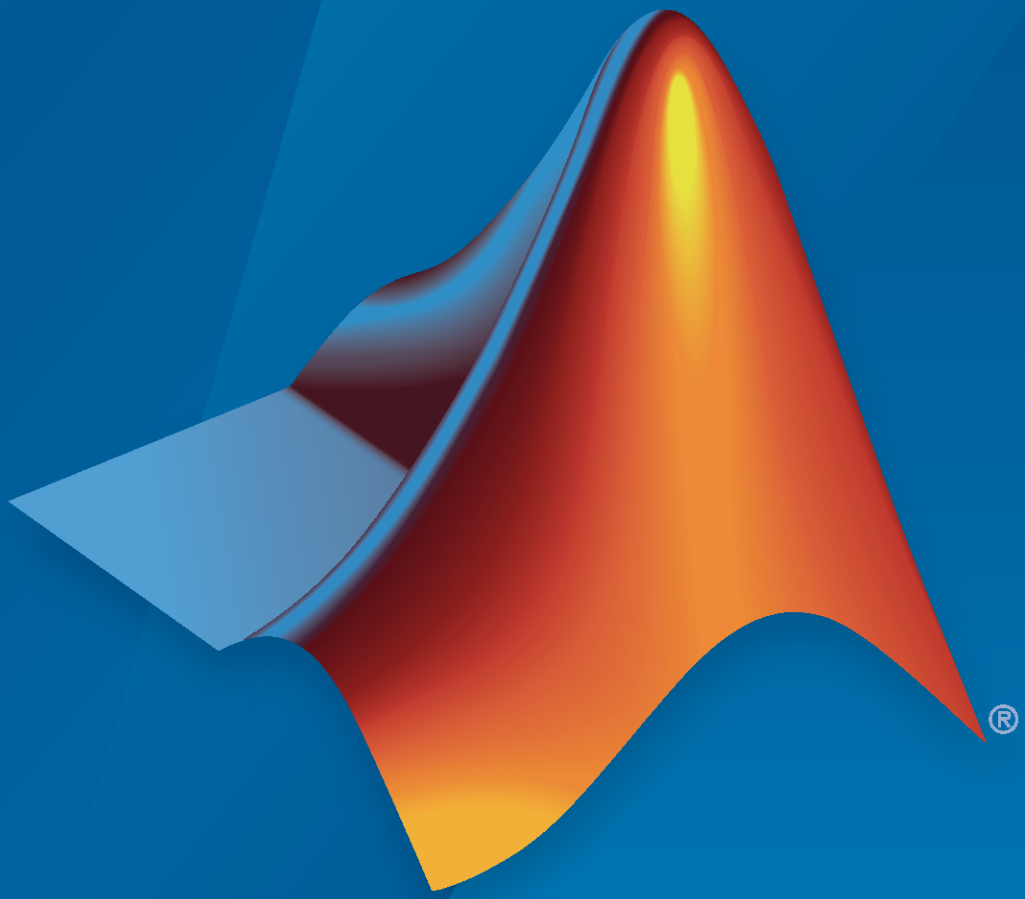
SerDes Toolbox™

Reference



MATLAB® & SIMULINK®

R2020a

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)
September 2019	Online only	Revised for Version 1.1 (Release 2019b)
March 2020	Online only	Revised for Version 1.2 (Release 2020a)

Contents

1 Functions

2 System Objects

3 Blocks

4 Apps

Functions

impulse2pulse

Pulse response from impulse response

Syntax

```
P = impulse2pulse(I,N,dt)
```

Description

`P = impulse2pulse(I,N,dt)` converts the impulse response `I` to a pulse response `P`, given the number of samples per symbol `N` and the uniform sampling interval `dt`.

Examples

Create Pulse Response from Impulse Response

Load the impulse response column matrix from a file.

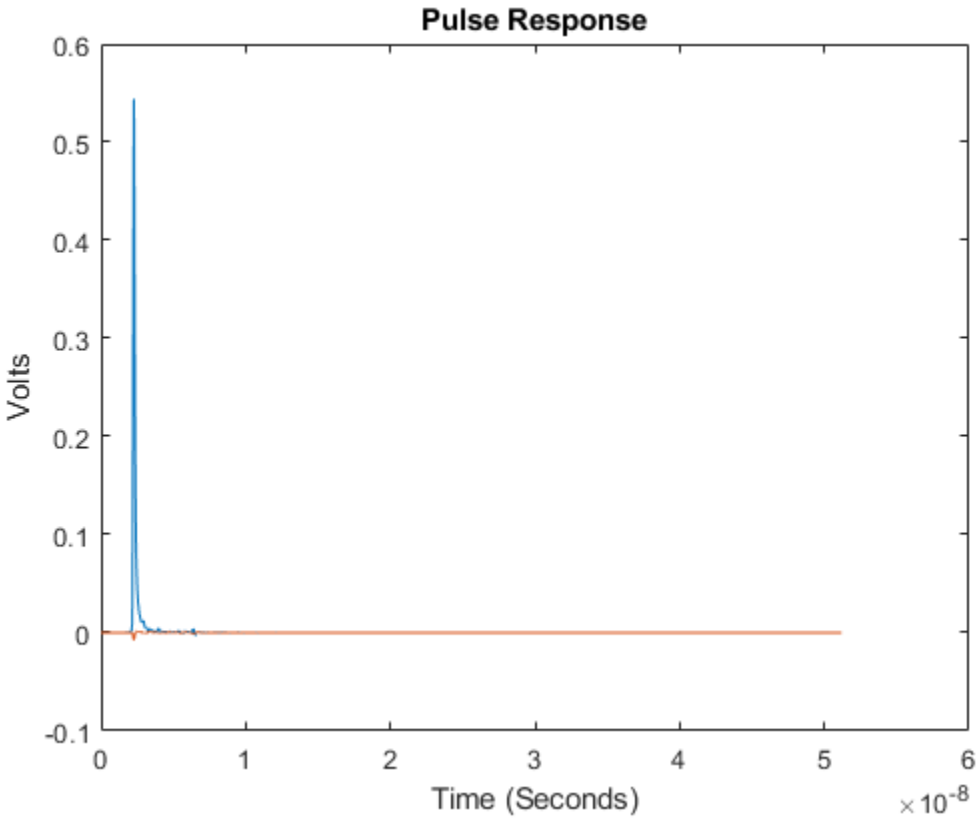
```
load('PulseResponseReflective100ps.mat');
```

Calculate the pulse response from the impulse response.

```
pulse = impulse2pulse(impulse,SamplesPerSymbol,dt);
```

Plot the pulse response.

```
figure
plot(t,pulse)
xlabel('Time (Seconds)')
ylabel('Volts')
title('Pulse Response')
```



Input Arguments

`I` — Impulse response

column matrix

Input impulse response, specified as a column matrix. The first column contains the primary impulse response and the subsequent columns (if any) contain the crosstalk impulse responses.

Data Types: `double`

`N` — Number of samples per symbol

positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

`dt` — Uniform sampling interval

positive real scalar

Uniform timestep of the waveform, specified as a real positive scalar in seconds.

Data Types: `double`

Output Arguments

P — Pulse response

column matrix

Converted pulse response, returned as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`impulse2step` | `pulse2impulse`

Introduced in R2020a

impulse2step

Step response from impulse response

Syntax

```
S = impulse2step(I,dt)
```

Description

`S = impulse2step(I,dt)` converts an impulse response `I` to a step response `S`, given the uniform sample interval `dt`.

Examples

Create Step Response from Impulse Response

Load the impulse response column matrix from a file.

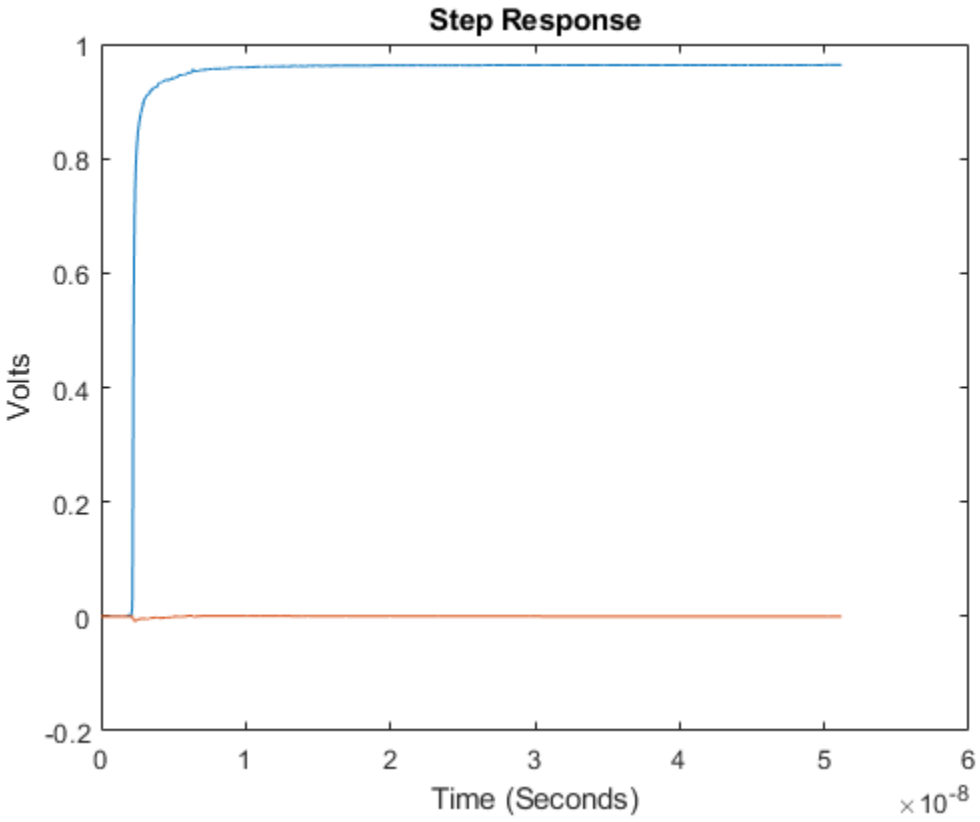
```
load('PulseResponseReflective100ps.mat');
```

Calculate the step response from the impulse response.

```
step = impulse2step(impulse,dt);
```

Plot the step response.

```
plot(t,step)
xlabel('Time (Seconds)')
ylabel('Volts')
title('Step Response')
```



Input Arguments

`I` — Impulse response

column matrix

Input impulse response, specified as a column matrix. The first column contains the primary impulse response and the subsequent columns (if any) contain the crosstalk impulse responses.

