SerDes Toolbox™

Reference

MATLAB® & SIMULINK®

R2019a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

SerDes Toolbox™ Reference

Revision History

March 2019	Online only	New for Version 1.0 (Release 2019a)

Contents

1

SerDes System Objects — Alphabetical List

serdes.AGC

Automatically adjusts gain to maintain output waveform amplitude

Description

`serdes.AGC` System object™ applies an adaptive variable gain to the input waveform to achieve a desired RMS output voltage. Averaging the RMS voltage over a specified number of symbols, `serdes.AGC` performs automatic gain control (AGC) by increasing or decreasing the gain, or keeping the gain constant.

To adjust the gain of the input signal:

1. Create the `serdes.AGC` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
agc = serdes.AGC
agc = serdes.AGC (Name,Value)
```

Description

`agc = serdes.AGC` returns an AGC object that modifies an input waveform according to the root-mean-squared property of the AGC block.

`agc = serdes.AGC (Name,Value)` returns an AGC object with each specified property set to specific value. Unspecified properties have default values.

Example: `agc = serdes.AGC('Mode',1)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

Main

`Mode` — AGC operating mode

`1` (default) | `0`

AGC operating mode, specified as `0` or `1`. `Mode` determines if the AGC adjusts the gain of input baseband signal or acts as a pass-through.

Mode Value	AGC Mode	AGC Operation
`0`	Off	`serdes.AGC` is bypassed, the input waveform remains unchanged.
`1`	On	`serdes.AGC` adjusts gain of input waveform to maintain `TargetRMSVoltage` in the output waveform.

Data Types: `double`

`TargetRMSVoltage` — Desired RMS voltage of output waveform

`0.3` (default) | real scalar in the range [1e-3, 10]

Desired RMS voltage of the output waveform, specified as a real scalar in the range [1e-3, 10] in V.

Data Types: `double`

Advanced

`SymbolTime` — Time of single symbol duration

`1e-10` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: `double`

`SampleInterval` — Uniform time step of waveform
`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

`Modulation` — Modulation scheme
`2` (default) | `4`

Modulation scheme, specified as `2` or `4`.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
4	Four-level pulse amplitude modulation (PAM4)

Data Types: `double`

`GainLimit` — Maximum allowed AGC gain
`10` (default) | real positive scalar

Maximum allowed AGC gain, specified as a real positive scalar. `GainLimit` provides a stable startup of the adaptive algorithm.

Data Types: `double`

`AveragingLength` — Averaging length
`100` (default) | real positive integer scalar

Averaging length, specified as a real positive integer. `AveragingLength` defines the number of symbol over which the RMS calculation of the input signal is made.

Data Types: `double`

`WaveType` — Input wave type form
`'Sample'` (default) | `'Impulse'` | `'Waveform'`

Input wave type form, specified as:

- `'Sample'` - A sample-by-sample input signal.

- `'Impulse'` - An impulse response input signal.
- `'Waveform'` - A bit-pattern waveform type of input signal, such as pseudorandom binary sequence (PRBS).

Data Types: `char`

Usage

Syntax

```
y = agc(x)
```

Description

```
y = agc(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal. If the `WaveType` is set to `'Sample'`, the input signal is a sample-by-sample signal specified as a scalar. If the `WaveType` is set to `'Impulse'`, the input signal is an impulse response vector signal.

Output Arguments

y — Gain adjusted output signal
scalar | vector

Gain adjusted output signal. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Generating Constant Level Output Signal

Use a `serdes.AGC` system object™ to reduce the amplitude of a waveform signal to maintain an rms voltage of `0.25` V.

Create a signal with two sinusoids, one at `250` Hz, and the other at `340` Hz. The sampling frequency is 800 Hz. The signal is corrupted with additive zero-mean random noise.

```
Fs = 10000;
L = 1000;
t = (0:L-1)'/Fs;
x = sin(2*pi*250*t) + 0.75*cos(2*pi*340*t);         % Original signal
y = x + .5*randn(size(x));                          % Noisy signal
```

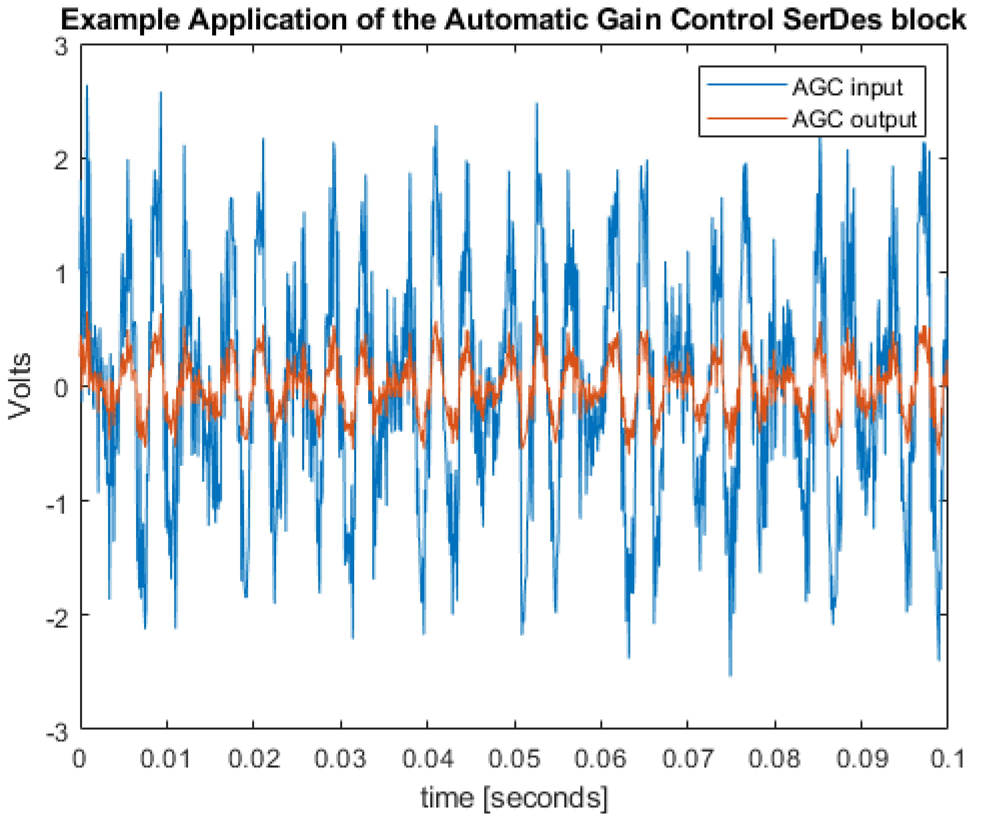
Find the frequency components of the signal using serdes.AGC.

```
agcblock = serdes.AGC('TargetRMSVoltage',0.25);
z = agcblock(y);
```

Plot the input and modified waveforms.

```
figure, plot(t,y,t,z)
legend('AGC input','AGC output')
title('Example Application of the Automatic Gain Control SerDes block');
```

```
xlabel('time [seconds]');
ylabel('Volts');
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

AGC | VGA | `serdes.VGA`

Introduced in R2019a

serdes.CDR

Performs clock data recovery function

Description

The serdes.CDR System object provides clock sampling times and estimates data symbols at the receiver using a Bang-Bang clock and data recovery (CDR) model.

To provide clock data locations:

1. Create the serdes.CDR object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
cdr = serdes.CDR
cdr = serdes.CDR (Name,Value)
```

Description

cdr = serdes.CDR returns a CDR object that determines the clock sampling times and estimates the data symbol according to the Bang-Bang CDR algorithm. It does not return or modify the incoming waveform.

cdr = serdes.CDR (Name,Value) returns a CDR object with each specified property set to specific value. Unspecified properties have default values.

Example: cdr = serdes.CDR('Count',8)

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

Main

`Count` — Early or late CDR count threshold to trigger phase update
`16` (default) | real positive integer >4

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer >4. Increasing the value of `Count` provides a more stable output clock phase at the expense of convergence speed. Because the bit decisions are made at the clock phase output, a more stable clock phase has a better bit error rate (BER).

Data Types: `double`

`Step` — Clock phase resolution
`0.0078` (default) | real scalar

Clock phase resolution, specified as a real scalar in fraction of unit interval (UI). `Step` is the inverse of the number of phase adjustments in CDR.

Data Types: `double`

`PhaseOffset` — Clock phase offset
`0` (default) | real scalar in the range [-0.5,0.5]

Clock phase offset, specified as a real scalar in the range [-0.5,0.5] in fraction of symbol time. `PhaseOffset` is used to manually shift clock probability distribution function (PDF) for better bit error rate (BER).

Data Types: `double`

`ReferenceOffset` — Reference clock offset impairment
`0` (default) | real scalar in the range [-3e-4,3e-4]

Reference clock offset impairment, specified as a real scalar in the range [-3e-4,3e-4] in parts per million (ppm). `ReferenceOffset` is the deviation between transmitter oscillator frequency and receiver oscillator frequency.

Data Types: `double`

`Sensitivity` — Sampling latch meta-stability voltage
`0` (default) | real scalar

Sampling latch meta-stability voltage, specified as a real scalar in V. If the data sample voltage lies within the region (+/-`Sensitivity`), there is a 50% probability of bit error..

Data Types: `double`

Advanced

`SymbolTime` — Time of single symbol duration
`1e-10` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: `double`

`SampleInterval` — Uniform time step of waveform
`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

`Modulation` — Modulation scheme
`2` (default) | `4`

Modulation scheme, specified as `2` or `4`.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
4	Four-level pulse amplitude modulation (PAM4)

Data Types: `double`

`WaveType` — Input wave type form
`'Sample'` (default) | `'Impulse'`

Input wave type form, specified as:

- `'Sample'` - A sample-by-sample input signal.
- `'Impulse'` - An impulse response input signal.

Data Types: `char`

Usage

Syntax

```
y = cdr(x)
```

Description

```
y = cdr(x)
```

Input Arguments

x — Input baseband signal
scalar

Input baseband signal. The input to the CDR must be applied as one sample at a time and not as a vector.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Clock Distribution Recovery with CDR

This example shows how to recover clock distribution using `serdes.CDR` system object™.

Use a symbol time of `100` ps and `16` samples per symbol. The channel has `5` dB loss.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
dt = SymbolTime/SamplesPerSymbol;
loss = 5;
chan = serdes.ChannelLoss('Loss',loss,'dt',dt,...
       'TargetFrequency',1/SymbolTime/2,'RiseTime',SamplesPerSymbol/4*dt);
```

Create a random data pattern using a pseudorandom binary sequence of order 10.

```
ord = 10;                       %PRBS order
nrz=prbs(ord,2^ord-1);
nrzPattern = nrz(:)' - 0.5;     %[0,1] --> [-0.5,0.5];
ChannelPulseResponse = impulse2pulse(chan.impulse, SamplesPerSymbol, dt);
waveprbs = pulse2wave(ChannelPulseResponse(:,1),nrzPattern,SamplesPerSymbol);
wave2 = [waveprbs; waveprbs];
```

Create the CDR object that utilizes NRZ modulation scheme.

```
CDR1 = serdes.CDR('Modulation',2,'Count',8,'Step',1/64,...
       'SymbolTime',SymbolTime,'SampleInterval',dt);
```

Initialize the outputs.

```
phase = zeros(1,length(wave2));
CDRearlyLateCount = zeros(1,length(wave2));
```

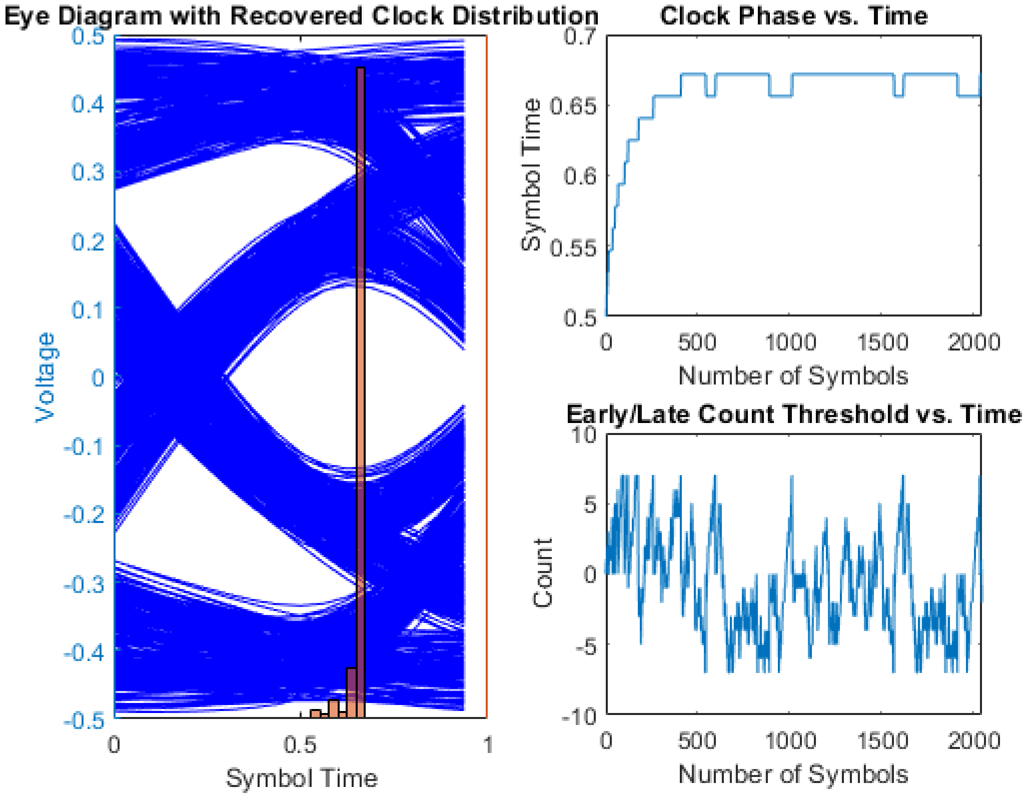
Feed the waveform one sample at a time through the CDR object.

```
for ii = 1:length(wave2)
      [phase(ii), ~, optional] = CDR1(wave2(ii));
```

```
        CDRearlyLateCount(ii) = optional.CDRearlyLateCount;
end
```

Plot the eye diagram with recovered clock distribution, clock phase vs. time, and early/late count threshold vs. time.

```
t = (0:length(wave2)-1)/SamplesPerSymbol;
teye = (0:SamplesPerSymbol-1)/SamplesPerSymbol;
eyed = reshape(wave2,SamplesPerSymbol,[]);
 figure,
subplot(2,2,[1,3]), yyaxis left, plot(teye,eyed, '-b'),
title('Eye Diagram with Recovered Clock Distribution')
xlabel('Symbol Time'), ylabel('Voltage')
yyaxis right,
histogram(phase,SamplesPerSymbol/2)
set(gca,'YTick',[])
subplot(2,2,2), plot(t,phase)
xlabel('Number of Symbols'), ylabel('Symbol Time');
title('Clock Phase vs. Time')
subplot(224), plot(t,CDRearlyLateCount)
xlabel('Number of Symbols'), ylabel('Count')
title('Early/Late Count Threshold vs. Time')
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

CDR

Introduced in R2019a

serdes.ChannelLoss

Create simple lossy transmission line model

Description

The `serdes.ChannelLoss` block constructs a lossy transmission line model for use in the **SerDes Designer** app and other exported Simulink® models in the SerDes Toolbox.

To construct the loss model from channel loss metric:

1. Create the `serdes.ChannelLoss` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
ChannelLoss = serdes.ChannelLoss
ChannelLoss = serdes.ChannelLoss(Name,Value)
```

Description

`ChannelLoss = serdes.ChannelLoss` returns a ChannelLoss object that modifies an input waveform with a lossy printed circuit board transmission line model according to the method outlined in the IEEE Standard 802.3bj-2014, normative section 39A.

`ChannelLoss = serdes.ChannelLoss(Name,Value)` returns a ChannelLoss object with each specified property set to specified value. Unspecified properties have default values.

Example: `ChannelLoss = serdes.ChannelLoss('Loss',5, 'TargetFrequency',14e9)` returns a ChannelLoss object that has a channel loss of 5 dB at 14 GHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

`Loss` — Channel power loss at target frequency
`8` (default) | real scalar

Channel loss at the target frequency, specified as a scalar in dB.

Data Types: `double`

`Target Frequency` — Frequency of the desired channel loss
`10e9` (default) | real positive scalar

Frequency for the desired channel loss, specified as a real positive scalar in Hz.

Data Types: `double`

`dt` — Sample interval
`1e-12` (default) | real positive scalar

Sample interval in s, specified as a real positive scalar.

Data Types: `double`

`Zc` — Differential channel characteristic impedance
`100` (default) | real positive scalar

Differential characteristic impedance of the channel, specified as a real positive scalar in Ohms.

Data Types: `double`

`TxR` — Single-ended impedance of transmitter analog model
`50` (default) | real nonnegative scalar

Single-ended impedance of the transmitter analog model, specified as a real nonnegative scalar in Ohms.

Data Types: `double`

`TxC` — Capacitance of transmitter analog model
`1e-12` (default) | real nonnegative scalar

Capacitance of the transmitter analog model, specified as a real nonnegative scalar in F.

Data Types: `double`

`RxR` — Single-ended impedance of receiver analog model
`50` (default) | real nonnegative scalar

Single-ended impedance of the receiver analog model, specified as a real nonnegative scalar in Ohms.

Data Types: `double`

`RxC` — Capacitance of receiver analog model
`1e-12` (default) | real nonnegative scalar

Capacitance of the receiver analog model, specified as a real nonnegative scalar in F.

Data Types: `double`

`RiseTime` — Rise time of stimulus input
`1e-11` (default) | real positive scalar

20%-80% rise time of the stimulus input to transmitter analog model, specified as a real positive scalar in s.

Data Types: `double`

`VoltageSwingIdeal` — Peak-to-peak voltage at the input of transmitter analog model
`1` (default) | real positive scalar

Peak-to-peak voltage at the input of transmitter analog model, specified as a real positive scalar in V.

Data Types: `double`

Usage

Syntax

```
y = ChannelLoss(x)
```

Description

```
y = ChannelLoss(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal.

Output Arguments

y — Estimated channel output
scalar | vector

Estimated channel output that includes the effect of a lossy printed circuit board transmission line model according to the method outlined in the IEEE Standard 802.3bj-2014, Normative section 39A.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

???

See Also

Introduced in R2019a

serdes.CTLE

Continuous time linear equalizer (CTLE) or peaking filter

Description

The `serdes.CTLE` System object processes a sample-by-sample input signal or analytically processes an impulse response vector input signal to remove distortions resulting from lossy channels.

To equalize the baseband signal using `serdes.CTLE`:

1. Create the `serdes.CTLE` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
ctle = serdes.CTLE
ctle = serdes.CTLE(Name,Value)
```

Description

`ctle = serdes.CTLE` returns a CTLE object that modifies an input waveform according to the pole zero transfer function defined in the object.

`ctle = serdes.CTLE(Name,Value)` returns a CTLE object with each specified property set to a specific value. Unspecified properties have default values.

Example: `ctle = serdes.CTLE('ACGain',5)` returns a CTLE object with gain at the peaking frequency set to 5 dB.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

Main

`Mode` — CTLE operating mode

`2` (default) | `0` | `1`

CTLE operating mode, specified as `0`, `1`, or `2`. `Mode` determines whether the CTLE is bypassed or not. If CTLE is not bypassed, then `Mode` also determines what transfer function is applied to the input waveform.

Mode Value	CTLE Mode	CTLE Operation
0	off	`serdes.CTLE` is bypassed and the input waveform remains unchanged.
1	fixed	`serdes.CTLE` applies the CTLE transfer function as specified by `ConfigSelect` to the input waveform.
2	adapt	If `WaveType` is set to `'Impulse'` or `'Waveform'`, then `serdes.CTLE` determines the CTLE transfer function for the best eye height opening and applies the transfer function to the input waveform.
		If `WaveType` is selected as `'Sample'`, then `serdes.CTLE` operates in the fixed mode.

Data Types: `double`

`ConfigSelect` — Select which member of transfer function family to apply in fixed mode

`0` (default) | real integer scalar

Select which member of the transfer function family to apply in fixed mode, specified as a real integer scalar.

Example: `ctle = serdes.CTLE('ConfigSelect',5,'Specification','DC Gain and Peaking Gain')` returns a CTLE object that selects the 6-th element of the `DCGain` and `PeakingGain` vector to apply to the filter transfer function.

Data Types: `double`

Specification — Input specification for CTLE response
`'DC Gain and Peaking Gain'` (default) | `'DC Gain and AC Gain'` | `'AC Gain and Peaking Gain'` | `'GPZ Matrix'`

Defines which inputs will be used for the CTLE transfer function family:

- `'DC Gain and Peaking Gain'` — CTLE response is specified from `DCGain`, `PeakingGain`, and `PeakingFrequency`.
- `'DC Gain and AC Gain'` — CTLE response is specified from `DCGain`, `ACGain`, and `PeakingFrequency`.
- `'AC Gain and Peaking Gain'` — CTLE response is specified from `ACGain`, `PeakingGain`, and `PeakingFrequency`.
- `'GPZ Matrix'` — CTLE response is specified from `GPZ`.

Data Types: `char`

PeakingFrequency — Approximate frequency at which CTLE transfer function peaks
`5e9` (default) | scalar | vector

Approximate frequency at which CTLE transfer function peaks in magnitude, specified as a scalar or a vector in Hz. If specified as a vector, then the vector length must be the same as the vectors in `ACGain`, `DCGain`, and `PeakingGain`.

Data Types: `double`

DCGain — Gain at zero frequency
`[0 -1 -2 -3 -4 -5 -6- -7 -8]` (default) | scalar | vector

Gain at zero frequency for the CTLE transfer function, specified as a scalar or a vector in dB. If specified as a vector, then the vector length must be the same as the vectors in `PeakingFrequency`, `ACGain`, and `PeakingGain`.

Data Types: `double`

PeakingGain — Difference between AC and DC gain

[0 1 2 3 4 5 6 7 8] (default) | scalar | vector

Peaking gain, specified as a vector in dB. It is the difference between ACGain and DCGain for the CTLE transfer function. If specified as a vector, then the vector length must be the same as the vectors in PeakingFrequency, ACGain, and DCGain.

Data Types: double

ACGain — Gain at the peaking frequency

0 | scalar | vector

Gain at the peaking frequency for the CTLE transfer function, specified as a scalar or vector in dB. If specified as a vector, then the vector length must be the same as the vectors in PeakingFrequency, DCGain, and PeakingGain.

Data Types: double

GPZ — Gain pole zero

matrix

Gain pole zero, specified as a matrix. GPZ explicitly defines the family of CTLE transfer functions by specifying the DCGain (dB) in column 1 and then poles and zeros in alternating columns. The poles and zeros are specified in Hz.

No repeated poles or zeros are allowed. Complex poles or zeros must have conjugates. The number of poles must be greater than number of zeros for system stability.

Data Types: double

Advanced

SymbolTime — Time of single symbol duration

100e-12 (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: double

SampleInterval — Uniform time step of waveform

6.25e-12 (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

WaveType — Input wave type form
`'Sample'` (default) | `'Impulse'` | `'Waveform'`

Input wave type form:

- `'Sample'` — A sample-by-sample input signal.
- `'Impulse'` — An impulse response input signal.
- `'Waveform'` — A bit-pattern waveform type of input signal, such as pseudorandom binary sequence (PRBS).

Data Types: `char`

Usage

Syntax

```
y = ctle(x)
```

Description

```
y = ctle(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal. If the `WaveType` is set to `'Sample'`, then the input signal is a sample-by-sample signal specified as scalars. If the `WaveType` is set to `'Impulse'`, then the input signal is an impulse response vector signal.

Output Arguments

y — Equalized CTLE output
scalar | vector

Equalized CTLE output waveform. If the input signal is a sample-by-sample signal specified as scalars, then the output is also scalar. If the input signal is an impulse response vector signal, then the output is also a vector.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Impulse Response Processing Using CTLE

This example shows how to process the impulse response of a channel using `serdes.CTLE` System object™.

Use a symbol time of `100` ps and `16` samples per symbol. The channel has `16` dB loss. The peaking frequency is `11` GHz.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
dbloss = 16;
DCGain = 0:-1:-26;
PeakingGain = 0:26;
PeakingFrequency = 11e9;
```

Calculate the sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the CTLE object. The object adaptively applies the optimum transfer function for the best eye height opening to the input impulse response.

```
CTLE1 = serdes.CTLE('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',2,'WaveType','Impulse',...
      'DCGain',DCGain,'PeakingGain',PeakingGain,...
      'PeakingFrequency',PeakingFrequency);
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
impulseIn = channel.impulse;
```

Process the impulse response with CTLE.

```
[impulseOut, Config] = CTLE1(impulseIn);
```

Display the adapted configuration.

```
fprintf('Adapted CTLE Configuration Selection is %g \n',Config)
```

```
Adapted CTLE Configuration Selection is 17
```

Convert the impulse responses to pulse, waveform, and eye diagram.

```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

pulseIn = impulse2pulse(impulseIn,SamplesPerSymbol,dt);
waveIn = pulse2wave(pulseIn,dataPattern,SamplesPerSymbol);
eyeIn = reshape(waveIn,SamplesPerSymbol,[]);

pulseOut = impulse2pulse(impulseOut,SamplesPerSymbol,dt);
waveOut = pulse2wave(pulseOut,dataPattern,SamplesPerSymbol);
eyeOut = reshape(waveOut,SamplesPerSymbol,[]);
```

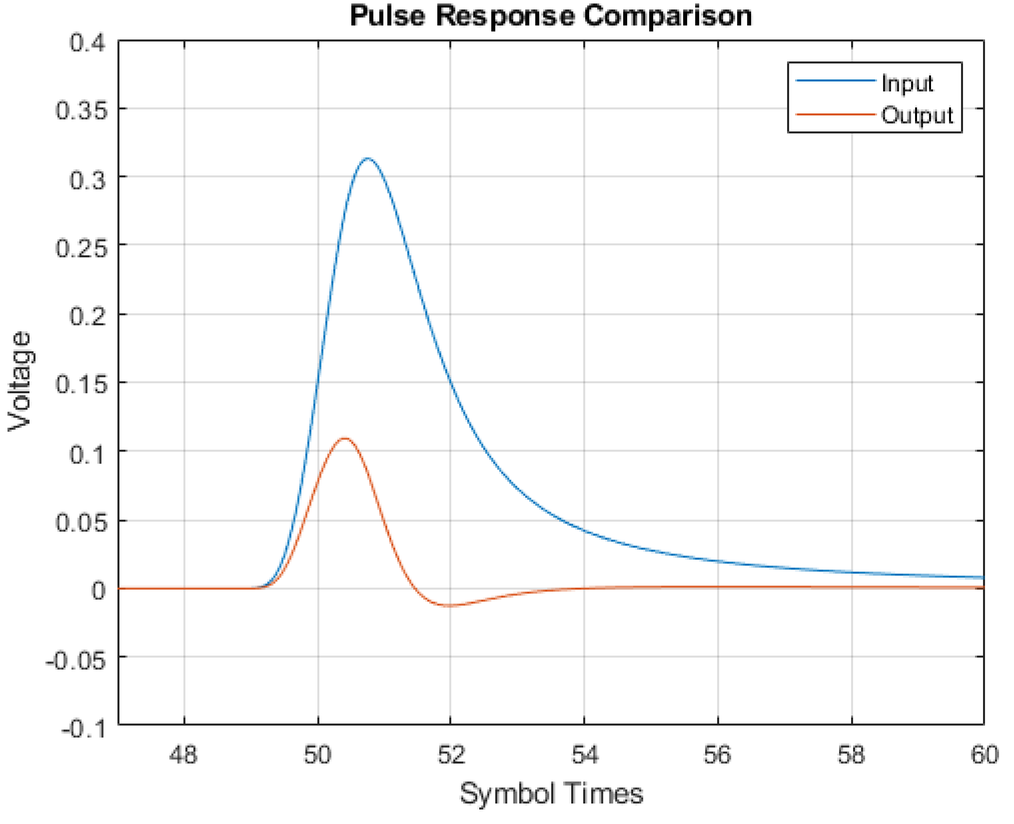
Create the time vectors.

```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

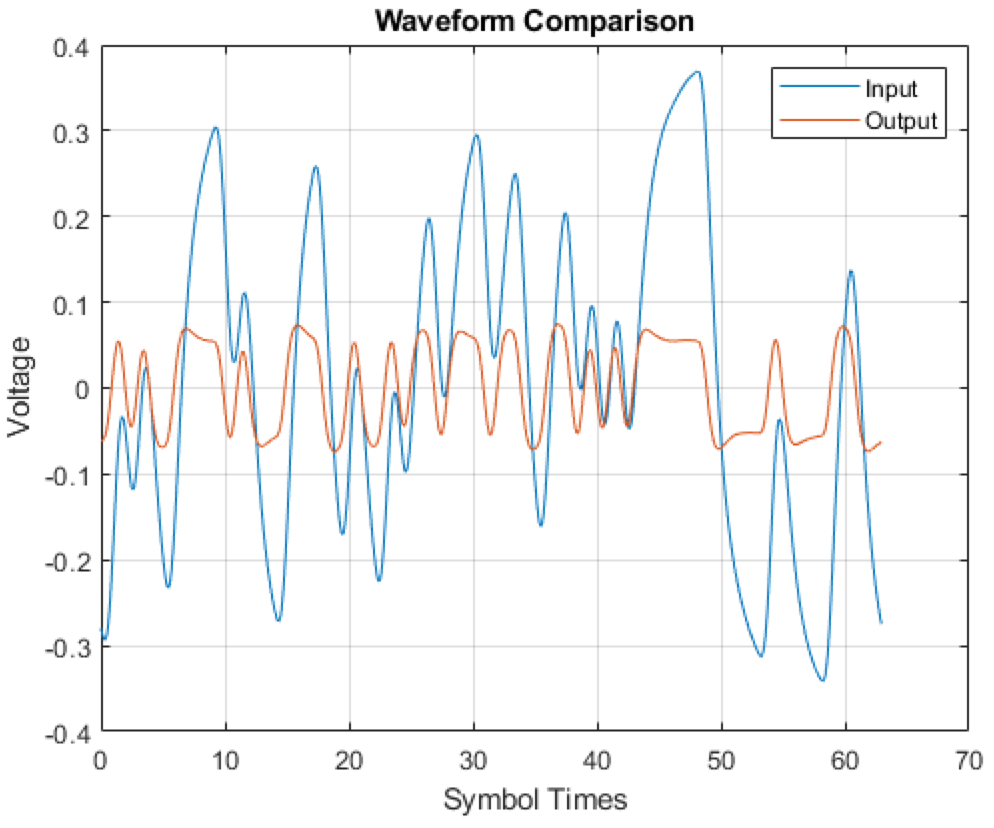
Plot pulse response comparison, waveform comparison, input, and output eye diagrams.

```
figure
plot(t,pulseIn,t,pulseOut)
```

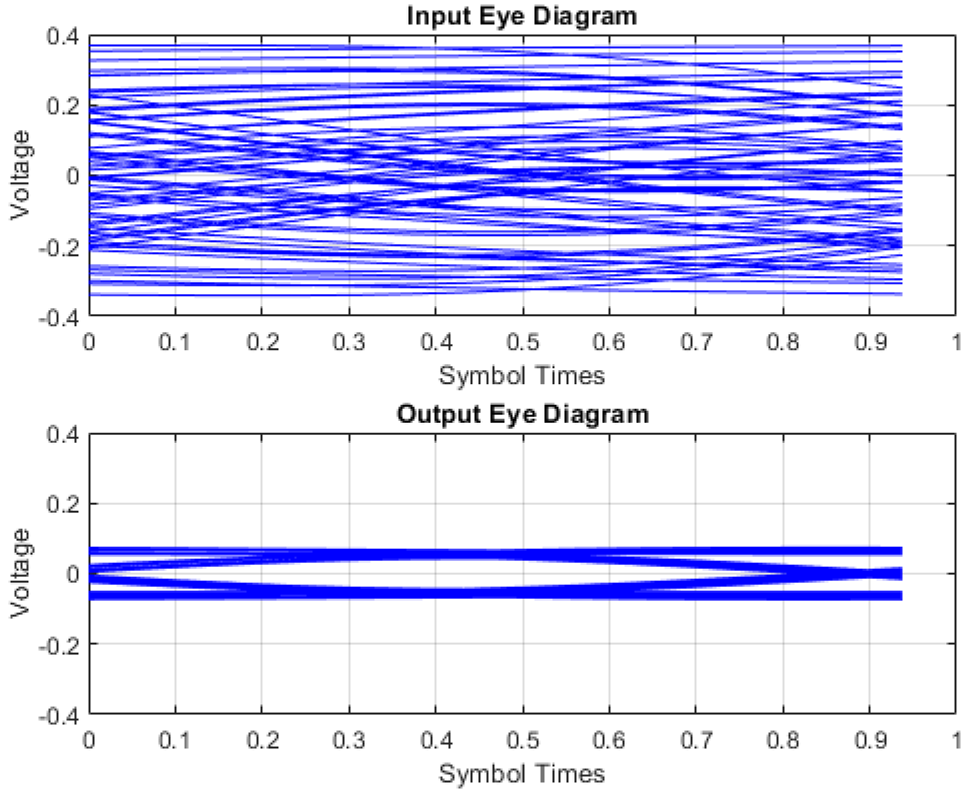
```
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('Symbol Times'),ylabel('Voltage')
grid on
axis([47 60 -0.1 0.4])
```




```
figure
plot(t2,waveIn,t2,waveOut)
legend('Input','Output')
title('Waveform Comparison')
xlabel('Symbol Times'),ylabel('Voltage')
grid on
```




```
figure
subplot(211),plot(teye,eyeIn,'b')
ax = axis;
xlabel('Symbol Times'),ylabel('Voltage')
grid on
title('Input Eye Diagram')
subplot(212),plot(teye,eyeOut,'b')
axis(ax);
xlabel('Symbol Times'),ylabel('Voltage')
grid on
title('Output Eye Diagram')
```



Sample-by-Sample Processing Using CTLE

This example shows how to process impulse response of a channel one sample at a time using `serdes.CTLE` System object™.

