Fix



Description: Fixed-Point FIR Filter
Library: Numeric, Signal Processing
Class: SDFFIR_Fix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFFIR_Fix.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
Taps	filter tap values	-.040609 -.001628 .17853 .37665 .37665 .17853 -.001628 -.040609	fix array	
Decimation	decimation ratio	1	int	[1, ∞)
DecimationPhase	decimation phase	0	int	[0, Decimation-1]
Interpolation	interpolation ratio	1	int	[1, ∞)
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum	
InputPrecision	precision of input signal, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision	
TapPrecision	precision of tap values, in bits	2.14	precision	
AccumulationPrecision	precision of accumulation, in bits	2.14	precision	
OutputPrecision	precision of output in bits and accumulation	2.14	precision	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		fix

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		fix

Notes/Equations

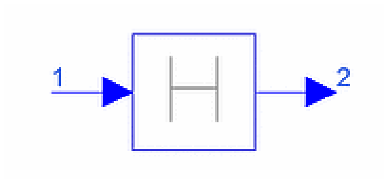
1. FIR implements a finite-impulse response filter with fixed-point capability. The default tap coefficients correspond to an eighth-order, equiripple, linear-phase, lowpass filter. The cutoff frequency is approximately one-third of the Nyquist frequency.
2. The filter coefficients are specified by the Taps parameter. During filter output computation, the precision of the filter taps is converted according to the TapPrecision parameter. The filter coefficients may be specified directly or these may be read from a file. To load filter coefficients from a file, replace the default coefficients with the string <*filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow the FIR filter to work as expected regardless of the directory where the simulation process actually runs). For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. This filter efficiently implements rational sample rate changes. When the Decimation ratio is ≥1, the filter behaves exactly as if it were followed by a DownSample component; similarly, when the Interpolation ratio is set, the filter behaves as if it were preceded by an UpSample component. However, the implementation is much more efficient than it would be using UpSample and DownSample. A polyphase structure is used internally, avoiding unnecessary use of memory and unnecessary multiplication by 0. Arbitrary sample-rate conversions by rational factors can be accomplished this way.
4. The DecimationPhase parameter is somewhat subtle. It is equivalent to the Phase parameter of the DownSample component. When decimating, samples are conceptually discarded (although a polyphase structure does not actually compute the discarded samples). For example, to decimate by a factor of 3, one of every 3 outputs is selected. The DecimationPhase parameter determines which of these is selected. If DecimationPhase is 0 (default), the most recent samples are selected.
5. When designing a multirate filter, avoid accidentally introducing aliasing. One may assume that the filter sample rate is the product of the Interpolation parameter and the input sample rate. Equivalently, one may use the product of the Decimation parameter and the output sample rate.
6. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision. TapPrecision indicates how many bits are used to represent the filter taps.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
7. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
8. See also: *FIR* (numeric), *FIR_Cx* (numeric), *DownSample* (numeric), *UpSample* (numeric).
9. For general information regarding numeric signal processing component signals, refer

to *Numeric Signal Processing Components* (numeric).

References

1. F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling," *Handbook of Digital Signal Processing*, Academic Press, 1987.
2. P. P. Vaidyanathan, "Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial," *Proc. of the IEEE*, vol. 78, no. 1, pp. 56-93, Jan. 1990.

Hilbert



Description: Hilbert transform
Library: Numeric, Signal Processing
Class: SDFHilbert
C++ Code: See *doc/sp_items/SDFHilbert.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
Interpolation	interpolation ratio	1		int	[1, ∞)
N	number of taps in the Hilbert filter	64		int	[1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. This component approximates the Hilbert transform of the input signal by using an FIR filter. The response is truncated symmetrically at $-N/2$ and $N/2$[1], which is accurate enough for some applications. For high accuracy it may be necessary to use the Parks-McClellan algorithm [2] to design a custom Hilbert transformer filter [1,3].
2. The Hilbert transform requires an infinite length set of FIR tap coefficients for accurate representation. This model approximates the Hilbert transform with a finite list of FIR taps. For practical accuracy, it is recommended $N \geq 64$.
3. See also: *FIR* (numeric).
4. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

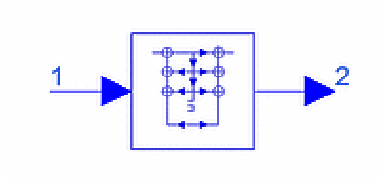
References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.
2. T. W. Parks and J. H. McClellan, "Chebyshev Approximation for Nonrecursive Digital

Filters With Linear Phase," *IEEE Trans. on Circuit Theory*, vol. 19, no. 2, pp. 189-194, March 1972.

3. L. R. Rabiner, J. H. McClellan, and T. W. Parks, "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation," *Proc. of the IEEE*, vol. 63, no. 4, pp. 595-610, April 1975.

IIR



Description: IIR Filter
Library: Numeric, Signal Processing
Class: SDFIIR
C++ Code: See *doc/sp_items/SDFIIR.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Gain	gain	1		real	(-∞, ∞)
Numerator	numerator coefficients	.5 .25 .1		real array	
Denominator	denominator coefficients	1 .5 .3		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

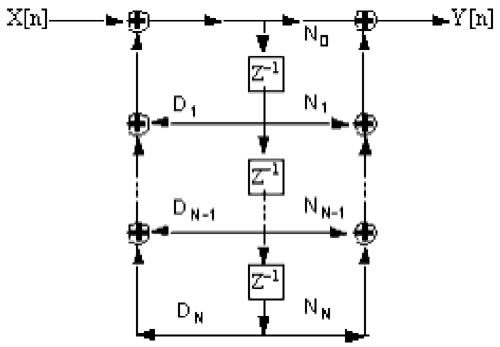
Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. IIR implements an infinite impulse response filter of arbitrary order in a direct form II as shown in IIR Filter Structure.
2. The parameters specify *H(z)*, the Z-transform of an impulse response *h(n)*. The output of IIR is the convolution of the input with *h(n)*.
   The transfer function is of the form
   $$H(z) = G\frac{N(z^{-1})}{D(z^{-1})}$$
   where
   Gain specifies *G*
   Numerator and Denominator specify $N(z^{-1})$ and $D(z^{-1})$, respectively.
   Both arrays start with the constant terms of the polynomial and decrease in powers of *z* (increase in powers of 1/*z*). (The constant term of *D* is not omitted, as is common in other programs that assume it has been normalized to unity.)

IIR Filter Structure

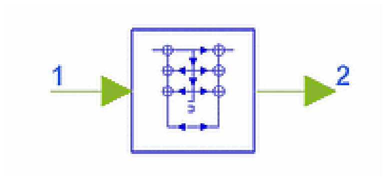


3. Numerator and Denominator array values can be specified directly or read from a file. To load values for a file, replace the default values with the string <*filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow obtain expected results regardless of the directory where the simulation process actually runs). For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
4. The numerical finite precision noise increases with the filter order. To minimize this distortion, expand the filter into a parallel or cascade form.
5. See also: *Biquad* (numeric), *IIR_Cx* (numeric), *IIR_Fix* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

IIR_Cx



Description: Complex IIR Filter
Library: Numeric, Signal Processing
Class: SDFIIR_Cx

Parameters

Name	Description	Default	Unit	Type	Range
Gain	gain	1.0		complex	
Numerator	numerator coefficients	(0.5, 0) (0.25, 0) (0.1, 0)		complex array	
Denominator	denominator coefficients	(1.0, 0) (0.5, 0) (0.3, 0)		complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		complex

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		complex

Notes/Equations

1. IIR_Cx implements a complex infinite impulse response (IIR) filter of arbitrary order in a direct form II realization.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. The parameters specify *H(z)*, the Z-transform of an impulse response *h(n)*. The output is the convolution of the input with *h(n)*. The transfer function is of the form
   $$H(z) = G\frac{N(z^{-1})}{D(z^{-1})}$$
   where
   Gain specifies *G*
   Numerator and Denominator specify $N(z^{-1})$ and $D(z^{-1})$, respectively.
   Both arrays start with the constant terms of the polynomial and decrease in powers of z (increase in powers of 1/z). (The constant term of *D* is not omitted, as is common in other programs that assume it has been normalized to unity.)
4. The Numerator and Denominator array values may be specified directly or these may be read from a file. To load array values for a file, replace the default values with the

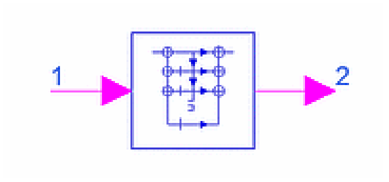
string <*filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow obtain expected results regardless of the directory where the simulation process actually runs).
For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

5. The numerical finite precision noise increases with the filter order. To minimize this distortion, it is often desirable to expand the filter into a parallel or cascade form.
6. See also: *Biquad* (numeric), *IIR* (numeric), *IIR_Fix* (numeric).
7. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

IIR_Fix



Description: Fixed IIR Filter
Library: Numeric, Signal Processing
Class: SDFIIR_Fix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFIIR_Fix.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
Gain	gain	1	real	(-∞, ∞)
Numerator	numerator coefficients	.5 .25 .1	real array	
Denominator	denominator coefficients	1 .5 .3	real array	
CoefPrecision	precision of coefficients	2.14	precision	
UseArrivingPrecision	use precision of arriving data: NO, YES	NO	enum	
InputPrecision	precision of input signal, in bits (used only if UseArrivingPrecision is set to NO)	2.14	precision	
AccumPrecision	precision of state, in bits	2.14	precision	
StatePrecision	precision of state, in bits	2.14	precision	
OutputPrecision	precision of output in bits and accumulation	2.14	precision	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		fix

Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		fix

Notes/Equations

1. IIR_Fix implements an infinite impulse response filter in a direct form II realization using fixed point arithmetic.

The transfer function is of the form

$$H(z) = G\frac{N(z^{-1})}{D(z^{-1})}$$

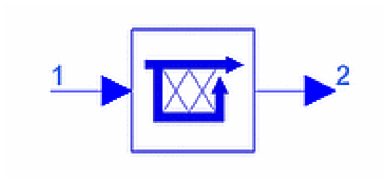
where
N() and *D()* are polynomials
Gain specifies *G*
Numerator and Denominator specify *N()* and *D()*, respectively.
Both arrays start with the constant terms of the polynomial and decrease in powers of z (increase in powers of 1/z). The coefficients are rounded to the precision given by CoefPrecision. (The constant term of D is not omitted, as is common in other programs that assume that it has been normalized to unity. Also, before the numerator and denominator coefficients are quantized, these are rescaled so that the leading denominator coefficient is unity. The gain is multiplied through the numerator coefficients as well.)

2. The numerical finite precision noise increases with the filter order. To minimize this distortion, expand the filter into a parallel or cascade form.
3. Quantization is performed in several places. First, the coefficients are quantized (rounded) to CoefPrecision. This is done after the coefficients have been rescaled to make the initial denominator coefficient unity. The input is optionally quantized (rounded) to precision specified by InputPrecision. The multiplication of the state by the coefficients preserves full precision, but the result is quantized to AccumPrecision after being added to other products. The state variables are stored with the precision given by StatePrecision. Before being sent out, the output values are quantized (rounded) to OutputPrecision.
4. The Numerator and Denominator array values may be specified directly or these may be read from a file. To load array values for a file, replace the default values with the string <*filename*, for example, "</filters/f1.txt", (use an absolute path name for the filename to allow obtain expected results regardless of the directory where the simulation process actually runs).
   For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. If the fixed-point operations cannot fit into the precision specified, overflow occurs with the overflow characteristic specified by OverflowHandler. If ReportOverflow = REPORT, after the simulation has finished the number of overflow errors (if any) will be reported. RoundFix identifies whether fixed-point computations are truncate or round method. If UseArrivingPrecision = NO, the input is cast to the precision specified by InputPrecision. TapPrecision indicates how many bits are used to represent the filter taps.
   For details on these fixed-point parameters refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
6. If UseArrivingPrecision = YES, then components that send a NULL particle on their first firing should not be connected at the input of this component. For example, when a Delay component is connected at its input, such a NULL particle has a precision of 1.0 and the output value will be forced to 0.
7. See also: *Biquad* (numeric), *IIR* (numeric), *IIR_Cx* (numeric).
8. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

Lattice



Description: Lattice Filter
Library: Numeric, Signal Processing
Class: SDFLattice
C++ Code: See *doc/sp_items/SDFLattice.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
ReflectionCoefs	reflection or PARCOR coefficients	0.804534 -0.820577 0.521934 -0.205		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

Pin Outputs

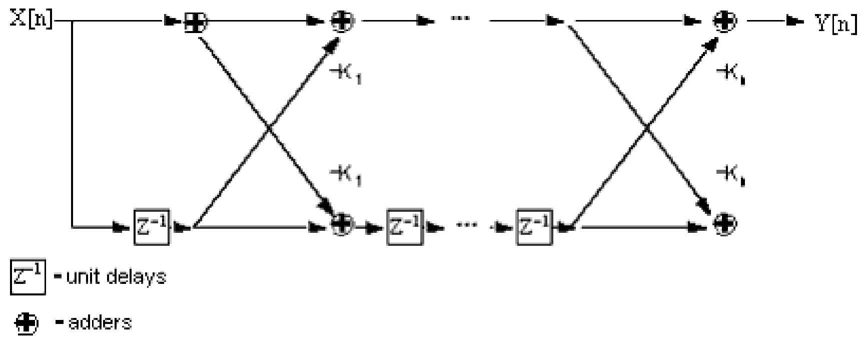
Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. Lattice implements a Lattice filter. The structure of this filter is shown in Lattice Filter Structure. The reflection (PARCOR) coefficients should be specified left to right, K_1 to K_n, as shown.

   Using the same coefficients in the RLattice component will result in the inverse transfer function.
2. The default reflection coefficients correspond to the optimal linear predictor for an AR process generated by filtering white noise with the following filter:
   $$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}}$$
   Because this filter is minimum phase, the transfer function of the lattice filter is $H^{-1}(z)$.