Data Types: `double`

`dt` — Uniform sampling interval

positive real scalar

Uniform timestep of the waveform, specified as a real positive scalar in seconds.

Data Types: `double`

Output Arguments

`S` — Step response

column matrix

Converted step response, returned as a column matrix. The first column contains the primary step response and the subsequent columns (if any) contain the crosstalk step responses.

Data Types: `double`

See Also

`impulse2pulse`

Introduced in R2020a

optPulseMetric

Pulse response metric for optimization routines

Syntax

```
metric = optPulseMetric(P,dt,N,B)
```

Description

`metric = optPulseMetric(P,dt,N,B)` calculates the eye height, eye width, eye area, and channel operating margin (COM) metrics at the target bit error rate (BER) level `B`. The function calculates these metrics from the pulse response matrix `P`, using the number of samples per symbol `N`, and the uniform sampling interval `dt`.

If there is no eye at the target BER level `B`, the `optPulseMetric` function increases the BER until a positive height is realized.

The resolution of the `optPulseMetric` function is limited by number of samples per symbol `N`.

The nonreturn to zero (NRZ) eye metrics are estimated at a BER level, where both the center and maximum eye heights are reported. If the eye is not open at the target BER, the BER is increased until a measurable eye contour is found. The actual BER used is reported by the value usedBER reported in the output `metric`.

Examples

Compare optPulseMetric Results to Statistical Eye Analysis

Define the system exploration parameters. Set the target bit error rate (BER) to 1e-9, symbol time to 100 ps, number of samples per symbol to 16, and channel loss to 4 dB.

```
targetBER = 1e-9;
SymbolTime = 100e-12;
N = 16;
dBLoss = 4;
```

Calculate the sample interval.

```
dt = SymbolTime/N;
```

Create the channel model at the specified loss.

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

Define a CTLE System object™.

```
ctle = serdes.CTLE('WaveType','Impulse',...
    'SampleInterval',dt,'SymbolTime',SymbolTime,'Mode',2);
```

Process the channel impulse response with the CTLE System object.

```
impulse = ctle(channelModel.impulse);
```

Convert the impulse response to a pulse response.

```
P = impulse2pulse(impulse,N,dt);
```

Calculate the fast optimization pulse metric.

```
metric = optPulseMetric(P,N,dt,targetBER);
```

Obtain the result of the `optPulseMetric` function throughout the whole eye.

In order to do that, first find the number of complete unit intervals in the pulse response.

```
nUI = floor(size(P,1)/N);
prMatrixSort = sort(abs(reshape(P(1:N*nUI),N,[])).',1,'descend');
```

Then convert the BER into the number of largest pulse response samples to get the `usedBER` contour. The mean height is the first row (largest value) of the sorted pulse response. The noise is the sum of sorted pulse response's next `nBER` sample intervals.

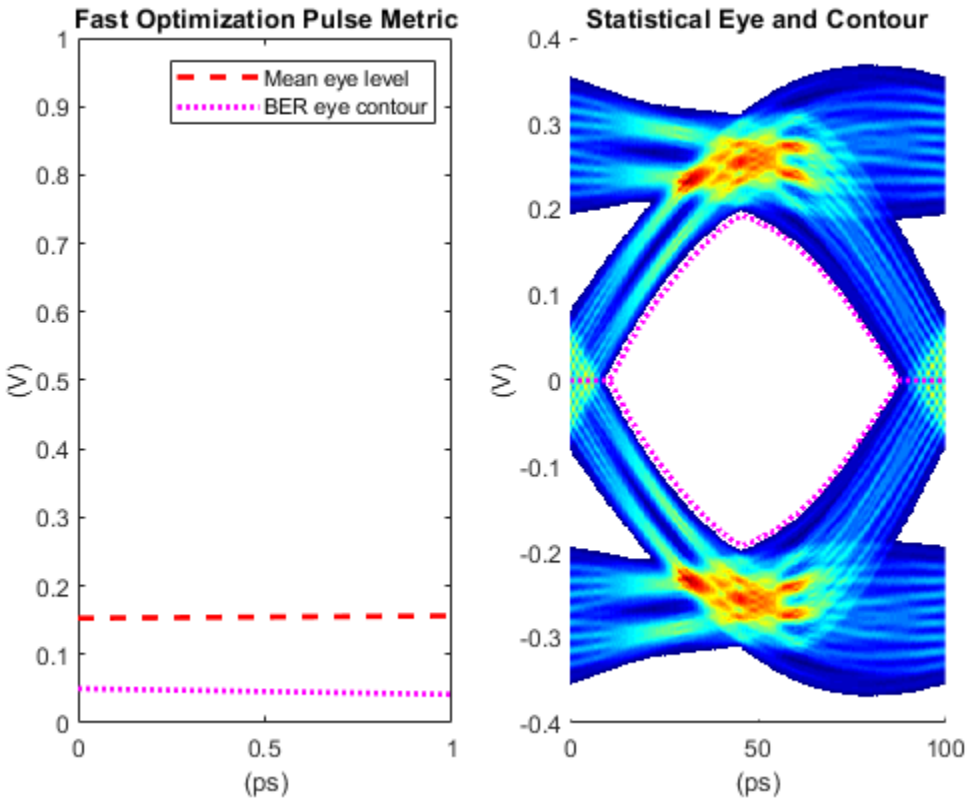
```
nBER = floor(min(abs(log(targetBER)/log(2)),nUI-1));
meanEyeHeight = prMatrixSort(1,:);
Noise =sum (prMatrixSort(2:nBER+1,:),1);
```

Calculate the statistical eye and the eye metrics.

```
[stateye,vh,th] = pulse2stateye(P,N,2);
th2 = th*SymbolTime*1e12;
[~,~,contours,~,EHmax,~,~,~,~,EW,~,~,~,~,eyeAreas,~,COM] = ...
          serdes.utilities.calculatePAMnEye(2,targetBER, ...
          th2(1),th2(end),vh(1),vh(end),stateye);
```

Plot the statistical eye and the results of the `optPulseMetric` function.

```
figure,
ax = axis;
subplot(1,2,1)
th4 = [(0:N-1),NaN,(0:N-1)]*dt*1e12;
plot(th4,[meanEyeHeight,NaN,-meanEyeHeight]/2,'r--',...
         th4,[meanEyeHeight-Noise,NaN,Noise-meanEyeHeight]/2,'m:','linewidth',2)
legend('Mean eye level','BER eye contour')
title('Fast Optimization Pulse Metric')
ylabel('(V)')
xlabel('(ps)')
axis(ax)

subplot(1,2,2)
hold all
imagesc(th2,vh,stateye)
plot(th2,contours,'m:','linewidth',2)
title('Statistical Eye and Contour')
axis('xy');
si_eyecmap = serdes.utilities.SignalIntegrityColorMap;
colormap(si_eyecmap)
xlabel('(ps)')
ylabel('(V)')
```



Compare the results of the statistical eye metrics and the fast optimization pulse metrics.

```
comparisonTable = table([metric.maxEyeHeight*1e3;metric.eyeWidth*1e12;...
          metric.eyeArea*1e12;metric.centerCOM; metric.usedBER], ...
          [EHmax*1e3; EW;eyeAreas; COM; targetBER],...
          'VariableNames',{'optPulseMetric','StatisticalEye'},...
          'RowNames',{'Max Eye Height (mV)','Eye Width (ps)','Eye Area [V*ps]',...
          'Center COM','BER'});
disp(comparisonTable)
```

```
                        optPulseMetric    StatisticalEye
                        ______________    ______________

    Max Eye Height (mV)        379.36            367.69
    Eye Width (ps)              93.75            79.347
    Eye Area [V*ps]            17.902            17.847
    Center COM                 11.597            10.853
    BER                         1e-09             1e-09
```

Input Arguments

P — Pulse response

column vector

Pulse response, specified as a column vector. The pulse response vector can be calculated from the channel impulse response vector.

Data Types: `double`

`dt` — Uniform sampling interval
real positive scalar

Uniform sampling interval of the waveform, specified as a real positive scalar in seconds.

Data Types: `double`

`N` — Number of samples per symbol
real positive scalar

Number of samples per symbol, specified as a real positive scalar.

Data Types: `double`

`B` — Target bit error rate
real positive scalar

Target bit error rate (BER), specified as a real positive scalar.

Data Types: `double`

Output Arguments

`metric` — Result of optPulseMetric function
structure

Result of the `optPulseMetric` function. The result is stored in a structure with fields containing this information:

Field	Description
`maxEyeHeight`	Maximum height of the inner eye in the `usedBER` contour curve.
`maxMeanEyeHeight`	Mean eye level height at the maximum inner eye height position.
`maxCOM`	Channel operating margin (COM) at the maximum inner eye height position.
`eyeArea`	Area inside the `usedBER` contour.
`eyeWidth`	Eye width of the `usedBER` contour.
`centerEyeHeight`	Eye height at the center of the `usedBER` contour.
`centerMeanEyeHeight`	Mean eye level height at the center of the eye.
`centerCOM`	Channel operating margin (COM) at the center of the eye.
`usedBER`	The bit error rate (BER) level used in the metric calculation that gives the nonzero eye height.