Use a symbol time of 100 ps and 16 samples per symbol. The channel has `16` dB loss. The peaking frequency is `11` GHz. Select `12`-th order pseudorandom binary sequence (PRBS), and simulate the first `500` symbols.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
```

```
dbloss = 16;
DCGain = 0:-1:-26;
PeakingGain = 0:26;
PeakingFrequency = 11e9;
ConfigSelect = 15;
prbsOrder = 12;
M = 500;
```

Calculate the sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the CTLE object. Since we are processing the channel one sample at a time, the input waveform is `'sample'` type. The object adaptively applies the optimum filter transfer function for the best eye height opening.

```
CTLE = serdes.CTLE('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',2,'WaveType','Sample',...
      'DCGain',DCGain,'PeakingGain',PeakingGain,...
      'PeakingFrequency',PeakingFrequency,...
      'ConfigSelect',ConfigSelect);
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
```

Create the eye diagram.

```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

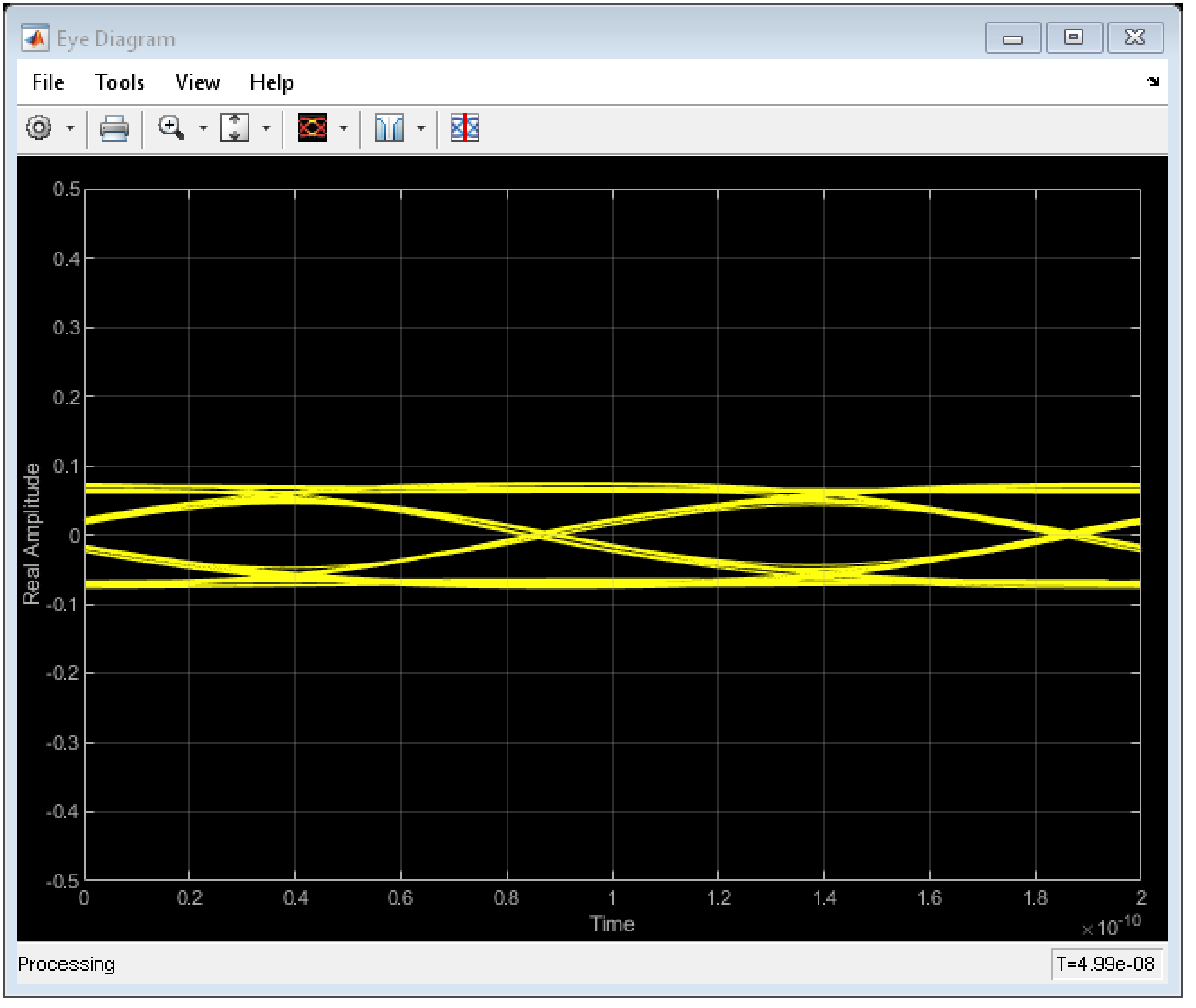
Initialize PRBS generator.

```
[dataBit,prbsSeed] = prbs(prbsOrder,1);
```

Loop through one symbol at at time.

```
  inwave = zeros(SamplesPerSymbol,1);
  outwave = zeros(SamplesPerSymbol,1);
  for ii = 1:M
      % Get new symbol
      [dataBit,prbsSeed] = prbs(prbsOrder,1,prbsSeed);
      inwave(1:SamplesPerSymbol) = dataBit-0.5;
```

```
    % Convolve input waveform with channel
    y = channel(inwave);

    % Process one sample at a time through the CTLE
    for jj = 1:SamplesPerSymbol
        outwave(jj) = CTLE(y(jj));
    end

    % Plot eye diagram
    eyediagram(outwave)
end
```


Eye Diagram
File Tools View Help
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
Real Amplitude
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
Time
× 10^-10
Processing
T=4.99e-08

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

AGC | CTLE | DFECDR | SaturatingAmplifier | `serdes.AGC` | serdes.DFECDR

Introduced in R2019a

serdes.DFECDR

Decision feedback equalizer (DFE) with clock and data recovery (CDR)

Description

The `serdes.DFECDR` System object adaptively processes a sample-by-sample input signal or analytically processes an impulse response vector input signal to remove distortions at post-cursor taps.

The decision feedback equalizer modifies baseband signals to minimize the intersymbol interference (ISI) at the clock sampling time. The DFE samples data at each clock tick and adjusts the amplitude of the waveform by a correction voltage. The correction voltage is determined by the previous *N* sampled unit interval (UI) values, where *N* is the number of DFE taps.

A clock and data recovery function provides the clock sampling location to the DFE. The clock recovery is a first order phase tracking CDR model.

To equalize the input signal:

1. Create the `serdes.DFECDR` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
dfecdr = serdes.DFECDR
dfecdr = serdes.DFECDR(Name,Value)
```

Description

`dfecdr = serdes.DFECDR` returns a DFECDR object that modifies an input waveform with the DFE and determines the clock sampling times. The system object estimates the data symbol according to the Bang-Bang CDR algorithm.

`dfecdr = serdes.DFECDR(Name,Value)` returns a DFECDR object with each specified property set to specified value. Unspecified properties have default values.

Example: `dfecdr = serdes.DFECDR('Mode',1)` returns a DFECDR object that applies specified DFE tap weights to input waveform.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

DFE Properties

Mode — DFE operating mode
2 (default) | 0 | 1

DFE operating mode, specified as `0`, `1`, or `2`. `Mode` determines what DFE tap weight values are applied to the input waveform.

Mode Value	DFE Mode	DFE Operation
`0`	off	`serdes.DFECDR` is bypassed and the input waveform remains unchanged.
`1`	fixed	`serdes.DFECDR` applies input DFE tap weights specified in `TapWeights` to the input waveform.

Mode Value	DFE Mode	DFE Operation
2	adapt	`serdes.DFECDR` adaptively determines the optimum DFE tap weights values and applies them to the input waveform.

Data Types: `double`

`TapWeights` — Initial DFE tap weights
`[0 0 0 0]` (default) | row vector

Initial DFE tap weights, specified as a row vector in volts (V). The length of the vector specifies the number of taps. Each vector element value specifies the strength of the tap at that element position. Setting a vector element value to zero only initializes the tap.

Data Types: `double`

`EqualizationGain` — Controls DFE tap weight update rate
`3e-6` (default) | scalar

Controls DFE tap weight update rate, specified as a unitless scalar. Increasing the value of `EqualizationGain` leads to a faster convergence of DFE adaptation at the expense of more noise in DFE tap values.

Data Types: `double`

`EqualizationStep` — DFE adaptive step resolution
`1e-6` (default) | real scalar

DFE adaptive step resolution, specified as a real scalar in volts (V). `EqualizationStep` specifies the minimum DFE tap change from one time step to the next to mimic hardware impairment. Setting `EqualizationStep` to zero yields DFE tap values without any resolution limitation.

Data Types: `double`

CDR Properties

`Count` — Early or late CDR count threshold to trigger phase update
`16` (default) | real positive integer >4

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer >4. Increasing the value of `Count` provides a more stable output clock phase at the expense of convergence speed. Because the bit decisions are made at the clock phase output, a more stable clock phase has a better bit error rate (BER).

Data Types: `double`

`ClockStep` — Clock phase resolution
`0.0078` (default) | real scalar

Clock phase resolution, specified as a real scalar in fraction of unit interval (UI). `ClockStep` is the inverse of the number of phase adjustments in CDR.

Data Types: `double`

`PhaseOffset` — Clock phase offset
`0` (default) | real scalar in the range [-0.5,0.5]

Clock phase offset, specified as a real scalar in the range [-0.5,0.5] in fraction of symbol time. `PhaseOffset` is used to manually shift the clock probability distribution function (PDF) for better BER.

Data Types: `double`

`ReferenceOffset` — Reference clock offset impairment
`0` (default) | real scalar in the range [-3e-4,3e-4]

Reference clock offset impairment, specified as a real scalar in the range [-3e-4,3e-4] in parts per million (ppm). `ReferenceOffset` is the deviation between transmitter oscillator frequency and receiver oscillator frequency.

Data Types: `double`

`Sensitivity` — Sampling latch meta-stability voltage
`0` (default) | real scalar

Sampling latch meta-stability voltage, specified as a real scalar in volts (V). If the data sample voltage lies within the region (+/-`Sensitivity`), there is a 50% probability of bit error.

Data Types: `double`

Advanced Properties

`SymbolTime` — Time of single symbol duration
`1e-10` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in seconds (s).

Data Types: `double`

`SampleInterval` — Uniform time step of waveform
`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in seconds (s).

Data Types: `double`

`Modulation` — Modulation scheme
2 (default) | 4

Modulation scheme, specified as 2 or 4.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
4	Four-level pulse amplitude modulation (PAM4)

Data Types: `double`

`WaveType` — Input wave type form
`'Sample'` (default) | `'Impulse'`

Input wave type form, specified as:

- `'Sample'` — A sample-by-sample input signal.
- `'Impulse'` — An impulse response input signal.

Data Types: `char`

Usage

Syntax

```
y = dfecdr(x)
```

Description

```
y = dfecdr(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal. If the `WaveType` is set to `'Sample'`, then the input signal is a sample-by-sample signal specified as a scalar. If the `WaveType` is set to `'Impulse'`, the input signal is an impulse response vector signal.

Output Arguments

y — Estimated channel output
scalar | vector

Estimated channel output. If the input signal is a sample-by-sample signal specified as a scalar, then the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Impulse Response Processing Using DFECDR

This example shows how to process impulse response of a channel using `serdes.DFECDR` system object™.

Use a symbol time of `100` ps. There are `16` samples per symbol. The channel has `14` dB loss.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
dbloss = 14;
NumberOfDFETaps = 2;
```

Calculate the sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the DFECDR object. The object adaptively applies optimum DFE tap weights to input impulse response.

```
DFE1 = serdes.DFECDR('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',2,'WaveType','Impulse','TapWeights',zeros(NumberOfDFETaps,1));
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
impulseIn = channel.impulse;
```

Process the impulse response with DFE.

```
[impulseOut,TapWeights] = DFE1(impulseIn);
```

Convert the impulse response to a pulse, a waveform and an eye diagram for visualization.

```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

pulseIn = impulse2pulse(impulseIn,SamplesPerSymbol,dt);
waveIn = pulse2wave(pulseIn,dataPattern,SamplesPerSymbol);
eyeIn = reshape(waveIn,SamplesPerSymbol,[]);

pulseOut = impulse2pulse(impulseOut,SamplesPerSymbol,dt);
```

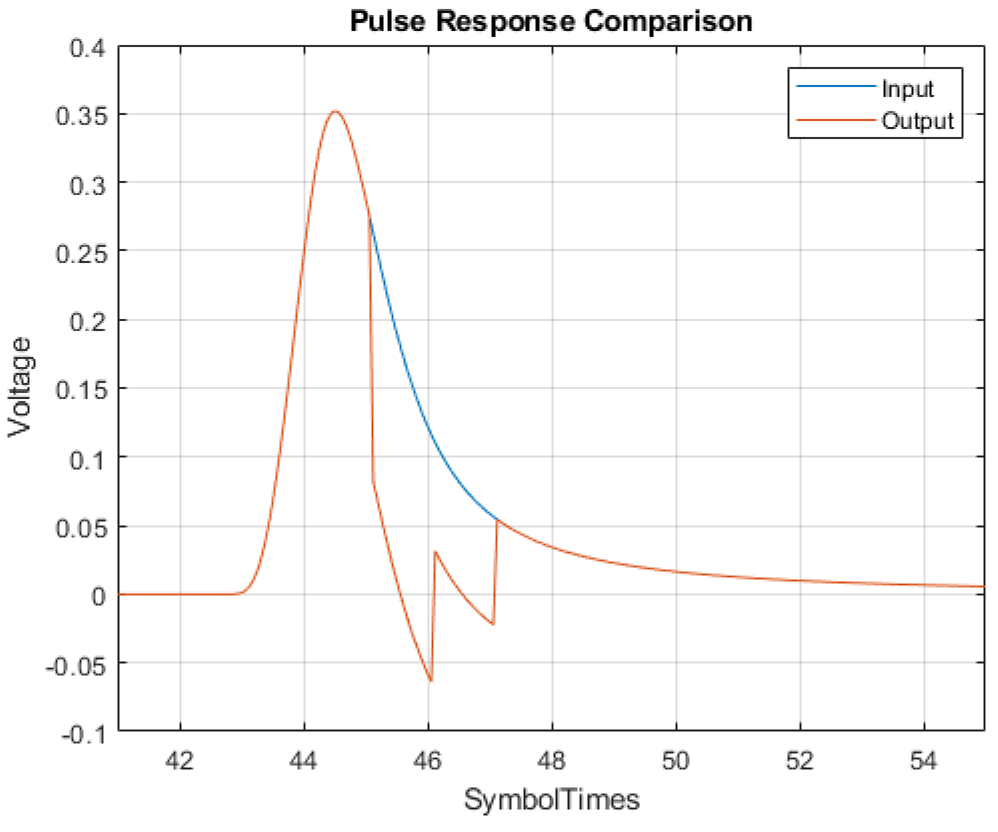
```
waveOut = pulse2wave(pulseOut,dataPattern,SamplesPerSymbol);
eyeOut = reshape(waveOut,SamplesPerSymbol,[]);
```

Create the time vectors.

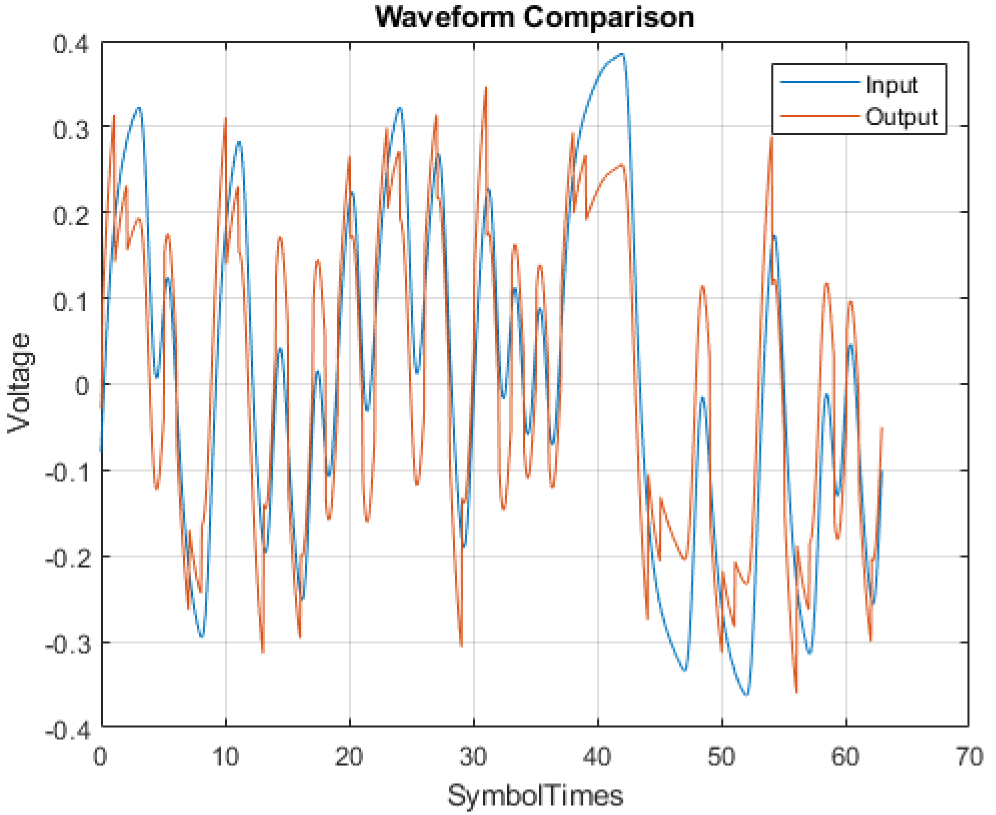
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

Plot the resulting waveforms.

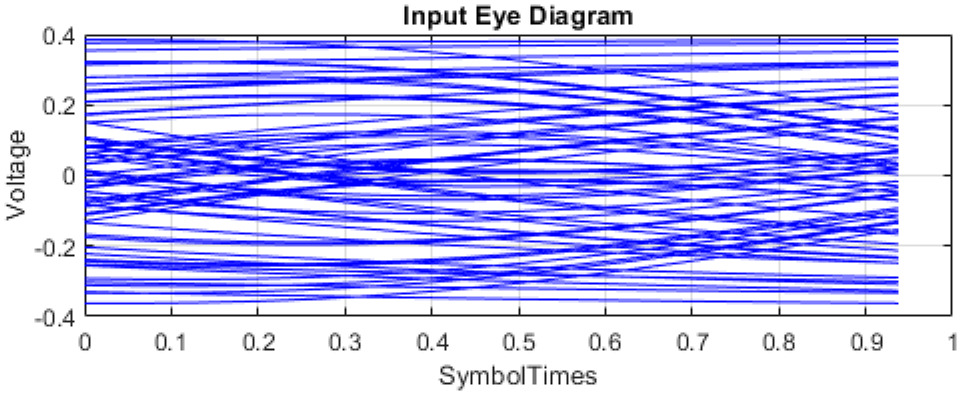
```
figure
plot(t,pulseIn,t,pulseOut)
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
axis([41 55 -0.1 0.4])
```

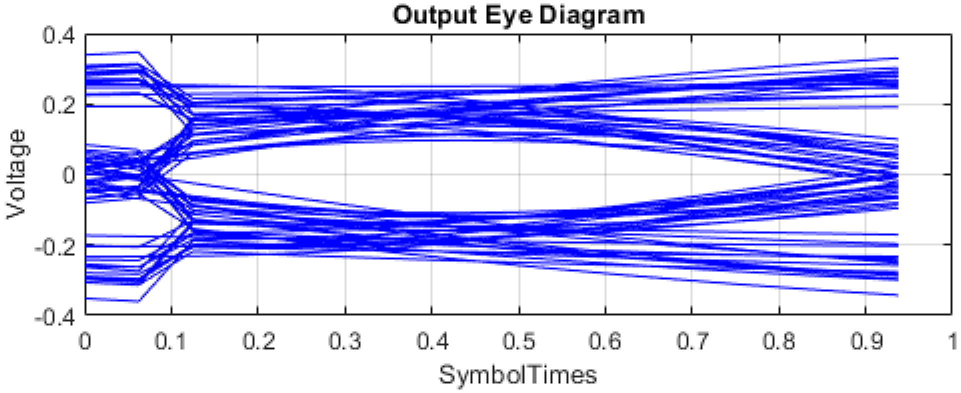



```
figure
plot(t2,waveIn,t2,waveOut)
legend('Input','Output')
title('Waveform Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
```




```
figure
subplot(211),plot(teye,eyeIn,'b')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
title('Input Eye Diagram')
subplot(212),plot(teye,eyeOut,'b')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
title('Output Eye Diagram')
```





Sample-by-Sample Processing Using DFECDR

This example shows how to process impulse response of a channel one sample at a time using `serdes.DFECDR` system object™.

Use a symbol time of `100` ps, with `8` samples per symbol. The channel loss is `14` dB. Select `12`-th order pseudorandom binary sequence (PRBS), and simulate the first 4000 symbols.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 8;
```

```
dbloss = 14;
NumberOfDFETaps = 2;
prbsOrder = 12;
M = 4000;
```

Calculate sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the DFECDR system object. Since we are processing the channel one sample at a time, the input waveform is `'sample'` type. The object adaptively applies the optimum DFE tap weights to input waveform.

```
DFE2 = serdes.DFECDR('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',2,'WaveType','Sample','TapWeights',zeros(NumberOfDFETaps,1),...
      'EqualizationStep',0,'EqualizationGain',1e-4);
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
```

Create the eye diagram.

```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

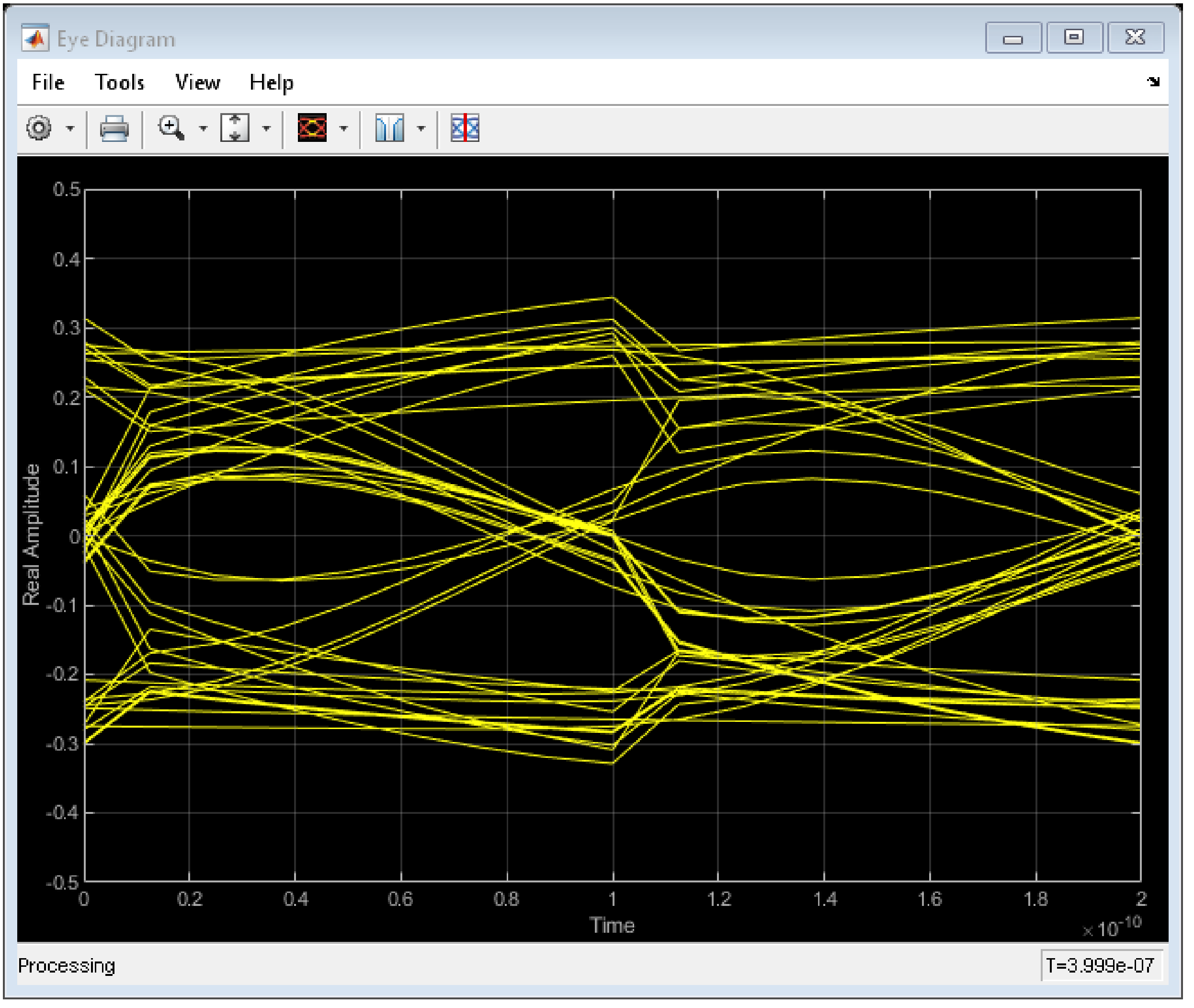
Initialize the PRBS generator.

```
[dataBit,prbsSeed]=prbs(prbsOrder,1);
```

Generate the sample-by-sample eye diagram.

```
%Loop through one symbol at a time.
inwave = zeros(SamplesPerSymbol,1);
outwave = zeros(SamplesPerSymbol,1);
dfeTapWeightHistory = nan(M,NumberOfDFETaps);

for ii = 1:M
  %Get new symbol
      [dataBit,prbsSeed]=prbs(prbsOrder,1,prbsSeed);
      inwave(1:SamplesPerSymbol) = dataBit-0.5;

   %Convolve input waveform with channel
      y = channel(inwave);
```

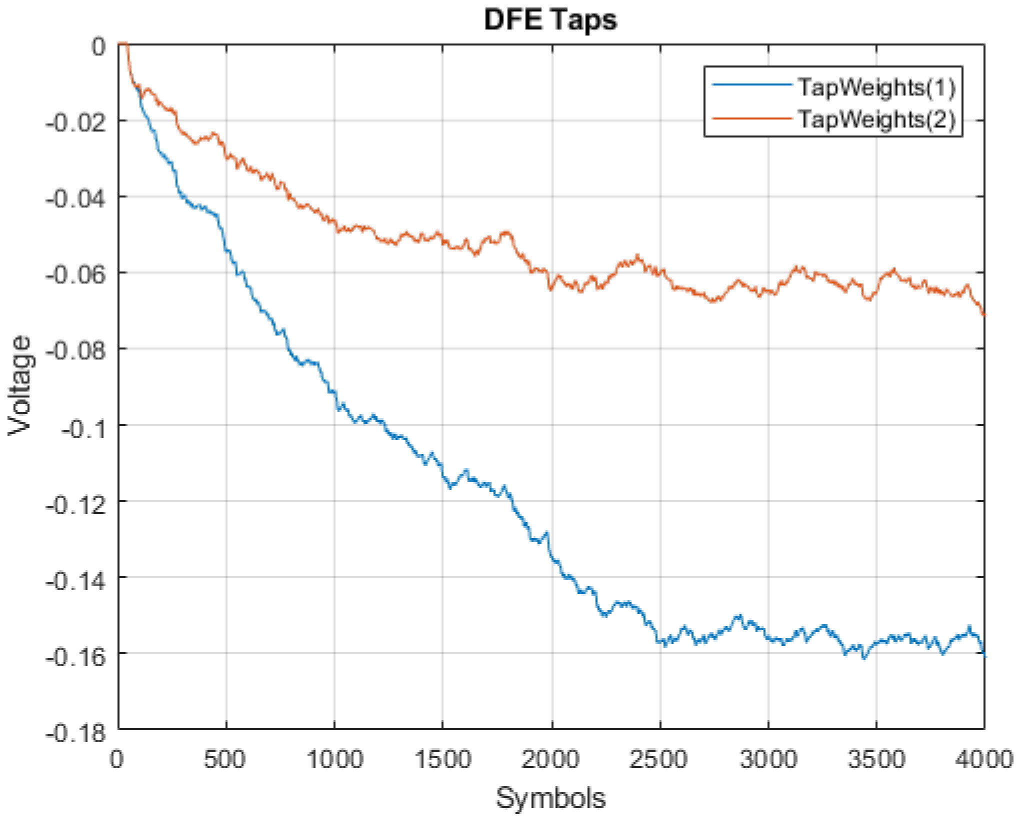
```
    %Process one sample at a time through the DFE
       for jj = 1:SamplesPerSymbol
           [outwave(jj),TapWeights] = DFE2(y(jj));
       end

     %Save DFE taps
       dfeTapWeightHistory(ii,:) = TapWeights;

     %Plot eye diagram
       eyediagram(outwave)
end
```



Plot the DFE adaptation history.