Lattice Filter Structure

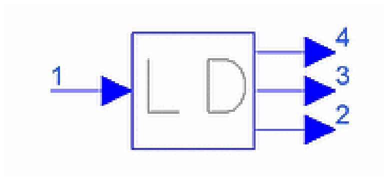


3. To read other reflection coefficients from a file, replace the default coefficients with *<filename>*. Use the full path of the filename so that the simulation will work correctly without regard to the directory from which it runs. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
4. The definition of reflection coefficients varies in the literature. The reflection coefficients in [2] and [3] are the negative of the ones used by Lattice, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
   The signs of the coefficients used in Lattice are appropriate for values given by the LevDur and Burg components.
5. See also: *BlockLattice* (numeric), *BlockRLattice* (numeric), *RLattice* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Prediction: A Tutorial Review," *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

LevDur



Description: FIR and lattice linear predictor coefficients
Library: Numeric, Signal Processing
Class: SDFLevDur
C++ Code: See *doc/sp_items/SDFLevDur.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Order	order of recursion	8		int	(0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	autocor	Autocorrelation estimate	real

Pin Outputs

Pin	Name	Description	Signal Type
2	lp	FIR linear predictor coefficients output.	real
3	refl	Lattice predictor coefficients output.	real
4	errPower	Prediction error power.	real

Notes/Equations

1. LevDur takes as inputs an autocorrelation function, or estimates produced by the Autocor component, and uses the Levinson-Durbin algorithm to compute both reflection coefficients and FIR linear predictor coefficients.
2. If the Autocor component is set so that its Unbiased parameter is 0, then the combined effect of Autocor and LevDur is called the autocorrelation algorithm. Order should be the same as the Autocor NoLags parameter.
3. On the errPower output, a sequence of Order+1 samples gives the prediction error power for each predictor order from 0 to Order. The first sample, which corresponds to the 0th-order predictor, is an estimate of the power of the input process. (For signals without noise, the errPower output can sometimes end up being a small negative number.)
4. The lp output gives the coefficients of an FIR filter that performs linear prediction for the input process. This set of coefficients is suitable for directly feeding the BlockFIR filter component. The number of coefficients produced is equal to Order.
5. The refl output is the reflection coefficients, suitable for feeding directly to the BlockLattice component, which will then generate the forward and backward prediction error. The number of coefficients produced is equal to Order.
6. The definition of reflection coefficients varies in the literature. The reflection coefficients in [2] and [3] are the negative of the ones generated by LevDur, which

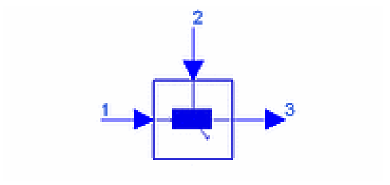
correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.

7. See also: *Autocor* (numeric), *BlockFIR* (numeric), *BlockLattice* (numeric).
8. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Linear Prediction: A Tutorial Review," *Proc. IEEE*, vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application* , Prentice-Hall, Englewood Cliffs, NJ, 1988
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

LMS



Description: LMS adaptive filter
Library: Numeric, Signal Processing
Class: SDFLMS
C++ Code: See *doc/sp_items/SDFLMS.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Taps	filter tap values	-.040609 -.001628 .17853 .37665 .37665 .17853 -.001628 -.040609		real array	
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
StepSize	adaptation step size	0.01		real	(0, ∞)
ErrorDelay	update loop delay	1		int	[1, ∞)
SaveTapsFile	filename in which to save final tap values			string	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	error		real

Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real

Notes/Equations

1. LMS is an adaptive filter using the least-mean square algorithm. The initial filter coefficients are given by the Taps parameter. The default initial coefficients give an 8th-order, linear phase lowpass filter. To read initial coefficients from a file, replace the default coefficients with <*filename*>, preferably specifying a complete path. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. LMS supports decimation, but not interpolation.
2. When used correctly, this LMS adaptive filter will adapt to try to minimize the mean-squared error of the signal at its error input [1]. The output of the filter should be compared to (subtracted from) some reference signal to produce an error signal. That error signal should be fed back to the error input. The ErrorDelay parameter must equal the total number of delays in the path from the output of the filter back

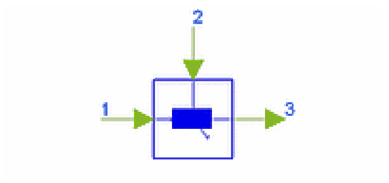
to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be greater than 0 or the simulation will deadlock.
The adaptation algorithm is the well-known LMS, or stochastic-gradient, algorithm.

3. If the SaveTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.
4. See also: *LMS_Cx* (numeric), *LMS_Leak* (numeric), *LMS_OscDet* (numeric).
5. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. S. Haykin, *Adaptive Filter Theory*, Prentice Hall: Englewood Cliffs, NJ. 1991. 2nd ed.

LMS_Cx



Description: Complex LMS adaptive filter
Library: Numeric, Signal Processing
Class: SDFLMS_Cx
C++ Code: See *doc/sp_items/SDFLMS_Cx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Taps	filter tap values	(-.040609,0.0) (-.001628,0.0) (.17853,0.0) (.37665,0.0)(.37665,0.0) (.17853,0.0) (-.001628,0.0) (-.040609,0.0)		complex array	
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
StepSize	adaptation step size	0.01		real	(0, ∞)
ErrorDelay	update loop delay	1		int	[1, ∞)
SaveTapsFile	filename in which to save final tap values			string	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		complex
2	error		complex

Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		complex

Notes/Equations

1. LMS_Cx implements an adaptive filter using the least-mean square algorithm. The initial filter coefficients are given by the Taps parameter. The default initial coefficients give an 8th-order, linear phase lowpass filter. To read initial coefficients from a file, replace the default coefficients with <*filename*>, preferably specifying a complete path. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

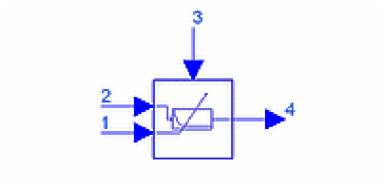
LMS_Cx supports decimation, but not interpolation.

2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. When used correctly, this LMS adaptive filter will adapt to try to minimize the mean-squared error of the signal at its error input [1]. The output of the filter should be compared (subtracted from) some reference signal to produce an error signal. That error signal should be fed back to the error input. The ErrorDelay parameter must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be greater than 0 or the simulation will deadlock.
   The adaptation algorithm is the well-known LMS, or stochastic-gradient algorithm.
4. If the SaveTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.
5. See also: *LMS* (numeric), *LMS_Leak* (numeric), *LMS_OscDet* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. S. Haykin, *Adaptive Filter Theory*, Prentice Hall: Englewood Cliffs, NJ. 1991. 2nd ed.

LMS_Leak



Description: LMS Adaptive Filter with Input Step Size
Library: Numeric, Signal Processing
Class: SDFLMS_Leak
C++ Code: See *doc/sp_items/SDFLMS_Leak.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Taps	filter tap values	-.040609 -.001628 .17853 .37665 .37665 .17853 -.001628 -.040609		real array	
Decimation	decimation ratio	1		int	[1, ∞)
DecimationPhase	decimation phase	0		int	[0, Decimation-1]
ErrorDelay	update loop delay	1		int	[1, ∞)
SaveTapsFile	filename in which to save final tap values			string	
Mu	coefficient update leak factor	0.0		real	(∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	error		real
3	step	Step-size for LMS algorithm.	real

Pin Outputs

Pin	Name	Description	Signal Type
4	signalOut		real

Notes/Equations

1. LMS_Leak is an LMS adaptive filter in which the step size is input (to the step input) every iteration. In addition, the Mu parameter specifies a leakage factor in the updates of the filter coefficients.
2. If two identical LMS_Leak filters are used as an adaptive predictive coder and decoder then, with Mu nearly equal to but greater than 0.0, the effects of channel errors between the coder and decoder will decay rather than accumulate. As Mu increases, the effects of channel errors decay more quickly, but the size of the error input increases also. See page 54 of Reference [1].
3. ErrorDelay must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation

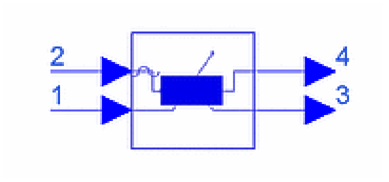
algorithm. The number of delays must be >0 or the simulation will deadlock.

4. If the SaveTapeFile string is non-null, a file will be created with the name given by that string, and the final tape values will be stored there after the run has completed.
5. See also: *LMS* (numeric), *LMS_Cx* (numeric), *LMS_OscDet* (numeric).
6. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. W. Honig and D. G. Messerschmitt, *Adaptive Filters*, Kluwer Academic Publishers, Norwood MA, 1985.

LMS_OscDet



Description: LMS adaptive filter with sinusoid detection
Library: Numeric, Signal Processing
Class: SDFLMS_OscDet
C++ Code

Parameters

Name	Description	Default	Unit	Type	Range
StepSize	adaptation step size	0.01		real	(0, ∞)
ErrorDelay	update loop delay	1		int	[1, ∞)
SaveTapsFile	filename in which to save final tap values			string	
InitialOmega	initial estimated angle, in radians	pi/4		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real
2	error		real

Pin Outputs

Pin	Name	Description	Signal Type
3	signalOut		real
4	cosOmega	Current estimated value of the cosine of the frequency of the dominate\nsinusoidal component of the input signal.	real

Notes/Equations

1. LMS_OscDet tries to lock onto the strongest sinusoidal component in the input signal, and outputs the current estimate of the cosine of the frequency of the strongest component. LMS_OscDet is a 3-tap least-mean square filter whose first and third coefficients are fixed while the second coefficient is adapted. It is a normalized version of the Direct Adaptive Frequency Estimation Technique.
2. The initial taps of this LMS filter are 0.5, −1, 0.5. The second tap is adapted while the others are held fixed. The second tap is equal to $-a_1$; its adaptation has the form

   $$y[n] = \frac{1}{2}x[n] - a_1[k]x[n-1] + \frac{1}{2}x[n-2]$$

   where
   $$a_1[k] = a_1[k-1] + 4\mu e[n]x[n-1]$$

   and *y[n]* is the output of this filter, which can be used as the error signal.

The step size term μ is fixed by the value of the StepSize parameter. You can effectively vary the step size by attenuating the error term as

$$e[n] = \frac{y[n]}{k}$$

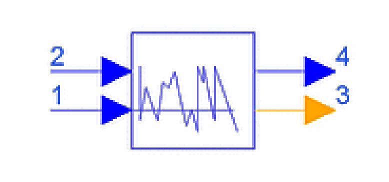
assuming that k = 1, 2, 3, and so forth. When the error becomes relatively small, this filter gives an estimate of the strongest sinusoidal component:

$$a_1 = \cos(\omega)$$

The taps here are scaled by one-half from those of other implementations; therefore, the output of the filter is also scaled by one-half. To compensate for this scaling, μ is multiplied by 2 relative to other implementations with full scale taps.

3. LMS_OscDet outputs the current value of a_1 on the cosOmega output port. The initial value is a_1 = 1 (0 frequency) so the initial value of the second tap is −1.
4. ErrorDelay must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be >0 or the simulation will deadlock.
5. If the SaveTapeFile string is non-null, a file will be created with the name given by that string, and the final tape values will be stored there after the run has completed.
6. See also: *LMS* (numeric), *LMS_Cx* (numeric), and *LMS_Leak* (numeric).
7. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

PattMatch



Description: Cross-correlation with template input
Library: Numeric, Signal Processing
Class: SDFPattMatch
C++ Code: See *doc/sp_items/SDFPattMatch.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
TempSize	number of samples in template	32		int	(0, ∞)
WinSize	number of samples in search template	176		int	[TempSize, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	templ	template input	real
2	window	window input	real

Pin Outputs

Pin	Name	Description	Signal Type
3	index	index output	int
4	values	cross-correlation output	real

Notes/Equations

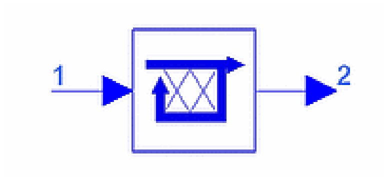
1. PattMatch accepts a template and a search window and tries to find the position in the search window where the template matches best. Every time the component fires, it reads TempSize samples from its templ input and WinSize samples from its window input. At the same time, it writes one sample to its index output and (WinSize − TempSize + 1) samples to its values output.
   The algorithm for finding the best template match position starts by placing the template at the left end of the window (first samples of template and window are aligned) and calculating the cross-correlation between them. Then the template is shifted across the window one sample at a time and the cross-correlation is computed at each step until the template reaches the right end of the window (last samples of template and window are aligned). The cross-correlation values are output on the values output. The index output is the value of the shift (in number of samples) that gives the largest cross-correlation.
2. The cross-correlation values are normalized against the energy of the window under the template:

$$C(n) = \frac{\sum_{m=0}^{T_{size}-1} T(m)W(m+n)}{\sum_{m=0}^{T_{size}-1} W(m+n)W(m+n)}$$

where *T* is the template, *W* is the window, *n* is the index value and *T* $_{size}$ equals TempSize.
Note that if the template is identical to a certain segment of the window, then the cross-correlation value *C(n)* for that segment will be 1.0. Therefore, the index with the highest cross-correlation value may not be the best match if that value is greater than 1.0.

3. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).
4. To access the example that shows how this component is used: from the Main window, choose **File > Open > Example > PtolemyDocExamples > Numeric_Signal_Processing_wrk**; from the Schematic window, choose **File > Open, PattMatch_example**.