Data Types: `struct`

More About

Channel Operating Margin

The channel operating margin (COM) is a ratio between the signal and the noise [1] and is given by the equation:

$$\mathrm{COM} = 20\log_{10}\frac{\mathrm{Signal}}{\mathrm{Noise}}$$

`optPulseMetric` estimates the signal amplitude from the pulse response cursor voltage (the mean eye level height) [2]. The noise amplitude is estimated at a given BER by the sum of intersymbol interference (ISI) voltages.

References

[1] IEEE 802.3bj 93 A.

[2] Common Electrical I/O (CEI) Implementation Agreement OIF-CEI-04.0, section 16.3.10.2.

Extended Capabilities

C/C++ Code Generation

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

See Also

Introduced in R2020a

prbs

Pseudorandom binary sequence

Syntax

```
p = prbs(O,N)
[PRBS,seed] = prbs(O,N)
[PRBS,seed] = prbs(O,N,seed)
```

Description

`p = prbs(O,N)` calculates a pseudorandom binary sequence `P` of order `O` and length `N`.

`[PRBS,seed] = prbs(O,N)` calculates a pseudorandom binary sequence and the seed needed to continue the sequence.

`[PRBS,seed] = prbs(O,N,seed)` calculates a pseudorandom binary sequence and the seed needed to continue the sequence using the `seed` value.

Examples

Create PRBS Pattern of Order 4

Create a vector using a pseudorandom binary sequence (PRBS) pattern of the order 4.

```
O = 4;
N = 2^O-1;
pattern1 = prbs(O,N);
```

Create another vector, this time using a PRBS pattern of order 4 one bit at a time.

```
pattern2 = zeros(1,N);
[pattern2(1),seed] = prbs(O,1);
    for ii = 2:N
        [pattern2(ii),seed] = prbs(O,1,seed);
    end
```

Verify that both patterns are the same.

```
disp(isequal(pattern1,pattern2))
```

```
   1
```

Input Arguments

O — Order of pseudorandom binary sequence

positive integer scalar

Order of the pseudorandom binary sequence pattern, specified as a positive integer scalar.

Data Types: `double`

`N` — Length of pseudorandom binary sequence
positive integer scalar

Length of the pseudorandom binary sequence pattern, specified as a positive integer scalar.

Data Types: `double`

Output Arguments

`PRBS` — Pseudorandom binary sequence pattern
vector

Pseudorandom binary sequence pattern of order `O` and length `N`, returned as a vector.

Data Types: `double`

`seed` — Seed value of pseudorandom binary sequence
vector

Seed value of the pseudorandom binary sequence pattern, returned as a vector. `seed` is used in subsequent calls to continue the PRBS pattern.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`pulse2wave` | `wave2pulse`

Introduced in R2020a

pulse2impulse

Impulse response from pulse response

Syntax

```
I = pulse2impulse(P,N,dt)
```

Description

`I = pulse2impulse(P,N,dt)` converts a pulse response `P` to an impulse response `I`, given the number of samples per symbol `N` and uniform sampling interval `dt`.

The function computes the step response from the pulse response. The derivative of the pulse response is the impulse response.

The function calculates the derivative with zero delay and is reasonably accurate upto one half of the Nyquist rate. But at the Nyquist rate, the derivative calculation diverges rapidly back to zero.

Examples

Create Impulse Response from Pulse Response

Load the pulse response column matrix from a file.

```
load('PulseResponseReflective100ps.mat');
```

Calculate the sample interval.

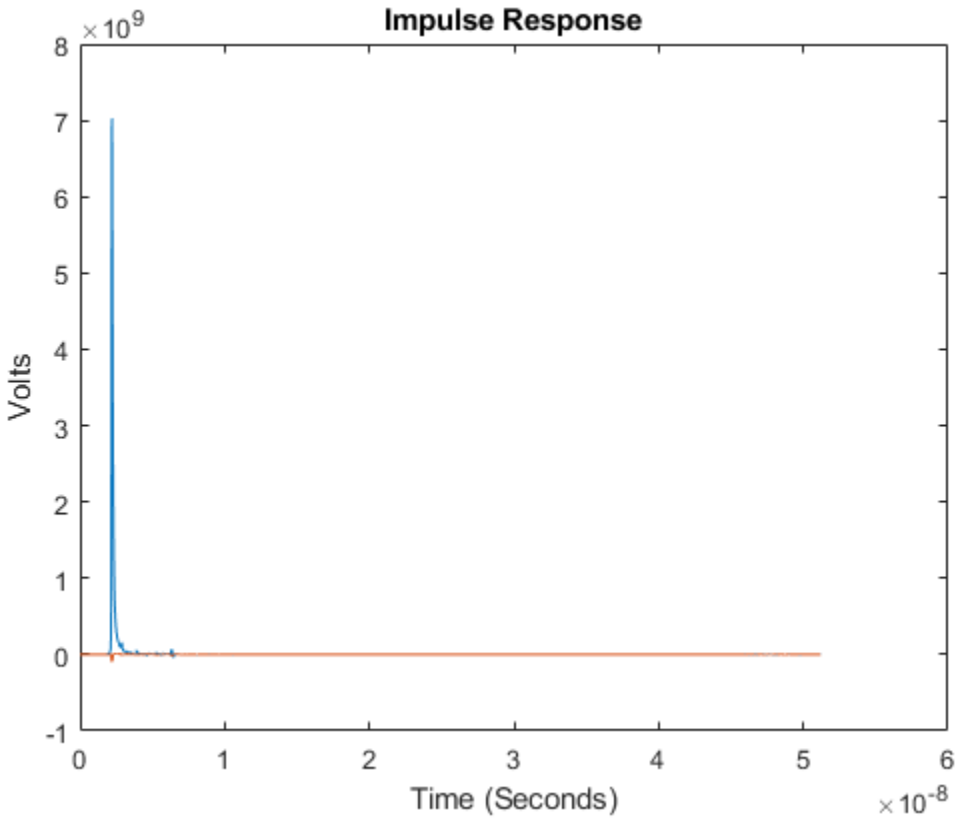
```
dt = SymbolTime/SamplesPerSymbol;
```

Calculate the impulse response.

```
I = pulse2impulse(pulse,SamplesPerSymbol,dt );
```

Plot the impulse response.

```
figure
plot(t,I)
xlabel('Time (Seconds)')
ylabel('Volts')
title('Impulse Response')
```



Input Arguments

P — Pulse response
column matrix

Input pulse response, specified as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

N — Number of samples per symbol
positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

dt — Uniform sampling interval
positive real scalar

Uniform timestep of the waveform, specified as a real positive scalar in seconds.

Data Types: `double`

Output Arguments

`I` — Impulse response

column matrix

Converted impulse response, returned as a column matrix. The first column contains the primary impulse response and the subsequent columns (if any) contain the crosstalk impulse responses.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`impulse2pulse` | `pulse2wave`

Introduced in R2020a

pulse2pda

Peak distortion analysis eye from pulse response

Syntax

```
[E,TH,D] = pulse2pda(P,N,M)
[E,TH,D] = pulse2pda( ___ ,DC)
```

Description

`[E,TH,D] = pulse2pda(P,N,M)` calculates the peak distortion analysis (PDA) eye from a pulse response `P` with `N` samples per symbol and `M` levels of modulation.

`[E,TH,D] = pulse2pda( ___ ,DC)` also maintains the DC offset present in the pulse response `P` if `DC` is set to `true`.

Examples

Create Peak Distortion Analysis Eye from Pulse Response

Load the pulse response column matrix from a file.

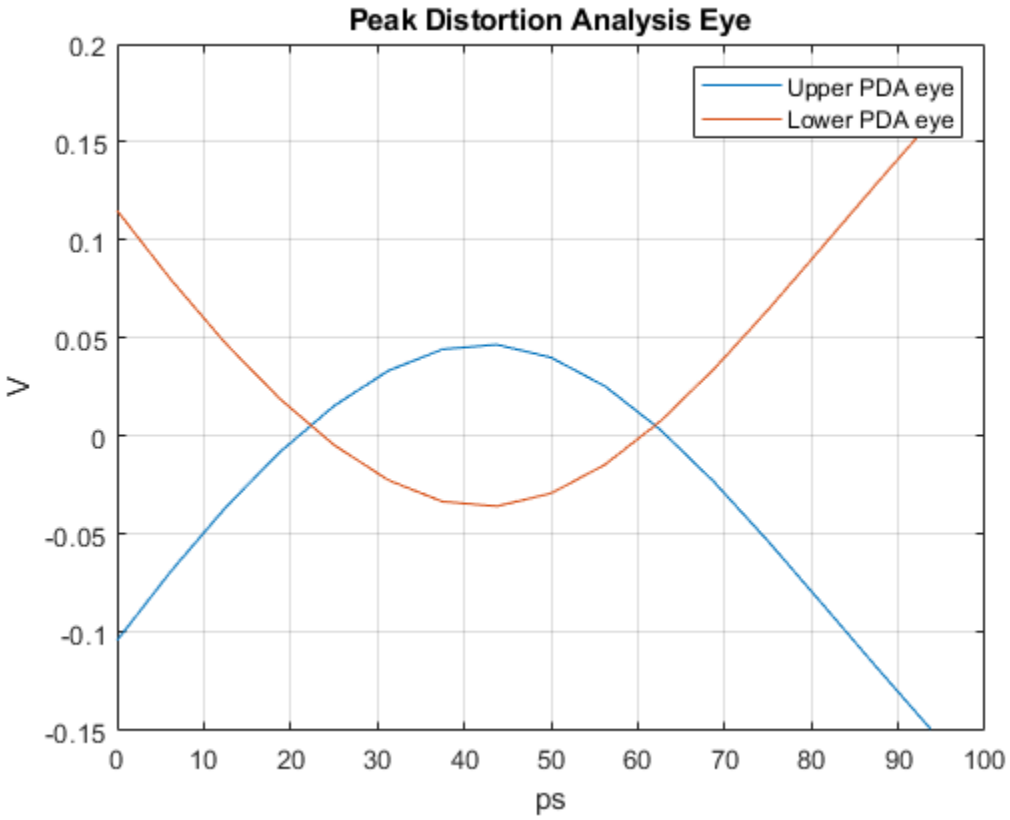
```
load('PulseResponseReflective100ps.mat');
```

Use the nonreturn to zero (NRZ) modulation scheme. Disregard any DC offset present in the pulse response.

```
M = 2;
DC = false;
```

Calculate and plot the peak distortion analysis (PDA) eye.

```
[pdaeye,th] = pulse2pda(pulse,SamplesPerSymbol,M,DC);
figure
t = th*SymbolTime*1e12;
plot(t,pdaeye)
legend('Upper PDA eye','Lower PDA eye')
xlabel('ps')
ylabel('V')
title('Peak Distortion Analysis Eye')
grid on
```



Input Arguments

`P` — Pulse response

column matrix

Input pulse response, specified as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

`N` — Number of samples per symbol

positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

`M` — Number of modulation levels

positive integer scalar greater than or equal to 2

Number of modulation levels, specified as a positive integer scalar. `M` defines the modulation scheme used in the peak distortion analysis (PDA) calculation.