```
figure
plot(dfeTapWeightHistory)
grid on
legend('TapWeights(1)','TapWeights(2)')
xlabel('Symbols')
```

```
ylabel('Voltage')
title('DFE Taps')
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

CDR | CTLE | DFECDR | `serdes.CDR` | `serdes.CTLE`

Introduced in R2019a

serdes.FFE

Models a feed-forward equalizer

Description

The `serdes.FFE` System object applies a feed-forward equalizer as a symbol-spaced finite-impulse response (FIR) filter to a sample-by-sample input signal or an impulse response vector input signal to reduce distortions due to channel loss impairments.

To equalize the baseband signal:

1. Create the `serdes.FFE` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
ffe = serdes.FFE
ffe = serdes.FFE(Name,Value)
```

Description

`ffe = serdes.FFE` returns an FFE object that modifies an input waveform according to the finite impulse response (FIR) transfer function defined in the object.

`ffe = serdes.FFE(Name,Value)` returns an FFE object with each specified property set to specified value. Unspecified properties have default values.

Example: `ffe = serdes.FFE('Mode',1)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

`Modulation` — Modulation scheme

2 (default) | 4

Modulation scheme, specified as `2` or `4`.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
4	Four-level pulse amplitude modulation (PAM4)

Data Types: `double`

Main

`Mode` — FFE operating mode

1 (default) | 0

FFE operating mode, specified as `0` or `1`. `Mode` determines whether FFE is bypassed or not.

Mode Value	FFE Mode	FFE Operation
0	Off	`serdes.FFE` is bypassed, the input waveform remains unchanged.
1	Fixed	`serdes.FFE` applies input FFE tap weights specified in `TapWeights` to input waveform.

Data Types: `double`

`TapWeights` — FFE tap weights

`[0 1 0 0 0]` (default) | row vector

FFE tap weights, specified as a row vector in V. The length of the vector specifies the number of taps. The vector element value specifies the strength of the tap at that element position. The tap with the largest magnitude is the main tap and therefore defines the number of pre- and post-taps.

Data Types: `double`

`Normalize` — Normalize tap weights

`'true'` (default) | `'false'`

Normalize tap weight vectors so that the sum of the absolute values of the `TapWeights` vector elements is one.

Data Types: `char`

Advanced

`SymbolTime` — Time of single symbol duration

`1e-10` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: `double`

`SampleInterval` — Uniform time step of waveform

`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

`WaveType` — Input wave type form

`'Sample'` (default) | `'Impulse'`

Input wave type form, specified as:

- `'Sample'` - A sample-by-sample input signal.
- `'Impulse'` - An impulse response input signal.

Data Types: `char`

Usage

Syntax

```
y = ffe(x)
```

Description

```
y = ffe(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal. If the `WaveType` is set to `'Sample'`, the input signal is a sample-by-sample signal specified as a scalar. If the `WaveType` is set to `'Impulse'`, the input signal must be an impulse response vector signal.

Output Arguments

y — Filtered channel output
scalar | vector

Filtered channel output. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Impulse Response Processing Using FFE

This example shows how to process impulse response of a channel using `serdes.FFE` system object™.

Use a symbol time of `100` ps and `16` samples per symbol. The channel has `16` dB loss.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
dbloss = 16;
```

Calculate the sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the FFE object with fixed mode of operation.

```
TapWeights = [0 0.7 -0.2 -0.10];
FFEMode = 1;
FFE1 = serdes.FFE('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',FFEMode,'WaveType','Impulse',...
      'TapWeights',TapWeights);
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
impulseIn = channel.impulse;
```

Process impulse response with FFE.

```
impulseOut = FFE1(impulseIn);
```

Convert the impulse responses to pulse, waveform and eye diagram for visualization.

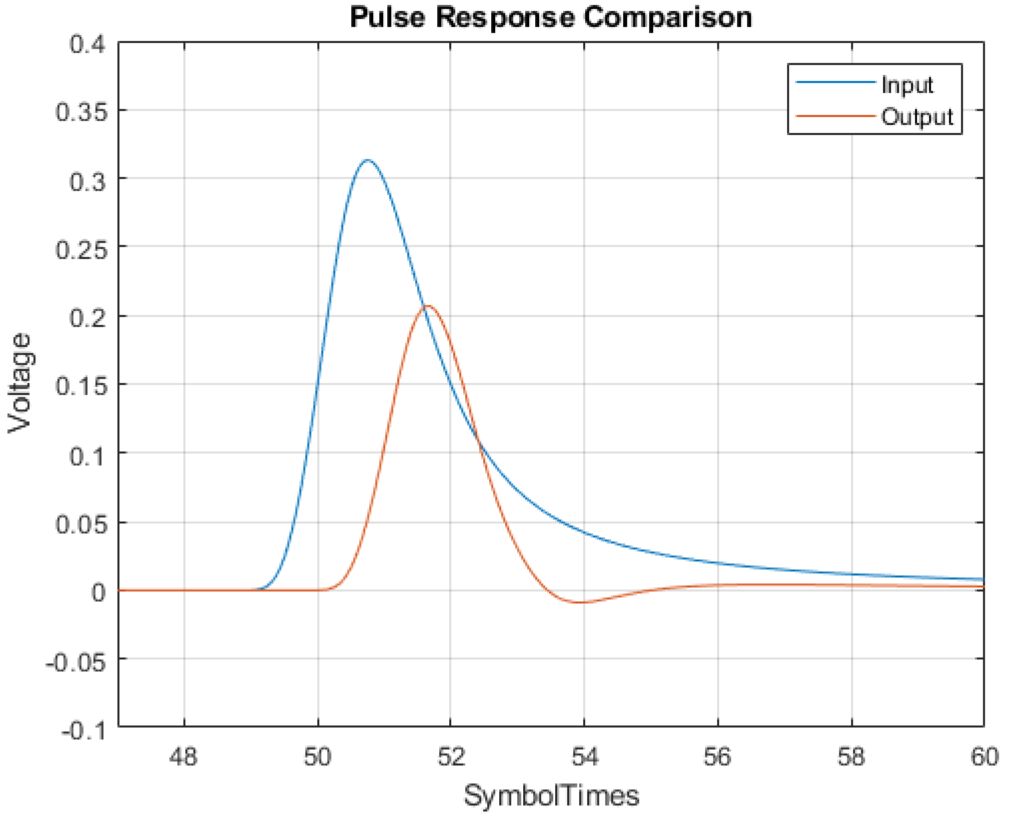
```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

pulseIn = impulse2pulse(impulseIn,SamplesPerSymbol, dt);
waveIn = pulse2wave(pulseIn,dataPattern,SamplesPerSymbol);
eyeIn = reshape(waveIn,SamplesPerSymbol,[]);

pulseOut = impulse2pulse(impulseOut,SamplesPerSymbol, dt);
waveOut = pulse2wave(pulseOut,dataPattern,SamplesPerSymbol);
eyeOut = reshape(waveOut,SamplesPerSymbol,[]);
```

Create the time vectors.

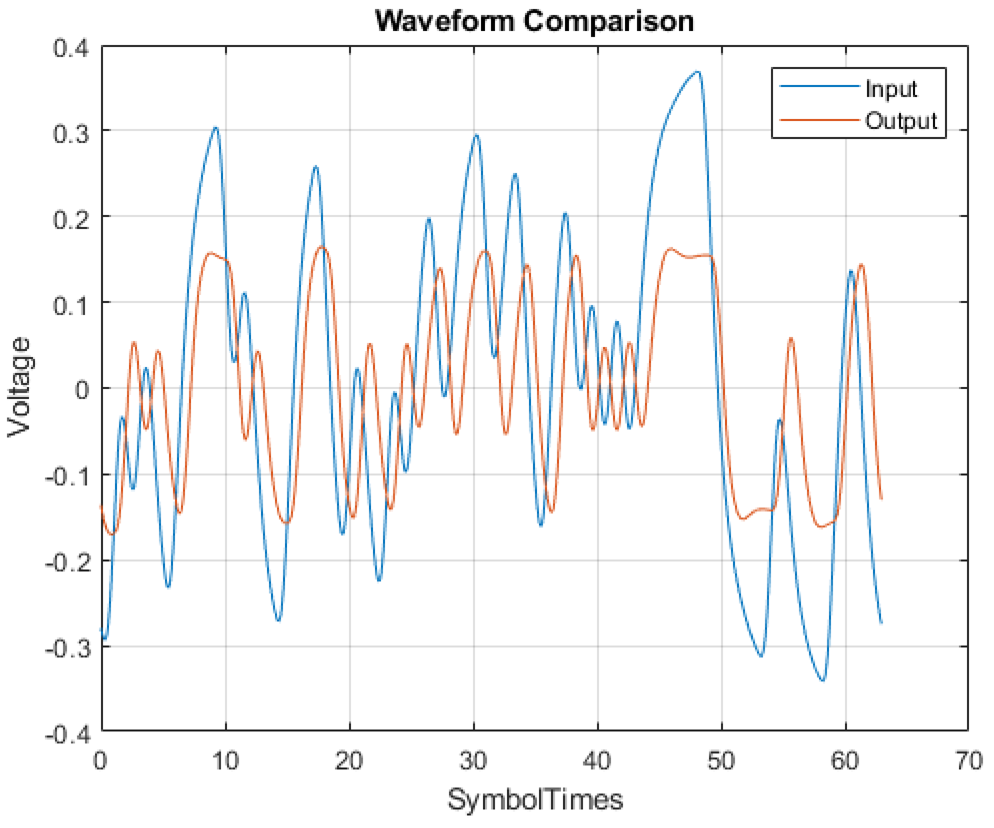
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

Plot the pulse response comparison, waveform comparison, and input and output eye diagrams.

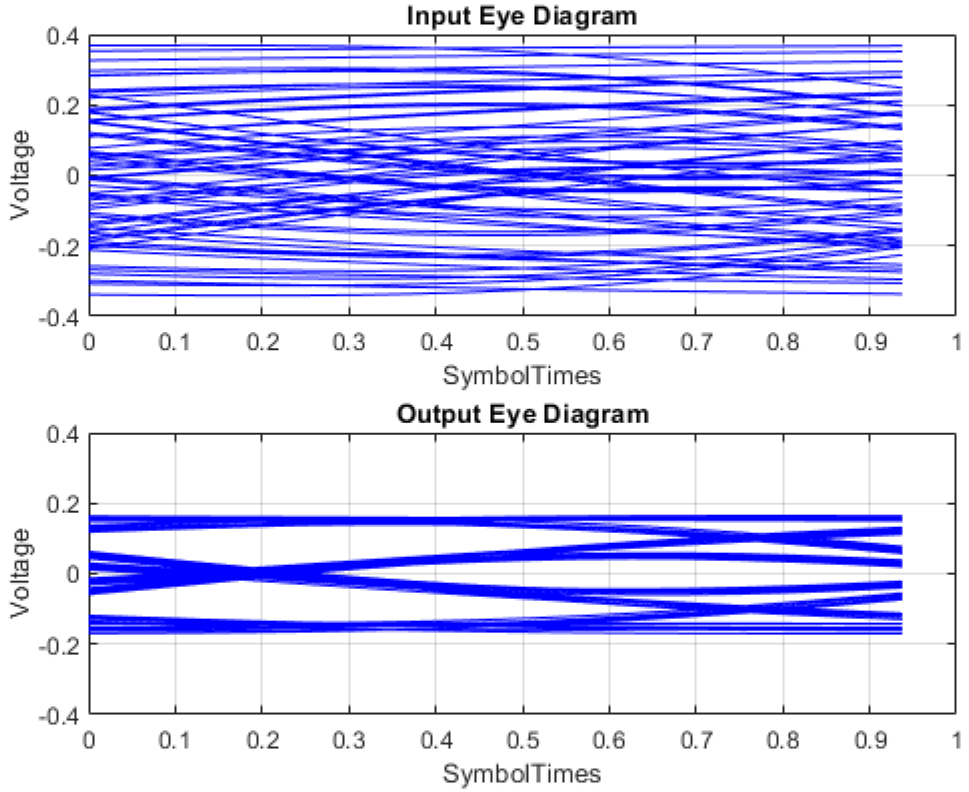
```
figure
plot(t,pulseIn,t,pulseOut)
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
axis([47 60 -0.1 0.4])
```




```
figure
plot(t2,waveIn,t2,waveOut)
legend('Input','Output')
title('Waveform Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
```




```
figure
subplot(211),plot(teye,eyeIn,'b')
ax = axis;
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
title('Input Eye Diagram')
subplot(212),plot(teye,eyeOut,'b')
axis(ax);
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
title('Output Eye Diagram')
```



Sample-by-Sample Processing Using FFE

This example shows how to process impulse response of a channel one sample at a time using `serdes.FFE` system object™.

Use a symbol time of `100` ps with `16` samples per symbol. The channel has `16` dB loss.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 16;
dbloss = 16;
```

Calculate the sample interval.

```
dt = SymbolTime/SamplesPerSymbol;
```

Create the FFE object with fixed mode.

```
FFEMode = 1;
TapWeights = [0 0.7 -0.2 -0.1];
FFE = serdes.FFE('SymbolTime',SymbolTime,'SampleInterval',dt,...
      'Mode',FFEMode,'WaveType','Sample',...
      'TapWeights',TapWeights);
```

Create the channel impulse response.

```
channel = serdes.ChannelLoss('Loss',dbloss,'dt',dt,...
      'TargetFrequency',1/SymbolTime/2);
```

Create the Eye Diagram.

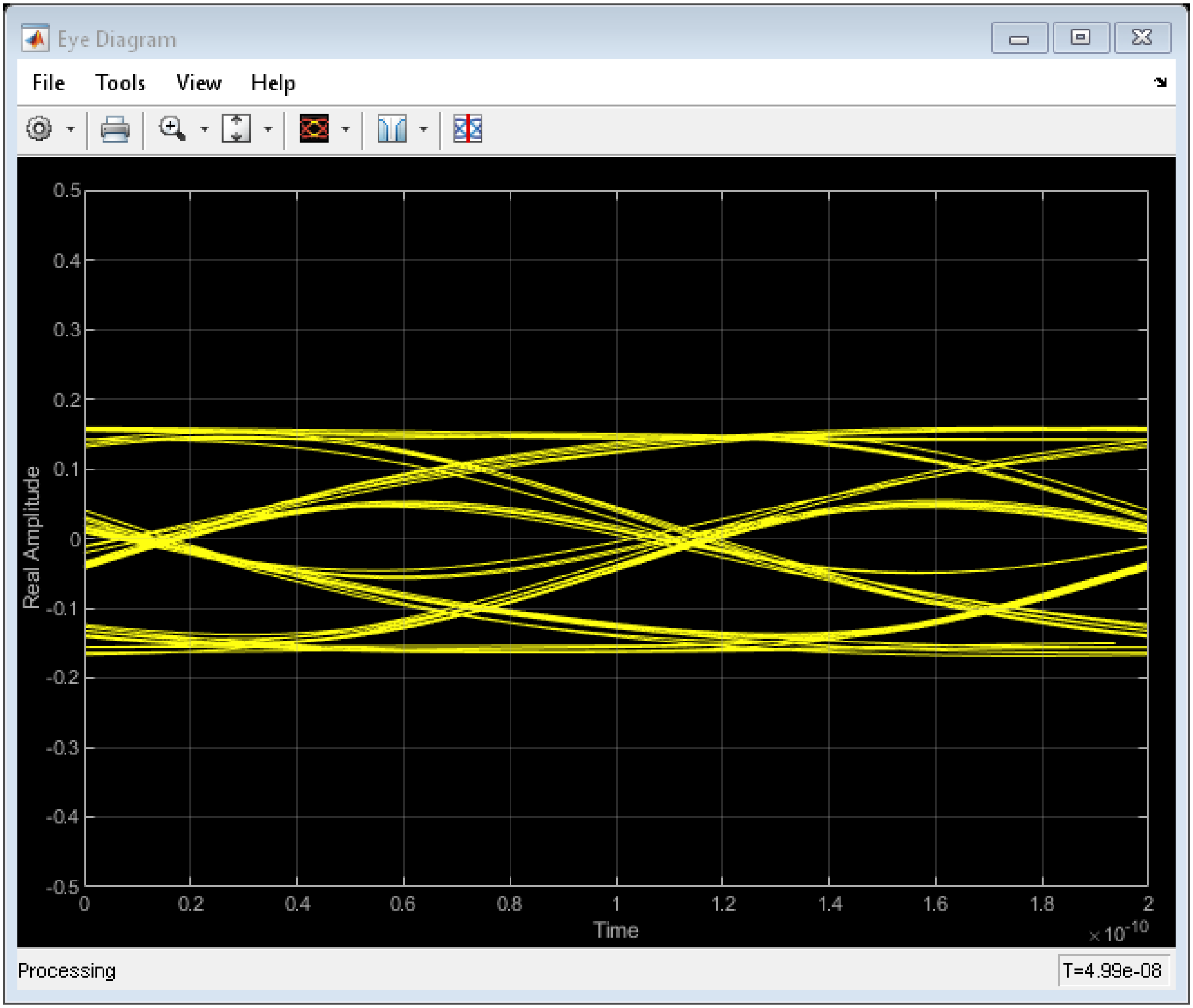
```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

Initialize the pseudorandom binary sequence (PRBS) code generator of order 12.

```
prbsOrder = 12;
M = 500;                                %number of symbols to simulate
[dataBit,prbsSeed]=prbs(prbsOrder,1);
```

Loop through one symbol at a time.

```
inwave = zeros(SamplesPerSymbol,1);
outwave = zeros(SamplesPerSymbol,1);
for ii = 1:M
      %Get new symbol
      [dataBit,prbsSeed]=prbs(prbsOrder,1,prbsSeed);
      inwave(1:SamplesPerSymbol) = dataBit-0.5;

      %convolve input waveform with channel
      y = channel(inwave);

      %process one sample at a time through the FFE
      for jj = 1:SamplesPerSymbol
          outwave(jj) = FFE(y(jj));
      end

      %Plot eye diagram
      eyediagram(outwave)
end
```


Eye Diagram
File Tools View Help
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
Real Amplitude
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
Time
×10^-10
Processing
T=4.99e-08

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

CTLE | FFE | `serdes.CTLE`

Introduced in R2019a

serdes.PassThrough

Propagates baseband signal without modification

Description

The `serdes.PassThrough` system object passes the input signal without any modification. This system object is used as a place holder within a SerDes system and as a template for user-authored system objects for use in SerDes Toolbox.

To propagate the signal through a `serdes.PassThrough`:

1. Create the `serdes.PassThrough` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
PassThrough = serdes.PassThrough
PassThrough = serdes.PassThrough (Name,Value)
```

Description

`PassThrough = serdes.PassThrough` returns an empty pass through object that returns the input signal unchanged.

`PassThrough = serdes.PassThrough (Name,Value)` returns an empty pass through object with each specified property set to specific value. Unspecified properties have default values.

Example: `SatAmp = serdes.PassThrough('Modulation',4)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

`Modulation` — Modulation scheme
2 (default) | 4

Modulation scheme, specified as 2 or 4.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
4	Four-level pulse amplitude modulation (PAM4)

Data Types: `double`

`SymbolTime` — Time of single symbol duration
`1e-10` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: `double`

`SampleInterval` — Uniform time step of waveform
`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

Usage

Syntax

```
y = PassThrough(x)
```

Description

```
y = PassThrough(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal.

Output Arguments

y — Unchanged output voltage
scalar | vector

Unchanged output voltage, as specified by the `serdes.PassThrough` object.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics

reset	Reset internal states of System object

Examples

Propagate Input Waveform Using PassThrough

This example shows how to propagate an input waveform without modification using a `serdes.PassThrough` system object™.

Create the incoming waveform.

```
t = linspace(0,12,101);
y1 = sin(t);
```

Create the PassThrough object.

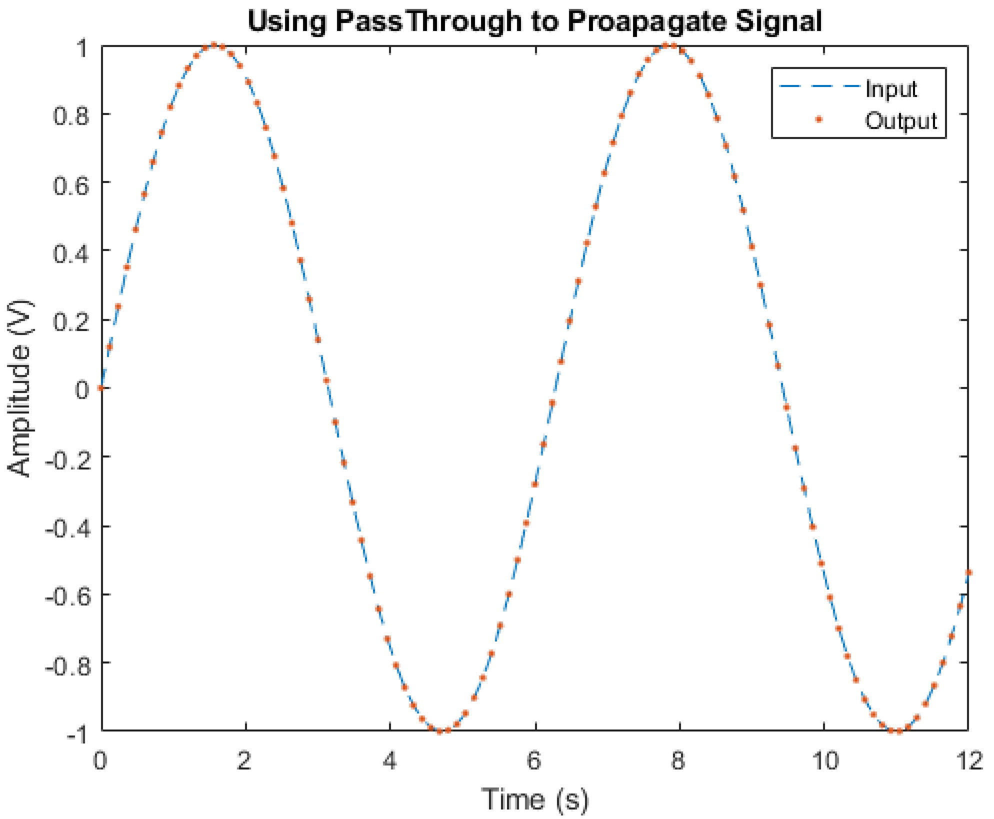
```
PT = serdes.PassThrough;
```

Process the input waveform with the PassThrough object.

```
y2 = PT(y1);
```

Plot the input and output waveforms.

```
figure, plot(t,y1,'--',t,y2,'.')
legend('Input','Output')
title('Using PassThrough to Proapagate Signal');
xlabel('Time (s)');
ylabel('Amplitude (V)');
```



Verify the equality of input and output signals.

```
isequal(y1,y2)
```

```
ans = logical
   1
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

CTLE | DFECDR | FFE | `serdes.CTLE` | `serdes.DFECDR` | `serdes.FFE`

Introduced in R2019a

serdes.SaturatingAmplifier

Models a saturating amplifier

Description

The `serdes.SaturatingAmplifier` System object scales the input waveform according to a voltage in vs. voltage out response. The voltage in vs. voltage out response is specified either by the soft clipping response defined by `Limit` and `Linear Gain`, or by the `VinVout` matrix.

To limit the voltage output to a specific value:

1. Create the `serdes.SaturatingAmplifier` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
SatAmp = serdes.SaturatingAmplifier
SatAmp = serdes.SaturatingAmplifier (Name,Value)
```

Description

`SatAmp = serdes.SaturatingAmplifier` returns an amplifier object that modifies the input signal so that the output voltage is clipped to a specific value defined by `Limit`.

`SatAmp = serdes.SaturatingAmplifier (Name,Value)` returns a CTLE object with each specified property set to specific value. Unspecified properties have default values.

Example: `SatAmp = serdes.SaturationgAmplifier('Limit',5)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

`Mode` — Amplifier operating mode
`1` (default) | `0`

Amplifier operating mode, specified as `0` or `1`. `Mode` determines whether the amplifier is bypassed or not.

Mode Value	Saturating Amplifier Mode	Saturating Amplifier Operation
`0`	Off	`serdes.SaturatingAmplifier` is bypassed, the input waveform remains unchanged.
`1`	On	`serdes.SaturatingAmplifier` scales the input waveform according to a voltage in vs. voltage out response.

Data Types: `double`

`Specification` — Input specification for limiting amplifier output
`Limit and Linear Gain` (default) | `VinVout`

Input specification for limiting amplifier output, specified as:

- `Limit and Linear Gain` - Creates a soft clipping voltage in vs. voltage out response with the values specified in `Limit` and `Linear Gain`.
- `VinVout` - Generates output voltages corresponding to input voltage specified in `VinVout`. If any input voltage point falls outside the specified values, the output for that particular input voltage is linearly interpolated.

Data Types: `char`

`Limit` — Clipping voltage for the limiting amplifier
`1.2` (default) | real positive scalar

Clipping voltage for the limiting amplifier, specified as a real positive scalar in V.

Data Types: `double`

`LinearGain` — Amplifier gain in the linear region
`1` (default) | real positive scalar

Amplifier gain in the linear region, specified as a unitless real positive scalar.

Data Types: `double`

`VinVout` — Input and corresponding output voltage response table
N×2 matrix

Input and corresponding output voltage response table, specified as an N×2 matrix in V.

Data Types: `double`

Usage

Syntax

```
y = SatAmp(x)
```

Description

```
y = SatAmp(x)
```

Input Arguments

x — Input baseband signal
scalar | vector

Input baseband signal.

Output Arguments

y — Clipped output voltage
scalar | vector

Clipped output voltage, as specified by the `serdes.SaturatingAmplifier` object.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Cliiping Input Waveform Using SaturatingAmplifier

This example shows how to clip an incoming sine wave using the `serdes.SaturatingAmplifier` system object™.

Define the input sine wave 250 Hz.

```
Fs = 10000;
L = 100;
t = (0:L-1)'/Fs;
x = sin(2*pi*250*t);
```

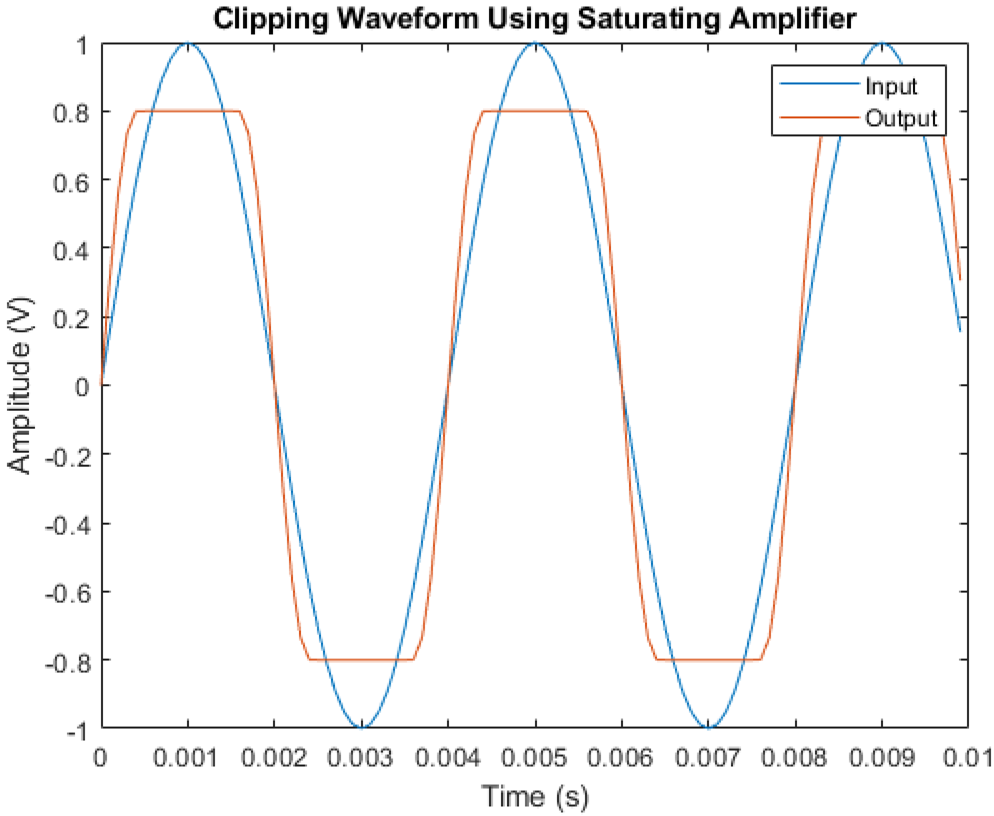
Construct the SaturatingAmplifier system object with a linear gain of 2, and gain limit of 0.8 V.

```
linearGain = 2;
limit = 0.8;
```

```
SaturatingAmplifier = serdes.SaturatingAmplifier('Mode',1,...
          'Limit',limit,'LinearGain',linearGain);
y = SaturatingAmplifier(x);
```

Plot the input and modified waveforms.

```
figure, plot(t,x,t,y)
legend('Input','Output')
title('Clipping Waveform Using Saturating Amplifier');
xlabel('Time (s)');
ylabel('Amplitude (V)');
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

AGC | SaturatingAmplifier | VGA | `serdes.AGC` | `serdes.VGA`

Introduced in R2019a

serdes.VGA

Models a variable gain amplifier

Description

The `serdes.VGA` system object scales the amplitude of the input waveform based on a gain specified by the user.

To scale the input signal:

1. Create the `serdes.VGA` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects? (MATLAB).

Creation

Syntax

```
vga = serdes.VGA
vga = serdes.VGA (Name,Value)
```

Description

`vga = serdes.VGA` returns a VGA object that modifies a input waveform according to the gain defined by the user.

`vga = serdes.VGA (Name,Value)` returns a VGA object with each specified property set to specific value. Unspecified properties have default values.

Example: `vga = serdes.VGA('ACGain',5)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects (MATLAB).

Main

`Mode` — VGA operating mode
`1` (default) | `0`

VGA operating mode, specified as `0` or `1`. `Mode` determines if the VGA adjusts the gain of input signal or acts as a pass-through.

Mode Value	VGA Mode	VGA Operation
`0`	Off	`serdes.VGA` is bypassed, the input waveform remains unchanged.
`1`	On	`serdes.VGA` scales the input waveform according to the specified `Gain`.

Data Types: `double`

`Gain` — Multiplicative gain used to scale the input waveform
`1` (default) | scalar

Multiplicative gain used to scale the input waveform, specified as a unitless scalar.

Data Types: `double`

Advanced

`SymbolTime` — Time of single symbol duration
`100e-12` (default) | real scalar

Time of a single symbol duration, specified as a real scalar in s.

Data Types: `double`

`SampleInterval` — Uniform time step of waveform
`6.25e-12` (default) | real scalar

Uniform time step of the waveform, specified as a real scalar in s.

Data Types: `double`

Usage

Syntax

```
y = vga(x)
```

Description

```
y = vga(x)
```

Input Arguments

x — Input signal
scalar | vector

Input signal to be scaled, specified as a scalar or vector.

Output Arguments

y — Scaled output signal
scalar | vector

Scaled output signal, returned as a scalar or vector corresponding to the input signal.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Scaling Input Waveform using VGA

This example shows how to apply variable gain to input waveform using `serdes.VGA` system object™.