RLattice



Description: Recursive Lattice Filter
Library: Numeric, Signal Processing
Class: SDFRLattice
C++ Code: See *doc/sp_items/SDFRLattice.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
ReflectionCoefs	reflection or PARCOR coefficients	0.804534 -0.820577 0.521934 -0.205		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	signalIn		real

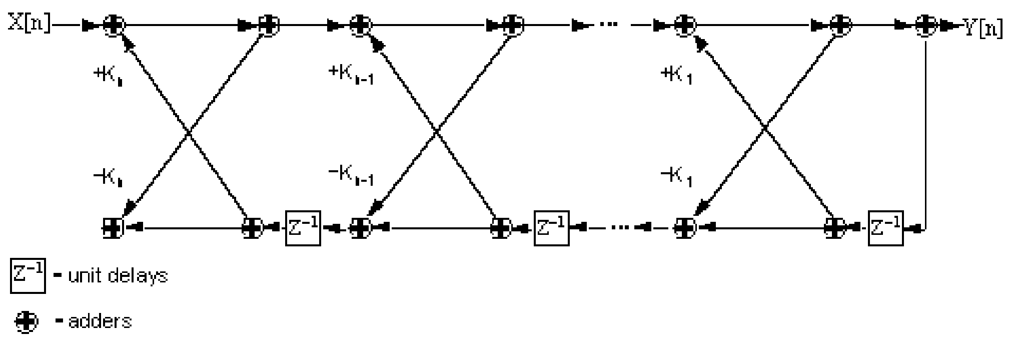
Pin Outputs

Pin	Name	Description	Signal Type
2	signalOut		real

Notes/Equations

1. RLattice implements a recursive lattice filter (also referred to as the Lattice inverse filter). The structure of this filter is:

RLattice Filter Structure



where Z $^{-1}$ are unit delays and + are adders. The reflection (or PARCOR) coefficients should be entered from K $_1$ to K $_n$, left to right, where K $_1$ through K $_n$ are specified as above.

2. Using the same coefficients in the Lattice component will result in the inverse transfer function.
3. The default reflection coefficients correspond to the optimal linear predictor for an AR process generated by filtering white noise with the following filter:

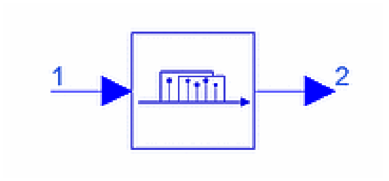
$$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}}$$

4. To read other reflection coefficients from a file, replace the default coefficients with <*filename*>. Use the full path of the filename so that the simulation will work correctly without regard to the directory from which it runs. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. The definition of reflection coefficients varies in the literature. The reflection coefficients in References [2] and [3] are the negative of the ones used by RLattice, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.
   The signs of the coefficients used in RLattice are appropriate for values given by the LevDur and Burg components.
6. See also: *BlockLattice* (numeric), *BlockRLattice* (numeric), *IIR* (numeric), *Lattice* (numeric).
7. For general information regarding numeric signal processing component signals, refer to *Numeric Signal Processing Components* (numeric).

References

1. J. Makhoul, "Linear Prediction: A Tutorial Review," *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.
2. S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
3. S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SlidWinAvg



Description: Sliding-Window Average
Library: Numeric, Signal Processing
Class: SDFSlidWinAvg
C++ Code: See *doc/sp_items/SDFSlidWinAvg.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
WindowSize	size of sliding window	3		int	(1, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. SlidWinAvg outputs the average of the last WindowSize input values.
   For the first (WindowSize −1) output samples for which less than WindowSize input samples are available, the missing values are assumed to be 0.
   This component is equivalent to an FIR filter with WidowSize taps all equal to 1/WindowSize.

Numeric Sources

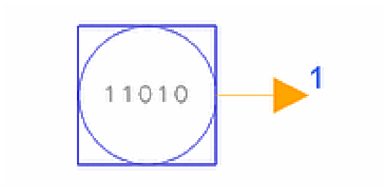
- *Bits* (numeric)
- *ComplexExp* (numeric)
- *Const* (numeric)
- *ConstCx* (numeric)
- *ConstFix* (numeric)
- *ConstInt* (numeric)
- *Cx M* (numeric)
- *DataPattern* (numeric)
- *DiagonalCx M* (numeric)
- *DiagonalFix M* (numeric)
- *DiagonalInt M* (numeric)
- *Diagonal M* (numeric)
- *Fix M* (numeric)
- *Float M* (numeric)
- *IdentityCx M* (numeric)
- *IdentityFix M* (numeric)
- *IdentityInt M* (numeric)
- *Identity M* (numeric)
- *IID Gaussian* (numeric)
- *IID Uniform* (numeric)
- *ImpulseFloat* (numeric)
- *Int M* (numeric)
- *NumericExpression* (numeric)
- *NumericSource* (numeric)
- *RampFix* (numeric)
- *RampFloat* (numeric)
- *RampInt* (numeric)
- *ReadFile* (numeric)
- *ReadFilePreProc* (numeric)
- *Rect* (numeric)
- *RectCx* (numeric)
- *RectCxDoppler* (numeric)
- *RectFix* (numeric)
- *SineGen* (numeric)
- *WaveForm* (numeric)
- *WaveFormCx* (numeric)
- *Window* (numeric)

The Numeric Sources component library contains scalar and matrix signal sources for floating-point (real), fixed-point, complex and integer data.

Some components accept parameter values that are arrays of data. The syntax for referencing arrays of data as parameter values includes an explicit list of values, a reference to a file that contains those values, or a combination of explicit values along with file references. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components operate with fixed-point numbers. These components use one or more parameters that define the characteristics of the fixed-point processing. These parameters include: OverflowHandler, OutputPrecision, RoundFix, ReportOverflow, and others. For details on the use of these parameters for fixed-point components a refer to *Parameters for Fixed-Point Components* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. The arithmetic used by these components is two's complement. Therefore, all precision values must specify at least one bit to the left of the decimal point (used as sign bit).

Bits



Description: Binary random bits output
Library: Numeric, Sources
Class: SDFBits

Parameters

Name	Description	Default	Symbol	Unit	Type	Range
Type	type of bit sequence, random or pseudo random: Random, Prbs	Random			enum	
ProbOfZero	probability of bit value being zero (used when Type=Random)	0.5			real	[0, 1]
LFSR_Length	Linear Feedback Shift Register length (used when Type=Prbs)	12	L		int	[2, 31]
LFSR_InitState	Linear Feedback Shift Register initial state (used when Type=Prbs)	1			int	[1, 2^L - 1]

Pin Outputs

Pin	Name	Description	Signal Type
1	output	output bit stream	int

Notes/Equations

Note
Due to a corrective change made to this component in the ADS 2008 release, its output bit sequence in ADS 2008 (and later releases) differs from the one generated in previous releases when Type=Prbs. In ADS 2008 (and later releases) the first output bit (in the period of $2^L - 1$ bits) is what used to be the last output bit (in the period of $2^L - 1$ bits) in releases prior to ADS 2008.

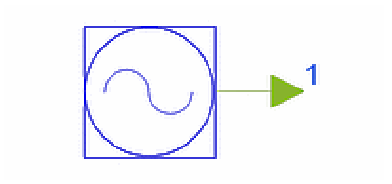
1. Bits generates random or pseudo-random binary bit sequences.
2. When Type = Random, Bits generates a random output bit sequence for which the probability of each bit being 0 is equal to ProbOfZero. If ProbOfZero is set to a value less than 0 it is considered to be equal to 0; if ProbOfZero is set to a value greater than 1 it is considered to be equal to 1.
   (The LFSR_Length and LFSR_InitState parameters are ignored in this mode.)
   The random bit sequence is generated by making use of the random number generator. Therefore, the bit pattern will be different for each instance of the Bits component. In addition, if other components that use the random number generator (for example, Noise, IID_Gaussian, RES with RTemp > −273.15) are added or removed from a design the output bit sequences from the Bits components will change.
   The output bit sequence is also dependent on the value of the DefaultSeed parameter

in the data flow controller (DF), which provides the initial seed for the random number generator.

- When DefaultSeed = 0, the initial seed value is obtained from the system time so the output bit sequence generated for each simulation will be different even if nothing else changes on the design.
- When DefaultSeed > 0, the output bit sequence generated for each simulation, though statistically random, has the same initial seed starting condition and therefore results in reproducible simulations.

3. When Type = Prbs, the output bit sequence is pseudo-random and is generated by using an LFSR (linear feedback shift register).
   The LFSR_Length parameter sets the LFSR length that, in turn, defines the period of the sequence ($2^L - 1$). If LFSR_Length is outside its valid range [2, 31], it is reset to its default value of 12.
   The LFSR_InitState parameter sets the initial state of the LFSR. If LFSR_InitState is outside its valid range [1, $2^L - 1$], it is reset to its default value of 1. The ProbOfZero parameter is ignored in this mode of operation. Since the random number generator is not used in this case, the output bit sequence does not depend on the DefaultSeed parameter of the DF controller.
   Two instances of the Bits source with Type set to Prbs and the same values for the LFSR_Length and LFSR_InitState parameters will generate the exact same output no matter what the DefaultSeed value is or if the rest of the design is modified.
   To get two or more uncorrelated pseudo-random bit sequences, place two or more Bits components, set their Type parameters to Prbs, their LFSR_Length parameters to the same value, and their LFSR_InitState parameters to different values. The maximum number of uncorrelated sequences one can generate with LFSRs of length L is $2^L - 1$.
4. See also: *LFSR* (numeric).
5. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ComplexExp



Description: Complex exponential source
Library: Numeric, Sources
Class: SDFComplexExp
Derived From: SineGen

Parameters

Name	Description	Default	Unit	Type	Range
RadiansPerSample	radians per sample	pi/50		real	(-∞, ∞)
InitialRadians	initial phase, in radians	0		real	(-∞, ∞)

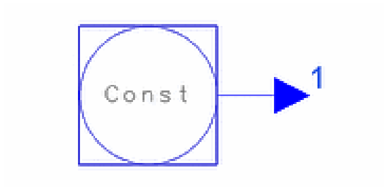
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	complex

Notes/Equations

1. ComplexExp generates the sequence of numbers given by
   $\cos(\omega \times n + \varphi) + j \times \sin(\omega \times n + \varphi)$, $n = 0, 1, \ldots$,
   where ω equals RadiansPerSample and φ equals InitialRadians.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Const



Description: Constant output
Library: Numeric, Sources
Class: SDFConst
C++ Code: See *doc/sp_items/SDFConst.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Level	value	0.0		real	(-∞, ∞)

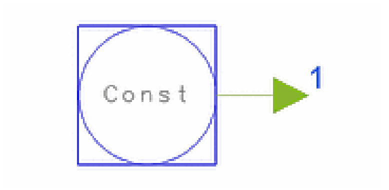
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. Const outputs a constant signal with a value given by the Level parameter (default 0.0).
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ConstCx



Description: Complex constant output
Library: Numeric, Sources
Class: SDFConstCx
C++ Code: See *doc/sp_items/SDFConstCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Real	real part	0.0		real	(-∞, ∞)
Imag	imaginary part	0.0		real	(-∞, ∞)

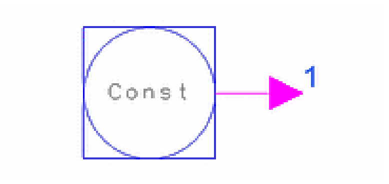
Pin Outputs

Pin	Name	Description	Signal Type
1	output		complex

Notes/Equations

1. ConstCx outputs a complex constant signal with the real part given by the Real parameter (default 0.0) and the imaginary part given by the Imag parameter (default 0.0).
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ConstFix



Description: Fixed-Point Constant Output
Library: Numeric, Sources
Class: SDFConstFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFConstFix.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
Level	constant value	0.0	fix	(-∞, ∞)
OutputPrecision	precision of output in bits and accumulation	2.14	precision	

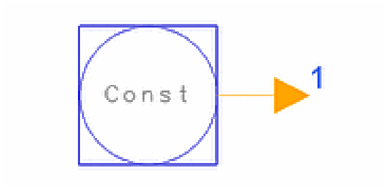
Pin Outputs

Pin	Name	Description	Signal Type
1	output		fix

Notes/Equations

1. ConstFix outputs a fixed-point constant signal with a value given by the Level parameter (default 0.0).
2. The output precision is specified using an *l.r* format: *l* is the number of bits to the left of the decimal place (including the sign bit); *r* is the number of bits to the right of the decimal place. For example, the precision *2.22* would represent a 24-bit fixed-point number with 1 sign bit, 1 integer bit, and 22 fractional bits.
3. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used a sign bit).
4. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ConstInt



Description: Integer constant output
Library: Numeric, Sources
Class: SDFConstInt
C++ Code: See *doc/sp_items/SDFConstInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Level	constant value	0		int	(-∞, ∞)

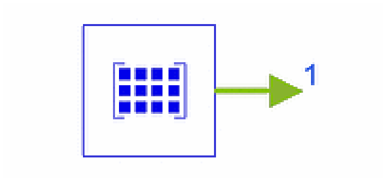
Pin Outputs

Pin	Name	Description	Signal Type
1	output		int

Notes/Equations

1. ConstInt outputs a constant signal with a value given by the Level parameter (default 0).
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Cx_M



Description: Complex Matrix Output
Library: Numeric, Sources
Class: SDFCx_M
Derived From: MatrixConstant

Parameters

Name	Description	Default	Type	Range
NumRows	the number of rows in the matrix	2	int	[1, ∞)
NumCols	the number of columns in the matrix	2	int	[1, ∞)
ComplexMatrixContents	complex valued elements of output matrix	1 j (-1) (-j)	complex array	

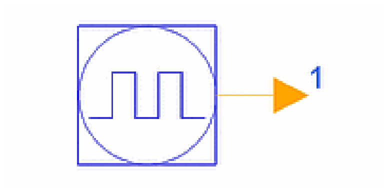
Pin Outputs

Pin	Name	Description	Signal Type
1	output		complex matrix

Notes/Equations

1. Cx_M produces a matrix with complex entries. Entries are read from the ComplexMatrixContents array parameter in rasterized order; for example, for an M × N matrix, the first row is filled from left to right using the first N values from the array.
   The ComplexMatrixContents value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, *<filename*.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   Value Types (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

DataPattern



Description: Patterned data source
Library: Numeric, Sources
Class: SDFDataPattern

Parameters

Name	Description	Default	Unit	Type	Range
DataPattern	data pattern: PN9, PN15, FIX4, _4_1_4_0, _8_1_8_0, _16_1_16_0, _32_1_32_0, _64_1_64_0	PN9		enum	

Pin Outputs

Pin	Name	Description	Signal Type
1	output	patterned data output	int

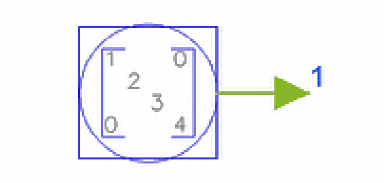
Notes/Equations

1. This model is used to generate one of eight patterned bit streams.
2. For the DataPattern parameter:
   - if PN9 is selected, a 511-bit pseudo-random test pattern is generated according to CCITT Recommendation O.153
   - if PN15 is selected, a 32767-bit pseudo-random test pattern is generated according to CCITT Recommendation O.151
   - if FIX4 is selected, a zero-stream is generated
   - if x_1_x_0 is selected, where *x* equals 4, 8, 16, 32, or 64, a periodic bit stream is generated, with the period being 2 × *x*. In one period, the first *x* bits are 1s and the second x bits are 0s.
3. For information regarding numeric source signals, refer to the *Numeric Sources* (numeric).