- If `M` = 2, the modulation scheme is nonreturn to zero (NRZ).
- If `M` = 4, the modulation scheme is four-level pulse amplitude modulation (PAM4).

Data Types: `double`

`DC` — Determine whether to maintain DC offsets in pulse response
`true` | `false`

Determine whether to maintain the DC offsets in pulse response `P`.

- If `DC` is set to `true`, the `pulse2pda` function maintains the DC offsets present in the pulse response.
- If `DC` is set to `false`, the `pulse2pda` function disregards the DC offsets present in the pulse response.

Data Types: `double`

Output Arguments

`E` — Peak distortion analysis eye
vector

Peak distortion analysis (PDA) eye, returned as a vector.

Data Types: `double`

`TH` — Time histogram bin centers
vector

Horizontal time histogram bin centers, returned as a vector.

Data Types: `double`

`D` — Upper pattern of PDA eye limit
vector

Upper pattern of the PDA eye limit, returned as a vector.

Data Types: `double`

See Also

`pulse2stateye`

Introduced in R2020a

pulse2stateye

Statistical eye from pulse response

Syntax

```
[S,VH,TH] = pulse2stateye(P,N,M)
[___,S1] = pulse2stateye(P,N,M)
pulse2stateye(P,N,M)
```

Description

`[S,VH,TH] = pulse2stateye(P,N,M)` calculates and plots the statistical eye `S` from a pulse response `P` with `N` samples per symbol and `M` levels of modulation.

The statistical eye is generated by progressively accumulating the histogram of each vertical slice `VH` by perturbing the ideal cursor voltage by the inter-symbol interference (ISI) voltages defined by the pulse response.

`[___,S1] = pulse2stateye(P,N,M)` also returns the symbol transition histograms and crosstalk histogram `S1`.

`pulse2stateye(P,N,M)` calculates and plots the statistical eye to the current figure.

Examples

Plot Statistical Eye from Pulse Response

Load the pulse response column matrix from a file.

```
load('PulseResponseReflective100ps.mat');
```

Use the nonreturn to zero (NRZ) modulation scheme.

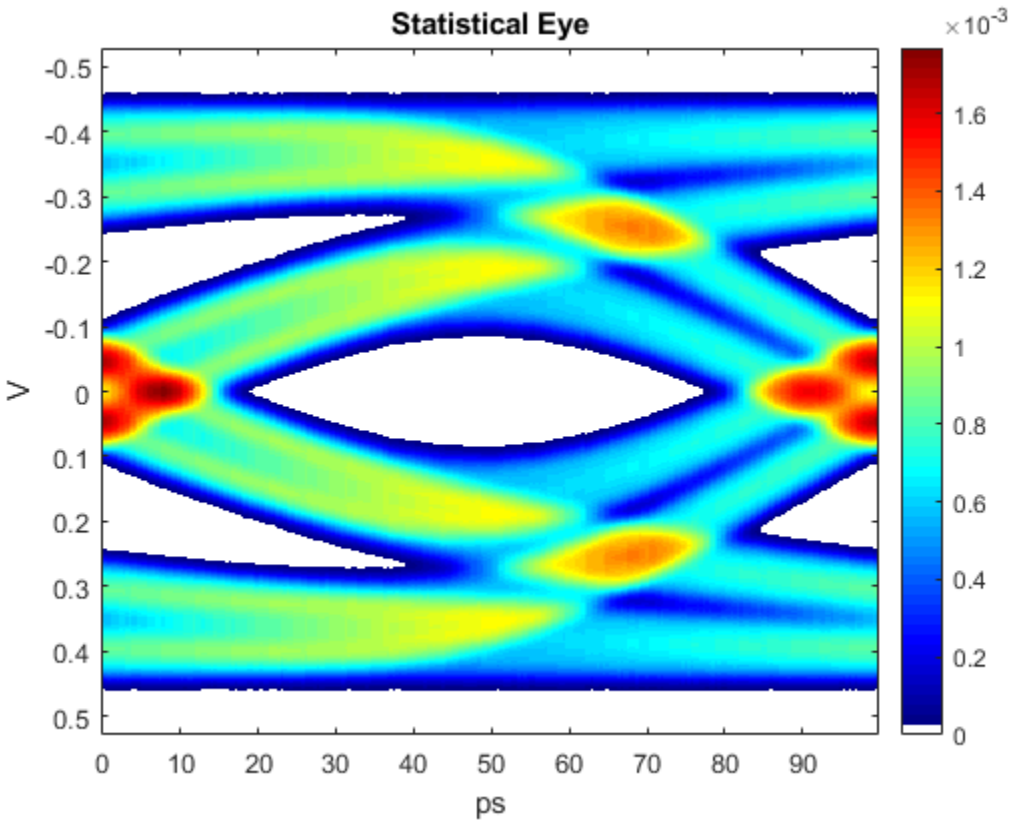
```
modulation = 2;
```

Calculate the statistical eye.

```
[stateye,vh,th] = pulse2stateye(pulse,SamplesPerSymbol,modulation);
```

Visualize the statistical eye using a color map.

```
cmap = serdes.utilities.SignalIntegrityColorMap;
figure,
imagesc(th*SymbolTime*1e12,vh,stateye)
colormap(cmap)
colorbar
xlabel('ps')
ylabel('V')
title('Statistical Eye')
```



Input Arguments

`P` — Pulse response

column matrix

Input pulse response, specified as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

`N` — Number of samples per symbol

positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

`M` — Number of modulation levels

positive integer scalar greater than or equal to 2

Number of modulation levels, specified as a positive integer scalar. `M` defines the modulation scheme used in the statistical eye calculation.

- If `M` = 2, the modulation scheme is nonreturn to zero (NRZ).
- If `M` = 4, the modulation scheme is four-level pulse amplitude modulation (PAM4).

Data Types: `double`

Output Arguments

`S` — Statistical eye matrix

matrix

Statistical eye of the pulse response `P`, returned as a matrix.

Data Types: `double`

`VH` — Voltage histogram bin centers

vector

Vertical voltage histogram bin centers, returned as a vector.

Data Types: `double`

`TH` — Time histogram bin centers

vector

Horizontal time histogram bin centers, returned as a vector.

Data Types: `double`

`S1` — Symbol transition histograms and crosstalk histogram

3-D matrix

Symbol transition histograms and crosstalk histogram, returned as a 3-D matrix.`S1` is used to calculate the statistical eye `S`.

Data Types: `double`

See Also

`pulse2pda`

Introduced in R2020a

pulse2wave

Data pattern waveform from pulse response

Syntax

```
W = pulse2wave(P,D,N)
```

Description

`W = pulse2wave(P,D,N)` converts the pulse response `P` into a voltage waveform `W`, given the symbol pattern `D` and the number of samples per symbol `N`.

The function uses a circular convolution technique to project the pulse response onto the data pattern.

Examples

Create Voltage Waveform from Pulse Response

Load the pulse response column matrix from a file.

```
load('PulseResponseReflective100ps.mat');
```

Find the pulse response column vector.