Create the input waveform.

```
t = linspace(0,12,101);
y1 = sin(t);
```

Create the VGA object with a scale factor of 3.

```
vga = serdes.VGA('Gain',3);
```

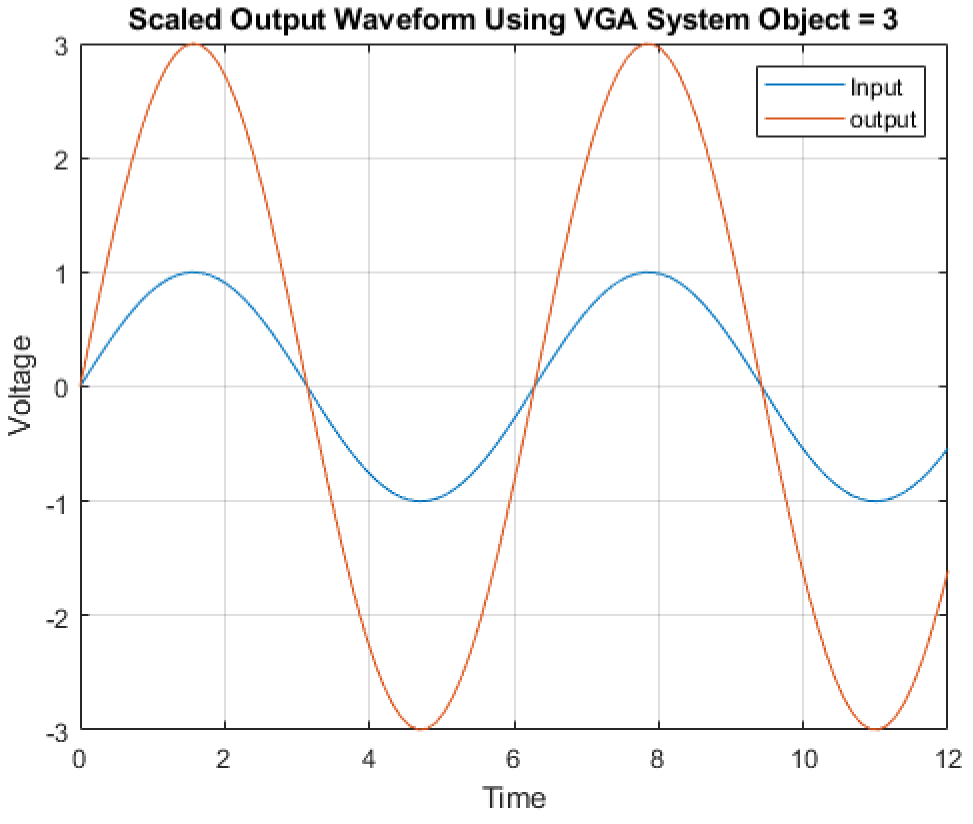
Process the input waveform with the VGA object.

```
y2 = vga(y1);
```