References

1. CCITT, Recommendation O.151(10/92).
2. CCITT, Recommendation O.153(10/92).

DiagonalCx_M



Description: Complex Diagonal Matrix Output
Library: Numeric, Sources
Class: SDFDiagonalCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)
DiagonalElements	complex diagonal elements of output matrix	1 j	complex array	

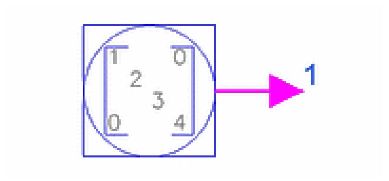
Pin Outputs

Pin	Name	Description	Signal Type
1	output		complex matrix

Notes/Equations

1. DiagonalCx_M outputs a diagonal matrix of size (RowsCols × RowsCols) with the diagonal elements given in the DiagonalElements parameter. All diagonal elements are complex numbers.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

DiagonalFix_M



Description: Fixed-Point Diagonal Matrix Output
Library: Numeric, Sources
Class: SDFDiagonalFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)
OutputPrecision	precision of output in bits and accumulation	2.14	string	
DiagonalElements	fixed-point diagonal elements of output matrix	1 -2	fix array	

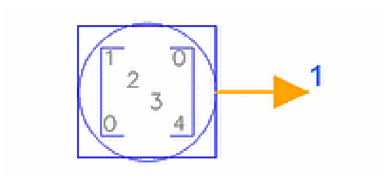
Pin Outputs

Pin	Name	Description	Signal Type
1	output		fix matrix

Notes/Equations

1. DiagonalFix_M outputs a diagonal matrix of size (RowsCols × RowsCols) with the diagonal elements given in the DiagonalElements parameter with the specified precision.
2. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used as sign bit).
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

DiagonalInt_M



Description: Integer Diagonal Matrix Output
Library: Numeric, Sources
Class: SDFDiagonalInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)
DiagonalElements	integer diagonal elements of output matrix	1 2	int array	

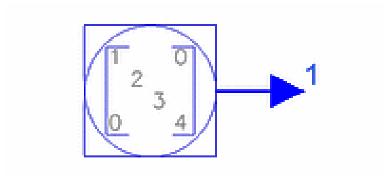
Pin Outputs

Pin	Name	Description	Signal Type
1	output		int matrix

Notes/Equations

1. DiagonalInt_M outputs a diagonal matrix of size (RowsCols × RowsCols) with the diagonal elements given in the DiagonalElements parameter. All diagonal elements are integer numbers.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Diagonal_M



Description: Diagonal Matrix Output
Library: Numeric, Sources
Class: SDFDiagonal_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)
DiagonalElements	diagonal elements of output matrix	1.0 2.0	real array	

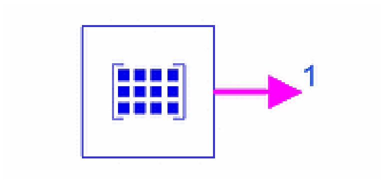
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real matrix

Notes/Equations

1. Diagonal_M outputs a diagonal matrix of size (RowsCols × RowsCols) with the diagonal elements given in the DiagonalElements parameter. All diagonal elements are floating-point (real) numbers.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Fix_M



Description: Fixed-Point Matrix Output
Library: Numeric, Sources
Class: SDFFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
NumRows	number of rows in output matrix	2	int	[1, ∞)
NumCols	number of columns in output matrix	2	int	[1, ∞)
FixMatrixContents	fixed-point elements of output matrix	1 -2 2 -2	fix array	
OutputPrecision	precision of output in bits and accumulation	2.14	precision	

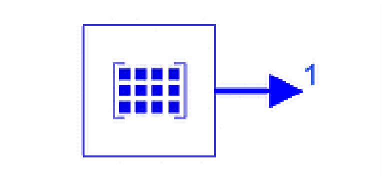
Pin Outputs

Pin	Name	Description	Signal Type
1	output		fix matrix

Notes/Equations

1. Fix_M generates a matrix with fixed-point entries. Entries are read from the FixMatrixContents array parameter in rasterized order; for example, for an M × N matrix, the first row is filled left to right using the first N values from the array. All entries have the same precision, as specified by OutputPrecision.
2. The FixMatrixContents value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, *<filename* . For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used a sign bit).
4. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Float_M



Description: Matrix Output
Library: Numeric, Sources
Class: SDFFloat_M
Derived From: MatrixConstant

Parameters

Name	Description	Default	Type	Range
NumRows	the number of rows in the matrix	2	int	[1, ∞)
NumCols	the number of columns in the matrix	2	int	[1, ∞)
FloatMatrixContents	floating-point(real) elements of matrix	1.0 -2.0 2.0 -2.0	real array	

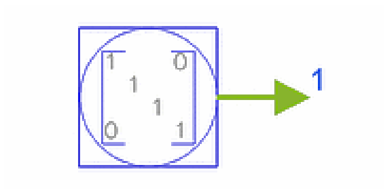
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real matrix

Notes/Equations

1. Float_M produces a matrix with floating-point (real) entries. Entries are read from the FloatMatrixContents array parameter in rasterized order; for example, for an M × N matrix, the first row is filled from left to right using the first N values from the array.
2. The FloatMatrixContents value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, <*filename* . For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

IdentityCx_M



Description: Complex Identity Matrix Output
Library: Numeric, Sources
Class: SDFIdentityCx_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)

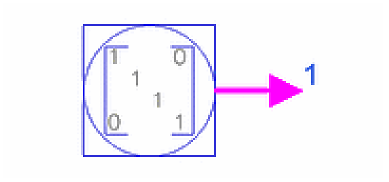
Pin Outputs

Pin	Name	Description	Signal Type
1	output		complex matrix

Notes/Equations

1. IdentityCx_M outputs an identity matrix of the specified size.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

IdentityFix_M



Description: Fixed-Point Identity Matrix Output
Library: Numeric, Sources
Class: SDFIdentityFix_M
Derived From: SDFFix

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)
OutputPrecision	precision of output in bits and accumulation	2.14	string	

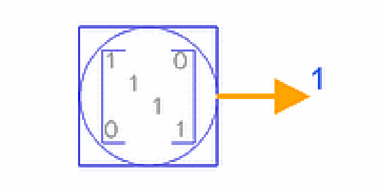
Pin Outputs

Pin	Name	Description	Signal Type
1	output		fix matrix

Notes/Equations

1. IdentityFix_M outputs an identity matrix of the specified size with the specified precision.
2. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used a sign bit).
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

IdentityInt_M



Description: Integer Identity Matrix Output
Library: Numeric, Sources
Class: SDFIdentityInt_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)

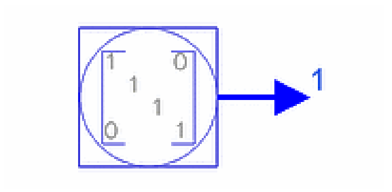
Pin Outputs

Pin	Name	Description	Signal Type
1	output		int matrix

Notes/Equations

1. IdentityInt_M outputs an identity matrix of the specified size.

Identity_M



Description: Identity Matrix Output
Library: Numeric, Sources
Class: SDFIdentity_M
Derived From: MatrixBase

Parameters

Name	Description	Default	Type	Range
RowsCols	number of rows and columns in output square matrix	2	int	[1, ∞)

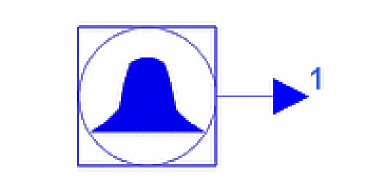
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real matrix

Notes/Equations

1. Identity_M outputs an identity matrix of the specified size.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

IID_Gaussian



Description: IID Gaussian Distributed Noise Output
Library: Numeric, Sources
Class: SDFIID_Gaussian
C++ Code: See *doc/sp_items/SDFIID_Gaussian.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Mean	mean of distribution	0.0		real	(-∞, ∞)
Variance	variance of distribution	1.0		real	(-∞, ∞)

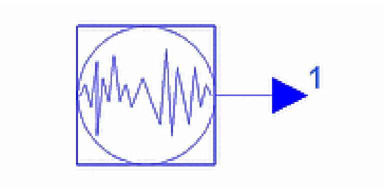
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. IID_Gaussian generates an identically independently distributed white Gaussian pseudo-random process with mean (default 0) and variance (default 1) specified by the Mean and Variance parameters.
2. The noise is random for each IID_Gaussian instance. The noise is dependent on the value of the DefaultSeed in the data flow controller (DF). When DefaultSeed = 0, the noise generated for each simulation is different. When DefaultSeed > 0, the noise generated for each simulation, though random, has the same initial seed starting condition and thus results in reproducible simulations.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

IID_Uniform



Description: IID Uniform Distributed Noise Output
Library: Numeric, Sources
Class: SDFIID_Uniform
C++ Code: See *doc/sp_items/SDFIID_Uniform.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Lower	lower limit	0.0		real	(-∞, ∞)
Upper	upper limit	1.0		real	[Lower, ∞)

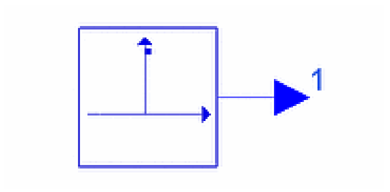
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. IID_Uniform generates an identically independently distributed uniformly distributed pseudo-random process. The output is uniformly distributed between Lower (default 0.0) and Upper (default 1.0) limits.
2. Noise is random for each IID_Uniform instance and is dependent on the value of the DefaultSeed in the data flow controller (DF). When DefaultSeed = 0, then the noise generated for each simulation is different; when DefaultSeed > 0, then the noise generated for each simulation, though random, has the same initial seed starting condition and thus results in reproducible simulations.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ImpulseFloat



Description: Impulse output
Library: Numeric, Sources
Class: SDFImpulseFloat
C++ Code: See *doc/sp_items/SDFImpulseFloat.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Level	height of impulse	1.0		real	(-∞, ∞)
Period	if greater than zero, period of impulse train	0		int	[0, ∞)
Delay	output delay	0		int	[0, ∞)

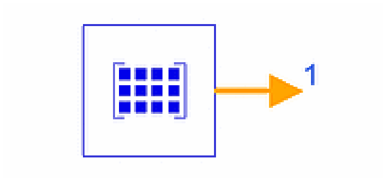
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. ImpulseFloat generates a single impulse or an impulse train, with an amplitude specified by Level (default 0.0). If Period (default 0) is equal to 0, then only a single impulse is generated; otherwise Period specifies the period of the impulse train. The impulse or impulse train is delayed by the amount specified by Delay.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Int_M



Description: Integer Matrix Output
Library: Numeric, Sources
Class: SDFInt_M
Derived From: MatrixConstant

Parameters

Name	Description	Default	Type	Range
NumRows	the number of rows in the matrix	2	int	[1, ∞)
NumCols	the number of columns in the matrix	2	int	[1, ∞)
IntMatrixContents	integer elements of output matrix	1 -2 2 -2	int array	

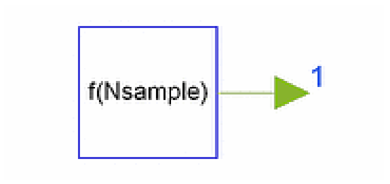
Pin Outputs

Pin	Name	Description	Signal Type
1	output		int matrix

Notes/Equations

1. Int_M produces a matrix with integer entries. Entries are read from the IntMatrixContents array parameter in rasterized order; for example, for an M × N matrix, the first row is filled from left to right using the first N values from the array.
2. The IntMatrixContents value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, <*filename* . For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

NumericExpression



Description: Numeric Expression Data output
Library: Numeric, Sources
Class: SDFNumericExpression

Parameters

Name	Description	Default	Type
Expression	expression, which can be function of "Nsample"	0.0+j*0.0	complex

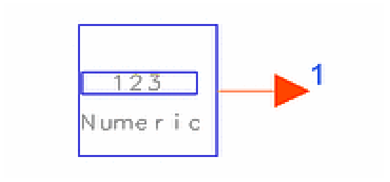
Pin Outputs

Pin	Name	Description	Signal Type
1	output	numeric source output signal	complex

Notes/Equations

1. This component is used to generate numeric data output evaluated using an expression. Expression can be any valid expression, following the syntax used for writing expression on a VAR block.
   If the Expression is dependent on predefined variable, *Nsample*, then the output will be dependent on the sample number, which is incremented for each firing of this component determined by the schedule.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

NumericSource



Description: Numeric signal generator using dataset
Library: Numeric, Sources
Class: SDFNumericSource

Parameters

Name	Description	Default	Type	Range
ControlSimulation	if set to YES, Period (or if Period=0 then the index of last data sample in the file) determines how long the simulation will run: NO, YES	NO	enum	
Periodic	if YES then output is periodic: NO, YES	YES	enum	
Period	period of the output waveform if Periodic=YES. If Period=0 then period is the index of the last data sample read	0	int	[0, ∞)
DataSet	dataSet file to construct Expression from		filename	
Expression	variable/sink name from dataset or a valid dataSet expression (data can be multi dimensional from 1-D to 3-D)		string	

Pin Outputs

Pin	Name	Description	Signal Type
1	output	Numeric source output signal	anytype

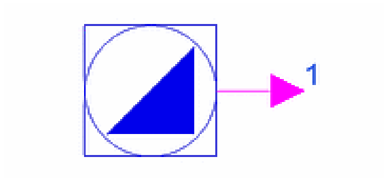
Notes/Equations

1. This component is used to generate numeric data output evaluated using a pre-generated dataset. Expression can be any valid expression using variables available in the dataset. The syntax used for writing expression is the same as writing an expression to display the data in a Data Display window.
   If the dataset was generated using a Sweep, and the expression results in multidimensional data, the output will be matrix data. The expression must evaluate into data that is up to 3-dimensional. Any expression that results in higher dimension (> 3-D) data will error out. To reduce the dimensionality, use the "[..., ::, ...]" operator.
   For example, consider a design that has a NumericSink *N1* and 3 levels of sweep. If such a dataset is used for generating data using NumericSource and the Expression was set to "N1", the simulation will error out saying it was 4- dimensional data. To fix it you can use "N1[0, ::, ::, ::]", which will now generate 3-dimensional matrix data at the output.
   If the length of simulation is larger than the available data in the dataset, use the Periodic and Period parameters to repeat the old data. The Periodic parameter must be set to YES for the output to repeat after the sample number equal to Period. If Periodic = YES and Period = 0, the Period will be the index on the last data read in

the dataset, and all of the data from the dataset will be read and repeated. If Periodic = NO, the output will be zero after all data is read.
If ControlSimulation = YES, Period will determine how long the simulation runs. If Period = 0, the simulation will run until the last data in the dataset is read.