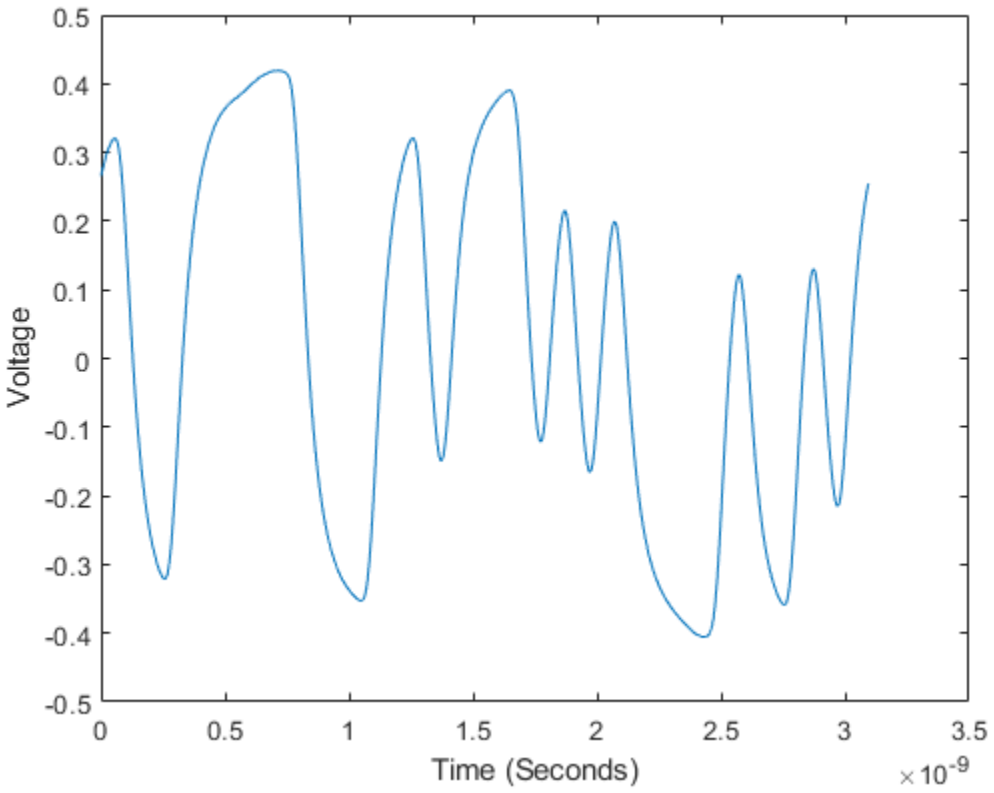
```
P = pulse(:,1) - pulse(1,1);
```

Create the symbol pattern of PRBS order 5.

```
data = prbs(5,2^5-1)-0.5;
```

Calculate and plot the voltage waveform from the pulse response.

```
waveform = pulse2wave(P,data,SamplesPerSymbol);
t = dt*(0:length(waveform)-1);
figure
plot(t,waveform)
xlabel('Time (Seconds)')
ylabel('Voltage')
```



Input Arguments

`P` — Pulse response

column matrix

Input pulse response, specified as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

`D` — Symbol pattern

vector

Symbol pattern, specified as a vector.

Data Types: `double`

`N` — Number of samples per symbol

positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

Output Arguments

`W` — Voltage waveform data pattern
column matrix

Voltage waveform data pattern, returned as a column matrix.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`pulse2impulse` | `wave2pulse`

Introduced in R2020a

wave2pulse

Pulse response from data pattern waveform

Syntax

```
P = wave2pulse(W,D,N)
P = wave2pulse( ___ ,K)
```

Description

`P = wave2pulse(W,D,N)` converts a data pattern waveform `W` to a pulse response `P`, given the symbol pattern `D` and the number of samples per symbol `N`.

`P = wave2pulse( ___ ,K)` converts a data pattern waveform to a truncated pulse response so that the first dimension of `P` is of size (`N`·`K`), where `K` is the desired length of the pulse response.

Examples

Recover Pulse Response from Data Pattern Waveform

Load pulse response column matrix from a file.

```
load('PulseResponseReflective100ps.mat');
```

Select the primary pulse response and remove any DC components.

```
P1 = pulse(:,1) - pulse(1,1);
```

Create a symbol pattern of PRBS order 7.

```
order = 7;
data = prbs(order,2^order-1)-0.5;
```

Create a data pattern waveform from the pulse response.

```
W1 = pulse2wave(P1,data,SamplesPerSymbol);
```

Apply memoryless nonlinearity to the waveform using a `serdes.SaturatingAmplifier` object. The saturating amplifier clips any voltage over `0.4` V.

```
SatAmp = serdes.SaturatingAmplifier('Limit',0.4);
W2 = SatAmp(W1);
```

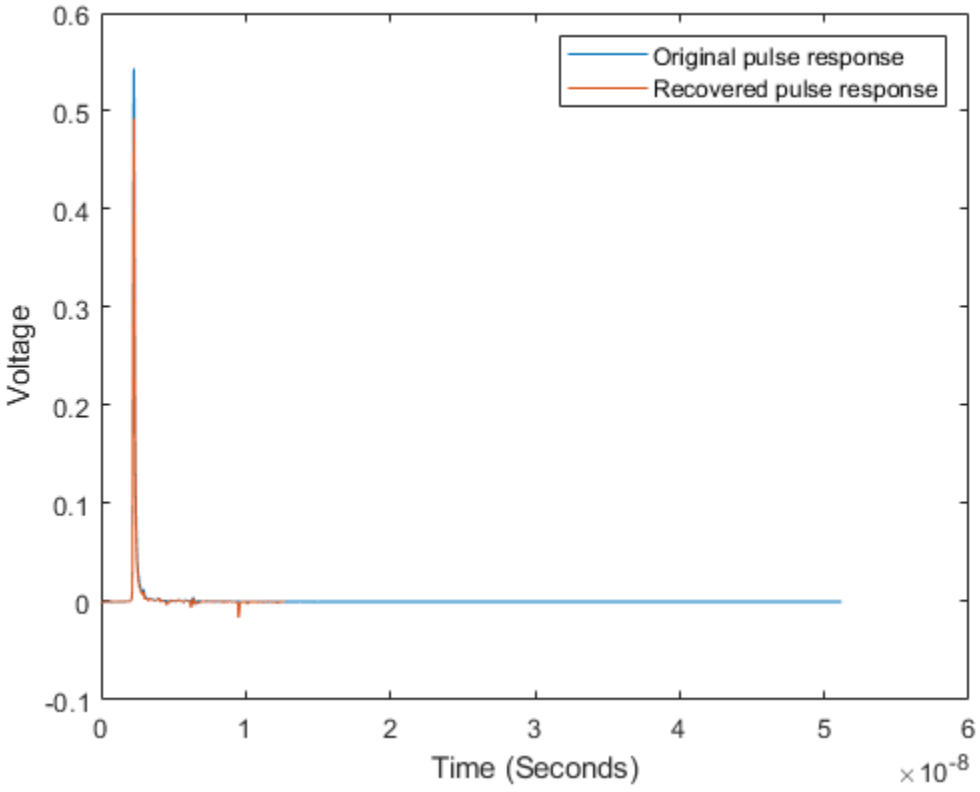
Recover the pulse response from the waveform `W2`.

```
P2 = wave2pulse(W2,data,SamplesPerSymbol);
```

Plot the original and recovered pulse responses.

```
t2 = dt*(0:length(P2)-1);
figure
```

```
plot(t,P1,t2,P2)
xlabel('Time (Seconds)')
ylabel('Voltage')
legend('Original pulse response','Recovered pulse response')
```



Input Arguments

`W` — Data pattern waveform

column vector

Data pattern waveform, specified as a column vector.

Data Types: `double`

`D` — Symbol pattern

vector

Symbol pattern contained within the data pattern waveform `W`, specified as a vector.

Data Types: `double`

`N` — Number of samples per symbol

positive integer scalar

Number of samples per symbol, specified as a positive integer scalar.

Data Types: `double`

K — Desired length of pulse response
positive integer scalar

Desired length of the pulse response, specified as a positive integer scalar in symbols.

Data Types: `double`

Output Arguments

P — Pulse response
column matrix

Converted pulse response, returned as a column matrix. The first column contains the primary pulse response and the subsequent columns (if any) contain the crosstalk pulse responses.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`pulse2wave`

Introduced in R2020a

System Objects

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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `agc = serdes.AGC('TargetRMSVoltage',0.5)` returns an AGC object with an output RMS voltage of 0.5 V.

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) | nonnegative real scalar in the range [0, 10]

Desired RMS voltage of the output waveform, specified as a nonnegative real scalar in the range [0, 10] in volts. Setting the `TargetRMSVoltage` to `0` results in an all zero output.

Data Types: `double`

Advanced

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

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

Data Types: `double`

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

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

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`

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

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

Data Types: `double`

`AveragingLength` — Averaging length for RMS calculation

`100` (default) | positive real integer

Averaging length, specified as a positive real 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:

- `'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.

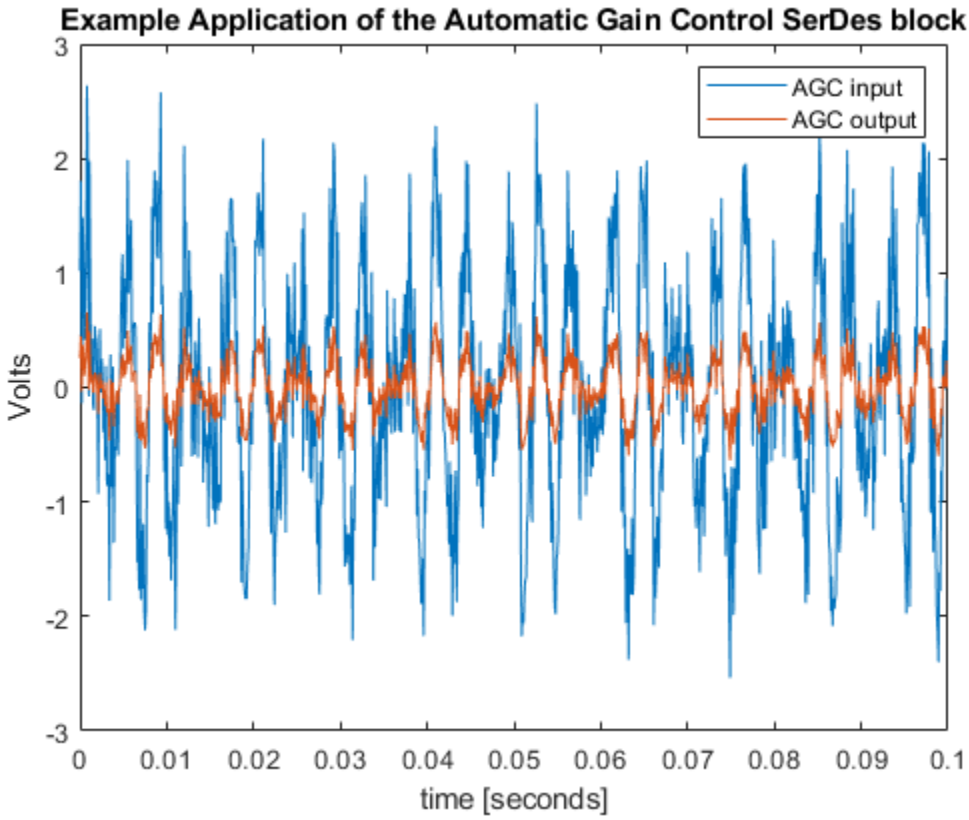
```
Fs = 10000;
L = 1000;
t = (0:L-1)'/Fs;
x = sin(2*pi*250*t) + 0.75*cos(2*pi*340*t);         % Original signal
y = x + .5*randn(size(x));                          % Noisy signal
```

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 first order phase tracking CDR model. For more information, see “Clock and Data Recovery in SerDes System”.