Plot the input and output waveforms.

```
figure
plot(t,y1,t,y2)
xlabel('Time')
ylabel('Voltage')
legend('Input','output')
```

```
grid on
title(sprintf('Scaled Output Waveform Using VGA System Object = %g',vga.Gain))
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

IBIS-AMI codegen is not supported in MAC.

See Also

AGC | VGA | `serdes.AGC`

Introduced in R2019a

Blocks — Alphabetical List

Analog Channel

Construct loss model from channel loss metric or impulse response

Library: SerDes Toolbox / Utilities

Description

The Analog Channel block constructs a loss model using a channel loss metric or an impulse response from another source in a SerDes Toolbox model. Analog model inputs are only used for IBIS file construction when using impulse response.

Ports

Input

`WaveIn` — Input signal
waveform

Input signal, specified as a waveform.

Data Types: `double`

Output

`WaveOut` — Modified output data
waveform

Modified output data that includes the effect of a lossy printed circuit board transmission line model according to the method outlined in the IEEE Standard 802.3bj-2014, normative section 39A.

Data Types: `double`

Parameters

Channel Model

`Channel model` — Source of channel model

`Loss model` (default) | `Impulse response`

Source of channel model.

- Select `Loss model` to model the analog channel from a loss model.
- Select `Impulse response` to model the analog channel from an impulse response.

Programmatic Use

- Use `get_param(gcb,'ChannelType')` to view the current **Channel model**.
- Use `set_param(gcb,'ChannelType',value)` to set a specific **Channel model**.

`Target frequency (Hz)` — Frequency for desired channel loss

`20e9` (default) | real positive scalar

Frequency for the desired channel loss, specified as a real positive scalar in Hz. It corresponds to the Nyquist frequency of the system.

Dependencies

This parameter is only available when `Loss model` is selected as **Channel model**.

Programmatic Use

- Use `get_param(gcb,'TargetFrequency')` to view the current value of **Target frequency**.
- Use `set_param(gcb,'TargetFrequency',value)` to set **Target frequency** to a specific value.

`Loss (dB)` — Channel loss at target frequency

`8` (default) | scalar

Channel loss at the target frequency, specified as a scalar in dB.

Dependencies

This parameter is only available when `Loss model` is selected as **Channel model**.

Programmatic Use

- Use `get_param(gcb,'Loss')` to view the current value of **Loss**.
- Use `set_param(gcb,'Loss',value)` to set **Loss** to a specific value.

`Impedance (Ohms)` — Channel characteristic impedance

real positive scalar

Characteristic impedance of the channel, specified as a real positive scalar in Ohms. **Impedance** depends on the setting of **Signaling** of the **CONFIGURATION** tab in the **SerDes Designer** app or in the Configuration block.

- If **Signaling** is set to `Differential`, the default value of **Impedance** is `100`.
- If **Signaling** is set to `Single-ended`, the default value of **Impedance** is `50`.

Dependencies

This parameter is only available when `Loss model` is selected as **Channel model**.

Programmatic Use

- Use `get_param(gcb,'Zc')` to view the current value of **Impedance**.
- Use `set_param(gcb,'Zc',value)` to set **Impedance** to a specific value.

`Impulse response` — User provided impulse response

`[zeros(1,63),1/SampleInterval,zeros(1,192)]` (default) | vector

User provided impulse response, specified as a unitless vector. **Impulse respnose** is used to construct a channel loss model from the user defined impulse response of the system.

Dependencies

This parameter is only available when `Impulse response` is selected as **Channel model**

Programmatic Use

- Use `get_param(gcb,'ImpulseResponse')` to view the current value of **Impulse response**.
- Use `set_param(gcb,'ImpulseResponse',value)` to set **Impulse response** to a specific value.

Analog Model

`Tx R (Ohms)` — Single-ended impedance of transmitter analog model

`50` (default) | real nonnegative scalar

Single-ended impedance of the transmitter analog model, specified as a real nonnegative scalar in Ohms.

Programmatic Use

- Use `get_param(gcb,'TxR')` to view the current value of **Tx R**.
- Use `set_param(gcb,'TxR',value)` to set **Tx R** to a specific value.

`Tx C (Ohms)` — Capacitance of transmitter analog model

`1e-13` (default) | real nonnegative scalar

Capacitance of the transmitter analog model, specified as a real nonnegative scalar in F.

Programmatic Use

- Use `get_param(gcb,'TxC')` to view the current value of **Tx C**.
- Use `set_param(gcb,'TxC',value)` to set **Tx C** to a specific value.

`Rx R (Ohms)` — Single-ended impedance of receiver analog model

`50` (default) | real nonnegative scalar

Single-ended impedance of the receiver analog model, specified as a real nonnegative scalar in Ohms.

Programmatic Use

- Use `get_param(gcb,'RxR')` to view the current value of **Rx R**.
- Use `set_param(gcb,'RxR',value)` to set **Rx R** to a specific value.

`Rx C (Ohms)` — Capacitance of receiver analog model

`1e-13` (default) | real nonnegative scalar

Capacitance of the receiver analog model, specified as a real nonnegative scalar in F.

Programmatic Use

- Use `get_param(gcb,'RxC')` to view the current value of **Rx C**.

- Use `set_param(gcb,'RxC',value)` to set **Rx C** to a specific value.

`Rise time (s)` — Rise time of stimulus input

`5e-12` (default) | real positive scalar

20%-80% rise time of the stimulus input to transmitter analog model, specified as a real positive scalar in s.

Programmatic Use

- Use `get_param(gcb,'RiseTime')` to view the current value of **Rise time**.
- Use `set_param(gcb,'RiseTime',value)` to set **Rise time** to a specific value.

`Voltage swing (V)` — Peak-to-peak voltage at the input of transmitter analog model

`1` (default) | real positive scalar

Peak-to-peak voltage at the input of transmitter analog model, specified as a real positive scalar in V.

Programmatic Use

- Use `get_param(gcb,'VoltageSwingIdeal')` to view the current value of **Voltage swing**.
- Use `set_param(gcb,'VoltageSwingIdeal',value)` to set **Voltage swing** to a specific value.

See Also

Configuration | Stimulus

Introduced in R2019a

AGC

Automatically adjusts gain to maintain output waveform amplitude
Library: SerDes Toolbox / Datapath Blocks

Description

The AGC block applies an adaptive variable gain to the input waveform to achieve a desired RMS output voltage. Averaging the RMS voltage over a specified number of symbols, AGC performs automatic gain control (AGC) by increasing or decreasing the gain, or keeping the gain constant.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal. The input signal can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Gain adjusted output signal
scalar | vector

Gain adjusted output signal. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The options are **Mode** and **Target RMS voltage**. By default, both parameters are selected.

`Mode` — AGC operating mode
`Off` (default) | `Fixed`

AGC operating mode, specified as:

- `Off` - AGC is bypassed, the input waveform remains unchanged.
- `Fixed` - AGC adjusts gain of input waveform to maintain **Target RMS voltage** in output waveform.

Programmatic Use

- Use `get_param(gcb,'Mode')` to view the current AGC **Mode**.
- Use `set_param(gcb,'Mode',value)` to set AGC to a specific **Mode**.

`Target RMS voltage` — Desired RMS voltage of output waveform
`0.1` (default) | real scalar in the range [1e-3, 10]

Desired RMS voltage of the output waveform, specified as a real scalar in the range [1e-3, 10] in V.

Programmatic Use

- Use `get_param(gcb,'TargetRMSVoltage')` to view the current value of **Target RMS voltage**.
- Use `set_param(gcb,'TargetRMSVoltage',value)` to set **Target RMS voltage** to a specific value.

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The options are **Mode** and **Target RMS voltage**. By default, both parameters are selected.

See Also

`serdes.AGC`

Introduced in R2019a

CDR

Models a clock data recovery circuit
Library: SerDes Toolbox / Datapath Blocks

Description

The CDR block provides clock sampling times and estimates data symbols at the receiver using a Bang-Bang clock and data recovery (CDR) model.

Ports

Input

`WaveIn` — Input baseband signal
scalar

Input baseband signal. The input to the CDR must be applied as one sample at a time and not as a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The options are **Phase offset** and **Reference offset**. By default, both parameters are selected.

`Phase offset` — Clock phase offset
`0` (default) | real scalar in the range [-0.5,0.5]

Clock phase offset, specified as a real scalar in the range [-0.5,0.5] in fraction of symbol time. **Phase offset** manually shifts clock probability distribution function (PDF) for better bit error rate (BER).

Programmatic Use

- Use `get_param(gcb,'PhaseOffset')` to view the current value of **Phase offset**.
- Use `set_param(gcb,'PhaseOffset',value)` to set CDR to a specific **Phase offset**.

`Reference offset` — Reference clock offset impairment

`0` (default) | real scalar in the range [-3e-4,3e-4]

Reference clock offset impairment, specified as a real scalar in the range [-3e-4,3e-4] in parts per million (ppm). **Reference offset** is the deviation between transmitter oscillator frequency and receiver oscillator frequency.

Programmatic Use

- Use `get_param(gcb,'ReferenceOffset')` to view the current value of **Reference offset**.
- Use `set_param(gcb,'ReferenceOffset',value)` to set CDR to a specific **Reference offset**.

`Early/late count threshold` — Early or late CDR count threshold to trigger phase update

`16` (default) | real positive integer >4

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer >4. Increasing the value of **Early/late count threshold** provides a more stable output clock phase at the expense of convergence speed. Because the bit decisions are made at the clock phase output, a more stable clock phase has a better bit error rate (BER).

Programmatic Use

- Use `get_param(gcb,'Count')` to view the current value of **Early/late count threshold**.
- Use `set_param(gcb,'Count',value)` to set CDR to a specific **Early/late count threshold**.

`Sensitivity` — Sampling latch meta-stability voltage

`0` (default) | real scalar

Sampling latch meta-stability voltage, specified as a real scalar in V. If the data sample voltage lies within the region (+/-**Sensitivity**), there is a 50% probability of bit error..

Programmatic Use

- Use `get_param(gcb,'Sensitivity')` to view the current value of **Sensitivity**.
- Use `set_param(gcb,'Sensitivity',value)` to set CDR to a specific **Sensitivity**.

See Also

Introduced in R2019a

Configuration

Configure system wide settings in a SerDes system model
Library: SerDes Toolbox / Utilities

Description

The Configuration block sets the system wide settings of a SerDes system such as symbol time, samples per symbol, target bit error rate (BER), modulation scheme,and signaling type. It also configures the IBIS and AMI models generation and customizes the AMI parameters.

Parameters

`Symbol time (s)` — Time of single symbol duration

`100e-12` (default) | real positive scalar

Time of a single symbol duration, specified as a real positive scalar in s.

Programmatic Use

- Use `get_param(gcb,'SymbolTime')` to view the current value of **Symbol time**.
- Use `set_param(gcb,'SymbolTime',value)` to set **Symbol time** to a specific value.

`Samples per symbol` — Data points per symbol

`16` (default) | `8` | `16` | `32` | `64` | `128`

Number of data points per symbol.

Programmatic Use

- Use `get_param(gcb,'SamplesPerSymbol')` to view the current value of **Samples per symbol**.
- Use `set_param(gcb,'SamplesPerSymbol',value)` to set **Samples per symbol** to a specific value.

`Sample interval (s)` — Uniform time step of waveform

`6.25e-12` (default) | real positive scalar

Uniform time step of the waveform, specified as a real positive scalar in s. This parameter is nontunable and is derived from **Symbol time** and **Samples per symbol**.

Programmatic Use

- Use `get_param(gcb,'SampleIntervalText')` to view the current value of **Sample interval**.

`Target BER` — Target bit error rate

`1e-6` (default) | real positive scalar

Target bit error rate used to generate eye-contours, specified as a unitless real positive scalar.

Programmatic Use

- Use `get_param(gcb,'TargetBER')` to view the current value of **Target BER**.
- Use `set_param(gcb,'TargetBER',value)` to set **Target BER** to a specific value.

`Modulation` — Modulation scheme

`NRZ` (default) | `PAM4`

Number of logic levels in modulation scheme.

- Select `NRZ` if modulation scheme has two logic levels.
- Select `PAM4` if modulation scheme has four logic levels.

Programmatic Use

- Use `get_param(gcb,'Modulation')` to view the current value of **Modulation**.
- Use `set_param(gcb,'Modulation',value)` to set **Modulation** to a specific value.

`Signaling` — Determine how signal is transmitted through wires

`Differential` (default) | `Single-ended`

Determine how the incoming signal is transmitted through wires.

- Select `Differential` to transmit the incoming signal using a differential pair of signals. The receiver responds to the difference between the two signals.

- Select `Single-ended` to transmit the incoming signal using a varying voltage. The receiver responds to the difference between the incoming signal and a reference or ground.

Signaling only effects the generated IBIS files. Voltage levels in Simulink does not change with changing the signaling type. **Signaling** also effects the **Impedance** of Analog Channel when the `Loss model` is selected as **Channel model**.

Programmatic Use

- Use `get_param(gcb,'Signaling')` to view the current value of **Signaling**.
- Use `set_param(gcb,'Signaling',value)` to set **Signaling** to a specific value.

`Plot statistical analysis after simulation` — Plot statistical analysis after simulation

button

Select to plot the statistical analysis (Init) results after the simulation is run. By default, this option is selected.

`Open SerDes IBIS-AMI Manager` — Open SerDes IBIS-AMI Manager

button

Click to open the SerDes IBIS-AMI Manager dialog box. Using this dialog box, you can set the IBIS and AMI file contents and export the IBIS-AMI model.

Set the IBIS and AMI model settings (model name, model type, corner percentage, bits to ignore) for the transmitter and receiver and specify file creation options in the **Export** tab of the SerDes IBIS-AMI Manager dialog box.

The **IBIS** tab of the SerDes IBIS-AMI Manager dialog box contains the .ibs file contents.

You can add customized AMI parameters using the **AMI-Tx** and **AMI-Rx** tabs. For more information, see “Customizing SerDes Toolbox Datapath Control Signals”..

See Also

Analog Channel | Stimulus

Topics

“Customizing SerDes Toolbox Datapath Control Signals”

Introduced in R2019a

CTLE

Models continuous time linear equalizer (CTLE)
Library: SerDes Toolbox / Datapath Blocks

Description