2. The variable specified in an expression cannot be a variable that represents matrix data generated using DSP designs.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RampFix



Description: Fixed-Point Ramp Output
Library: Numeric, Sources
Class: SDFRampFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFRampFix.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
OutputPrecision	precision of output in bits and accumulation	2.14	precision	
Step	increment from one sample to the next	1.0	fix	(-∞, ∞)
Value	initial (or latest) value output by RampFix	0.0	fix	(-∞, ∞)

Pin Outputs

Pin	Name	Description	Signal Type
1	output		fix

Notes/Equations

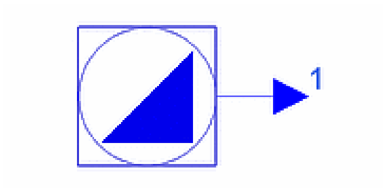
1. RampFix generates a ramp signal, starting at Value (default 0.0) and incrementing by the step size specified by Step (default 1.0).
2. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used a sign bit).
3. The value of the Step and Value parameters and their precision in bits can be specified using two different notations.
   Specifying only a value in the dialog box would create a fixed-point number with the default precision, which has a total length of 32 bits with the number of integer bits set as required by the value of the parameter. For example, the default value 1.0 creates a fixed-point object with precision 2.30, and a value like 0.5 would create one with precision 1.31.
   An alternate way of specifying the value and the precision is to use the parentheses notation, which will be interpreted as (value, precision). For example, (2.546, 3.5)

would create a fixed-point object by casting the double-precision floating-point (real) number 2.546 to a fixed-point precision of 3.5.
This component has three *precision* specifications:

- OutputPrecision given by designer
- Step parameter precision (default or given by designer)
- Value parameter precision (default or given by designer)
  Certain conditions must be satisfied to get reasonable results.
- the Step parameter precision should not have more integer or fractional bits than OutputPrecision. Otherwise, the extra (if any) fractional bits will be handled according to the value of the RoundFix parameter and the extra (if any) integer bits will be handled according to the value of the OverflowHandler parameter.
- if Value is not equal to 0, the OutputPrecision should not have more integer or fractional bits than Value parameter precision. Otherwise, the extra (if any) fractional bits will be handled according to the value of the RoundFix parameter and the extra (if any) integer bits will be handled according to the value of the OverflowHandler parameter.
  Examples (OverflowHandler = wrapped and RoundFix = TRUNCATE is assumed):
- Specifying OutputPrecision = "5.1" and Step = 0.25, will result in a constant output equal to the value of the Value parameter possibly wrapped and truncated to fit the output precision.
- Specifying OutputPrecision = "5.1", Step = 0.5 and Value = 4.0 (default precision is 4.28) will result in an output starting at 4.0, incrementing by 0.5 at each step and *saturating* when 7.5 is reached.
- Specifying OutputPrecision = "4.1", Step = 0.75 and Value = "(3.0, 4.1)" will result in an output starting at 3.0, incrementing by 0.5 at each step and wrapping to −8 after 7.5 is reached. The same output is obtained if Value has other precisions specified that have more integer or fractional bits than OutputPrecision. For example, "(3.0, 6.3)" will produce the same results.

4. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RampFloat



Description: Ramp output
Library: Numeric, Sources
Class: SDFRampFloat
C++ Code: See *doc/sp_items/SDFRampFloat.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Step	increment from one sample to the next	1.0		real	(-∞, ∞)
Value	initial value output	0.0		real	(-∞, ∞)

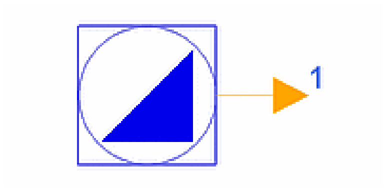
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. RampFloat generates a ramp signal, starting at Value (default 0.0) and incrementing by the step size (default 1.0) specified by the Step parameter.
   Because doubles have finite precision, the maximum value that RampFloat can output is Step/DBL_EPSILON. For example, for a Step of 1, the maximum is 1FFFFFFFFFFFFF, or 9007199254740991. After that value, the output will remain constant.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RampInt



Description: Integer ramp output
Library: Numeric, Sources
Class: SDFRampInt
C++ Code: See *doc/sp_items/SDFRampInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Step	increment from one sample to the next	1		int	(-∞, ∞)
Value	initial value output	0		int	(-∞, ∞)

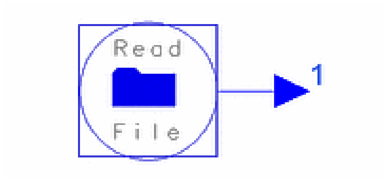
Pin Outputs

Pin	Name	Description	Signal Type
1	output		int

Notes/Equations

1. RampInt generates an integer ramp signal, starting at Value (default 0) and incrementing by the step size specified by Step (default 1).
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ReadFile



Description: Waveform output from file
Library: Numeric, Sources
Class: SDFReadFile
C++ Code: See *doc/sp_items/SDFReadFile.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
FileName	input file name	file.txt		filename	
ControlSimulation	control simulation: NO, YES	NO		enum	
OutputType	output type: zero padded, periodic	periodic		enum	

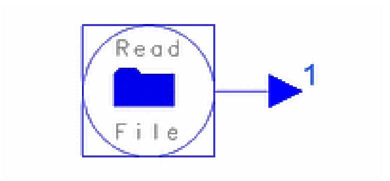
Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. ReadFile reads ASCII data from a file. The simulation can be halted at end of file, the file contents can be periodically repeated, or the file contents can be padded with zeroes.
2. The input file is to be a text file that contains real array data in ADS Ptolemy format. For details on this file format refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

ReadFilePreProc



Description: Waveform output from file with preprocessing using a shell command
Library: Numeric, Sources
Class: SDFReadFilePreProc
Derived From: ReadFile

Parameters

Name	**Description**	**Default**	**Unit**	**Type**	**Range**
FileName	input file name	file.txt		filename	
ControlSimulation	control simulation: NO, YES	NO		enum	
OutputType	output type: zero padded, periodic	periodic		enum	
PerlFile	data file pre-processing perl script			filename	

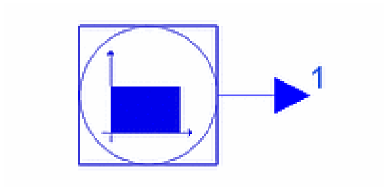
Pin Outputs

Pin	**Name**	**Description**	**Signal Type**
1	output		real

Notes/Equations

1. ReadFilePreProc pre-processes the ASCII datafile specified in the FileName parameter, using the perl script provided in PerlFile parameter. It is equivalent to executing the command `perl PerlFile FileName' then using the results as ASCII input to the design. The original datafile is not modified; instead, the processed file is temporarily saved in the data directory (under the name *tmp<InstanceName>.txt*) and removed at the end of simulation. The simulation can be halted at the end of file, the file contents can be periodically repeated, or these can be padded with zeroes.
2. The resulting file must be a text file that contains real array data in ADS Ptolemy format. For details on this file format refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. Use of this component is demonstrated in the **File > Open > Example > PtolemyDocExamples > Numeric_Sources_wrk**. Open the networks design *ReadFilePreProc_example*.
4. Also see: *ReadFile* (numeric).
5. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Rect



Description: Rectangular pulse output
Library: Numeric, Sources
Class: SDFRect
C++ Code: See *doc/sp_items/SDFRect.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Height	height of rectangular pulse	1.0		real	(-∞, ∞)
Width	width of rectangular pulse	8		int	[0, ∞)
Period	if greater than zero, repetition period of pulse stream	0		int	[0, ∞)

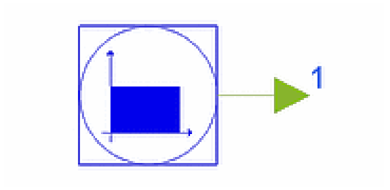
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	real

Notes/Equations

1. Rect generates a rectangular pulse of height and width specified by Height and Width. If Period > 0, the pulse is repeated with the given period.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RectCx



Description: Complex rectangular pulse output
Library: Numeric, Sources
Class: SDFRectCx
C++ Code: See *doc/sp_items/SDFRectCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Height	height of rectangular pulse	1.0		complex	
Width	width of rectangular pulse	240		int	[0, ∞)
Period	period of pulse stream	1024		int	[0, ∞)

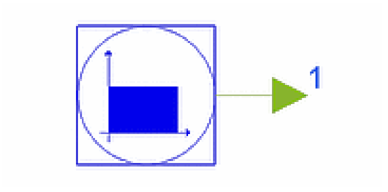
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	complex

Notes/Equations

1. RectCx generates a complex rectangular pulse specified by Height and Width. If Period > 0, the pulse is repeated with the given period.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RectCxDoppler



Description: Complex rectangular Doppler pulse output
Library: Numeric, Sources
Class: SDFRectCxDoppler
C++ Code: See *doc/sp_items/SDFRectCxDoppler.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Width	width of rectangular pulse	240		int	[0, ∞)
Period	period of pulse stream	1024		int	[0, ∞)
Bandwidth	signal bandwidth	1.0e9		real	[0.0, ∞)
Te	duration time	30.0*10^-6		real	[0.0, ∞)
Fe	emission frequency	3.0e9		real	[0.0, ∞)
Fsimu	simulation frequency	8.0e6		real	[0.0, ∞)
Vn	target velocity	150.0		real	[0.0, ∞)
Tp	pulse period	1.0e-3		real	[0.0, ∞)
Np	pulse number	16		int	[0, ∞)
Fpor	carrier frequency	3.0e9		real	[0.0, ∞)
C	light speed	3.0e8		real	[0.0, 3e8)
SNRn	signal-to-noise ratio	10.0		real	[0, ∞)
SqrPthn	square root of noise power	1.0		real	[0, ∞)
Sdelay	target delay	0		real	[0.0, ∞)

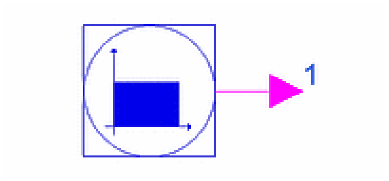
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	complex

Notes/Equations

1. RectCxDoppler generates a complex rectangular pulse of width specified by Width. If Period > 0, the pulse is repeated with the given period.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

RectFix



Description: Fixed-Point Rectangular Pulse Output
Library: Numeric, Sources
Class: SDFRectFix
Derived From: SDFFix
C++ Code: See *doc/sp_items/SDFRectFix.html* under your installation directory.

Parameters

Name	Description	Default	Type	Range
OverflowHandler	output overflow characteristic: wrapped, saturate, zero_saturate, warning	wrapped	enum	
ReportOverflow	simulation overflow error report option: DONT_REPORT, REPORT	REPORT	enum	
RoundFix	fixed-point computations, assignments, and data type conversions option: TRUNCATE, ROUND	TRUNCATE	enum	
Height	height of rectangular pulse	1.0	fix	(-∞, ∞)
Width	width of rectangular pulse	8	int	[0, ∞)
Period	period of pulse stream	0	int	[0, ∞)
OutputPrecision	precision of output in bits and accumulation	2.14	precision	

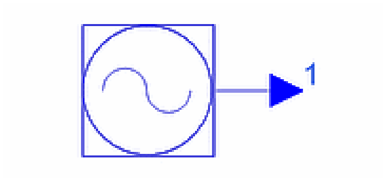
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	fix

Notes/Equations

1. RectFix generates a fixed-point rectangular pulse specified by Height and Width. If Period > 0, the pulse is repeated with the given period.
2. OutputPrecision is specified using an *l.r* format, where *l* is the number of bits to the left of the decimal place (including the sign bit) and *r* is the number of bits to the right of the decimal place. For example, the precision *2.22* would represent a 24-bit fixed-point number with 1 sign bit, 1 integer bit, and 22 fractional bits.
3. This component uses two's-complement arithmetic; the values of the OutputPrecision parameter given by the designer must specify at least 1 bit to the left of the decimal place (used a sign bit).
4. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

SineGen



Description: Sine wave output
Library: Numeric, Sources
Class: SDFSineGen
C++ Code: See *doc/sp_items/SDFSineGen.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
RadiansPerSample	radians per sample	pi/50		real	(-∞, ∞)
InitialRadians	initial phase, in radians	0		real	(-∞, ∞)

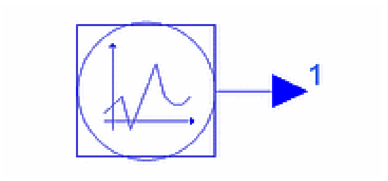
Pin Outputs

Pin	Name	Description	Signal Type
1	output	output signal	real

Notes/Equations

1. SineGen generates the sequence of numbers given by sin(ω × n +Φ), n = 0, 1, ... , where ω equals RadiansPerSample and Φ equals InitialRadians.
2. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

WaveForm



Description: Waveform output
Library: Numeric, Sources
Class: SDFWaveForm
C++ Code: See *doc/sp_items/SDFWaveForm.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Value	waveform values	1 -1		real array	
ControlSimulation	control simulation: NO, YES	NO		enum	
Periodic	periodic output: NO, YES	YES		enum	
Period	period of waveform when greater than zero	0		int	[0, ∞)

Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. Waveform outputs a waveform specified by Value. You can get periodic signals with any period, and halt a simulation at the end of the given waveform. Waveform Outputs summarizes the operations.
   Value can be specified directly or read from a file. To use data from a file, replace the default coefficients with the string, <*filename*. For details using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation. The size of the array is currently limited to 20,000 samples. The complete file is be read and its contents stored in an array. To read longer files use the ReadFile component, which reads one sample at a time and therefore uses less storage.