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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `cdr = serdes.CDR('Count',8)` returns a CDR object with early or late CDR count threshold of 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).

`Count` also controls the bandwidth of the CDR which is approximately calculated by using the equation:

$$\text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Early/late threshold count} \cdot \text{Step}}$$

Data Types: `double`

`Step` — Clock phase resolution

`0.0078` (default) | real scalar

Clock phase resolution, specified as a real scalar in fraction of symbol time. `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 ≤300

Reference clock offset impairment, specified as a real scalar ≤300 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. 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:

- `'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));
```

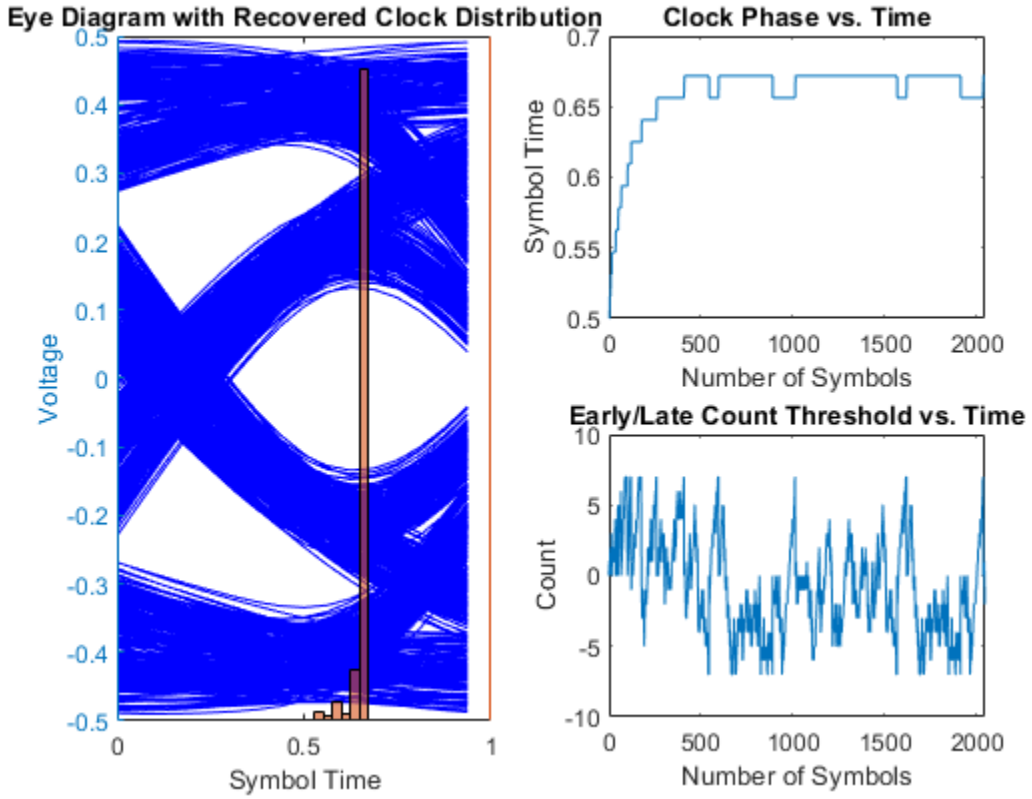
Feed the waveform one sample at a time through the CDR object.

```
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 | DFECDR | `serdes.DFECDR`

Topics
“Clock and Data Recovery in SerDes System”

Introduced in R2019a

serdes.ChannelLoss

Create simple lossy transmission line model

Description

The `serdes.ChannelLoss` System object constructs a lossy transmission line model for use in the **SerDes Designer** app and other exported Simulink® models in the SerDes Toolbox. For more information, see “Analog Channel Loss in SerDes System”.

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 [1].

`ChannelLoss = serdes.ChannelLoss(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. 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 real scalar in dB.

Data Types: `double`

`TargetFrequency` — Frequency of desired channel loss
`1e10` (default) | positive real scalar

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

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

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

Data Types: `double`

`EnableCrosstalk` — Include crosstalk in simulation
`false` (default) | `true`

Set `EnableCrosstalk` to `true` to include crosstalk in the simulation. By default, `EnableCrosstalk` is set to `false`.

`CrosstalkSpecification` — Specify magnitude of near and far end aggressors
`CEI-28G-SR` (default) | `CEI-25G-LR` | `CEI-28G-VSR` | `100GBASE-CR4` | `Custom`

Specify the magnitude of the near and far end aggressors. You can choose to include maximum allowed crosstalk for specifications such as `100GBASE-CR4`, `CEI-25G-LR`, `CEI-28G-SR`, `CEI-28G-VSR`, or you can specify your own custom crosstalk integrated crosstalk noise (ICN) level.

`fb` — Baud rate for ICN calculation
`14e9` (default) | positive real scalar

Baud rate used for integrated crosstalk noise (ICN) calculation, specified as a positive real scalar in hertz. `fb` is the inverse of the symbol time.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true`.

Data Types: `double`

`FEXTICN` — Desired integrated noise level of far end aggressor
`15e-3` (default) | nonnegative real scalar

Desired integrated crosstalk noise (ICN) level of the far end aggressor, specified as a nonnegative real scalar in volts. ICN specifies the strength of the crosstalk.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

Data Types: `double`

`Aft` — Amplitude factor of far end crosstalk aggressor
`1.200` (default) | positive real scalar

Amplitude factor of the far end crosstalk aggressor, specified as a positive real scalar in volts.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

Data Types: `double`

`Tft` — Rise time of far end crosstalk aggressor

`9.6e-12` (default) | positive real scalar

Rise time of the far end crosstalk aggressor, specified as a positive real scalar in seconds.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

Data Types: `double`

`NEXTICN` — Desired integrated noise level of near end aggressor

`10e-3` (default) | nonnegative real scalar

Desired integrated crosstalk noise (ICN) level of the near end aggressor, specified as a nonnegative real scalar in volts. ICN specifies the strength of the crosstalk.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

Data Types: `double`

`Ant` — Amplitude factor of near end crosstalk aggressor

`1.200` (default) | positive real scalar

Rise time of the near end crosstalk aggressor, specified as a positive real scalar in seconds.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

Data Types: `double`

`Tnt` — Rise time of near end crosstalk aggressor

`9.6e-12` (default) | positive real scalar

Rise time of the near end crosstalk aggressor, specified as a positive real scalar in seconds.

Dependencies

This property is only available when `EnableCrosstalk` is set to `true` and `CrosstalkSpecification` is set to `Custom`.

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 “Analog Channel Loss in SerDes System”.

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

Processing Ideal Sinusoid Using ChannelLoss Model

This example shows how to process an ideal sinusoidal input waveform with the ChannelLoss model and check that it modifies the amplitude of the waveform in a reasonable way.

Define the system parameters. Use a symbol time of 100 ps with 8 samples per symbol. The amplitude of the input signal is 1 V. The channel loss is 3 dB.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 8;
a0 = 1;
Loss = 3;
```

Calculate the sample interval. Define a time vector that is 30 symbols long.

```
dt = SymbolTime/SamplesPerSymbol;
t = (0:SamplesPerSymbol*30)*dt;
```

Create the sinusoidal input waveform.

```
F = 1/SymbolTime/2;      %Fundamental frequency
inputWave = a0*sin(2*pi*F*t);
```

Create the `channelModel` object at the specified loss for near ideal transmitter and receiver termination.

```
channelModel = serdes.ChannelLoss('Loss',Loss,'dt',dt,...
      'TargetFrequency',F,'TxR',50,'TxC',1e-14,...
      'RxR',50,'RxC',1e-14);
```

Process the input waveform using the `channelModel` object.

```
 outputWave = channelModel(inputWave);
```

Calculate the output amplitudes.

```
a1 = max(outputWave);                              %Output amplitude
aideal = a0*10^(-abs(channelModel.Loss)/20);       %Theoretical output amplitude
```

Generate the frequency response.

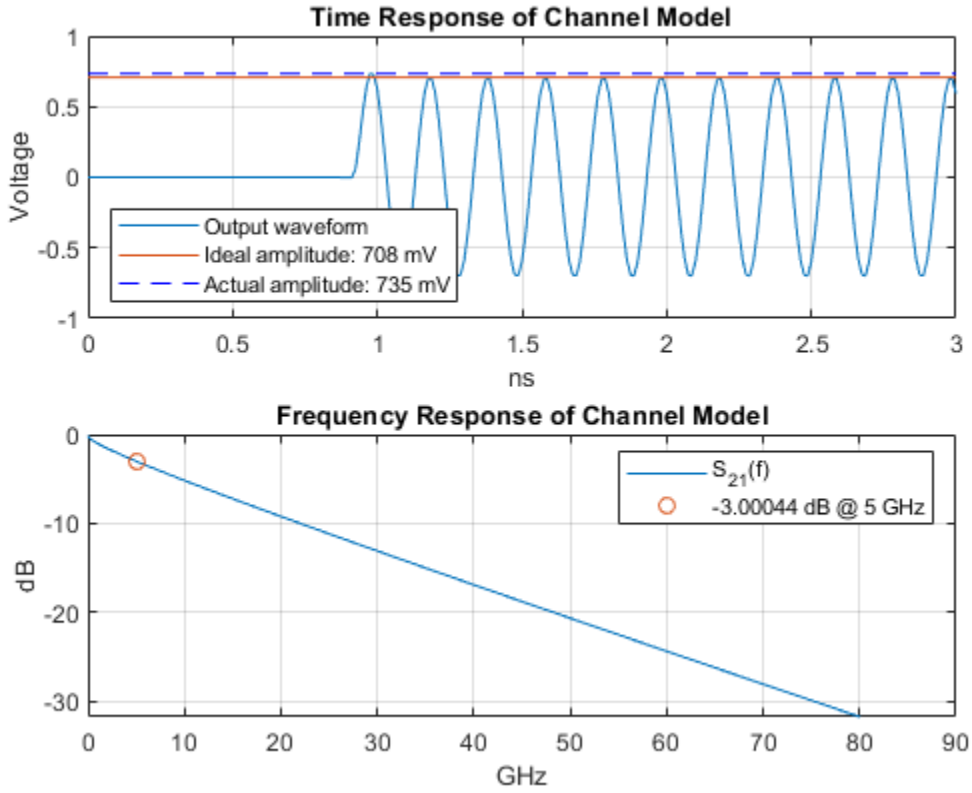
```
s21 = channelModel.s21;
f = (0:length(s21)-1)*channelModel.dF;
```

Determine the loss at the target frequency of the frequency response.