The CTLE block applies a linear peaking filter to equalize the frequency response of a sample-by-sample input signal. The equalization process reduces distortions resulting from lossy channels.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal. The input signal can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Equalized CTLE output
scalar | vector

Equalized CTLE output waveform. If the input signal is a sample-by-sample signal specified as a scalar, then the output is also scalar. If the input signal is an impulse response vector signal, then the output is also a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Parameters included in IBIS-AMI model
`Mode` | `Config select`

Choose parameters to be included in IBIS-AMI models. The options are “Mode” on page 2-0 and “Configuration select” on page 2-0. By default, both parameters are selected. Deselecting a parameter removes the said parameter from the AMI parameter file, hard-coding the current value to the IBIS-AMI model.

`Mode` — CTLE operating mode
`Off` (default) | `Fixed` | `Adapt`

CTLE operating mode:

- `Off` — CTLE is bypassed and the input waveform remains unchanged.
- `Fixed` — CTLE applies the CTLE transfer function as specified by **Configuration select** to the input waveform.
- `Adapt` — If the input signal is an impulse response vector or a pseudorandom binary sequence (PRBS), then the CTLE determines the CTLE transfer function for the best eye height opening and applies the transfer function to the input waveform.

  If the input signal is a sample-by-sample scalar, then the CTLE operates in the `Fixed` mode.

Programmatic Use

- Use `get_param(gcb,'Mode')` to view the current CTLE **Mode**.
- Use `set_param(gcb,'Mode',value)` to set CTLE to a specific **Mode**.

Data Types: `char`

`Configuration select` — Select which member of transfer function to apply in fixed mode
`1` (default) | `0` | `2` | `3` | `4` | `5` | `6` | `7` | `8`

Select which transfer function configuration to apply in fixed mode, specified as a real integer scalar. Depending on the **Specification**, **Configuration select** specifies which gain coefficient is applied to the filter transfer function.

For example, setting **Configuration select** to *n* and **Specification** to `'DC Gain and Peaking Gain'` selects the (*n*+1)-th element in the **DC gain** and **Peaking gain** vectors to be applied to the filter transfer function.

Programmatic Use

- Use `get_param(gcb,'ConfigSelect')` to view the current value of **Configuration Select**.
- Use `set_param(gcb,'ConfigSelect',value)` to set **Configuration Select** to a specific value.

Data Types: `double`

Specification — Input specification for CTLE response

`'DC Gain and Peaking Gain'` (default) | `'DC Gain and AC Gain'` | `'AC Gain and Peaking Gain'` | `'GPZ Matrix'`

Defines which inputs will be used for the CTLE transfer function family:

- `'DC Gain and Peaking Gain'` — CTLE response is specified from **DC gain**, **Peaking gain**, and **Peaking frequency**.
- `'DC Gain and AC Gain'` — CTLE response is specified from **DC gain**, **AC gain**, and **Peaking frequency**.
- `'AC Gain and Peaking Gain'` — CTLE response is specified from **AC gain**, **Peaking gain**, and **Peaking frequency**.
- `'GPZ Matrix'` — CTLE response is specified from **Gain pole zero matrix**.

Programmatic Use

- Use `get_param(gcb,'Specification')` to view the current CTLE **Specification**.
- Use `set_param(gcb,'Specification',value)` to set CTLE to a specific **Specification**.

Data Types: `char`

DC gain (dB) — Gain at zero frequency

`[0:-1:-8]` (default) | scalar | vector

Gain at zero frequency for the CTLE transfer function, specified as a scalar or a vector in dB. If specified as a vector, the vector length must be the same as the vectors in **Peaking gain**, **AC gain**, and **Peaking gain**.

Dependencies

This parameter is only available when **Specification** is set to `'DC Gain and Peaking Gain'` or `'DC Gain and AC Gain'`.

Programmatic Use

- Use `get_param(gcb,'DCGain')` to view the current value of **DC gain**.
- Use `set_param(gcb,'DCGain',value)` to set **DC gain** to a specific value.

Data Types: `double`

`Peaking gain (dB)` — Difference between AC and DC gain

`[0:8]` (default) | scalar | vector

Peaking gain, specified as a scalar or vector in dB. **Peaking gain** is the difference between **AC gain** and **DC gain** for the CTLE transfer function. If specified as a vector, the vector length must be the same as the vectors in **DC gain**, **AC gain**, and **Peaking frequency**.

Dependencies

This parameter is only available when **Specification** is set to `'DC Gain and Peaking Gain'` or `'AC Gain and Peaking Gain'`.

Programmatic Use

- Use `get_param(gcb,'PeakingGain')` to view the current value of **Peaking gain**.
- Use `set_param(gcb,'PeakingGain',value)` to set **Peaking gain** to a specific value.

Data Types: `double`

`AC gain (dB)` — Gain at peaking frequency

`0` (default) | scalar | vector

Gain at the peaking frequency for the CTLE transfer function, specified as a scalar or vector in dB. If specified as a vector, the vector length be the same as the vectors in **DC gain**, **Peaking gain**, and **Peaking frequency**.

Dependencies

This parameter is only available when **Specification** is set to `'DC Gain and AC Gain'` or `'AC Gain and Peaking Gain'`.

Programmatic Use

- Use `get_param(gcb,'ACGain')` to view the current value of **AC gain**.
- Use `set_param(gcb,'ACGain',value)` to set **AC gain** to a specific value.

Data Types: `double`

`Peaking frequency (Hz)` — Approximate frequency at which CTLE transfer function peaks

`14e9` (default) | scalar | vector

Approximate frequency at which CTLE transfer function peaks in magnitude, specified as a scalar or a vector in GHz. If specified as a vector, the vector length must be the same as the vectors in **DC gain**, **AC gain**, and **Peaking gain**.

Dependencies

This parameter is not available when **Specification** is set to `'GPZ Matrix'` .

Programmatic Use

- Use `get_param(gcb,'PeakingFrequency')` to view the current value of **Peaking frequency**.
- Use `set_param(gcb,'PeakingFrequency',value)` to set **Peaking frequency** to a specific value.

Data Types: `double`

`Gain pole zero matrix` — Gain pole zero

matrix

Gain pole zero, specified as a matrix. **Gain pole zero matrix** explicitly defines the family of CTLE transfer functions by specifying the **DC gain** (dB) in column 1 and then poles and zeros in alternating columns. The poles and zeros are specified in Hz.

No repeated poles or zeros are allowed. Complex poles or zeros must have conjugates. The number of poles must be greater than number of zeros for system stability.

Example: To create a gain pole zero matrix with three poles and two zeroes, input the matrix as follows: [G, P1, Z1, P2, Z2, P3].

Dependencies

This parameter is only available when **Specification** is set to `'GPZ Matrix'`.

Programmatic Use

- Use `get_param(gcb,'GPZ')` to view the current value of **Gain pole zero**.
- Use `set_param(gcb,'GPZ',value)` to set **Gain pole zero** to a specific value.

Data Types: `double`

See Also

AGC | DFECDR | SaturatingAmplifier | `serdes.AGC` | `serdes.CTLE` | serdes.DFECDR

Introduced in R2019a

DFECDR

Decision feedback equalizer (DFE) with clock and data recovery (CDR)
Library: SerDes Toolbox / Datapath Blocks

Description

The DFECDR block adaptively processes a sample-by-sample input signal or analytically processes an impulse response vector input signal to remove distortions at post-cursor taps.

The decision feedback equalizer modifies baseband signals to minimize the intersymbol interference (ISI) at the clock sampling time. The DFE samples data at each clock tick and adjusts the amplitude of the waveform by a correction voltage. The correction voltage is determined by the previous *N* sampled unit interval (UI) values, where *N* is the number of DFE taps.

A clock and data recovery function provides the clock sampling location to the DFE.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal. The input signal can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Estimated channel output
scalar | vector

Estimated channel output. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The options are **Mode**, **Tap weights**, **Phase offset** and **Reference offset**. By default, all four parameters are selected.

DFE

`Mode` — DFE operating mode
`Off` (default) | `Fixed` | `Adapt`

DFE operating mode, specified as:

- `Off` - DFECDR is bypassed, the input waveform remains unchanged.
- `Fixed` - DFECDR applies input DFE tap weights specified in **Tap weights** to input waveform.
- `Adapt` - DFECDR adaptively determines the optimum DFE tap weights values and apply to input waveform.

Programmatic Use

- Use `get_param(gcb,'Mode')` to view the current DFECDR **Mode**.
- Use `set_param(gcb,'Mode',value)` to set DFECDR to a specific **Mode**.

`Tap weights` — Initial DFE tap weights
`[0 0 0 0]` (default) | row vector

Initial DFE tap weights, specified as a row vector in V. The length of the vector specifies the number of taps. The vector element value specifies the strength of the tap at that element position. Setting a vector element value to zero only initializes the tap.

Programmatic Use

- Use `get_param(gcb,'TapWeights')` to view the current DFECDR **Tap weights**.
- Use `set_param(gcb,'TapWeights',value)` to set DFECDR to a specific **Tap weights** vector.

`Adaptive gain` — Controls DFE tap weight update rate

`3e-06` (default) | scalar

Controls DFE tap weight update rate, specified as a unitless scalar. Increasing the value of **Adaptive gain** leads to a faster convergence of DFE adaptation at the expense of more noise in DFE tap values.

Programmatic Use

- Use `get_param(gcb,'EqualizationGain')` to view the current DFECDR **Adaptive gain** value.
- Use `set_param(gcb,'EqualizationGain',value)` to set DFECDR to a specific value of **Adaptive gain**.

`Adaptive step size` — DFE adaptive step resolution

`1e-6` (default) | real scalar

DFE adaptive step resolution, specified as a real scalar in V. **Adaptive step size** specifies the minimum DFE tap change from one time step to the next to mimic hardware impairment. Setting **Adaptive step size** to `0` yields DFE tap values without any resolution limitation.

Programmatic Use

- Use `get_param(gcb,'EqualizationStep')` to view the current DFECDR **Adaptive step size** value.
- Use `set_param(gcb,'EqualizationStep',value)` to set DFECDR to a specific value of **Adaptive step size**.

CDR

`Phase offset (UI)` — Clock phase offset

`0` (default) | scalar in the range [-0.5,0.5]

Clock phase offset, specified as a real scalar in the range [-0.5,0.5] in fraction of symbol time. **Phase offset** is used to manually shift clock probability distribution function (PDF) for better BER.

Programmatic Use

- Use `get_param(gcb,'PhaseOffset')` to view the current DFECDR **Phase offset** value.
- Use `set_param(gcb,'PhaseOffset',value)` to set DFECDR to a specific value of **Phase offset**.

`Reference offset (ppm)` — Reference clock offset impairment

`0` (default) | real scalar in the range [-3e-4,3e-4]

Reference clock offset impairment, specified as a real scalar in the range [-3e-4,3e-4] in parts per million (ppm). **Reference offset** is the deviation between transmitter oscillator frequency and receiver oscillator frequency.

Programmatic Use

- Use `get_param(gcb,'ReferenceOffset')` to view the current DFECDR **Reference offset** value.
- Use `set_param(gcb,'ReferenceOffset',value)` to set DFECDR to a specific value of **Reference offset**.

`Early/late count threshold` — Early or late CDR count threshold to trigger phase update

`16` (default) | real positive integer >4

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer >4. Increasing the value of `Count` provides a more stable output clock phase at the expense of convergence speed. Because the bit decisions are made at the clock phase output, a more stable clock phase has a better bit error rate (BER).

Programmatic Use

- Use `get_param(gcb,'Count')` to view the current DFECDR **Early/late count threshold** value.

- Use `set_param(gcb,'Count',value)` to set DFECDR to a specific value of **Early/late count threshold**.

`Step (UI)` — Clock phase resolution

`0.0078` (default) | real scalar

Clock phase resolution, specified as a real scalar in fraction of unit interval (UI). **Step** is the inverse of the number of phase adjustments in the CDR.

Programmatic Use

- Use `get_param(gcb,'ClockStep')` to view the current DFECDR **Step** value.
- Use `set_param(gcb,'ClockStep',value)` to set DFECDR to a specific value of **Step**.

`Sensitivity (V)` — Sampling latch meta-stability voltage

`0` (default) | real scalar

Sampling latch meta-stability voltage, specified as a real scalar in V. If the data sample voltage lies within the region (+/-**Sensitivity**), there is a 50% probability of bit error.

Programmatic Use

- Use `get_param(gcb,'Sensitivity')` to view the current DFECDR **Sensitivity** value.
- Use `set_param(gcb,'Sensitivity',value)` to set DFECDR to a specific value of **Sensitivity**.

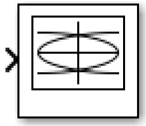
See Also

`serdes.DFECDR`

Introduced in R2019a

Eye Diagram Scope

Display eye diagram of time-domain signal



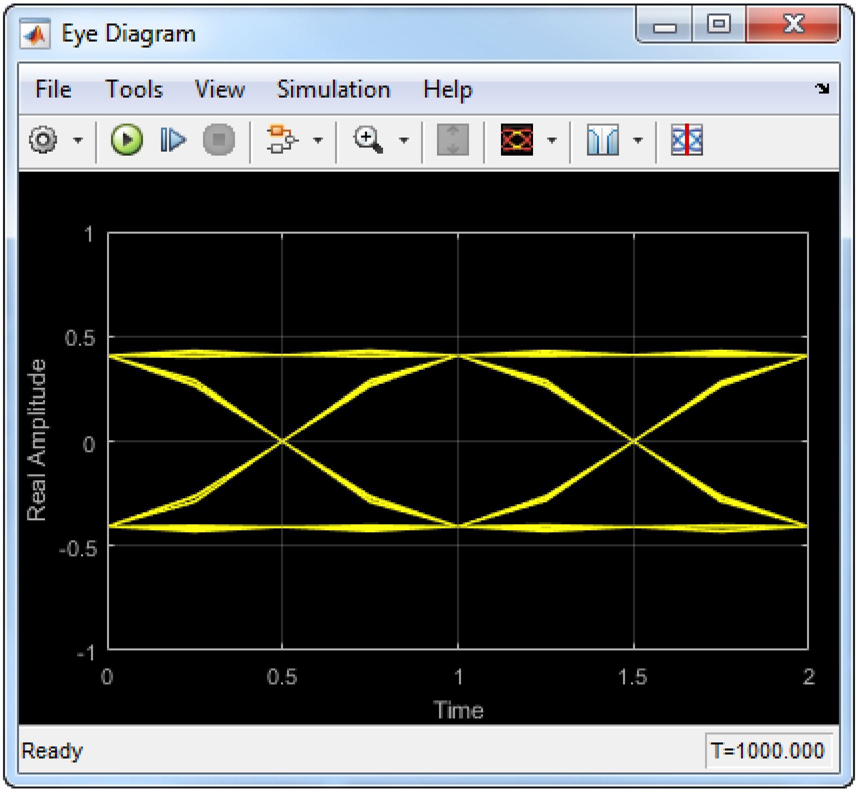
Library

Comm Sinks

Description

The Eye Diagram block displays multiple traces of a modulated signal to produce an eye diagram. You can use the block to reveal the modulation characteristics of the signal, such as the effects of pulse shaping or channel distortions.

The Eye Diagram block has one input port. This block accepts a column vector or scalar input signal. The block accepts a signal with the following data types: double, single, base integer, and fixed point. All data types are cast as double before the block displays results.

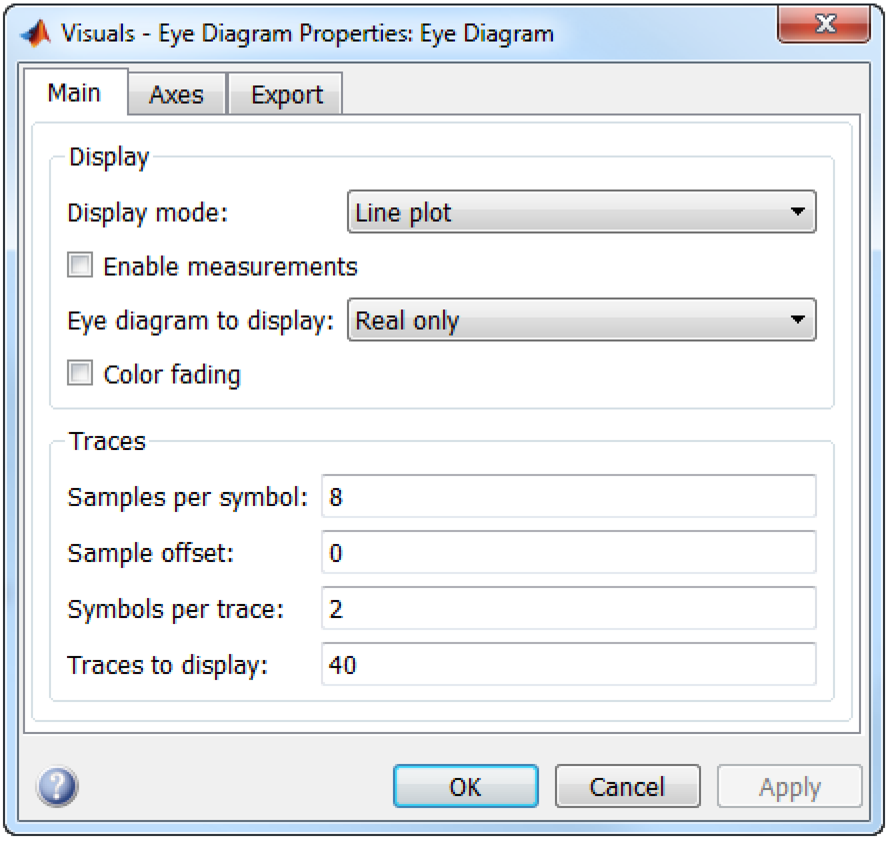


Dialog Box

To modify the eye diagram display, select **View > Configuration Properties** or click the **Configuration Properties** button (). Then select the **Main**, **2D color histogram**, **Axes**, or **Export** tabs and modify the settings.

Visuals — Eye Diagram Properties

Main Tab



Display mode

Display mode of the eye diagram, specified as `Line plot` or `2D color histogram`. Selecting `2D color histogram` makes the histogram tab available. This parameter is tunable.

Enable measurements

Select this check box to enable eye measurements of the input signal.

Show horizontal (jitter) histogram

Select this radio button to display the jitter histogram. This parameter is available when **Display mode** is `2D color histogram` and **Enable measurements** is selected. This can also be accessed by using the histogram button drop down on the toolbar.

Show vertical (noise) histogram

Select this radio button to display the noise histogram. This parameter is available when **Display mode** is `2D color histogram` and **Enable measurements** is selected. This can also be accessed by using the histogram button drop down on the toolbar.

Show horizontal bathtub curve

Select this check box to display the horizontal bathtub curve. This parameter is available when **Enable measurements** is selected. This can also be accessed by using the bathtub curve button on the toolbar.