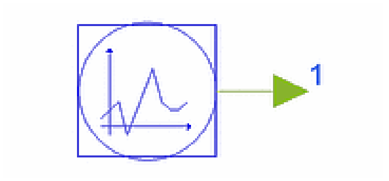
   Waveform Outputs

StopSimulation	Periodic	Period	Operation
do not stop	yes	0	period is length of waveform
do not stop	yes	N>0	period is N
do not stop	no	any	output the waveform once, then zeros
stop at end	any	any	stop after outputting the waveform once

2. For information regarding numeric source signals, refer to *Numeric Sources*

(numeric).

WaveFormCx



Description: Complex waveform output
Library: Numeric, Sources
Class: SDFWaveFormCx
C++ Code: See *doc/sp_items/SDFWaveFormCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Value	waveform values	(1) (-1)		complex array	
ControlSimulation	control simulation: NO, YES	NO		enum	
Periodic	periodic output: NO, YES	YES		enum	
Period	period of waveform when greater than zero	0		int	[0, ∞)

Pin Outputs

Pin	Name	Description	Signal Type
1	output		complex

Notes/Equations

1. WaveFormCx outputs a complex waveform as specified by Value. You can get periodic signals with any period, and halt a simulation at the end of the given waveform. Waveform Operations are summarized below.
The Value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, <*filename*. The size of the array is currently limited to 20,000 samples. The entire file will be read and its contents stored in an array. To read longer files, use the ReadFile component, which reads one sample at a time and therefore uses less storage.

Waveform Operations

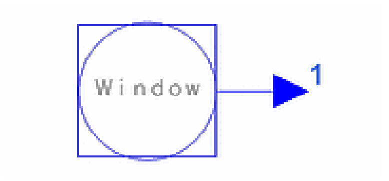
StopSimulation	Periodic	Period	Operation
do not stop	yes	0	period is length of waveform
do not stop	yes	N>0	period is N
do not stop	no	any	output the waveform once, then zeros
stop at end	any	any	stop after outputting the waveform once

2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the

ADS Ptolemy Simulation (ptolemy) documentation.

3. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

Window



Description: Window data
Library: Numeric, Sources
Class: SDFWindow
C++ Code: See *doc/sp_items/SDFWindow.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Name	name of window function to generate (Rectangle, Bartlett, Hanning, Hamming, Blackman, SteepBlackman, or Kaiser)	Hanning		string	
Length	length of window function to produce	256		int	[4, ∞)
Period	period of the output	0		int	[0, ∞)
WindowParameters	array of values for the window	0		real array	

Pin Outputs

Pin	Name	Description	Signal Type
1	output		real

Notes/Equations

1. Window generates standard window functions or periodic repetitions of standard window functions. One period of samples is produced at each simulation. It produces output values that are samples of a standard windowing function.
2. Length is the length of the window to produce; most window functions have a 0 value at the first and last sample.
3. Period specifies the period of the output signal. The window will be zero-padded if required. Period = 0 means a period equal to Length.
   A negative period will produce one window, then output 0 for all later samples. A period of less than the window length will be equivalent to a period of the window length (that is, Period = 0).
4. For the Kaiser window, the first entry in WindowParameters is taken as the beta parameter that is proportional to the stopband attenuation of the window.
5. The WindowParameters value may be specified directly or these may be read from a file. To use data from a file, replace the default coefficients with the string, <*filename*. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
6. For information regarding numeric source signals, refer to *Numeric Sources* (numeric).

References

1. Leland Jackson, *Digital Filters and Signal Processing*, 2nd ed., Kluwer Academic Publishers, ISBN 0-89838-276-9, 1989.

Numeric Special Functions

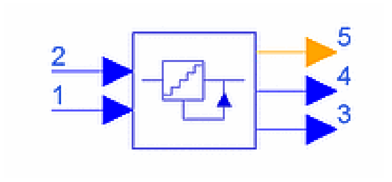
- *AdaptLinQuant* (numeric)
- *Compress* (numeric)
- *DeadZone* (numeric)
- *Dirichlet* (numeric)
- *Expand* (numeric)
- *LatchClocked* (numeric)
- *Limit* (numeric)
- *LinQuantIdx* (numeric)
- *MuLaw* (numeric)
- *OrderTwoInt* (numeric)
- *PcwzLinear* (numeric)
- *Polynomial* (numeric)
- *Quant* (numeric)
- *QuantIdx* (numeric)
- *Quantizer* (numeric)
- *Quantizer2D* (numeric)
- *SchmittTrig* (numeric)
- *Table* (numeric)
- *TableCx* (numeric)
- *TableInt* (numeric)
- *Toggle* (numeric)
- *Unwrap* (numeric)

The numeric special functions components provide data processing functions common to communication systems such as signal quantizers, signal compressor, signal expandors and other block that operate on single data points or arrays of data that are integer, double precision floating-point (real), or complex values. Each component accepts a specific class of signal and outputs a resultant signal. (These components do not accept any matrix class of signal.)

If a component receives another class of signal, the received signal is automatically converted to the signal class specified as the input of the component. Auto conversion from a higher to a lower precision signal class may result in loss of information. The auto conversion from timed, complex or floating-point (real) signals to a fixed signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the auto conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30, and a value of 0.5 would create one of precision of 1.31. For details on conversions between different classes of signals, refer to *Conversion of Data Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

Some components accept parameter values that are arrays of data. The syntax for referencing arrays of data as parameter values includes an explicit list of values, a reference to a file that contains those values, or a combination of explicit values along with file references. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.

AdaptLinQuant



Description: Adaptive linear quantizer
Library: Numeric, Special Functions
Class: SDFAdaptLinQuant
C++ Code: See *doc/sp_items/SDFAdaptLinQuant.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Bits	number of bits	8		int	[1, 31]

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real
2	inStep		real

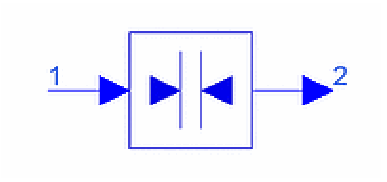
Pin Outputs

Pin	Name	Description	Signal Type
3	amplitude		real
4	outStep		real
5	stepLevel		int

Notes/Equations

1. AdaptLinQuant quantizes the input to the number of levels given by 2^{Bits}. The quantization levels are uniformly spaced at the step size given by the inStep input value and are odd symmetric about zero. Therefore, the *high* threshold is (2^{Bits-1})(inStep/2), and the *low* threshold is the negative of the *high* threshold.
2. Rounding to the nearest level is performed. The output level will equal *high* only if the input level equals or exceeds *high*. If the input is below *low*, then the quantized output will equal *low*.
3. The quantized value is output on the amplitude port as a floating-point (real) value, the step size is output on the outStep port as a floating-point (real) value, and the index of the quantization level is output on the stepLevel port as a non-negative integer between 0 and 2^{Bits-1}.
4. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Compress



Description: Compression part of a compander
Library: Numeric, Special Functions
Class: SDFCompress
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Type	compression law: MU-law, A-law	MU-law		enum	
CompressionK	compression constant	1		real	
Max	maximum input value magnitude	1		real	(0.0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

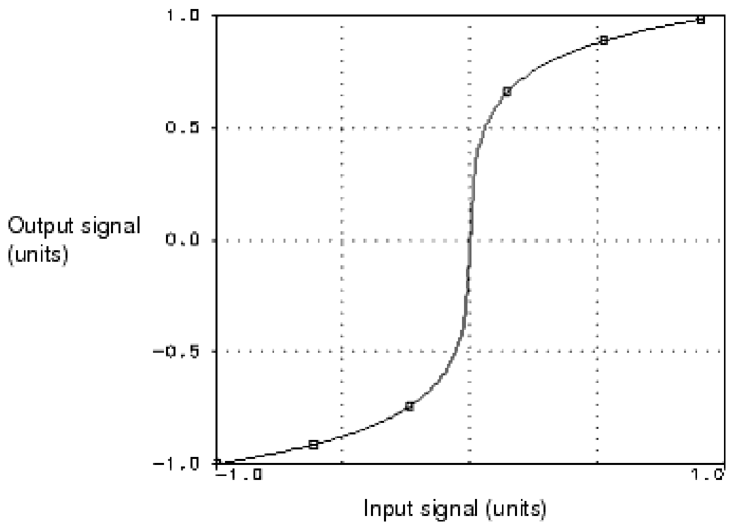
Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Compress can be used to obtain the MU-law and A-law compression characteristics. The output signal is always a baseband signal.
2. Let *x′ (n) = x(n)/Max*
   MU-law:
   $$y(n) = V_M \frac{\operatorname{sgn}[x'(n)]\ln\{1.0 + \mu|x'(n)|\}}{\ln(1.0+\mu)} \text{ for } \mu \ge 0$$
   A-law:
   $$y(n) = \begin{cases} V_M \dfrac{\operatorname{sgn}[x'(n)]A|x'(n)|}{1.0 + \ln(A)} & \text{for } |x'(n)| < 1/A \\ V_M \dfrac{\operatorname{sgn}[x'(n)]\{1 + \ln[A|x'(n)|]\}}{1 + \ln(A)} & \text{for } |x'(n)| \ge 1/A \end{cases}$$
   where
   y(n) is the Output for sample *n*
   x(n) is the Input for sample *n*
   V_M is Max, the maximum input value magnitude
   μ is the compression constant for MU-law
   A is the compression constant for A-law

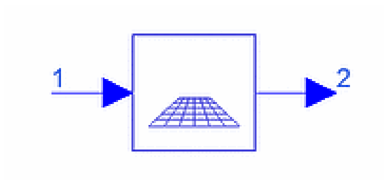
3. The output signal versus input signal plot of the Compress component, with Type = MU-law, CompressionK = 255, and Max = 1V, is shown below.

Compress Signal Plot



1. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

DeadZone



Description: Dead Zone Nonlinearity
Library: Numeric, Special Functions
Class: SDFDeadZone
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
K	magnitude gain	1		real	(-∞, 0.0) or (0.0, ∞)
Low	lower dead zone value	0		real	(-∞, High)
High	higher dead zone value	1		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

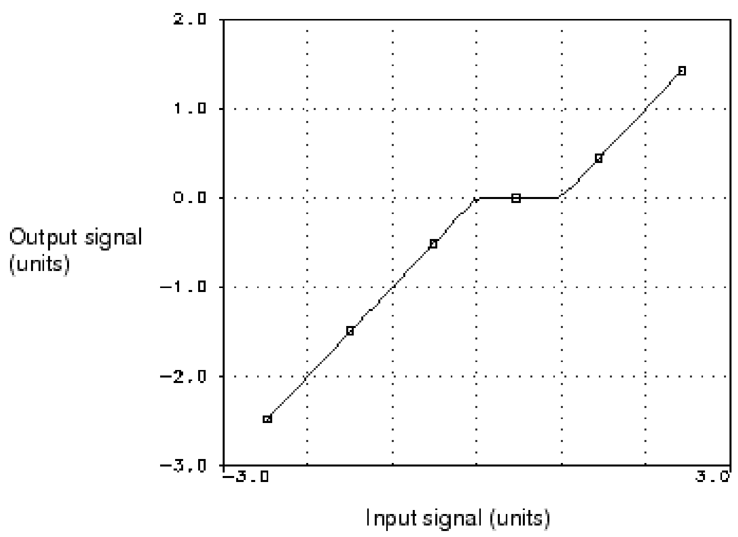
1. DeadZone models a dead zone nonlinearity. Its output is always a floating-point (real) signal.

$$y(n)= \begin{cases} K(x(n)-V_h) & \text{for} \quad x(n) > V_h \\ K(x(n)-V_l) & \text{for} \quad x(n) < V_l \\ 0 & \text{otherwise} \end{cases}$$

where:
y(n) is the output for sample *n*
x(n) is the input for sample *n*
K is the magnitude of the gain
V_h is the *High* dead zone value
V_l is the *Low* dead zone value

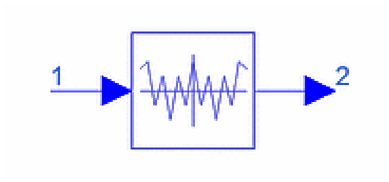
2. The output signal versus input signal plot of DeadZone, with *K* = 1, Low = 0 and High = 1, is shown below.