```
f1 = find(f>channelModel.TargetFrequency,1,'first');
LossAtTarget = interp1(f(f1-1:f1),db(s21(f1-1:f1)),channelModel.TargetFrequency);
```

Plot the time and frequency response of the channel model.

```
tns = t*1e9;
thline = [tns(1),tns(end)];
fghz = f*1e-9;
figure
subplot(211)
plot(tns,outputWave,thline,aideal*[1 1],thline,a1*[1 1],'b--'),
grid on
xlabel('ns'),ylabel('Voltage')
title('Time Response of Channel Model')
legend('Output waveform',...
sprintf('Ideal amplitude: %g mV',round(aideal*1e3)),...
sprintf('Actual amplitude: %g mV',round(a1*1e3)),'Location','southwest')
subplot(212)
plot(fghz,db(s21),...
channelModel.TargetFrequency*1e-9,LossAtTarget,'o')
title('Frequency Response of Channel Model')
legend('S_{21}(f)',sprintf('%g dB @ %g GHz',LossAtTarget,channelModel.TargetFrequency*1e-9))
grid on
xlabel('GHz')
ylabel('dB')
```

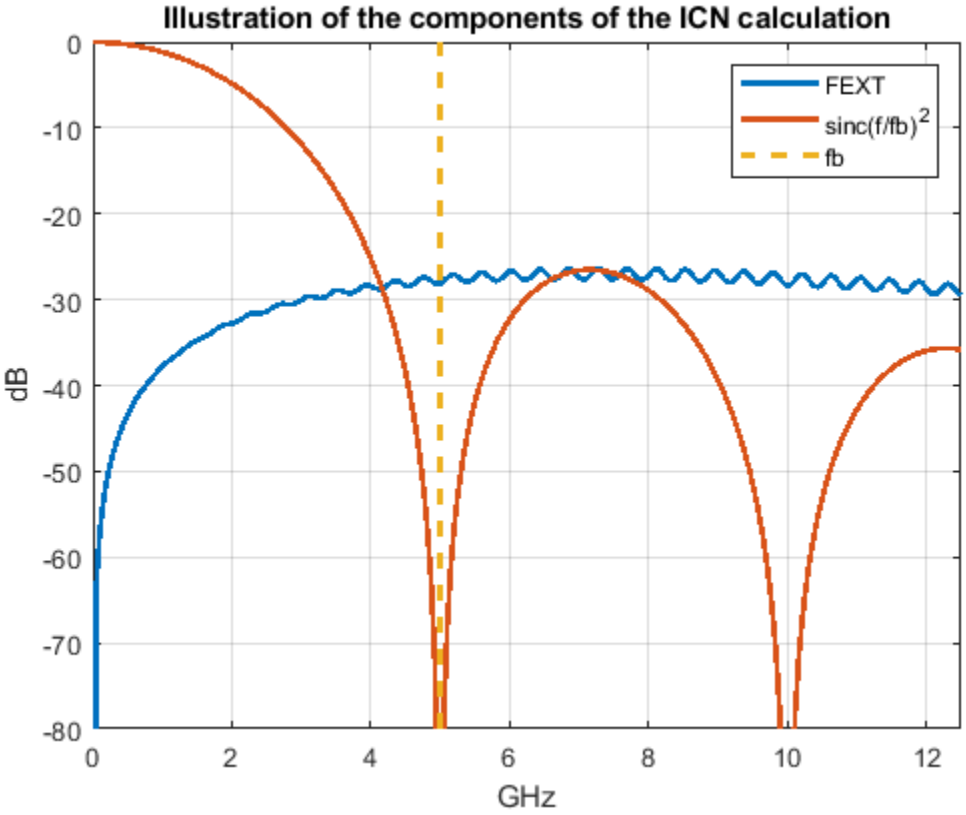


More About

Integrated Crosstalk Noise (ICN)

ICN is a frequency domain metric where the crosstalk is multiplied by a weighting function and then numerically integrated from 50 MHz to the baud rate (f_b). If there are multiple aggressors, their power are summed together before combining with the weighting function.

The time domain signal does not excite all frequencies evenly. The power spectral density (PSD) of a baseband time domain excitation follows a sinc-squared type response. The weighting function mimics the excitation of the PSD and shapes the PSD by including the effects of the receiver bandwidth and the transmitter rise time.



The total ICN is calculated by root-sum-squaring the FEXT ICN and NEXT ICN values together.

$$W_{ft}(f) = \left(\frac{A_{ft}^2}{4f_b}\right)\mathrm{sinc}^2\left(\frac{f}{f_b}\right)\left[\frac{1}{1+\left(f/f_{ft}\right)^4}\right]\left[\frac{1}{1+\left(f/f_{ft}\right)^8}\right]$$

$$\sigma_{fx} = \left(2\Delta f\sum_n W_{ft}(f_n)10^{-MDFEXT_{loss}(f_n)/10}\right)^{1/2}$$

$$\sigma_{nx} = \left(2\Delta f\sum_n W_{ft}(f_n)10^{-MDNEXT_{loss}(f_n)/10}\right)^{1/2}$$

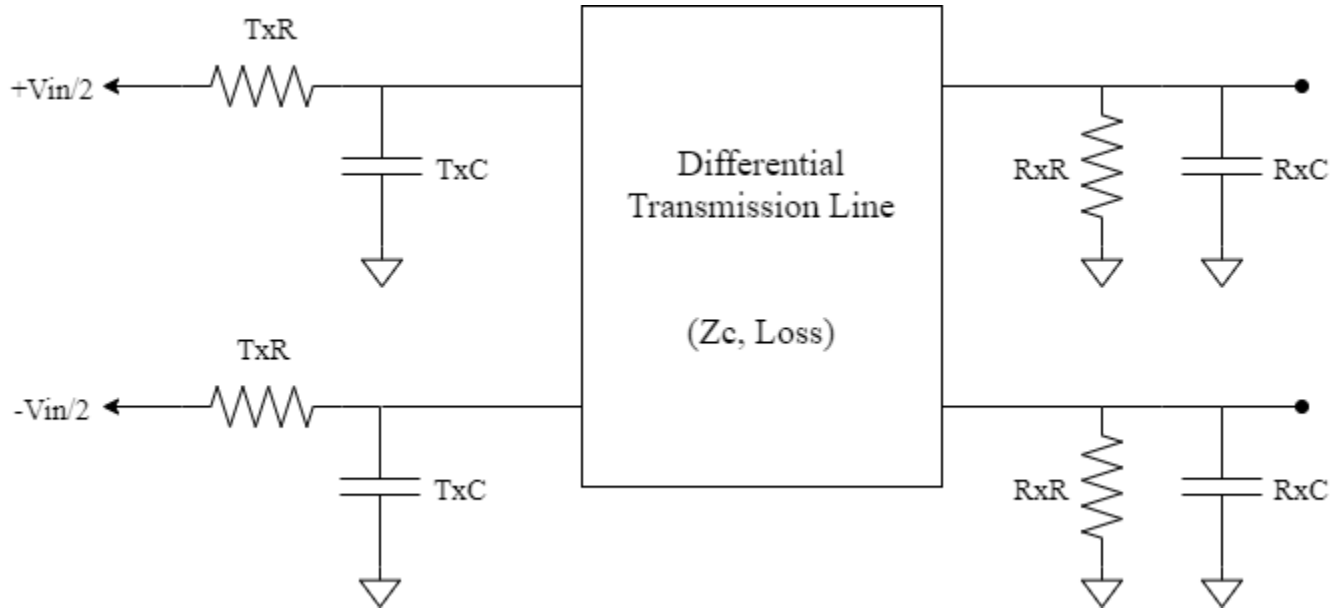
$$\sigma_x = \sqrt{\sigma_{fx}^2 + \sigma_{nx}^2}$$

Algorithms

Creating Transmission Line Model

To obtain a lossy printed circuit board (PCB) transmission line (T-line) model with a given Loss at the `TargetFrequency`, two T-lines of length 100 mm and 150 mm are created and loss evaluated at the Target Frequency. These two data points are used to extrapolate to the transmission line length needed to achieve the requested loss. The transmission line model is a analytic equation based on the method described in [1].

This transmission line, with the requested loss is then combined with the Tx and Rx single ended termination resistance and capacitance as illustrated below:



Creating Far End Crosstalk

The impact felt on a victim line from a far end crosstalk aggressor is proportional to the rate of change of the aggressor waveform [2]. So, you can estimate the shape of a FEXT time domain signal with the derivative of the through response lossy impulse response.

$$I_{\mathrm{FEXT}}(t) = k_{\mathrm{FEXT}}\frac{dI(t)}{dt}$$

where, k_{FEXT} is a scale factor that scales the $I_{\mathrm{FEXT}}(t)$ so that it has user specified ICN value.

To calculate the ICN of the signal, transform the signal to frequency domain using Fourier transform.

$$H_{\mathrm{FEXT}}(f) = \mathscr{F}[I_{\mathrm{FEXT}}(t)]$$

The magnitude of the scale factor k_{FEXT} is: $k_{\mathrm{FEXT}}(f) = -\frac{ICN_{\mathrm{FEXT}}}{\mathbb{ICN}_{\mathrm{FEXT}}(H_{\mathrm{FEXT}}(f))}$,

where $\mathbb{ICN}$ is the integrated crosstalk noise operator.

The sign of k_{FEXT} is negative since in typical transmission lines in inhomogeneous dielectrics, the inducting coupling is generally greater than capacitive coupling. As a result, the forward crosstalk pulse has the opposite magnitude from the magnitude of the aggressor signal.

Creating Near End Crosstalk

To calculate the near end crosstalk, note that the frequency domain NEXT response is similar in shape (not in magnitude) to the victim's return loss (S_{11} or S_{11}).