Show vertical bathtub curve

Select this check box to display the vertical bathtub curve. This parameter is available when **Enable measurements** is selected. This can also be accessed by using the bathtub curve button on the toolbar.

Eye diagram to display

Select either `Real only` or `Real and imaginary` to display one or both eye diagrams. To make eye measurements, this parameter must be `Real only`. This parameter is tunable.

Color fading

Select this check box to fade the points in the display as the interval of time after they are first plotted increases. The default value is `false`. This parameter is available only when the **Display mode** is `Line plot`. This property is tunable.

Samples per symbol

Number of samples per symbol. Use with **Symbols per trace** to determine the number of samples per trace. This parameter is tunable.

Sample offset

Sample offset, specified as a nonnegative integer smaller than the product of **Samples per symbol** and **Symbols per trace**. The offset provides the number of samples to omit before plotting the first point. This parameter is tunable.

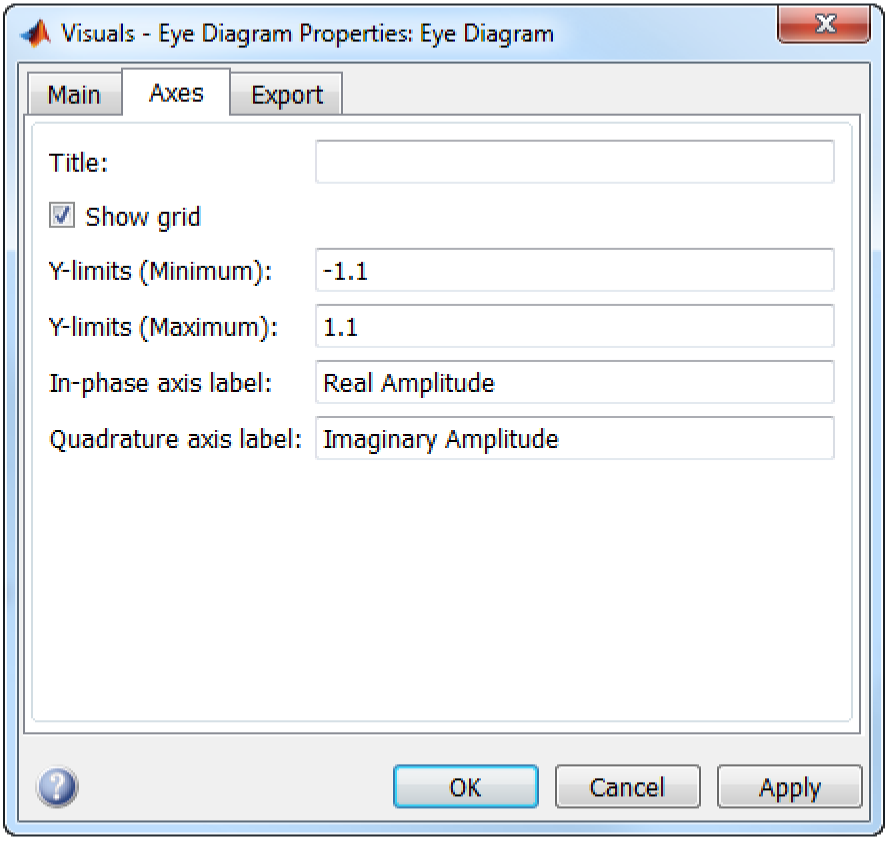
Symbols per trace

Number of symbols plotted per trace, specified as a positive integer. This parameter is tunable.

Traces to display

Number of traces plotted. This parameter is available only when the **Display mode** is `Line plot`. This parameter is tunable.

Axes Tab



Title

Label that appears above the eye diagram plot. By default, the plot has no title. This parameter is tunable.

Show grid

Toggle this check box to turn the grid on and off. This parameter is tunable.

Y-limits (Minimum)

Minimum value of the *y*-axis. This parameter is tunable.

Y-limits (Maximum)

Maximum value of the *y*-axis. This parameter is tunable.

Real axis label

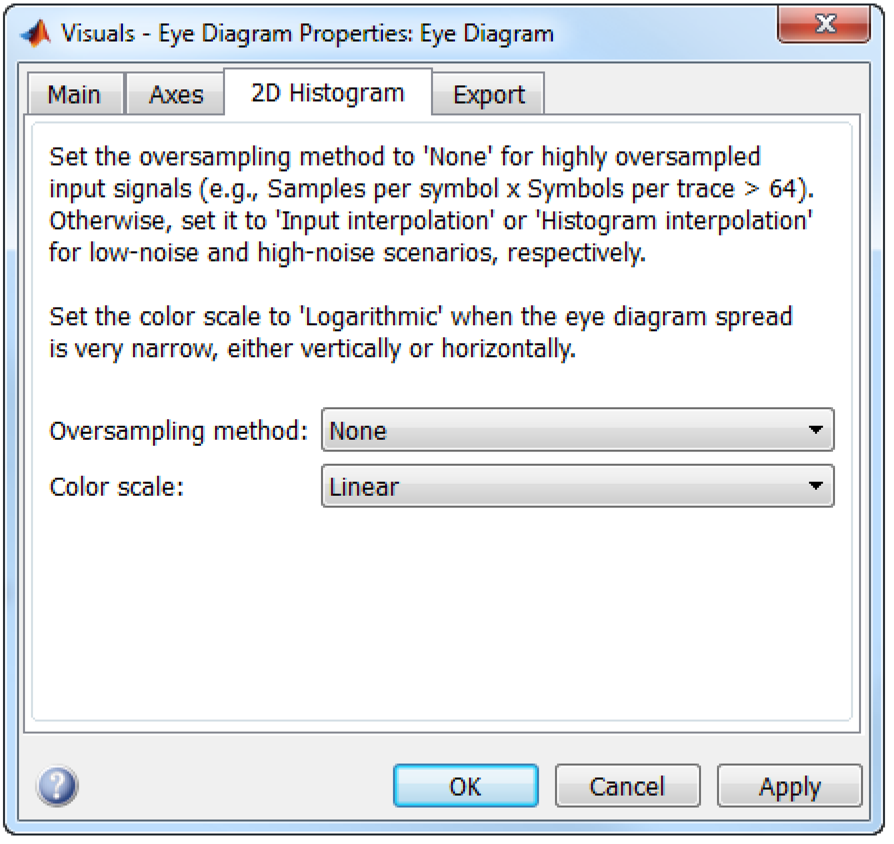
Text that the scope displays along the real axis. This parameter is tunable.

Imaginary axis label

Text that the scope displays along the imaginary axis. This parameter is tunable.

2D Histogram Tab

The 2D histogram tab is available when you click the histogram button or when the display mode is set to `2D color histogram`.



Oversampling method

Oversampling method, specified as `None`, `Input interpolation`, or `Histogram interpolation`. This parameter is tunable.

To plot eye diagrams as quickly as possible, set the **Oversampling method** to `None`. The drawback to not oversampling is that the plots look pixelated when the number of samples per trace is small. To create smoother, less-pixelated plots using a small number of samples per trace, set the **Oversampling method** to `Input interpolation` or `Histogram interpolation`. `Input interpolation` is the faster of the two interpolation methods and produces good results when the signal-to-noise ratio (SNR) is high. With a lower SNR, this oversampling method is not recommended because it introduces a bias to the centers of the histogram ranges. `Histogram interpolation` is not as fast as the other techniques, but it provides good results even when the SNR is low.

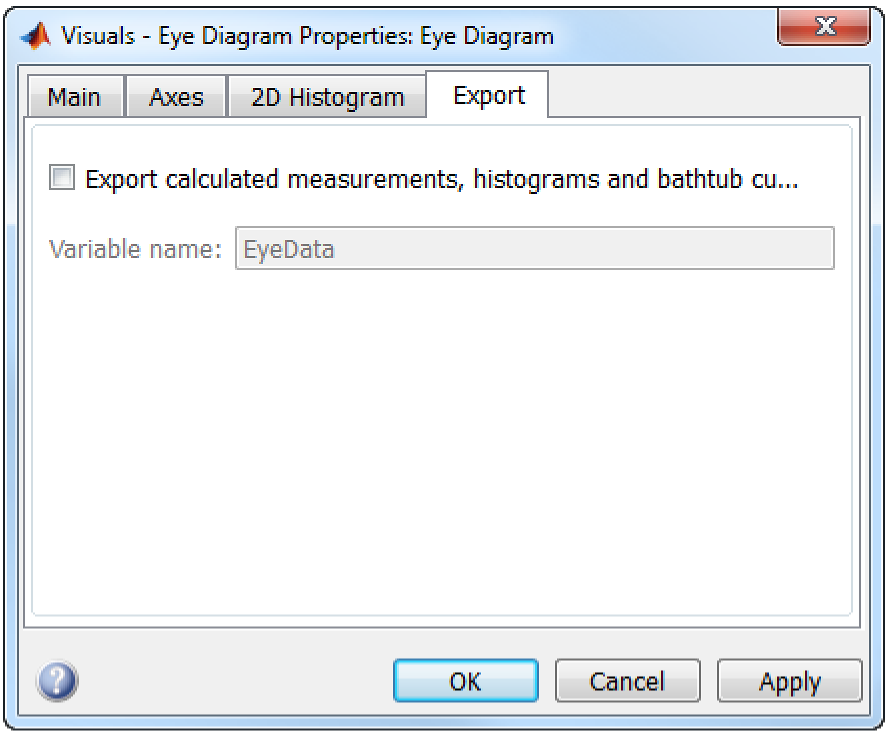
Color scale

Color scale of the histogram plot, specified as either `Linear` or `Logarithmic`. Set **Color scale** to `Logarithmic` if certain areas of the eye diagram include a disproportionate number of points. This parameter is tunable.

Reset

The toolbar contains a histogram reset button , which resets the internal histogram buffers and clears the display. This button is not available when the display mode is set to `Line plot`.

Export Tab



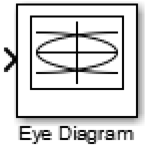
Export measurements

Select this check box export the eye diagram measurements to the MATLAB® workspace. This parameter is tunable.

Variable name

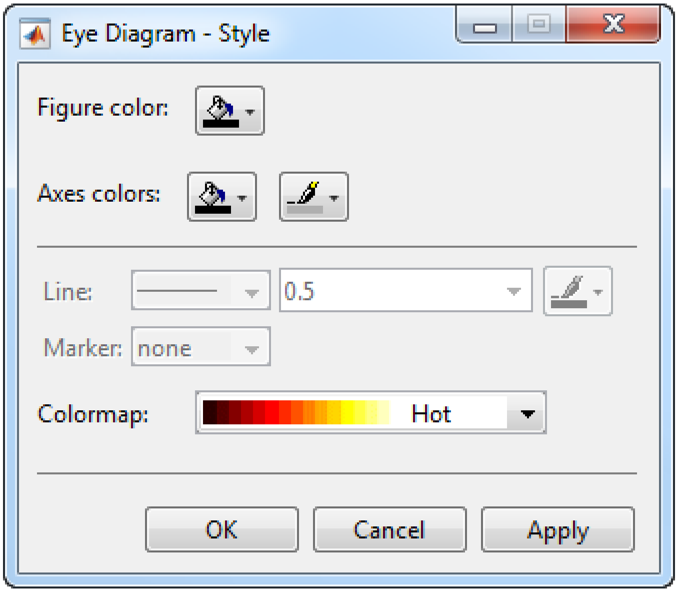
Specify the name of the variable to which the eye diagram measurements are saved. The default is `EyeData`. This parameter is tunable. The data is saved as a structure having these fields:

- `MeasurementSettings`
- `Measurements`
- `JitterHistogram`
- `NoiseHistogram`
- `HorizontalBathtub`
- `VerticalBathtub`
- `BlockName`



Style Dialog Box

In the **Style** dialog box, you can customize the style of the active display. You can change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display. To open this dialog box, select **View > Style**.



Properties

Figure color

Specify the background color of the scope figure. By default, the figure color is black.

Axes colors

Specify the fill and line colors for the axes.

Line

Specify the line style, line width, and line color for the displayed signal.

Marker

Specify data point markers for the selected signal. This parameter is similar to the `Marker` property for MATLAB Handle Graphics® plot objects.

Specifier	Marker Type
none	No marker (default)
○	Circle
□	Square
×	Cross
•	Point
+	Plus sign
✳	Asterisk
◇	Diamond
▽	Downward-pointing triangle
△	Upward-pointing triangle
◁	Left-pointing triangle
▷	Right-pointing triangle
☆	Five-pointed star (pentagram)
✡	Six-pointed star (hexagram)

Colormap

Specify the colormap of the histogram plots as one of these schemes: `Parula`, `Jet`, `HSV`, `Hot`, `Cool`, `Spring`, `Summer`, `Autumn`, `Winter`, `Gray`, `Bone`, `Copper`, `Pink`, `Lines`, or `Custom`. This parameter is active when the Eye Diagram is in `Histogram` mode. The default is `Hot`. If you select `Custom`, a dialog box pops up from which you can enter code to specify your own colormap.

Measurements

To open the measurements panel, click on the **Eye Measurements** button or select Tools > Measurements > Eye Measurements from the toolbar menu.

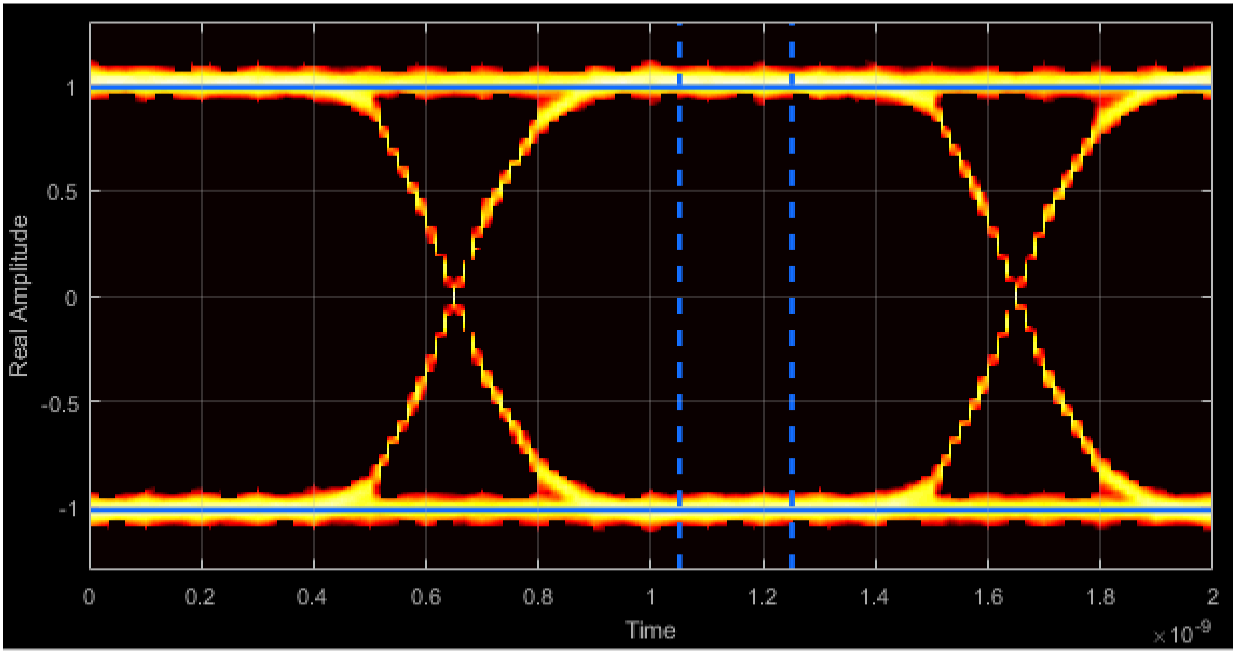
Note

- For amplitude measurements, at least one bin per vertical histogram must reach 10 hits before the measurement is taken, ensuring higher accuracy.

- For time measurements, at least one bin per horizontal histogram must reach 10 hits before the measurement is taken.
- When an eye crossing time measurement falls within the [-0.5/Fs, 0) seconds interval, the time measurement wraps to the end of the eye diagram, i.e., the measurement wraps by 2*Ts seconds (where Ts is the symbol time). For a complex signal case, the analyze method issues a warning if the crossing time measurement of the in-phase branch wraps while that of the quadrature branch does not (or vice versa). To avoid the time-wrapping or a warning, add a half-symbol duration delay to the current value in the MeasurementDelay property of the eye diagram object. This additional delay repositions the eye in the approximate center of the scope.

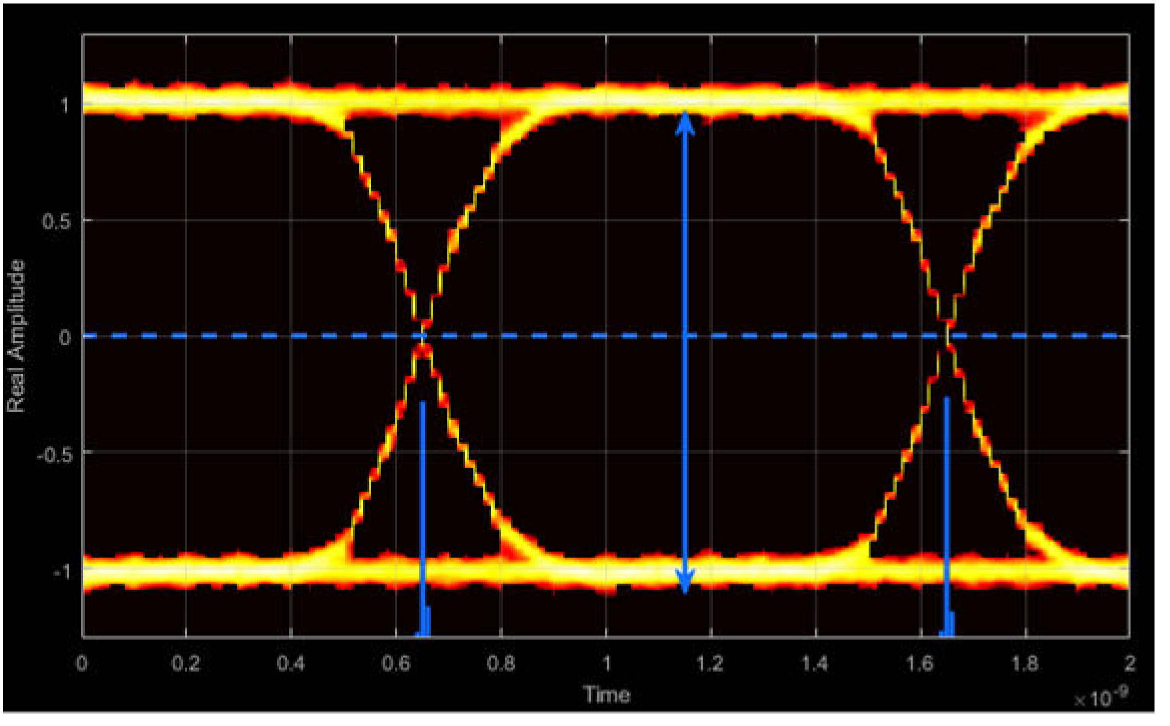
Eye Levels — Amplitude level used to represent data bits

Eye level is the amplitude level used to represent data bits. For the displayed NRZ signal, the levels are –1 V and +1 V. The eye levels are calculated by averaging the 2-D histogram within the eye level boundaries.



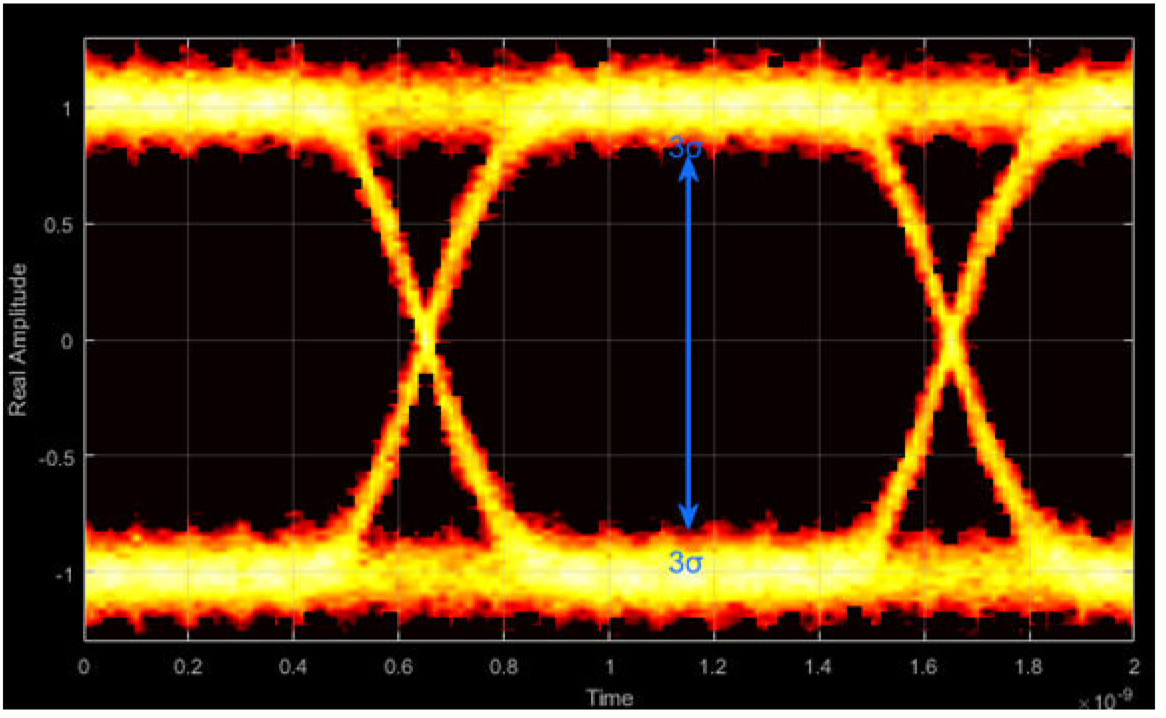
`Eye Amplitude` — Distance between eye levels

Eye amplitude is the distance in V between the mean value of two eye levels.



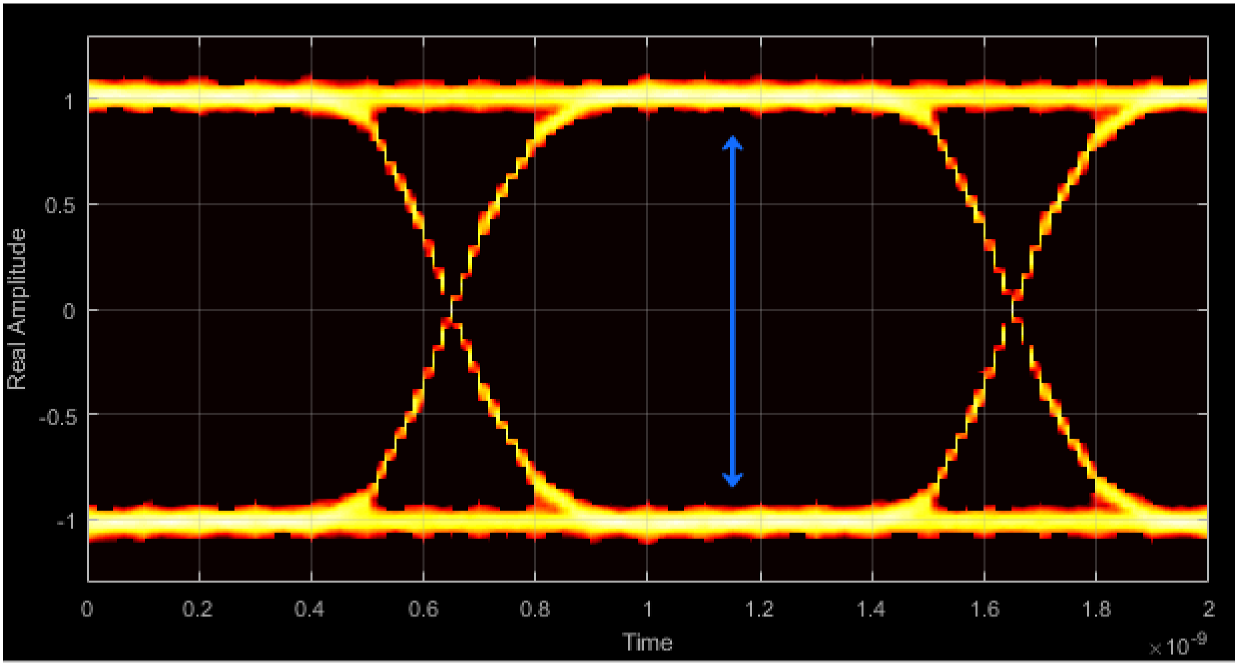
`Eye Height` — Statistical minimum distance between eye levels

Eye height is the distance between μ – 3σ of the upper eye level and μ + 3σ of the lower eye level. μ is the mean of the eye level and σ is the standard deviation.



`Vertical Opening` — Distance between BER threshold points

The vertical opening is the distance between the two points that correspond to the BER threshold. For example, for a BER threshold of 10^{-12}, these points correspond to the 7σ distance from each eye level.



Eye SNR — Signal-to-noise ratio

The eye SNR is the ratio of the eye level difference to the difference of the vertical standard deviations corresponding to each eye level:

$$\mathrm{SNR} = \frac{L_1 - L_0}{\sigma_1 - \sigma_0},$$

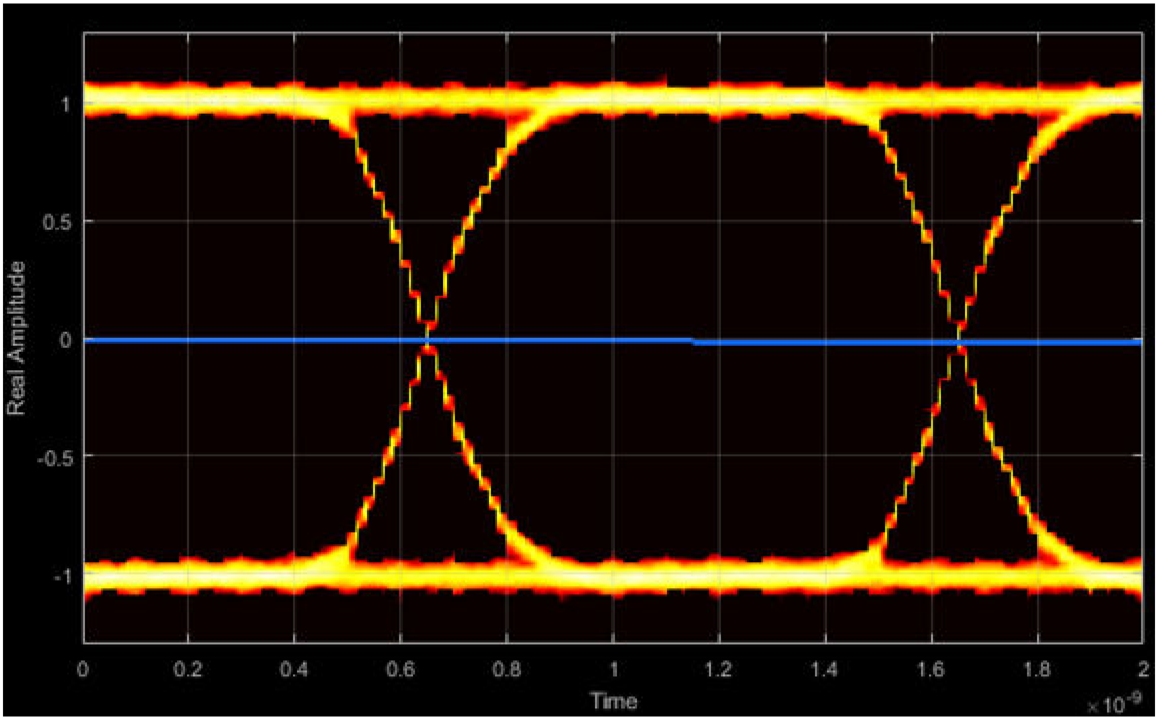
where L_1 and L_0 represent the means of the upper and lower eye levels and σ_1 and σ_0 represent their standard deviations.

Q Factor — Quality factor

The Q factor is calculated using the same formula as the Eye SNR. However, the standard deviations of the vertical histograms are replaced with those computed with the dual-Dirac analysis.

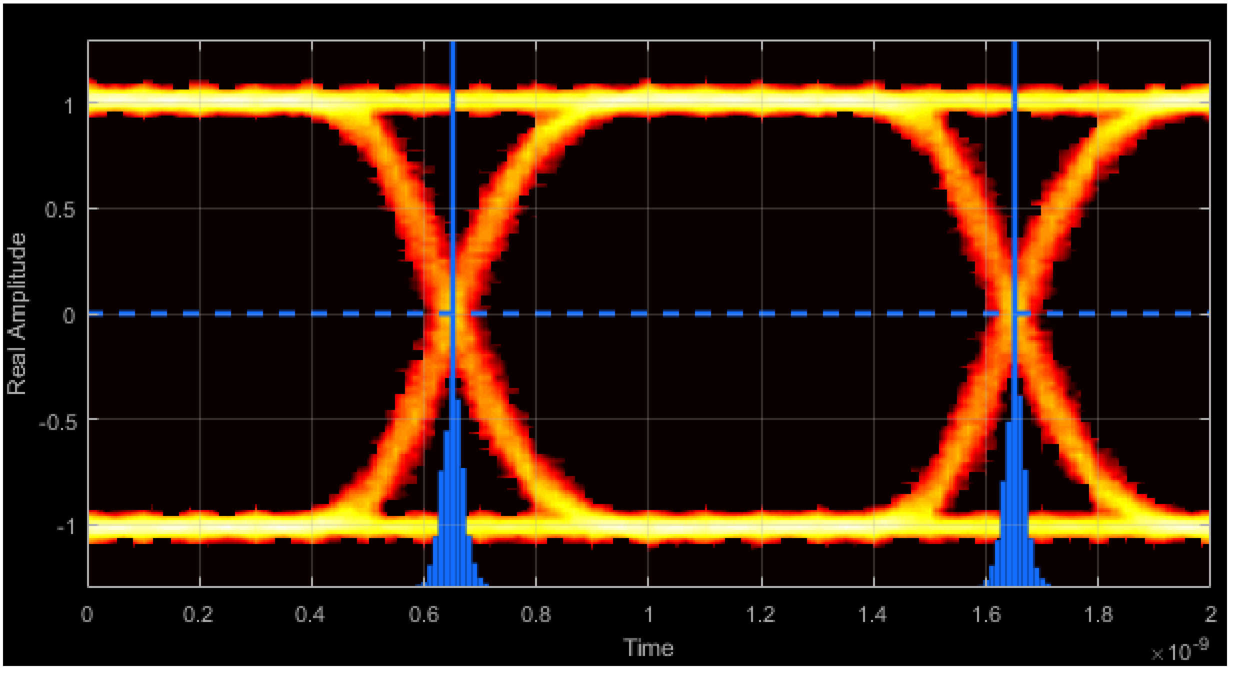
Crossing Levels — Amplitude levels for eye crossings

The crossing levels are the amplitude levels at which the eye crossings occur.



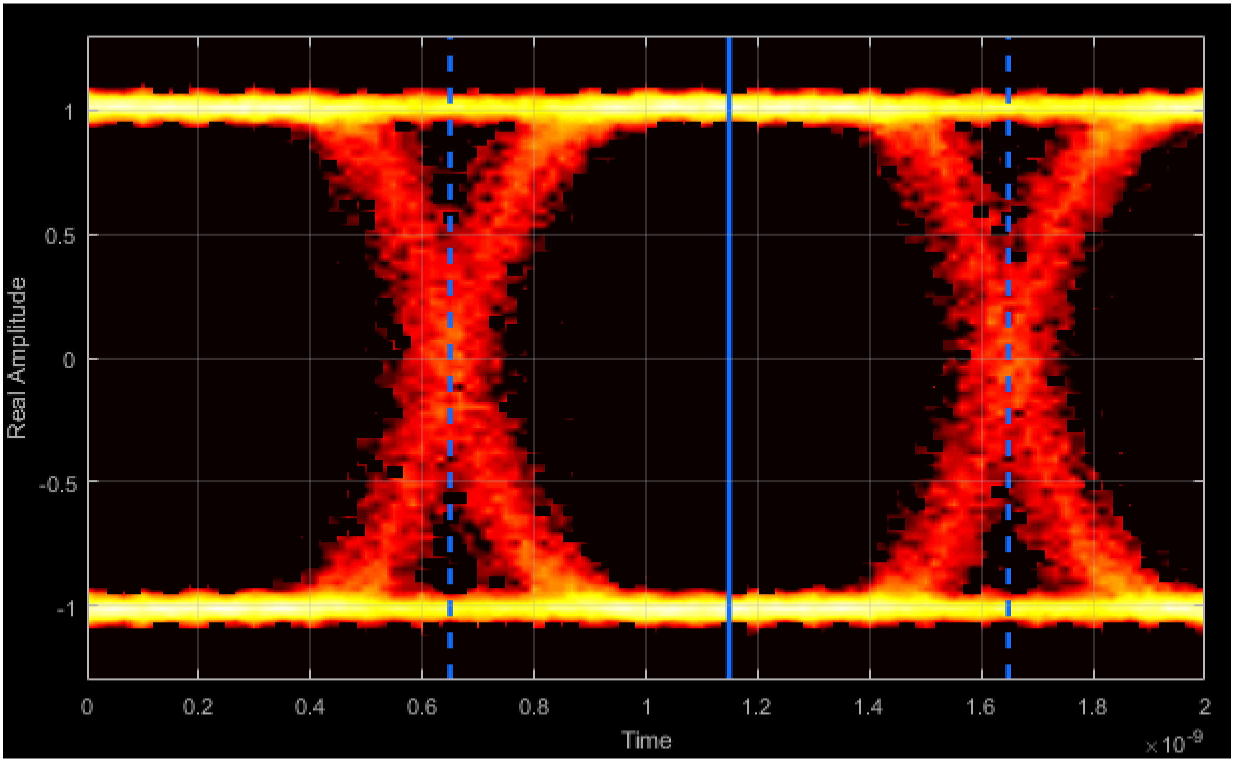
`Crossing Times` — Times for which crossings occur

The crossing times are the times at which the crossings occur. The times are computed as the mean values of the horizontal (jitter) histograms.



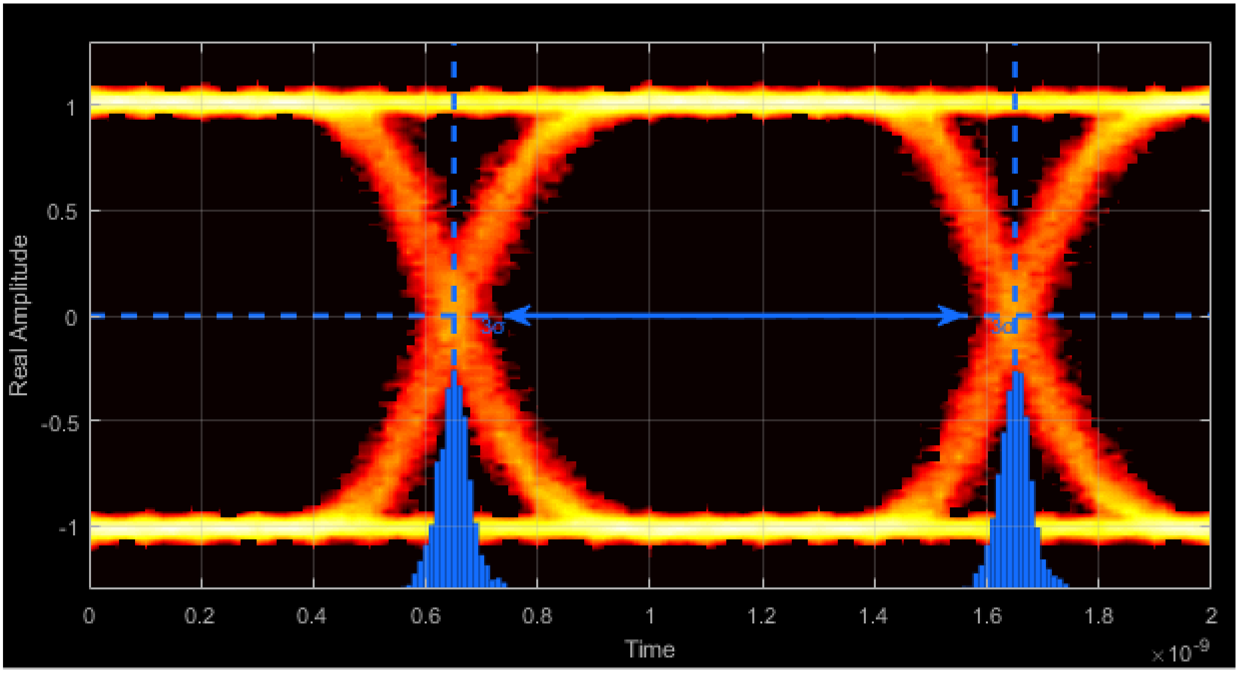
Eye Delay — Mean time between eye crossings

Eye delay is the midpoint between the two crossing times.



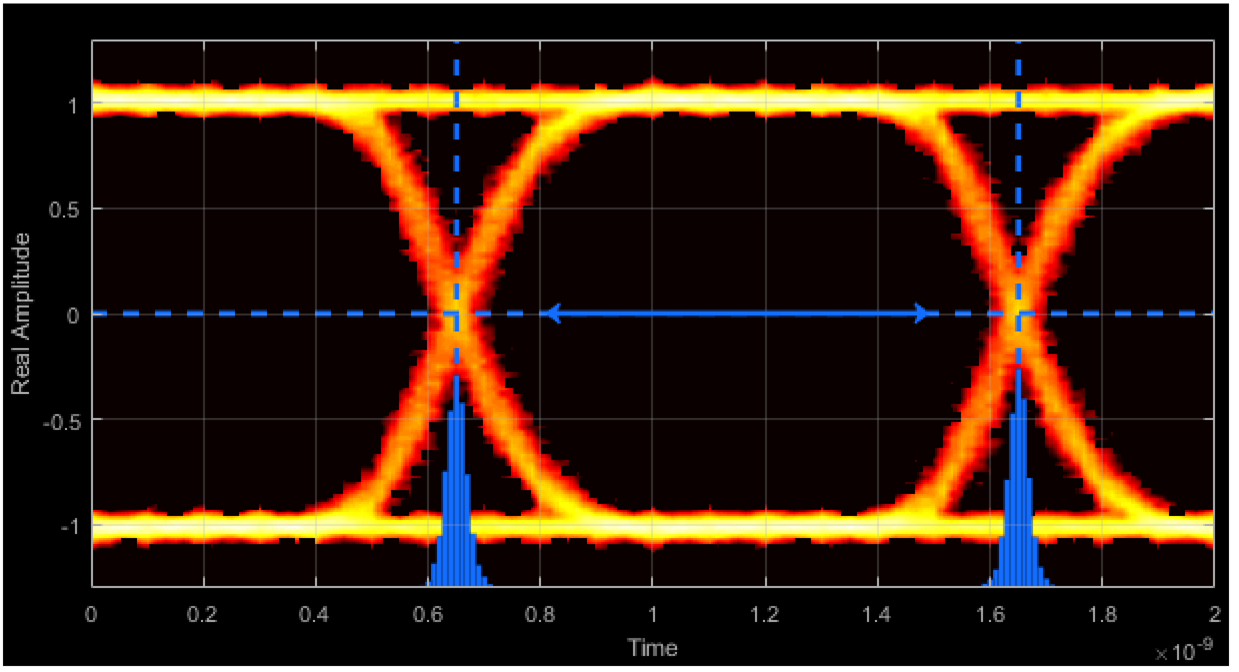
Eye Width — Statistical minimum time between eye crossings

Eye width is the horizontal distance between $\mu + 3\sigma$ of the left crossing time and $\mu - 3\sigma$ of the right crossing time. μ is the mean of the jitter histogram and σ is the standard deviation.



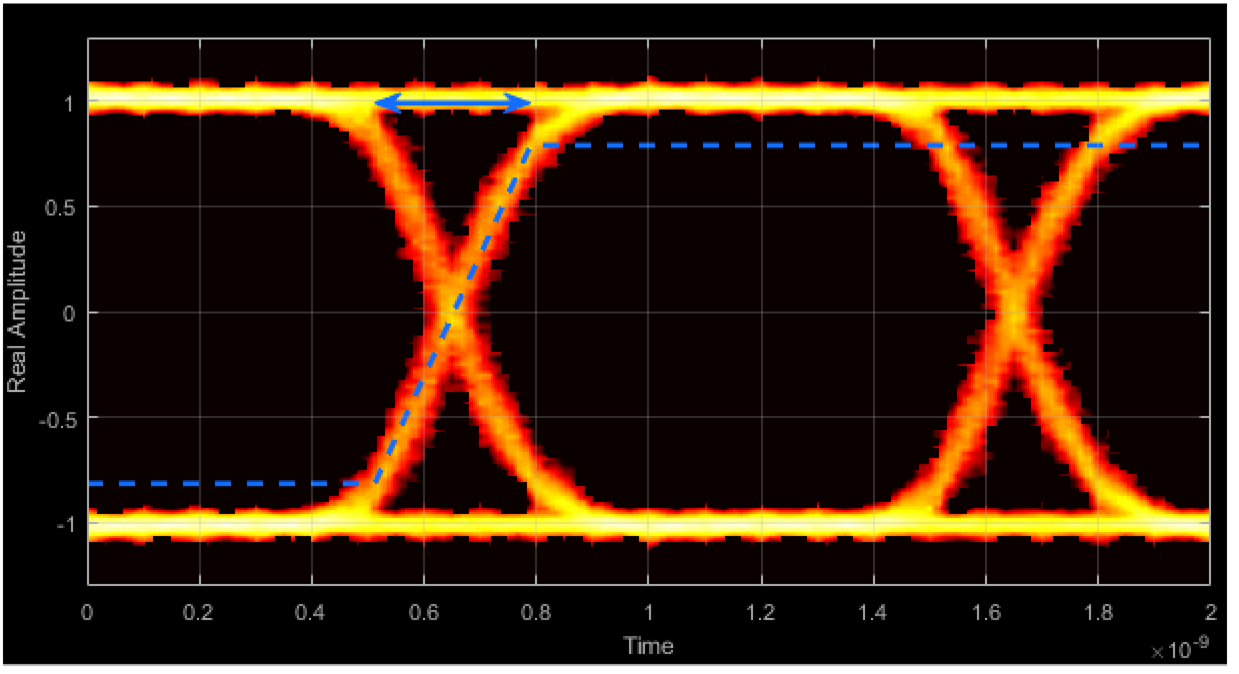
`Horizontal Opening` — Time between BER threshold points

The horizontal opening is the distance between the two points that correspond to the BER threshold. For example, for a 10^{-12} BER, these two points correspond to the 7σ distance from each crossing time.



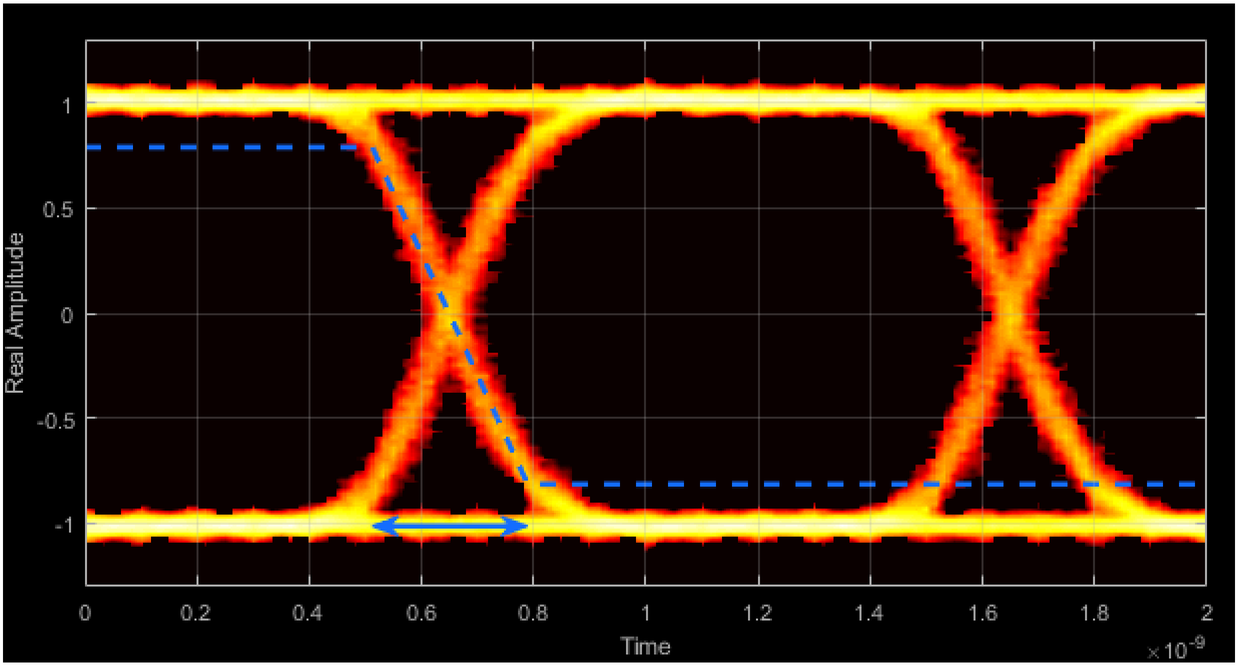
Rise Time — Time to transition from low to high

Rise time is the mean time between the low and high thresholds defined in the eye diagram. The default thresholds are 10% and 90% of the eye amplitude.



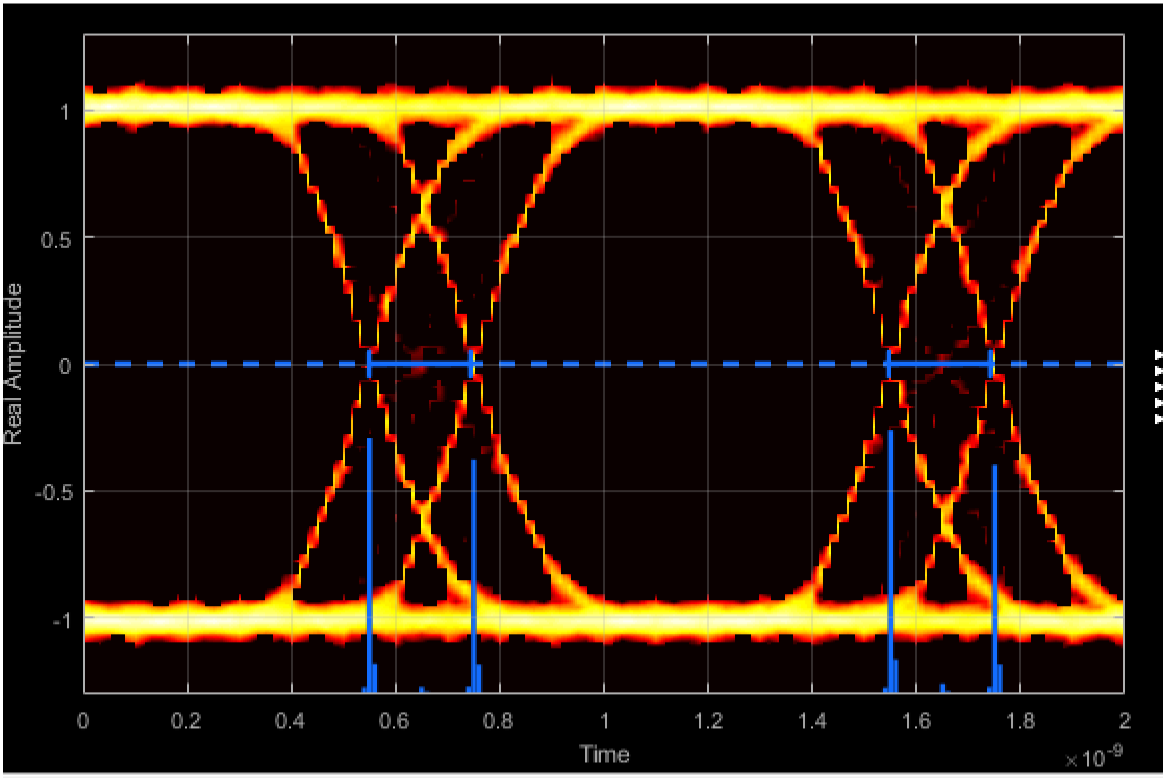
`Fall Time` — Time to transition from high to low

Fall time is the mean time between the high and low thresholds defined in the eye diagram. The default thresholds are 10% and 90% of the eye amplitude.



`Deterministic Jitter` — Deterministic deviation from ideal signal timing

The deterministic jitter (DJ) is the distance between the two peaks of the dual-Dirac histograms. The probability density function (PDF) of DJ is composed of two delta functions.



Random Jitter — Random deviation from ideal signal timing

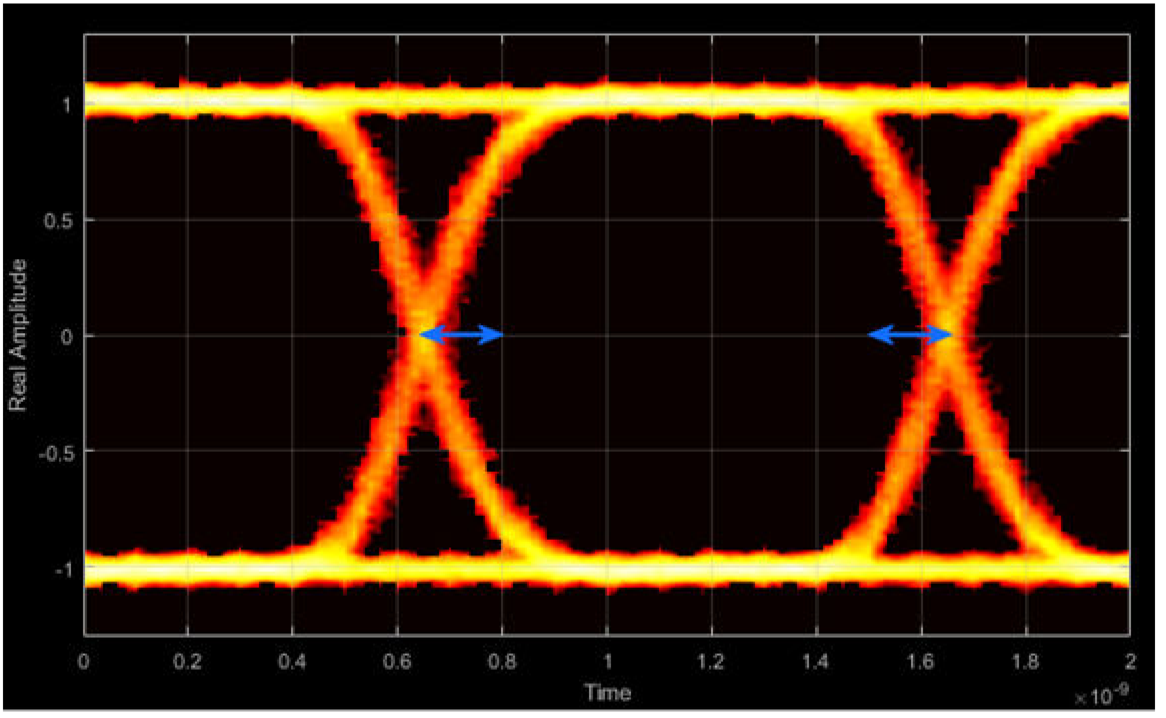
The random jitter (RJ) is the Gaussian unbounded jitter component. The random component of jitter is modeled as a zero-mean Gaussian random variable with a specified standard-deviation, σ. The random jitter is computed as:

$$\mathrm{RJ} = (Q_L + Q_R)\sigma \, ,$$

where

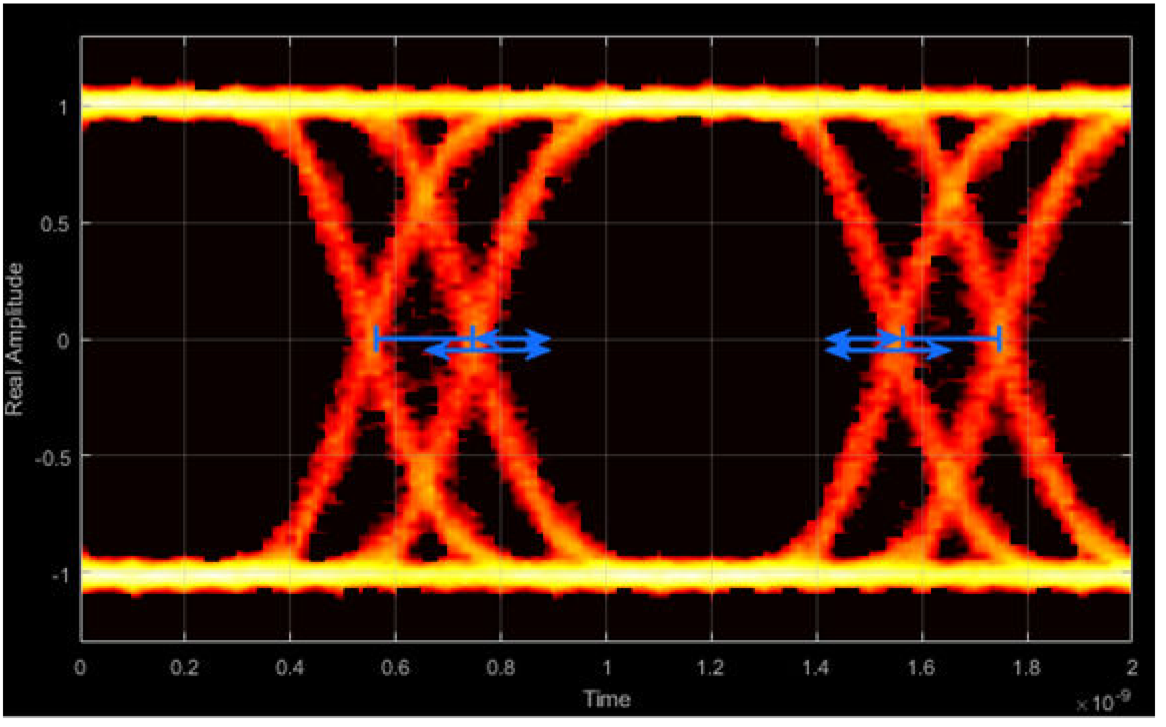
$$Q = \sqrt{2}\mathrm{erfc}^{-1}\left(2\frac{BER}{\rho}\right) \, .$$

BER is the specified BER threshold. ρ is the amplitude of the left and right Dirac function, which is determined from the bin counts of the jitter histograms.

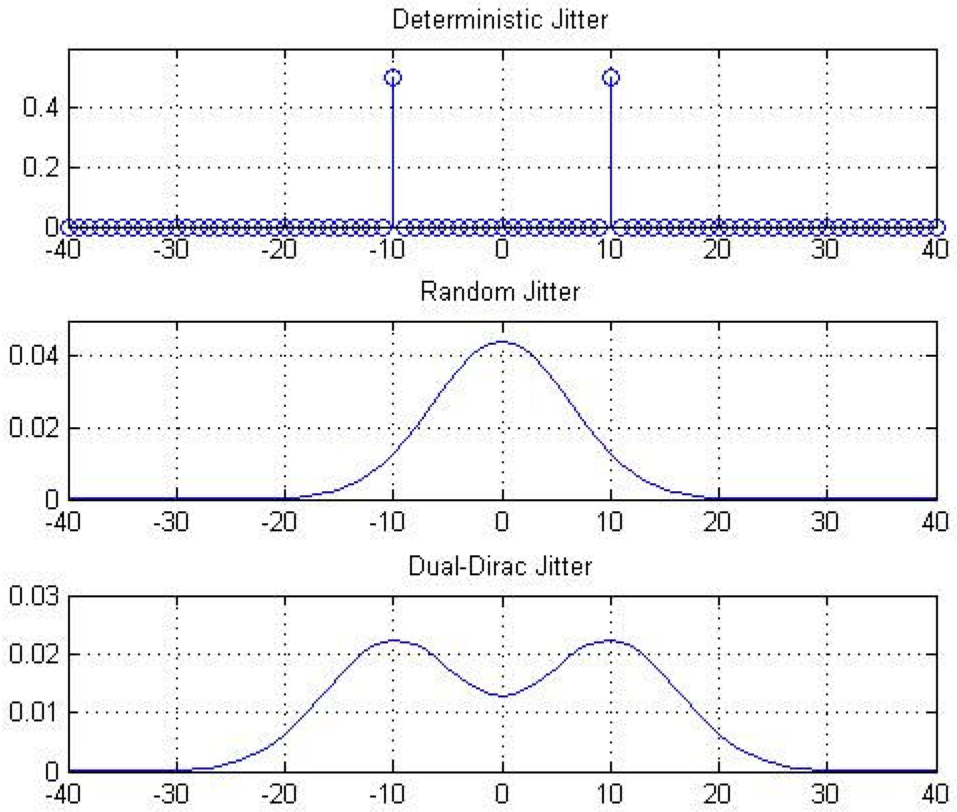


`Total Jitter` — Deviation from ideal signal timing

Total jitter (TJ) is the sum of the deterministic and random jitter, such that TJ = DJ + RJ.

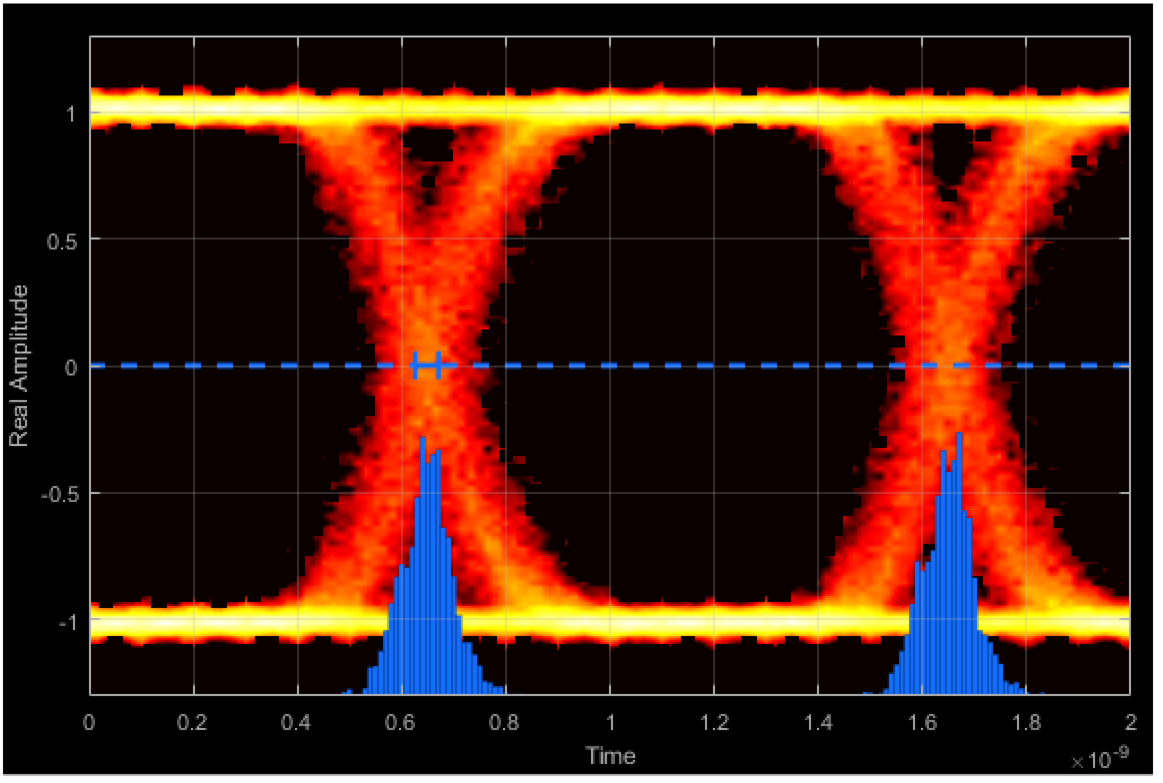


The total jitter PDF is the convolution of the DJ PDF and the RJ PDF.



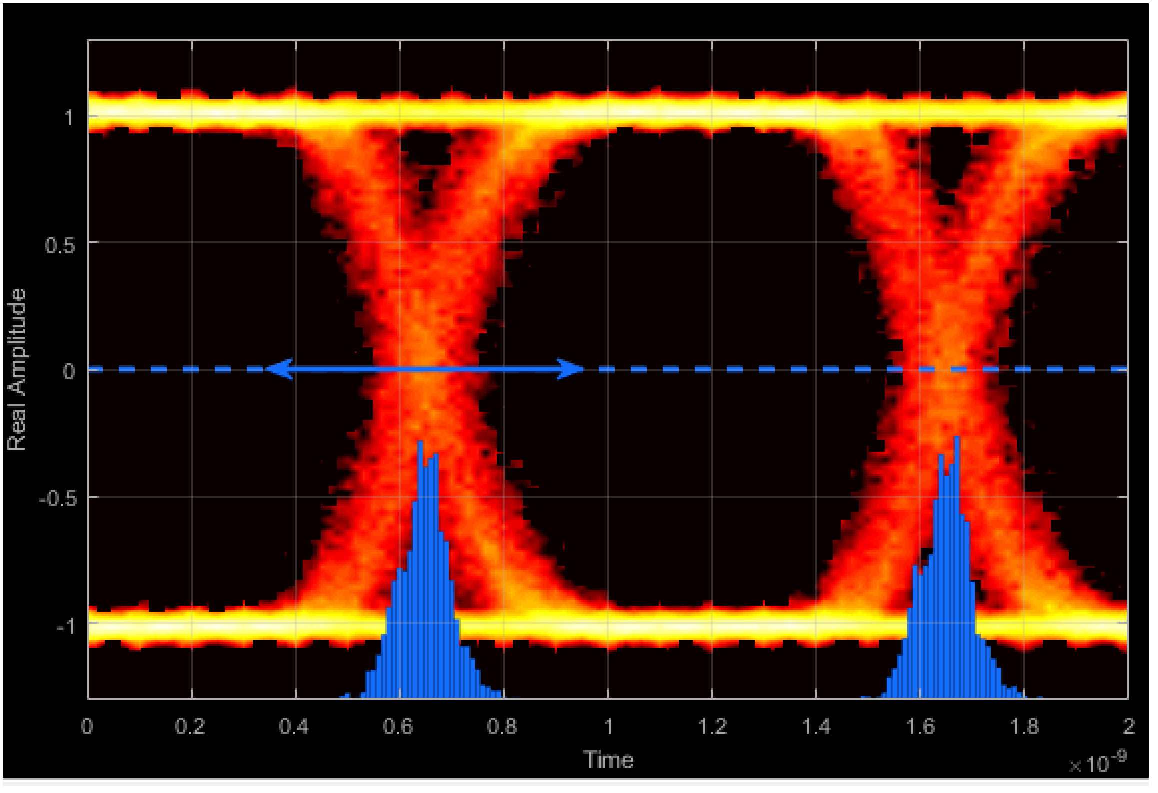
`RMS Jitter` — Standard deviation of jitter

RMS jitter is the standard deviation of the jitter calculated in the horizontal (jitter) histogram at the decision boundary.



`Peak-to-Peak Jitter` — Distance between extreme data points of histogram

Peak-to-peak jitter is the maximum horizontal distance between the left and right nonzero values in the horizontal histogram of each crossing time.



Measurement Settings

To change measurement settings, first select **Enable measurements**. Then, in the **Eye Measurements** pane, click the arrow next to **Settings**. You can control these measurement settings.

`Eye level boundaries` — Time range for calculating eye levels

`[40 60]` (default) | two-element vector

Time range for calculating eye levels, specified as a two-element vector. These values are expressed as a percentage of the symbol duration. Tunable.

`Decision boundary` — Amplitude level threshold

`0` (default) | scalar

Amplitude level threshold in V, specified as a scalar. This parameter separates the different signaling regions for horizontal (jitter) histograms. This parameter is tunable, but the jitter histograms reset when the parameter changes.

For non-return-to-zero (NRZ) signals, set **Decision boundary** to 0. For return-to-zero (RZ) signals, set **Decision boundary** to half the maximum amplitude.

`Rise/Fall Thresholds` — Amplitude levels of the rise and fall transitions

`[10 90]` (default) | two-element vector

Amplitude levels of the rise and fall transitions, specified as a two-element vector. These values are expressed as a percentage of the eye amplitude. This parameter is tunable, but the crossing histograms of the rise and fall thresholds reset when the parameter changes.

`Hysteresis` — Amplitude tolerance of the horizontal crossings

`0` (default) | scalar

Amplitude tolerance of the horizontal crossings in V, specified as a scalar. Increase hysteresis to provide more tolerance to spurious crossings due to noise. This parameter is tunable, but the jitter and the rise and fall histograms reset when the parameter changes.

`BER threshold` — BER used for eye measurements

`1e-12` (default) | nonnegative scalar from 0 to 0.5

BER used for eye measurements, specified as a nonnegative scalar from 0 to 0.5. The value is used to make measurements of random jitter, total jitter, horizontal eye openings, and vertical eye openings. Tunable.

`Bathtub BERs` — BER values used to calculate openings of bathtub curves

`[0.5 0.1 0.01 0.001 0.0001 1e-05 1e-06 1e-07 1e-08 1e-09 1e-10 1e-11 1e-12]` (default) | vector

BER values used to calculate openings of bathtub curves, specified as a vector whose elements range from 0 to 0.5. Horizontal and vertical eye openings are calculated for each of the values specified by this parameter. To enable this parameter, select **Show horizontal bathtub curve**, **Show vertical bathtub curve**, or both. Tunable.

`Measurement delay` — Duration of initial data discarded from measurements

`0` (default) | nonnegative scalar

Duration of initial data discarded from measurements, in seconds, specified as a nonnegative scalar.

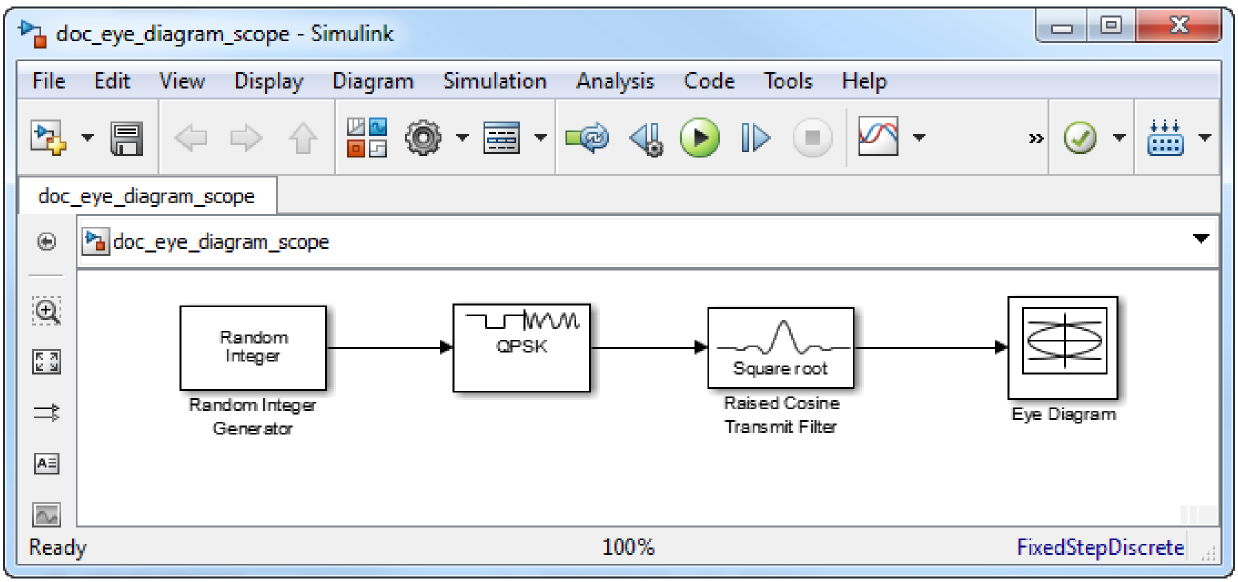
Examples

View Eye Diagram

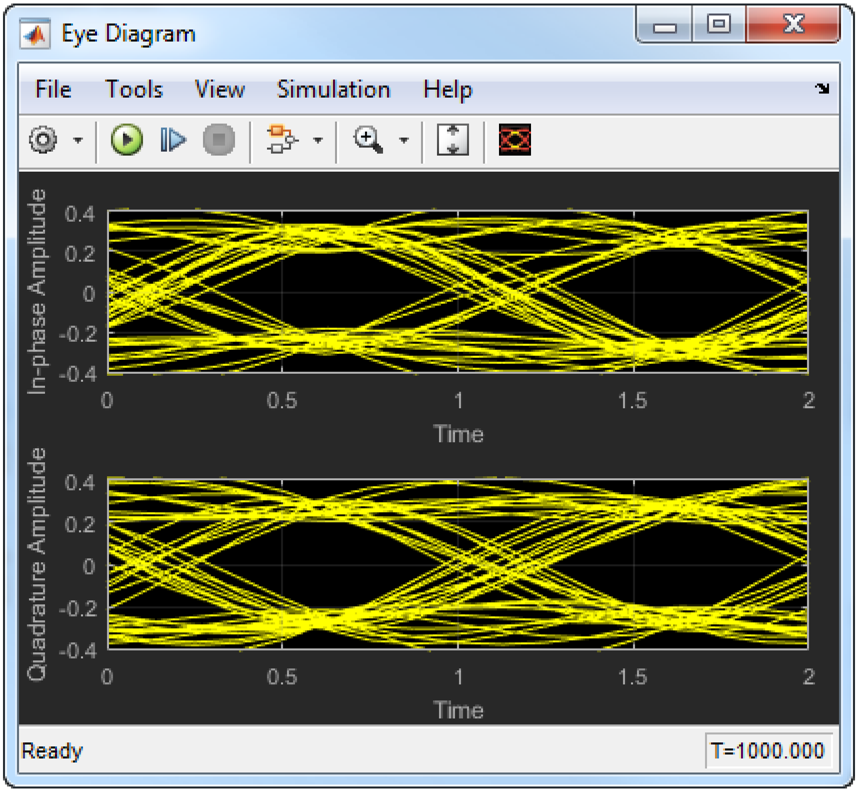
Display the eye diagram of a filtered QPSK signal using the Eye Diagram block.

Load the doc_eye_diagram_scope model from the MATLAB command prompt.

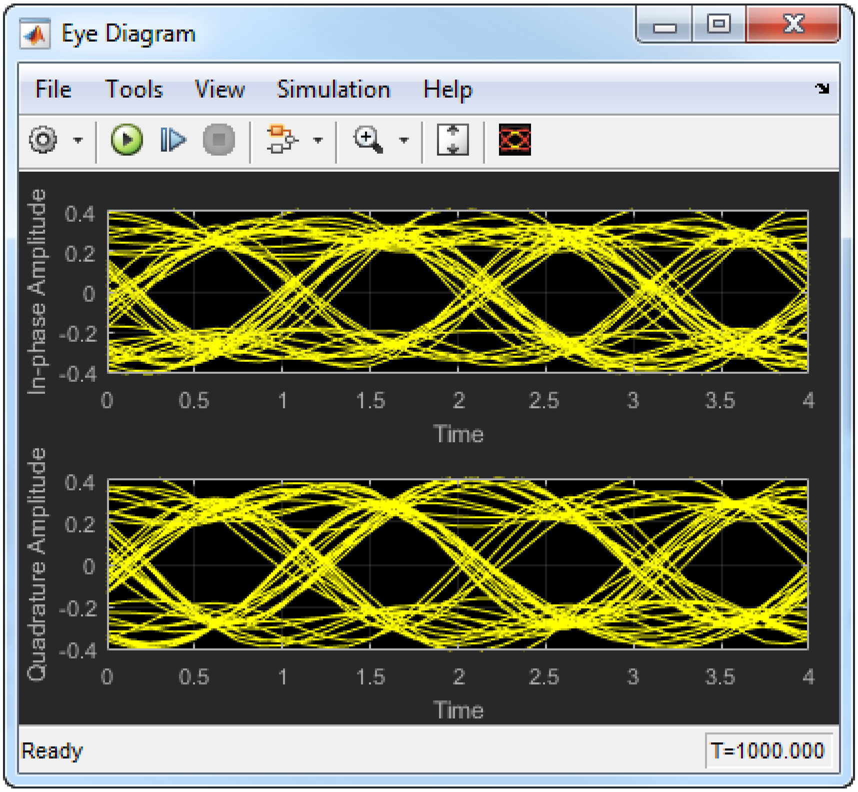
```
doc_eye_diagram_scope
```



Run the model and observe that two symbols are displayed.



Open the configuration parameters dialog box. Change the **Symbols per trace** parameter to 4. Run the simulation and observe that four symbols are displayed.



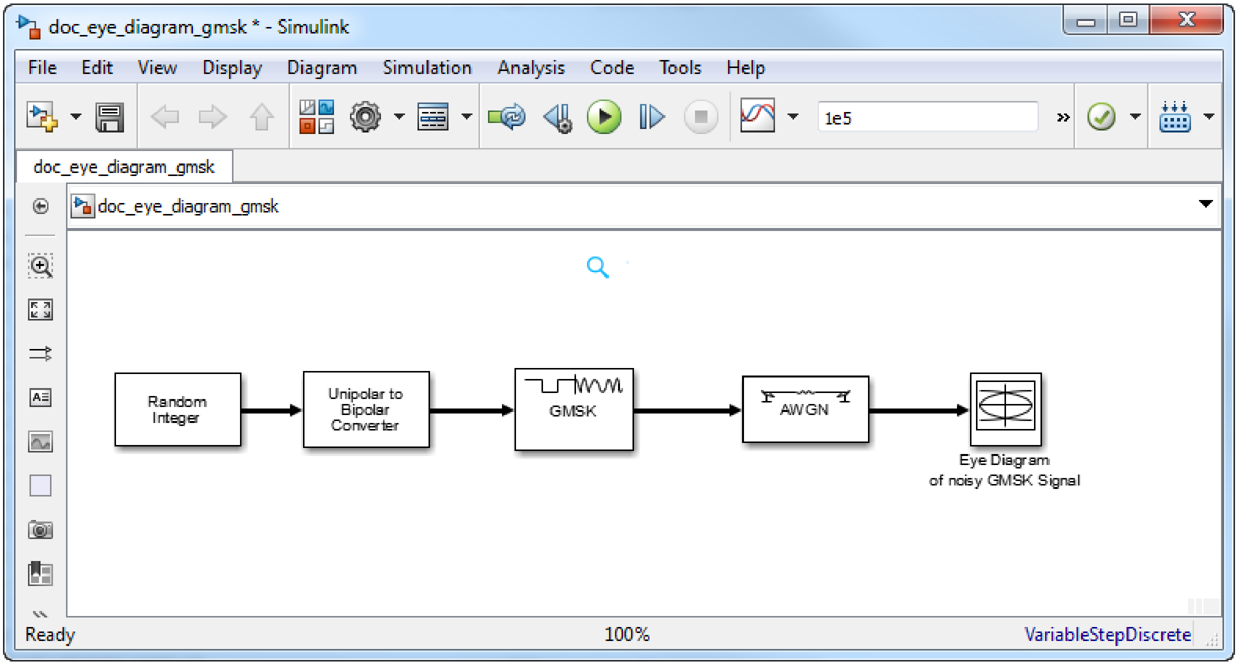
Try changing the Raised Cosine Transmit Filter parameters or changing additional Eye Diagram parameters to see their effects on the eye diagram.

Histogram Plots

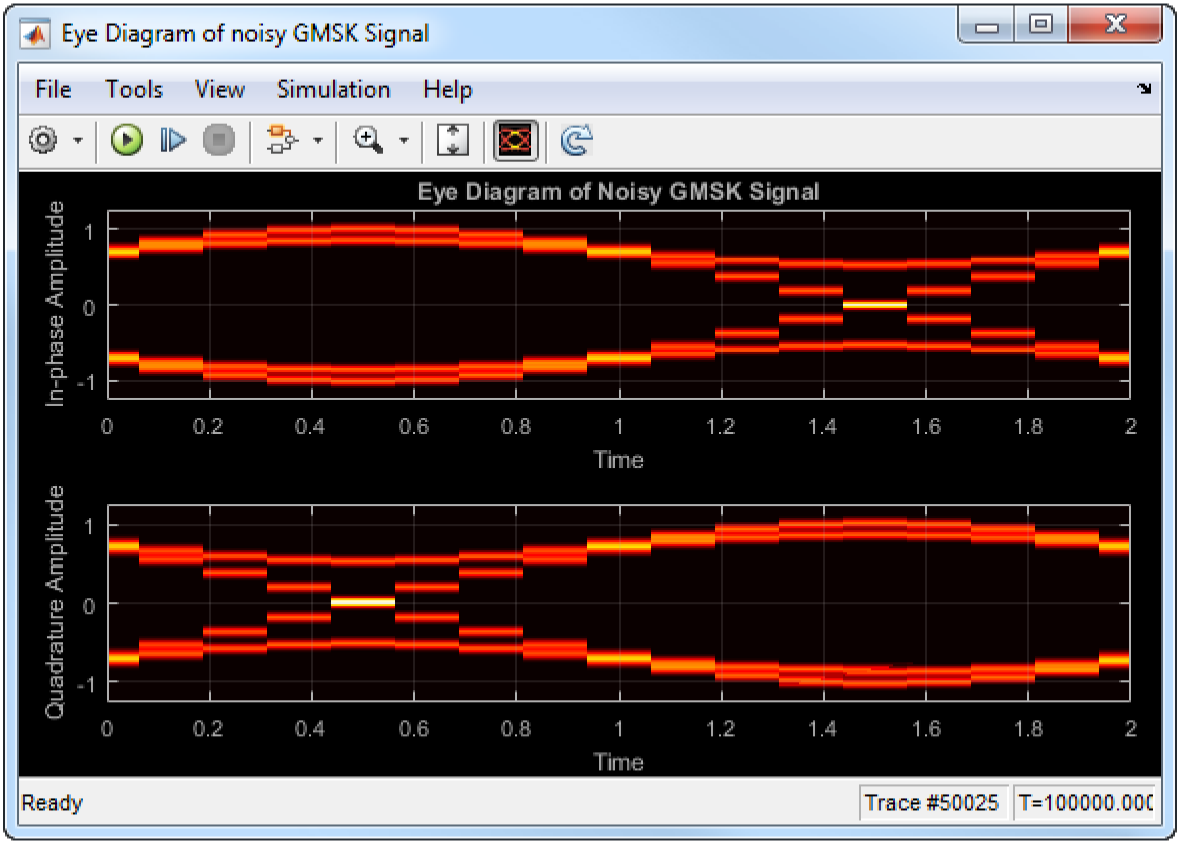
Display histogram plots of a noisy GMSK signal.

Load the doc_eye_diagram_gmsk model from the MATLAB command prompt.