DeadZone Signal Plot



3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Dirichlet



Description: Dirichlet (aliased sinc) function
Library: Numeric, Special Functions
Class: SDFDirichlet
C++ Code: See *doc/sp_items/SDFDirichlet.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
N	length of Dirichlet kernel	10		int	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	The input x to the Dirichlet kernel.	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	The output of the Dirichlet kernel.	real

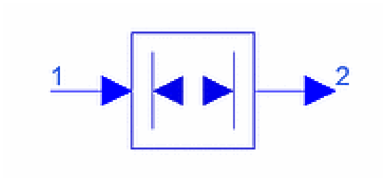
Notes/Equations

1. Dirichlet computes the normalized Dirichlet kernel (also called the aliased sinc function).
2. The value of the normalized Dirichlet kernel at x = 0 is always 1, and the normalized Dirichlet kernel oscillates between −1 and +1. The normalized Dirichlet kernel is periodic in x with a period of either 2 π when N is odd or 4 π when N is even.
3. The Dirichlet kernel is the discrete-time Fourier transform (DTFT) of a sampled pulse function. The parameter N is the length of the pulse [1].
   See also: *Sinc* (numeric) component.
4. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

Expand



Description: Expander part of a compander
Library: Numeric, Special Functions
Class: SDFExpand
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
Type	compression law: MU-law, A-law	MU-law		enum	
CompressionK	compression constant	1		real	
Max	maximum input value magnitude	1		real	(0.0, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. Expand can be used to obtain the A-law and MU-law expansion characteristics. The output of this component is always a baseband signal.
2. The following equations describe the characteristics of the component:
   Let
   $x'(n) = x(n)/V_M$

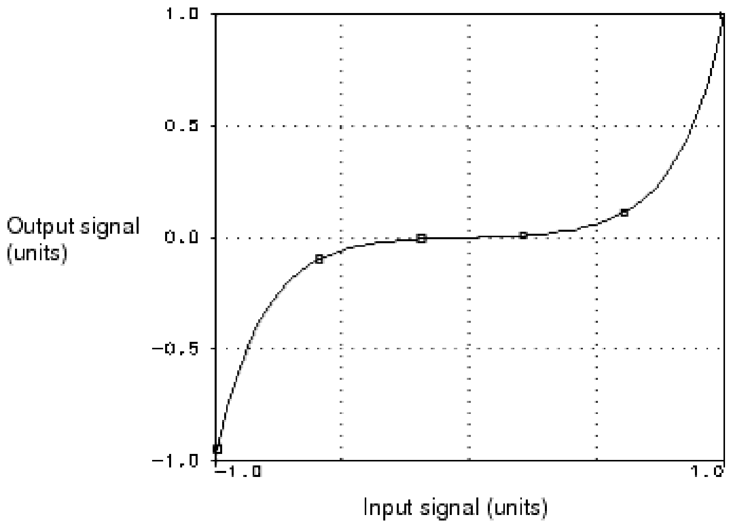
   Then
   MU-law:
   $$y(n) = \frac{V_M}{\mu}\operatorname{sgn}(x'(n))((1+\mu)^{|x(n)|}-1)$$
   A-law:
   $$y(n) = \begin{cases} \frac{V_M(1+ln(A))}{A}x'(n) & \text{for } x'(n) < 1/A \\ \frac{V_M}{A}\operatorname{sgn}(x'(n))e^{(|x'(n)|(1+ln(A))-1)} & \text{for } x'(n) > 1/A \end{cases}$$

where:
y(n) is the output for sample *n*
x(n) is the input for sample *n*
V_M is Max, the maximum input value magnitude

μ is the compression constant for MU-Law
A is the compression constant for A-Law

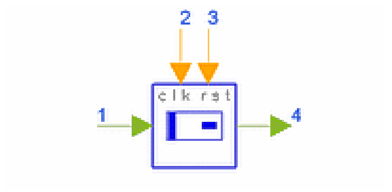
3. The output signal versus input signal plot of the Expand component, with Type = MU-law, CompressionK = 255, and Max = 1V, is shown below.

Expand Component Signal Plot



4. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

LatchClocked



Description: Data Latch with Clock Input
Library: Numeric, Special Functions
Class: SDFLatchClocked
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
ResetCx	complex output when reset pin is high	0.0		complex	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	complex
2	clock	clock signal	int
3	reset	reset signal	int

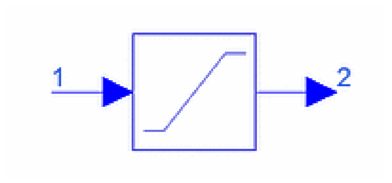
Pin Outputs

Pin	Name	Description	Signal Type
4	output	output signal	complex

Notes/Equations

1. LatchClocked can be used to latch complex numbers. The input is latched with the positive edge of the clock. The outputs can be reset asynchronously to the values specified by input2 and input3 by setting the signal at the reset pin to high.
   The component is positive edge sensitive to the clock input and level sensitive to the reset input. The reset signal is asynchronous.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Limit



Description: Limiter
Library: Numeric, Special Functions
Class: SDFLimit
C++ Code: See *doc/sp_items/SDFLimit.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
K	magnitude gain	1.0		real	(-∞ 0.0) or (0.0, ∞)
Bottom	lower output saturation value	0.0		real	(-∞, Top)
Top	higher output saturation value	1.0		real	(-∞, ∞)
Type	type of limiting curve: linear, atan	linear		enum	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Limit can be used to model two different types of limiting nonlinearities. The output is always a floating-point (real) signal.
2. If Type = linear

$$y(n)\begin{cases} V_l & if\, x(n) < \dfrac{V_l}{K} \\ Kx(n) & if\, \dfrac{(V_l)}{K} \le x(n) \le \dfrac{V_h}{K} \\ V_h & if\, x(n) > \dfrac{V_h}{K} \end{cases}$$

3. If Type = atan
   y(n) = offset + scale * atan((K*x(n) - offset) / scale), where:
   y(n) is the output for sample n
   x(n) is the input for sample n

K is the magnitude gain
scale = (V_h - V_l)/pi

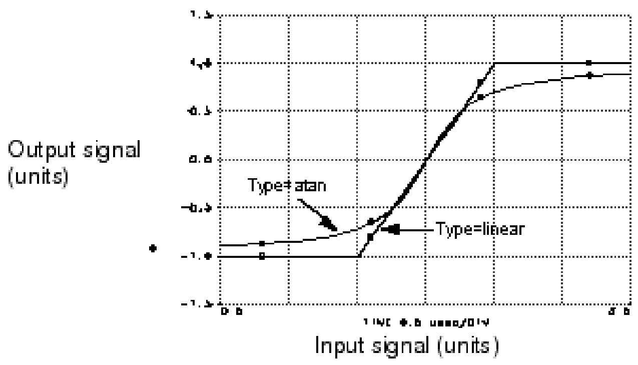
offset = (V_h + V_l)/2

V_l is the lower output saturation value (Bottom)

V_h is the higher output saturation value (Top)

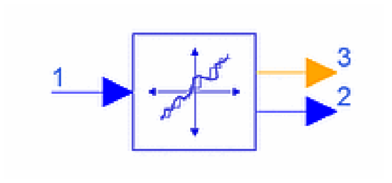
4. The output signal versus input signal plot of Limit (parameters K = 1, V_l = −1, and V_h = 1) is shown below for linear and atan types.

Limit Component Signal Plot



5. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

LinQuantIdx



Description: Uniform quantizer with step number output
Library: Numeric, Special Functions
Class: SDFLinQuantIdx
C++ Code: See *doc/sp_items/SDFLinQuantIdx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Levels	number of quantization levels	128		int	[1, ∞)
Low	lower limit of signal excursion	-3.0		real	(-∞, High)
High	upper limit of signal excursion	3.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

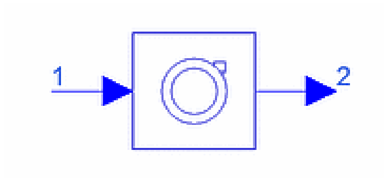
Pin Outputs

Pin	Name	Description	Signal Type
2	amplitude		real
3	stepNumber		int

Notes/Equations

1. LinQuantIdx quantizes the input value to the number of levels given by the *Levels* parameter plus 1. The quantization levels are uniformly spaced between Low and High inclusive. Rounding down is performed-the output level will equal High if the input level equals or exceeds High; if the input is below *Low*, the quantized output will equal Low. The quantized value is output to the SignalOut port, while the index of the quantization is output to the StepNumber port. This integer output is useful for components that need an integer input.
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

MuLaw



Description: Mu law compressor
Library: Numeric, Special Functions
Class: SDFMuLaw
C++ Code: See *doc/sp_items/SDFMuLaw.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Compress	enable compression	1		int	
Mu	mu parameter, a positive integer	255		int	[0, ∞)
Denom	denominator of mu-law definition	1.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

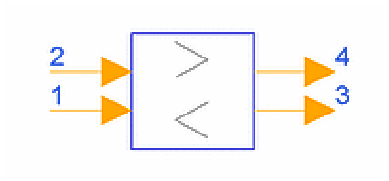
Notes/Equations

1. MuLaw transforms the input using a logarithmic mapping if the Compress parameter is true. In telephony, applying μ−law to 8-bit sampled data is called companding and is used to quantize the dynamic range of speech more accurately [1]. The transformation is defined in terms of the non-negative integer parameter Mu:
$$y(n) = \frac{\ln\{1.0 + \mu|x(n)|\}}{\ln(1.0 + \mu)} \text{ for } \mu \geq 0$$
where
y(n) is the output for sample *n*
x(n) is the input for sample *n*
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

References

1. S. Haykin, *Communication Systems* 3rd ed., John Wiley Sons, 1994, p. 380.

OrderTwoInt



Description: Ordered Two Integer Output
Library: Numeric, Special Functions
Class: SDFOrderTwoInt
C++ Code: See *doc/sp_items/SDFOrderTwoInt.html* under your installation directory.

Pin Inputs

Pin	Name	Description	Signal Type
1	upper		int
2	lower		int

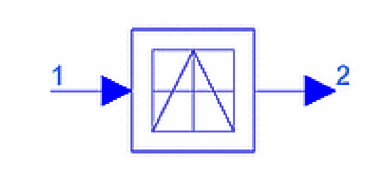
Pin Outputs

Pin	Name	Description	Signal Type
3	greater		int
4	lesser		int

Notes/Equations

1. OrderTwoInt takes two inputs and outputs the greater and lesser of the two integer inputs.
   $$y_1 = \max(x_1, x_2)$$
   $$y_2 = \min(x_1, x_2)$$
   where
   y_1 is the greater output
   y_2 is the lesser output
   x_1 is the upper input
   x_2 is the lower input
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

PcwzLinear



Description: Piecewise Linear Map Output
Library: Numeric, Special Functions
Class: SDFPcwzLinear
C++ Code: See *doc/sp_items/SDFPcwzLinear.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Breakpoints	endpoints and breakpoints in the mapping	(-1.0,-1.0) (0.0,1.0) (1.0,-1.0)		complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

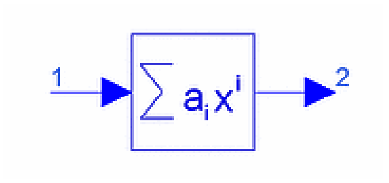
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. PcwzLinear implements a piecewise linear mapping from the input to the output. Mapping is given by a sequence of (x,y) pairs that specify breakpoints in the function; the sequence of x values must be increasing. The function implemented by this component can be represented by drawing straight lines between the (x,y) pairs, in sequence. (Each input value will be treated as a point on the x axis; the corresponding y value will be assigned to the output.)
   Default mapping is the *tent* map, in which inputs between -1.0 and 0.0 are linearly mapped into the range -1.0 to 1.0.
   Inputs between 0.0 and 1.0 are mapped into the same range, but with opposite slope, 1.0 to -1.0. If the input is outside the range specified in the x values of the breakpoints, then the appropriate extreme value will be used for the output. Therefore, for the default map: if the input is -2.0, the output will be -1.0; if the input is +2.0, the output will again be -1.0.
2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Polynomial



Description: Polynomial input-output relationship
Library: Numeric, Special Functions
Class: SDFPolynomial

Parameters

Name	Description	Default	Unit	Type	Range
Coefficients	Polynomial coefficients, 0-th order coefficient first	0 1		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

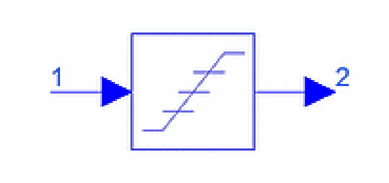
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

1. This component models a system with a polynomial input-output relationship. If the input is x, the output is $y = c_0 + c_1 \times x + c_2 \times x^2 + \ldots + c_N \times x^N$, where N is the order of the polynomial and $c_0, \ldots, c_N$ are the elements of the Coefficients parameter.
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Quant



Description: Quantizer
Library: Numeric, Special Functions
Class: SDFQuant
C++ Code: See *doc/sp_items/SDFQuant.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Thresholds	quantization thresholds, in increasing order	0.0		real array	
Levels	output levels. If empty use 0, 1, 2, ...			real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

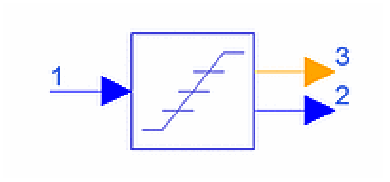
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Quant quantizes the input value to one of N+1 possible output levels using N thresholds.
   - For input ≤ n th threshold, but > all previous thresholds, the output will be the n th level.
   - For input > all thresholds, the output is N+1 th level.
   - For input < all thresholds, the output is 0 th level.
2. If the level is specified, there must be one more level than thresholds. The default value for level is 0, 1, 2, ... N.
   This component takes on the order of log N steps to find the right level, whereas the linear quantizer component LinQuantIdx takes a constant amount of time. Therefore, for linear quantization, use the LinQuantIdx component.
3. Assume that the Thresholds parameter is set to (8.1, 9.2, 10.3) and that the Levels parameter is not set so that the default values of (0.0, 1.0, 2.0, 3.0) are used. An input of -1.5 would give an output of 0.0; an input of 8.2 would give an output of 1.0; and, an input of 15.5 would give an output of 3.0.
4. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

QuantIdx



Description: Quantizer with Step Number Output
Library: Numeric, Special Functions
Class: SDFQuantIdx
C++ Code: See *doc/sp_items/SDFQuantIdx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Thresholds	quantization thresholds, in increasing order	0.0		real array	
Levels	output levels. If empty use 0, 1, 2, ...			real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

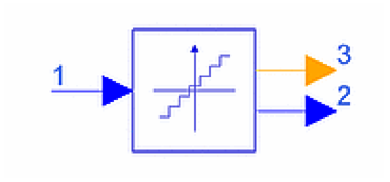
Pin	Name	Description	Signal Type
2	output		real
3	stepNumber	Level of the quantization from 0 to N-1	int

Notes/Equations

1. QuantIdx quantizes the input value to one of N+1 possible output levels using N thresholds. This component also outputs the quantization level (stepNumber). For an input less than or equal to the n th threshold, but larger than all previous thresholds, the output will be the n th level. If the input is greater than all thresholds, the output is the N+1 th level. If the input is less than all thresholds, the output is the 0 th level.
2. If the level is specified, there must be one more level than thresholds. The default value for level is 0, 1, 2, ... N. This component takes on the order of log N steps to find the right level, whereas the linear quantizer component LinQuantIdx takes a constant amount of time. Therefore, for linear quantization, use the LinQuantIdx component.
3. Assume that the Thresholds parameter is set to (8.1, 9.2, 10.3) and that the Levels parameter is not set so that the default values of (0.0, 1.0, 2.0, 3.0) are used. An input of -1.5 would give an output of 0.0; an input of 8.2 would give an output of 1.0; and, an input of 15.5 would give an output of 3.0.
4. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
5. For information regarding numeric special function component signals, refer to

Numeric Special Functions (numeric).