```
doc_eye_diagram_gmsk
```

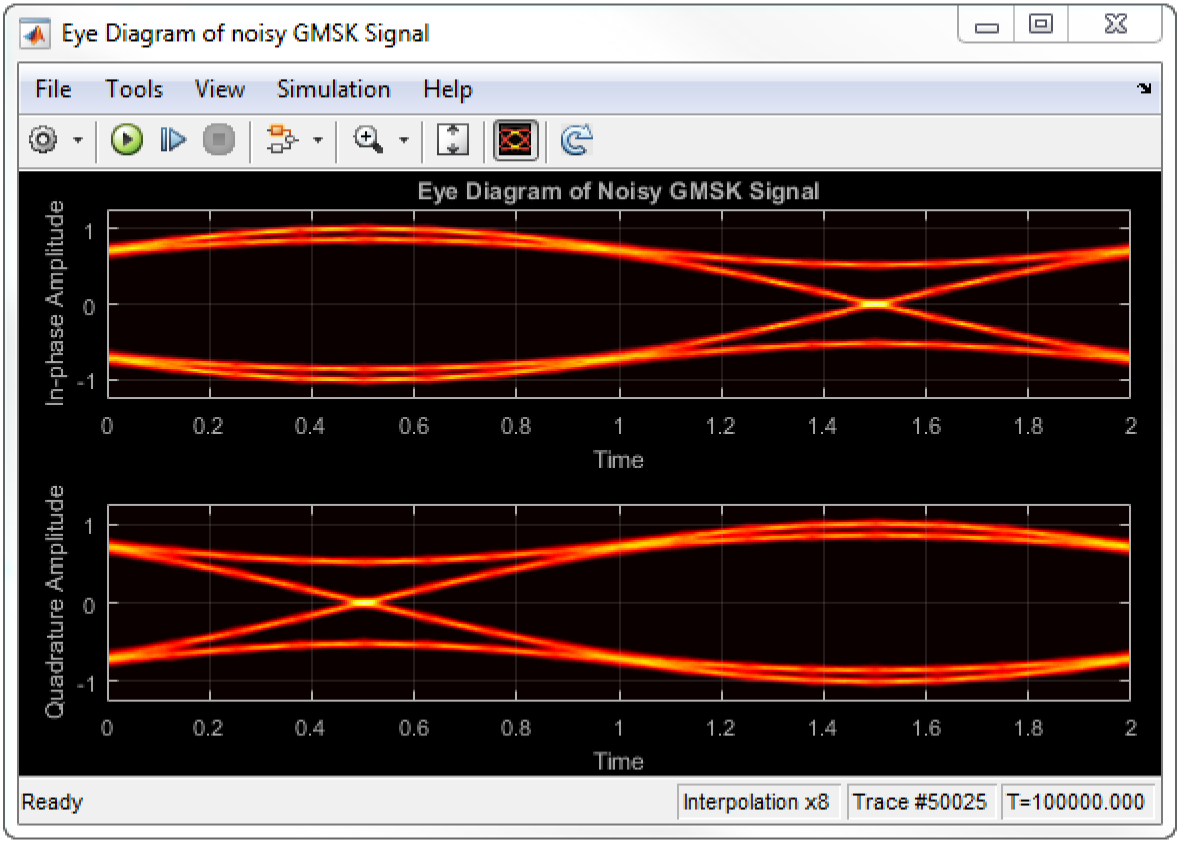


Run the model. The eye diagram is configured to show a histogram without interpolation.



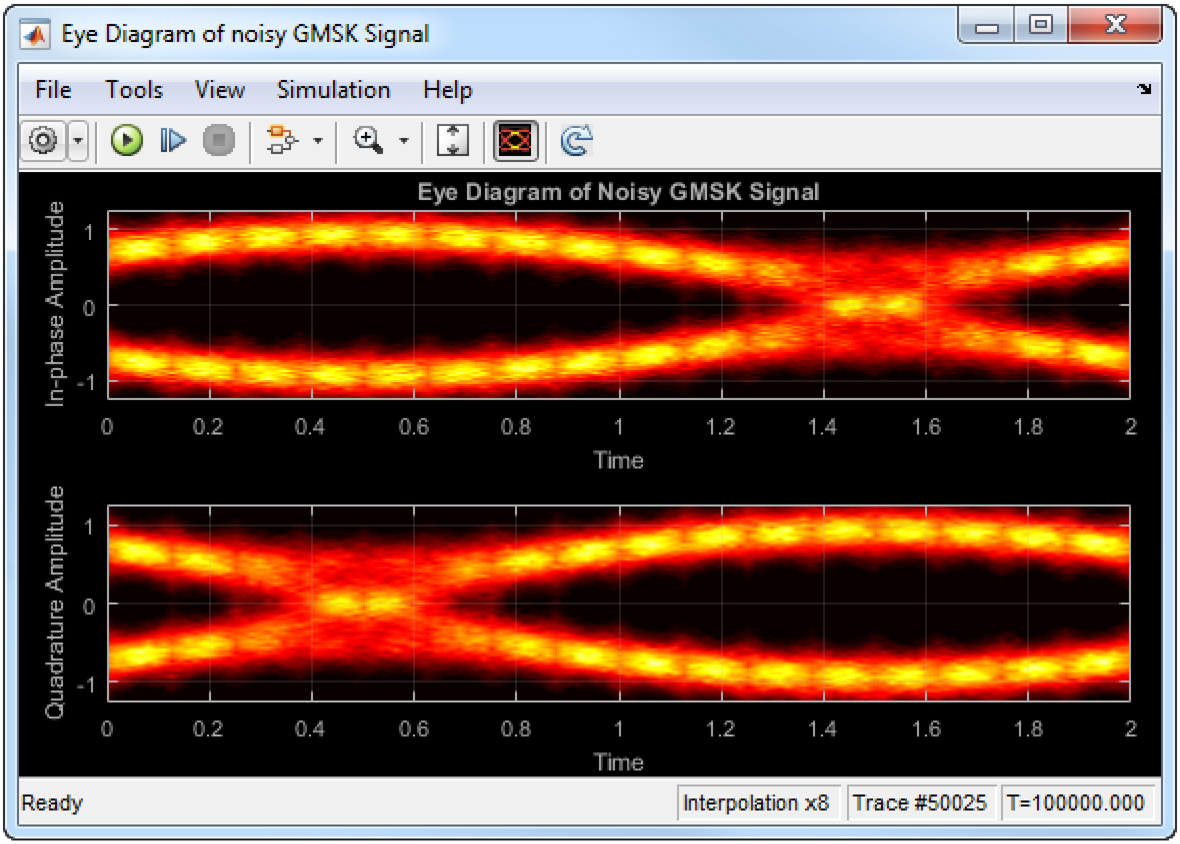
The lack of interpolation results in a plot having piecewise-continuous behavior.

Open the **2D Histogram** tab of the Configuration Properties dialog box. Set the **Oversampling method** to `Input interpolation`. Run the model.



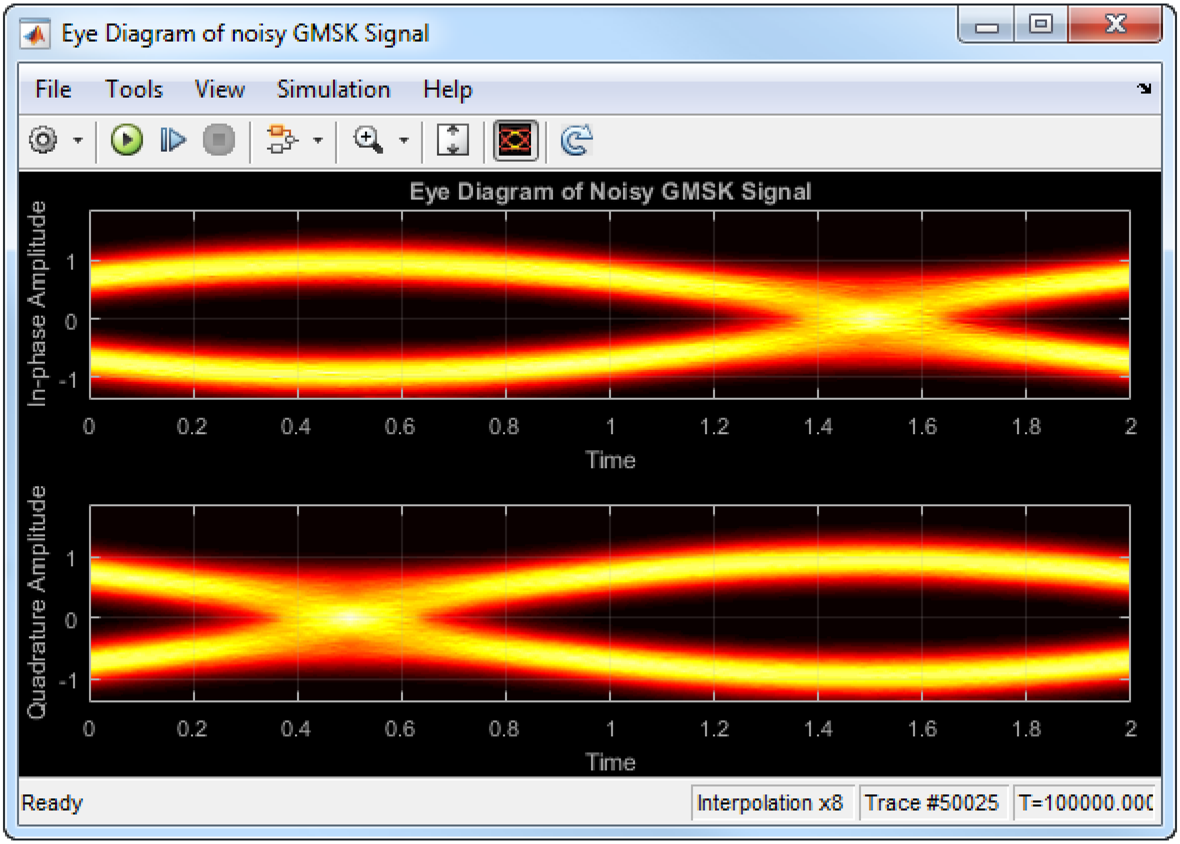
The interpolation smooths the eye diagram.

On the AWGN Channel block, change **SNR (dB)** from 25 to 10. Run the model.



Observe that vertical striping is present in the eye diagram. This striping is the result of input interpolation, which has limited accuracy in low-SNR conditions.

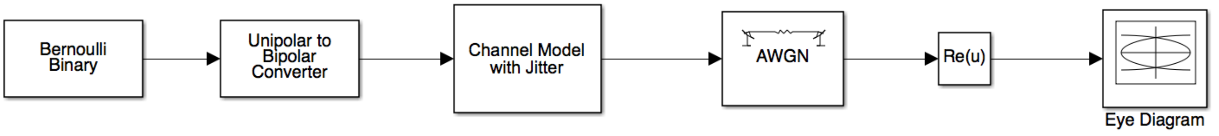
Set the **Oversampling method** to `Histogram interpolation`. Run the model.



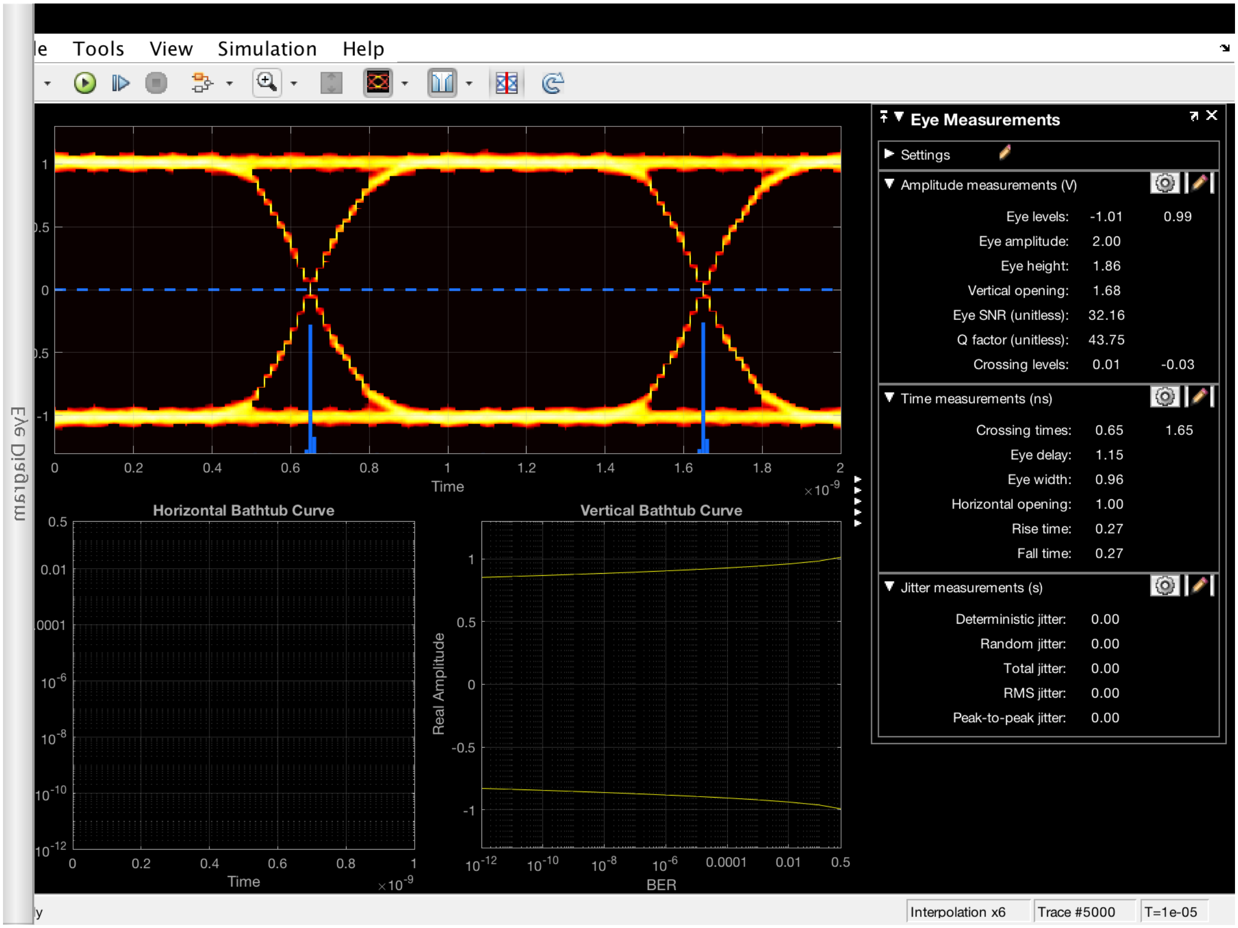
The eye diagram plot now renders accurately because the histogram interpolation method works for all SNR values. This method is not as fast as the other techniques and results in increased execution time.

Visualize Random and Deterministic Jitter

Open the model. The model generates bipolar data, adds deterministic and random jitter, applies white noise, displays the resulting eye diagram.

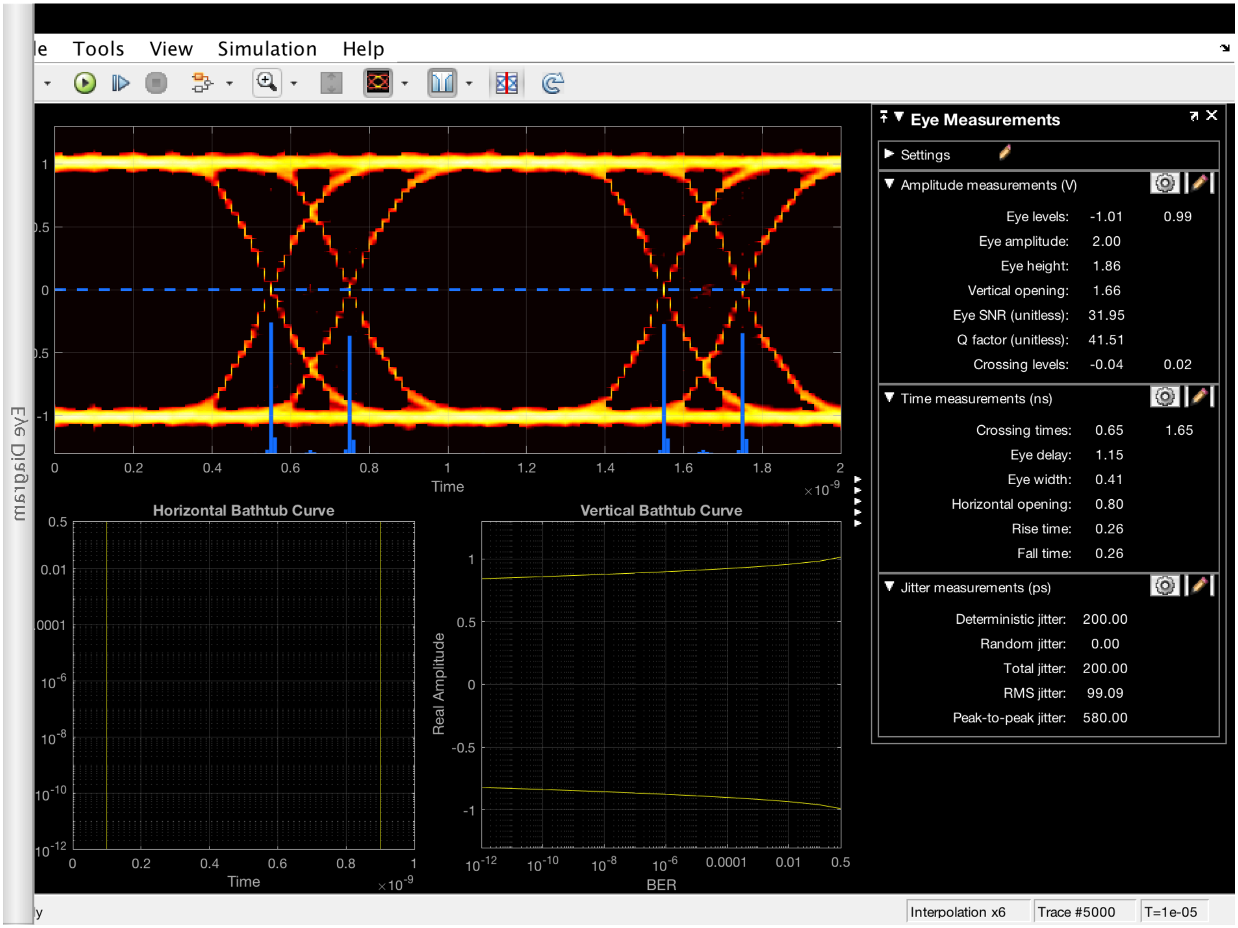


Run the model.



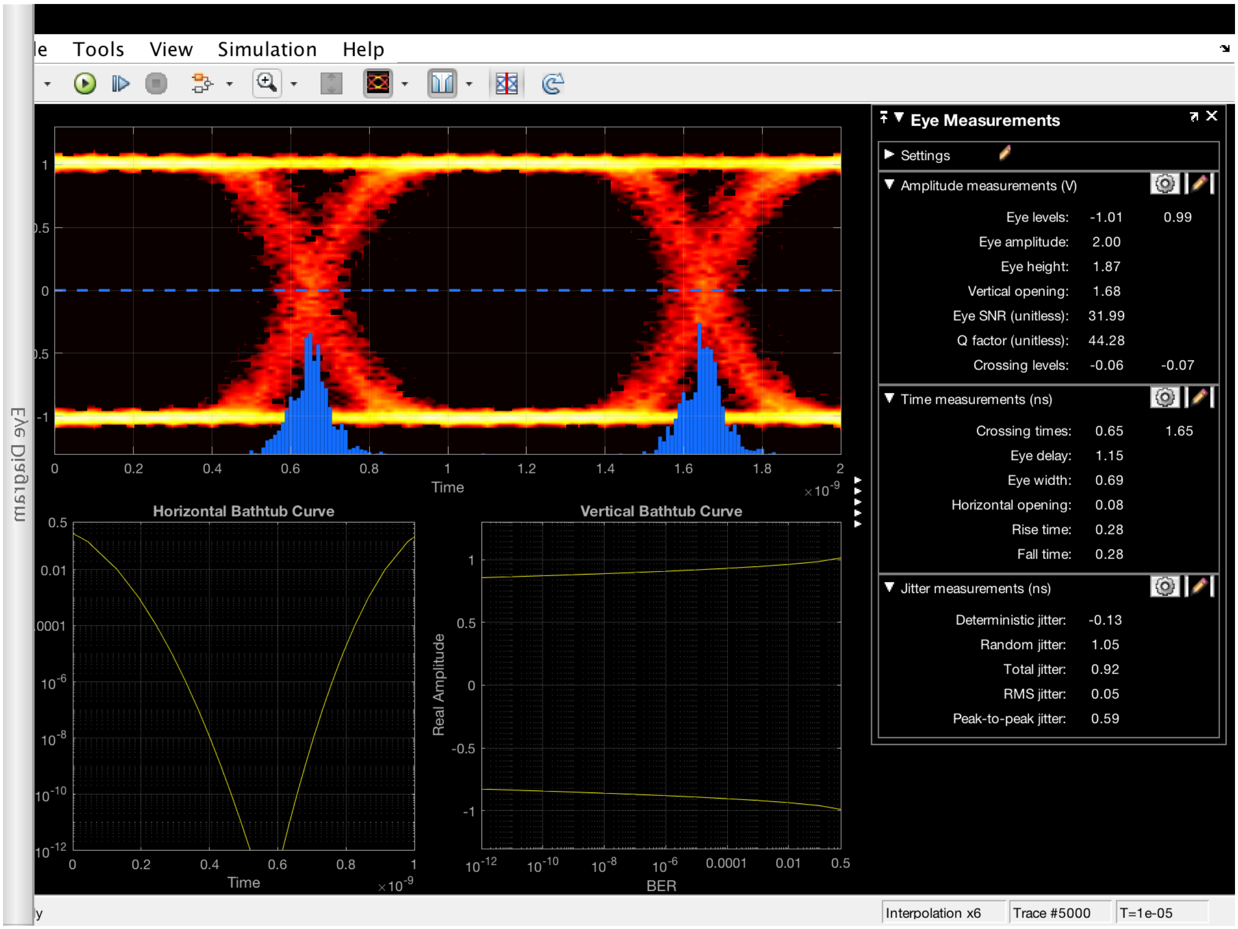
The signal shows clean crossings as there is no jitter.

To show the effect of the deterministic jitter, set the **Deterministic jitter** parameter to `100e-12` in the `Channel Model with Jitter` block. Run the model.



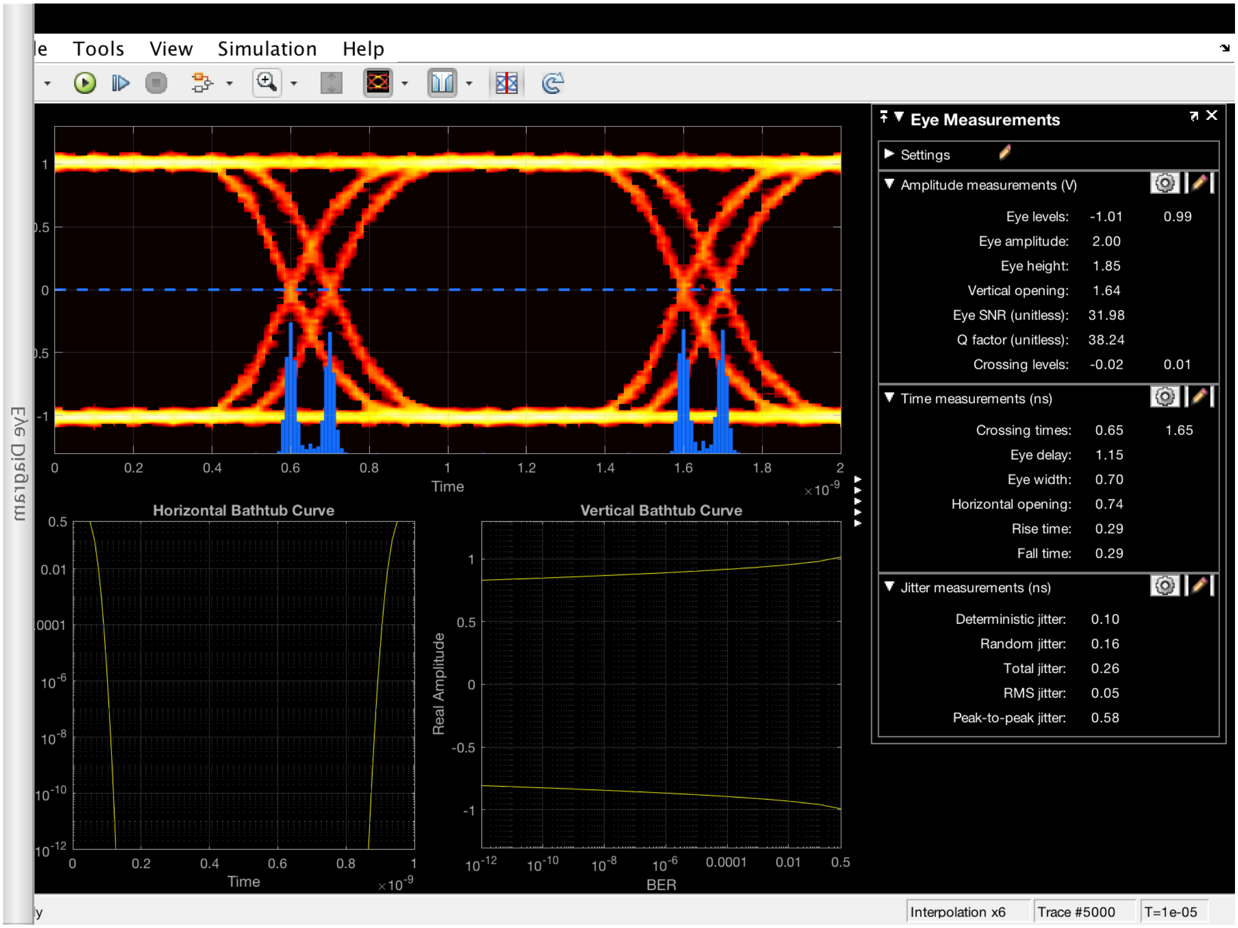
The deterministic jitter is shown by the separation between the two peaks in the jitter histogram.

To show the effect of the RMS jitter, set the **Deterministic jitter** parameter to `0` and set the **RMS jitter** parameter to `50e-12`. Run the model.



The RMS jitter is shown by the fuzziness around each of the crossings.

Set the RMS jitter to `10e-12` and the deterministic jitter to `50e-12`. Run the model.



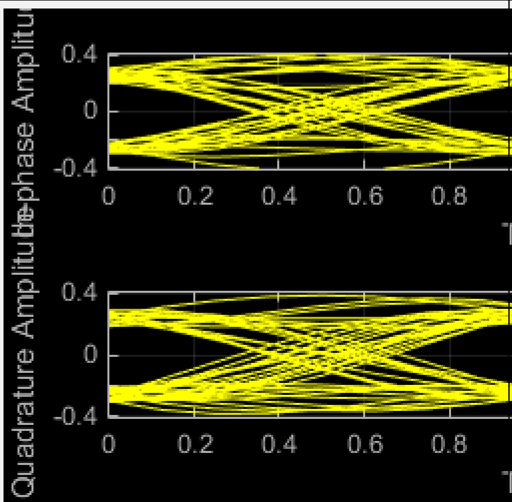
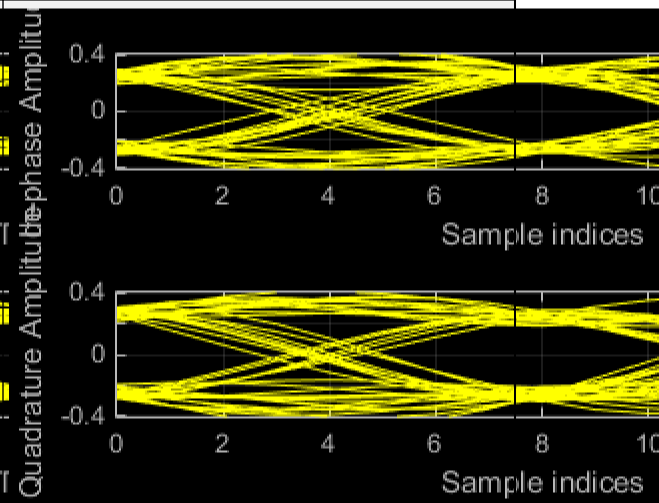
The signal shows the effects of both jitter types.

Definitions

Using Eye Diagram in Conditionally Executed Subsystems

When an Eye Diagram block is placed in a conditionally executed subsystem, for example in a triggered or enabled subsystem:

- Input size must be an integer multiple of `SamplesPerSymbol` * `SymbolsPerTrace`
- Sample offset must be zero
- The rightmost part of the display is intentionally omitted. This figure compares typical eye diagram display when placed in a normal system versus one placed in a conditionally executed subsystem.

Eye Diagram Plot in Normal System	Eye Diagram Plot in Conditionally Executed Subsystem
	
In a regular Eye Diagram, the rightmost part is a line between the last sample of a trace and the first sample of the next trace.	In conditionally executed subsystems, these traces may be non-contiguous, thus this rightmost segment could corrupt the display and is omitted.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block is excluded from the generated code when code generation is performed on a system containing this block.

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

This block can be used for simulation visibility in subsystems that generate HDL code, but is not included in the hardware implementation.

See Also

Blocks

System Objects

Introduced in R2014a

FFE

Models a feed-forward equalizer
Library: SerDes Toolbox / Datapath Blocks

Description

The FFE block applies a feed-forward equalizer as a symbol-spaced finite-impulse response (FIR) filter to a sample-by-sample input signal or an impulse response vector input signal to reduce distortions due to channel loss impairments.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal. The input signal can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Filtered channel output
scalar | vector

Filtered channel output. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The options are **Mode** and **Tap weights**. By default, both parameters are selected.

`Mode` — FFE operating mode
`Off` (default) | `Fixed`

FFE operating mode, specified as:

- `Off` - FFE is bypassed, the input waveform remains unchanged.
- `Fixed` - FFE applies input FFE tap weights specified in **Tap weights** to input waveform.

Programmatic Use

- Use `get_param(gcb,'Mode')` to view the current FFE **Mode**.
- Use `set_param(gcb,'Mode',value)` to set FFE to a specific **Mode**.

`Tap weights` — FFE tap weights
`[0,1,0,0,0]` (default) | row vector

FFE tap weights, specified as a row vector in V. The length of the vector specifies the number of taps. The vector element value specifies the strength of the tap at that element position. The tap with the largest magnitude is the main tap and therefore defines the number of pre- and post-taps.

Programmatic Use

- Use `get_param(gcb,'TapWeights')` to view the current FFE **Tap weights**.
- Use `set_param(gcb,'TapWeights',value)` to set FFE to a specific **Tap weights** vector.

`Normalize` — Normalize tap weights
button

Select to normalize tap weight vectors so that the sum of the absolute values of the `TapWeights` vector elements is one.

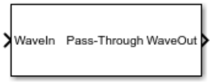
See Also

`serdes.FFE`

Introduced in R2019a

PassThrough

Propagates baseband signal without modification
Library: SerDes Toolbox / Datapath Blocks



Description

The PassThrough block passes the input signal without any modification. This block is used as a place holder within a SerDes system and as a template for user-authored blocks for use in SerDes Toolbox.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal, can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Unchanged output voltage
scalar | vector

Unchanged output voltage. The PassThrough block does not modify the input voltage in any way and returns the same output as the input.

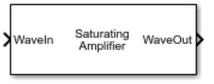
Data Types: `double`

See Also

Introduced in R2019a

SaturatingAmplifier

Models a saturation amplifier
Library: SerDes Toolbox / Datapath Blocks



Description

The SaturatingAmplifier block scales the input waveform according to a voltage in vs. voltage out response. The voltage in vs. voltage out response is specified either by the soft clipping response defined by **Limit** and **Linear Gain**, or by the **VinVout** matrix.

Ports

Input

`WaveIn` — Input baseband signal
scalar | vector

Input baseband signal, can be a sample-by-sample signal specified as a scalar, or an impulse response vector signal.

Data Types: `double`

Output

`WaveOut` — Clipped output voltage
scalar | vector

Clipped output voltage, as specified by the SaturatingAmplifier block. If the input signal is a sample-by-sample signal specified as a scalar, the output is also scalar. If the input signal is an impulse response vector signal, the output is also a vector.

Data Types: `double`

Parameters

`IBIS-AMI parameters` — Choose parameters to be included in IBIS-AMI model
button

Choose which parameters to be included in IBIS-AMI models. The only option is **Mode**, which is selected by default.

`Mode` — Amplifier operating mode
`Off` (default) | `On`

Amplifier operating mode, specified as:

- `Off` - SaturatingAmplifier is bypassed, the input waveform remains unchanged.
- `On` - SaturatingAmplifier scales the input waveform according to a voltage in vs. voltage out response.

Programmatic Use

- Use `get_param(gcb,'Mode')` to view the current saturating amplifier operating **Mode**.
- Use `set_param(gcb,'Mode',value)` to set amplifier to a specific **Mode**.

`Specification` — Input specification for limiting amplifier output
`Limit and Linear Gain` (default) | `VinVout`

Input specification for limiting amplifier output, specified as:

- `Limit and Linear Gain` - Creates a soft clipping voltage in vs. voltage out response with the values specified in **Limit** and **Linear Gain**.
- `VinVout` - Generates output voltages corresponding to input voltage specified in **VinVout**. If any input voltage point falls outside the specified values, the output for that particular input voltage is interpolated.

Programmatic Use

- Use `get_param(gcb,'Specification')` to view the current **Specification** of saturating amplifier.
- Use `set_param(gcb,'Specification',value)` to set saturating amplifier to a specific **Specification**.

`Limit` — Clipping voltage for the limiting amplifier

`1.2` (default) | real positive scalar

Clipping voltage for the limiting amplifier, specified as a real positive scalar in V.

Dependencies

This parameter is only available when **Specification** is selected as `Limit and Linear Gain`

Programmatic Use

- Use `get_param(gcb,'Limit')` to view the current value of **Limit** of saturating amplifier.
- Use `set_param(gcb,'Limit',value)` to set **Limit** to a specific value.

`LinearGain` — Amplifier gain in the linear region

`1` (default) | real positive scalar

Amplifier gain in the linear region, specified as a unitless real positive scalar.

Dependencies

This parameter is only available when **Specification** is selected as `Limit and Linear Gain`

Programmatic Use

- Use `get_param(gcb,'LinearGain')` to view the current value of **LinearGain** of saturating amplifier.
- Use `set_param(gcb,'LinearGain',value)` to set **LinearGain** to a specific value.

`VinVout` — Input and corresponding output voltage response table

N×2 matrix

Input and corresponding output voltage response table, specified as an N×2 matrix in V.

Dependencies

This parameter is only available when **Specification** is selected as `VinVout`

Programmatic Use

- Use `get_param(gcb,'VinVout')` to view the current **VinVout** table value of saturating amplifier.

- Use `set_param(gcb,'VinVout',value)` to set **VinVout** to a specific value.

See Also

`serdes.SaturatingAmplifier`

Introduced in R2019a

Stimulus

Set pseudorandom binary sequence (PRBS) pattern and number of symbols to simulate in SerDes model

Library: SerDes Toolbox / Utilities

Description

The Stimulus sets the PRBS pattern and the number of symbols to simulate in a SerDes Toolbox model.

Ports

Output

`WaveOut` — Output signal with specific PRBS pattern

vector

Output pattern with a specific PRBS pattern, specified as a vector.

Data Types: `double`

Parameters

PRBS — Order of the pseudorandom binary sequence

`11` (default) | `7` | `9` | `13` | `15` | `20` | `23` | `31` | `47`

Order of the pseudorandom binary sequence.

Dependencies

This parameter is only tunable when **Custom stimulus** option is deselected.

Programmatic Use

- Use `get_param(gcb,'PRBS')` to view the current value of **PRBS**.
- Use `set_param(gcb,'PRBS',value)` to set **PRBS** to a specific value.

`Number of symbols` — Length of PRBS pattern used for simulation
`2000` (default) | positive integer

Length of the PRBS pattern used for simulation, specified as a positive integer.

Dependencies

This parameter is only tunable when **Custom stimulus** option is deselected.

Programmatic Use

- Use `get_param(gcb,'NumberOfSymbols')` to view the current value of **Number of symbols**.
- Use `set_param(gcb,'NumberOfSymbols',value)` to set **Number of symbols** to a specific value.

`Custom stimulus` — Select to input a custom stimulus
button

Select to input a custom stimulus. By default, this option is deselected.

If you enable this option, you can manually enter a vector containing the input voltages at sample interval spacing as your stimulus.

Example: `[zeros(1,(SymbolTime/SampleInterval)),ones(1,(SymbolTime/SampleInterval))]-0.5`

See Also
Analog Channel | Configuration

Introduced in R2019a

VGA

Models a variable gain amplifier
Library: SerDes Toolbox / Datapath Blocks

Description

The VGA block scales the amplitude of the input waveform based on a gain specified by the user.

Ports

Input

`WaveIn` — Input signal
scalar | vector

Input signal to be scaled, specified as a scalar or vector.

Data Types: `double`

Output

`WaveOut` — Scaled output signal
scalar | vector

Scaled output signal, returend as a scalar or vector corresponding to the input signal.

Data Types: `double`

Parameters

Mode — VGA operating mode

Off (default) | On

VGA operating mode, specified as Off or On. Mode determines if the VGA adjusts the gain of input signal or acts as a pass-through.

VGA Mode	VGA Operation
Off	VGA is bypassed, the input waveform remains unchanged.
On	VGA scales the input waveform according to the specified Gain.

Programmatic Use

- Use get_param(gcb,'Mode') to view the current VGA **Mode**.
- Use set_param(gcb,'Mode',value) to set VGA to a specific **Mode**.

Data Types: double

Gain — Multiplicative gain used to scale the input waveform

1 (default) | scalar

Multiplicative gain used to scale the input waveform, specified as a unitless scalar.

Programmatic Use

- Use get_param(gcb,'Gain') to view the current value of **Gain**.
- Use set_param(gcb,'Gain',value) to set VGA **Gain** to a specific value.

Data Types: double

IBIS-AMI parameters — Choose parameters to be included in IBIS-AMI model

button

Choose which parameters to be included in IBIS-AMI models. The options are **Mode** and **Gain**. By default, both parameters are selected.

See Also

Introduced in R2019a

SerDes Apps — Alphabetical List

SerDes Designer

Design and analyze SerDes systems for export to Simulink, MATLAB and IBIS-AMI

Description

The **SerDes Designer** app generates the SerDes Designer tree required to generate IBIS-AMI models. Start from the app to develop initial SerDes architecture using statistical analysis and manage developed models.

Using this app, you can:

- Create fully compliant IBIS(Input/Output Buffer Information Specification)-AMI(Algorithmic Modeling Interface) models and perform statistical analysis.
- Generate MATLAB scripts for further customization and statistical and time domain analysis.
- Export Simulink models for further customization, statistical and time domain analysis, and IBIS-AMI model generation.

To know more about this app, see "Design SerDes System and Export IBIS-AMI Model".

Open the SerDes Designer App

- MATLAB Toolstrip: In the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `serdesDesigner`.

Examples

- "Design SerDes System and Export IBIS-AMI Model"
- "PCIe4 Transmitter/Receiver IBIS-AMI Model"

Programmatic Use

`serdesDesigner` opens a new session of the **SerDes Designer** app, enabling you to design and analyze a SerDes system.

`serdesDesigner(serdesDesign)` opens the **SerDes Designer** app and loads the `serdesDesign` file saved from the previous session.

Limitations

IBIS-AMI codegen is not supported in MAC.

More about

Configuring SerDes System

The **SerDes Designer** app provides built-in configuration settings for customizing your SerDes system. From the app toolstrip, go to **CONFIGURATION** tab, and select relevant settings.

Parameter Name	Default Value	Description
Symbol Time (ps)	`100`	
Samples per Symbol	`16`	Choose between `8`, `16`, `32`, `64`, and `128`
Target BER	`1e-6`	
Modulation	`NRZ`	Choose between `NRZ` and `PAM4`.
Signalling	`Signaling`	Choose between `Differential` and `Single Ended`.

Setting Up Transmitter and Receiver

Use the **AnalogOut** subsystem to set up the transmitter.

Use the **AnalogIn** subsystem to set up the receiver.

From the app toolstrip, go to the **BLOCKS** tab, and use the relevant blocks. The app provides the following building blocks:

- FFE
- CTLE
- DFECDR
- CDR
- AGC
- VGA
- SaturatingAmplifier
- PassThrough

Statistical Analysis

From the app toolstrip, go to **ANALYSIS** tab, and select **Add Plots** to perform statistical (Init) analysis. By default, **Auto-Analyze** is selected, and plot results are automatically updated with each change in the SerDes system. You can deselect the **Auto-Analyze**, and update the plot at your preference by clicking the **Analyze** button.

You can view the following plots from the app:

- Pulse Response
- Statistical Eye
- PRBS Waveform
- Contours
- Bathtub
- Report
- BER

Exporting SerDes System

From the app toolstrip, go to **EXPORT** tab. You can either:

- **Export SerDes System to Simulink**

- **Generate MATLAB Code for SerDes System**
- **Make IBIS-AMI Model for SerDes System**

Extended Support with Other Compilers and Products

Note

- If you have Simulink license, you can export Simulink and IBIS-AMI models from the app.
- If you have a supported compiler, you can compile the SerDes system in that compiler from the app. List of supported compilers are: IBIS-AMI Microsoft Visual C++ 2017 v15.0, IBIS-AMI Microsoft Visual C++ 2015 v14.0, IBIS-AMI Microsoft Visual C++ 2013 v12.0, IBIS-AMI MinGW64, and IBIS-AMI GNU gcc/g++ .
- If you have the following licenses: MATLAB Coder™, Simulink Coder, and Embedded Coder®, you can keep your C files during .dll file generation. Otherwise, your C files will be deleted during the .dll file generation.

See Also

Blocks
AGC | CDR | CTLE | DFECDR | FFE | PassThrough | SaturatingAmplifier | VGA

Objects
`serdes.AGC` | `serdes.CDR` | `serdes.CTLE` | `serdes.DFECDR` | `serdes.FFE` | `serdes.PassThrough` | `serdes.SaturatingAmplifier` | `serdes.VGA`

Topics
"Design SerDes System and Export IBIS-AMI Model"
"PCIe4 Transmitter/Receiver IBIS-AMI Model"

Introduced in R2019a