Quantizer



Description: Quantizer Using CodeBook
Library: Numeric, Special Functions
Class: SDFQuantizer
C++ Code: See *doc/sp_items/SDFQuantizer.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
FloatCodebook	possible output values	0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

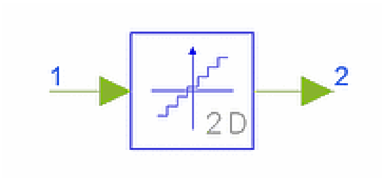
Pin Outputs

Pin	Name	Description	Signal Type
2	output	Closest value in the codebook	real
3	outIndex	Index of the closest value in the codebook	int

Notes/Equations

1. Quantizer quantizes the input value to the nearest output value in the given codebook. The nearest value is found by a full search of the codebook, so this component will be significantly slower than either the Quant or LinQuantIdx components. The absolute value of the difference is used as a distance measure. The index of the closest value in the codebook is also output.
2. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Quantizer2D



Description: 2-dimensional quantizer
Library: Numeric, Special Functions
Class: SDFQuantizer2D
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
VxMax	maximum real output level	1.0		real	(-∞, ∞)
VxMin	minimum real output level	-1.0		real	(-∞, VxMax)
Nx	number of real output levels	16		int	[1, ∞)
VyMax	maximum imaginary output level	1.0		real	(-∞, ∞)
VyMin	minimum imaginary output level	-1.0		real	(-∞, VyMax)
Ny	number of imaginary output levels	16		int	[1, ∞)
QuantList	user-defined quantization points			complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	complex

Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	complex

Notes/Equations

1. The complex number input is mapped to one of a finite set of complex numbers. Any arbitrary set of points can be specified as the set of output points by using a file or a list, or else the parameters VxMax, VxMin, Nx, VyMax, VyMin and Ny can be used to set up a rectangular grid of output points.
   The ability to specify output points by a file or a list can be used to define arbitrary 2D quantizers. Each input is mapped to the nearest output point, where the metric used to determine the nearest output point is the Euclidean distance. This type of a quantizer is also referred to as a Voronoi or a nearest neighbor vector quantizer [1]. 2D Quantizer with Three Output Points shows an example where three output points P1, P2, and P3 have been specified. The entire 2D plane is then divided into 3 regions, R1, R2, and R3, which are shown by the dotted lines. Any input point in region R1 is mapped to the output point P1 (and similarly for the other regions). 2D Quantizer with Output Points On a Grid illustrates how a rectangular grid of output points can be set up by using the parameters VxMax, VxMin, Nx, VyMax,

VyMin and Ny.
Due to the regular lattice structure of this quantizer, it can be implemented efficiently in terms of speed. Therefore, it is more efficient to use this second method of specifying a quantizer than using a file or a list of output points.
When a file or list is used to specify the list of output points, data is entered for the QuantList parameter as an ordered list of complex values.
Data entered as an explicit array has the form:

```
QuantList = "(1, 0) (0.707, 0.707) (0, 1) ( − 0.707, 0.707) ( − 1, 0) ( −
0.707, − 0.707) (0, − 1) (0.707, − 0.707)"
```

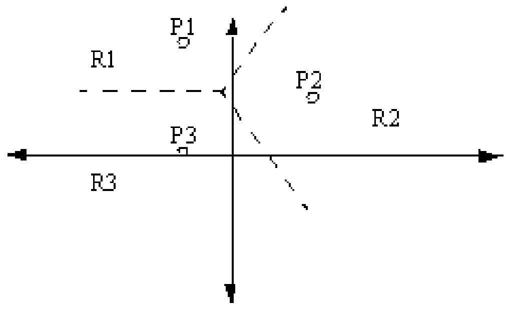
As an alternative from an explicit list, this dataset can be contained in a text file and referenced by name as follows:

```
QuantList = "<myquantlist.cx"
```

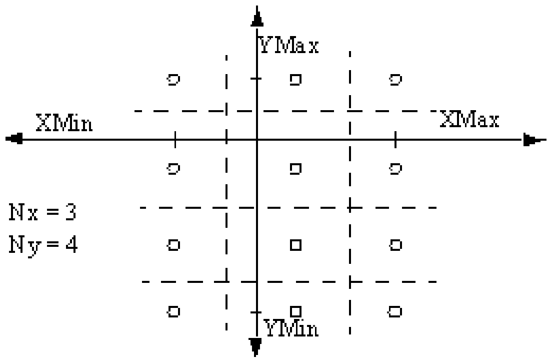
where the file named *myquantlist.cx* must be located in the current workspace data subdirectory. If not in the data subdirectory, then the file name must include the full directory path as the prefix to the file name. The contents of this file is simply the complex values where the separator can be a comma, space, tab, or new line, with one or more complex pairs per line:
(1, 0) (0.707, 0.707)
(0, 1) (−0.707, 0.707)
(−1, 0) (−0.707, −0.707)
(0, −1) (0.707, −0.707)
This above dataset can be used to create a quantizer for an 8PSK receiver whose signal set consists of 8 points equally spaced on a unit circle. Quantizer2D shows the points and the decision regions (in dotted lines) for this quantizer.

2. For details on complex parameter values, refer to *Complex-Valued Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
   For details on using complex arrays of data, refer to *Value Types* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

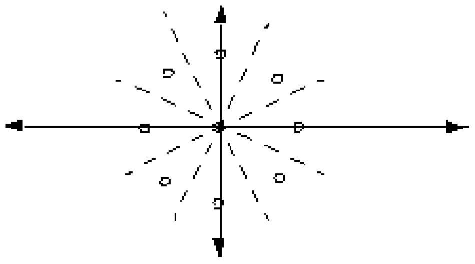
2D Quantizer with Three Output Points



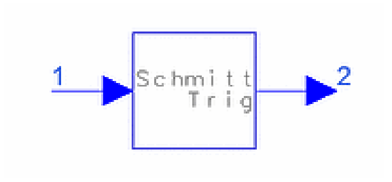
2D Quantizer with Output Points On a Grid



Quantizer2D



SchmittTrig



Description: Schmitt Trigger
Library: Numeric, Special Functions
Class: SDFSchmittTrig
Derived From: baseOmniSysNumericStar

Parameters

Name	Description	Default	Unit	Type	Range
ILow	lower input trigger value	-1		real	(-∞, IHigh)
IHigh	higher input trigger value	1		real	(-∞, ∞)
OLow	lower output trigger value	-1		real	(-∞, OHigh)
OHigh	higher output trigger value	1		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input	input signal	real

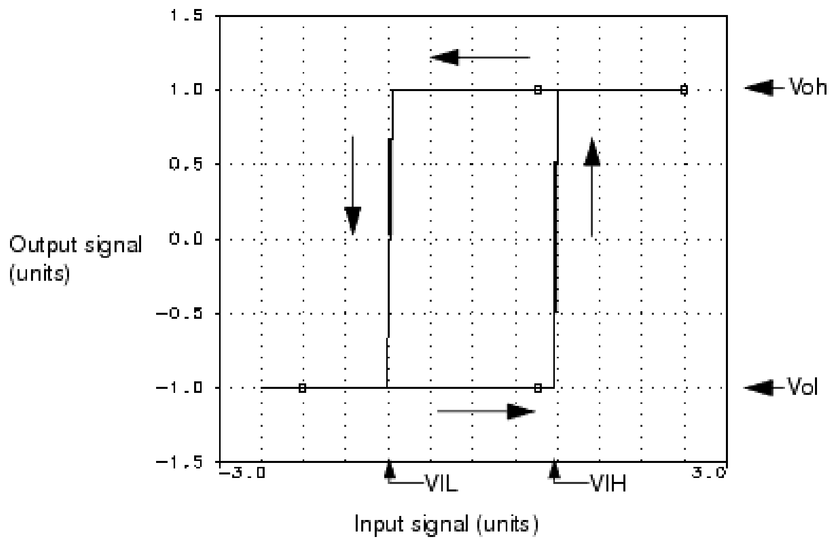
Pin Outputs

Pin	Name	Description	Signal Type
2	output	output signal	real

Notes/Equations

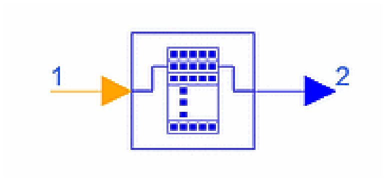
1. SchmittTrig is a Schmitt trigger with programmable levels. The output is always a floating-point (real) signal.
2. The output signal versus input signal plot, with parameters ILow = −1, IHigh = 1, OLow = −1, and OHigh = 1, is shown below.

SchmittTrig Signal Plot



3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Table



Description: Indexed Lookup Table Output
Library: Numeric, Special Functions
Class: SDFTable
C++ Code: See *doc/sp_items/SDFTable.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Values	table of values to output	{-1, 1}		real array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

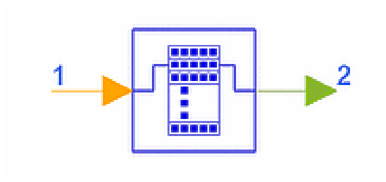
Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Table implements a real-valued lookup table indexed by an integer-valued input. The input values must be between 0 and $N - 1$, inclusive, where N is the size (number of elements) of the *Values* parameter. N must be less than 20,000. The first element of the *Values* parameter is indexed by an input with value 0. The last element of the *Values* parameter is indexed by an input with value $N - 1$. An error occurs if the input value is outside the interval $[0, N - 1]$.
2. Example. Let's assume the *Values* parameter is set to {-1.0, -0.333, 0.333, 1.0} (the 4 signal levels of a PAM-4 system). If the input signal values are 0, 0, 3, 1, 0, 1, 3, 2, 3, 1, 0, 2, then the output signal values will be -1.0, -1.0, 1.0, -0.333, -1.0, -0.333, 1.0, 0.333, 1.0, -0.333, -1.0, 0.333.
3. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
4. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

TableCx



Description: Indexed Complex Lookup Table Output
Library: Numeric, Special Functions
Class: SDFTableCx
C++ Code: See *doc/sp_items/SDFTableCx.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Values	table of values to output	{(1), (j), (-1), (-j), (0), (1), (j), (1)}		complex array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

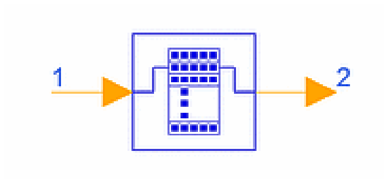
Pin Outputs

Pin	Name	Description	Signal Type
2	output		complex

Notes/Equations

1. TableCx implements a complex-valued lookup table indexed by an integer-valued input. The input must lie between 0 and $N - 1$, inclusive, where N is the size of the table. The table of values listed for the Values parameter must be less than 20,000 values long. Its first component is indexed by a zero-valued input. An error occurs if the input value is out of the array bounds.
   The input must be in the range: $0 \le$ input < size of Values.
2. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

TableInt



Description: Indexed Integer Lookup Table Output
Library: Numeric, Special Functions
Class: SDFTableInt
C++ Code: See *doc/sp_items/SDFTableInt.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
Values	table of values to output	{-1, 1}		int array	

Pin Inputs

Pin	Name	Description	Signal Type
1	input		int

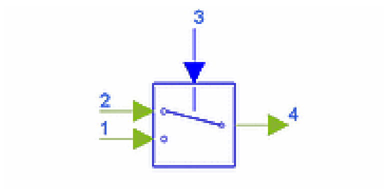
Pin Outputs

Pin	Name	Description	Signal Type
2	output		int

Notes/Equations

1. TableInt implements an integer-valued lookup table indexed by an integer-valued input. The input must lie between 0 and $N - 1$, inclusive, where N is the size of the table. The table of values listed for the Values parameter must be less than 20,000 values long. Its first component is indexed by a zero-valued input. An error occurs if the input value is out of the array bounds.
   The input must be in the range: $0 \le$ input < size of Values.
2. For details on using arrays of data for parameter values, refer to *Understanding Parameters* (ptolemy) in the *ADS Ptolemy Simulation* (ptolemy) documentation.
3. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Toggle



Description: Data Toggle with Clock Input
Library: Numeric, Special Functions
Class: SDFToggle
Derived From: baseOmniSysNumericStar

Pin Inputs

Pin	Name	Description	Signal Type
1	input1	input signal 1	complex
2	input2	input signal 2	complex
3	control	control signal	real

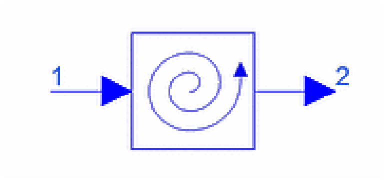
Pin Outputs

Pin	Name	Description	Signal Type
4	output	output signal	complex

Notes/Equations

1. Let
   $v_1(t)$ = input1
   $v_2(t)$ = input2
   $v_3(t)$ = control
   $v_4(t)$ = output,
   then
   $$v_4(t) = \begin{cases} v_2(t) & \text{when } v_3(t) \geq 0.5 \\ v_1(t) & \text{when } v_3(t) < 0.5 \end{cases}$$
   Here, $v_1(t)$, $v_2(t)$ and $v_4(t)$ are complex valued signals with real and imaginary parts. If $v_3(t)$ is complex valued, its imaginary part is ignored and only the real part is considered.
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).

Unwrap



Description: Unwrap phase
Library: Numeric, Special Functions
Class: SDFUnwrap
C++ Code: See *doc/sp_items/SDFUnwrap.html* under your installation directory.

Parameters

Name	Description	Default	Unit	Type	Range
OutPhase	initial output phase	0.0		real	(-∞, ∞)
PrevPhase	initial wrapped phase of input signal for computing the first phase difference (phase change)	0.0		real	(-∞, ∞)

Pin Inputs

Pin	Name	Description	Signal Type
1	input		real

Pin Outputs

Pin	Name	Description	Signal Type
2	output		real

Notes/Equations

1. Unwrap unwraps a phase plot, removing discontinuities of magnitude 2 π. Unwrap assumes that the phase never changes by more than π in one sample period; it also assumes that the input is in the range $[-\pi, \pi]$.
2. For information regarding numeric special function component signals, refer to *Numeric Special Functions* (numeric).