$$H_{\mathrm{NEXT}}(f) = k_{\mathrm{NEXT}} \cdot S_{11}(f)$$

Then the scale factor k_{NEXT} is: $k_{\mathrm{NEXT}} = -\frac{ICN_{\mathrm{NEXT}}}{\mathbb{ICN}(S_{11}(f))}$

And the time domain NEXT signal is derived from the inverse Fourier transform.

$I_{\mathrm{NEXT}}(t) = \mathscr{F}^{-1}[k_{\mathrm{NEXT}} \cdot S_{11}(f)]$

References

[1] IEEE 802.3bj-2014. "IEEE Standard for Ethernet Amendment 2: Physical Layer Specifications and Management Parameters for 100 Gb/s Operation Over Backplanes and Copper Cables." https://standards.ieee.org/standard/802_3bj-2014.html.

[2] Stephen Hall and Howard Heck. *Advanced Signal Integrity for High-Speed Digital Designs*. Hoboken, NJ: Wiley Press, 2009.

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

Analog Channel | Configuration | **SerDes Designer**

Topics

“Analog Channel Loss in SerDes System”

Introduced in R2019a

serdes.CTLE

Continuous time linear equalizer (CTLE) or peaking filter

Description

The `serdes.CTLE` System object applies a linear peaking filter to equalize a sample-by-sample input signal or to analytically process an impulse response vector input signal. The equalization process reduces distortions resulting from lossy channels. The filter is a real one-zero two-pole (1z/2p) filter, unless you define the gain-pole-zero (GPZ) matrix.

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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. 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 the Init subsystem calls to the `serdes.CTLE`. The `serdes.CTLE` determines the CTLE transfer function for the best eye height opening and applies the transfer function to the input waveform for time domain simulation. This optimized transfer function is used by the CTLE for entire time domain simulation. For more information about the Init subsystem, see “Statistical Analysis in SerDes Systems”
		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.

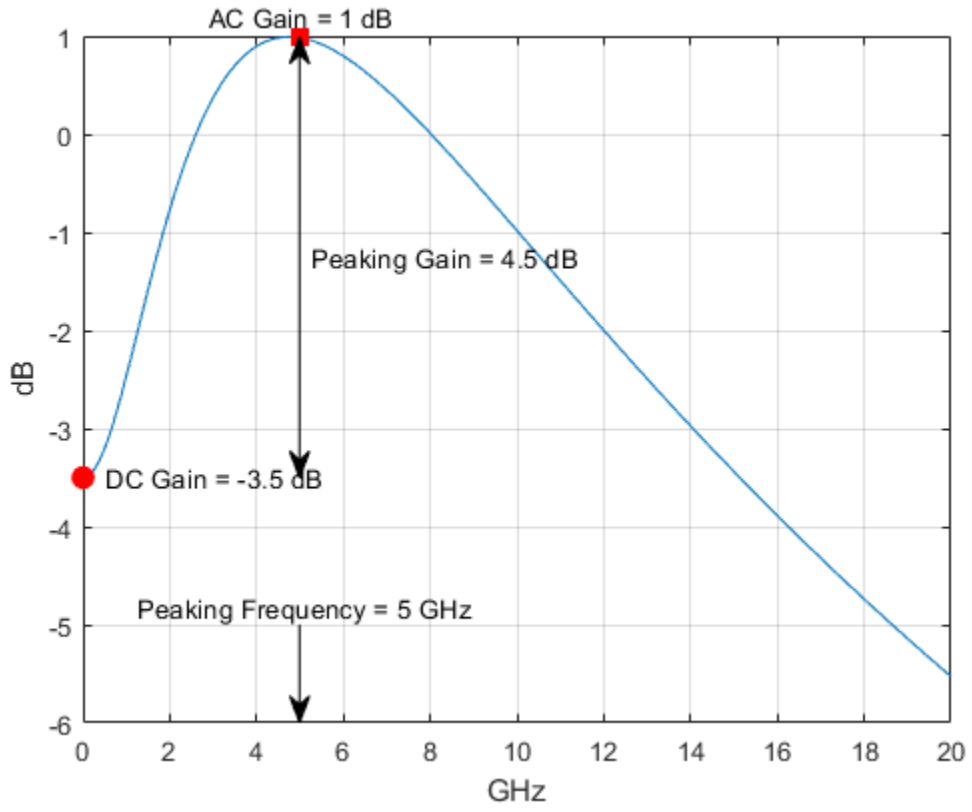
Data Types: `double`

`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. There are five inputs which can be used to define the CTLE transfer function family: “DCGain” on page 2-0 , “PeakingGain” on page 2-0 , “ACGain” on page 2-0 , “PeakingFrequency” on page 2-0 , and “GPZ” on page 2-0 .

You can define the CTLE response from any two of the three gains and peaking frequency or you can define the GPZ matrix for the CTLE.



- Select `'DC Gain and Peaking Gain'` to specify CTLE response from `DCGain`, `PeakingGain`, and `PeakingFrequency`.
- `'DC Gain and AC Gain'` to specify CTLE response from `DCGain`, `ACGain`, and `PeakingFrequency`.
- `'AC Gain and Peaking Gain'` to specify CTLE response from `ACGain`, `PeakingGain`, and `PeakingFrequency`.
- `'GPZ Matrix'` to specify CTLE response 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 scalar, it is converted to match the length of `ACGain`, `DCGain`, and `PeakingGain` by scalar expansion. 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 scalar, it is converted to match the length of `PeakingFrequency`, `ACGain`, and

`PeakingGain` by scalar expansion. 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 scalar, it is converted to match the length of `PeakingFrequency`, `ACGain`, and `DCGain` by scalar expansion. 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 scalar, it is converted to match the length of `PeakingFrequency`, `DCGain`, and `PeakingGain` by scalar expansion. 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 the first column and then poles and zeros in alternating columns. The poles and zeros are specified in Hz. Additional rows in the matrix define additional configurations, which can be selected using the `ConfigSelect`.

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`
Complex Number Support: Yes

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.

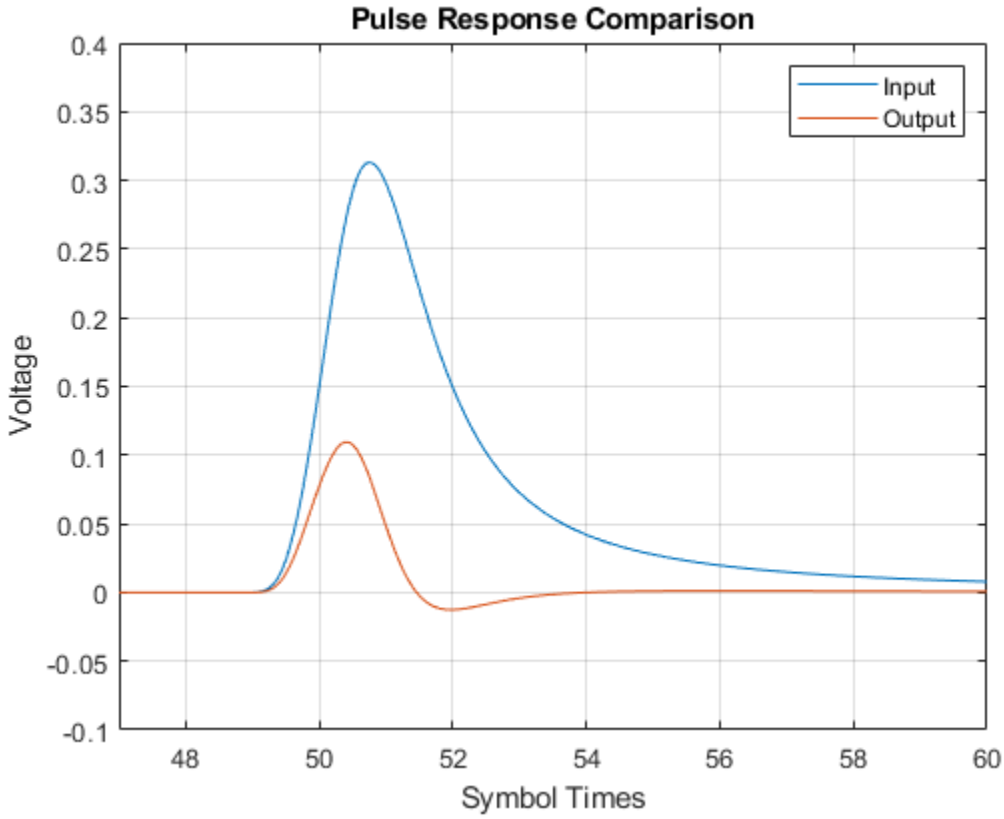
```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

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.

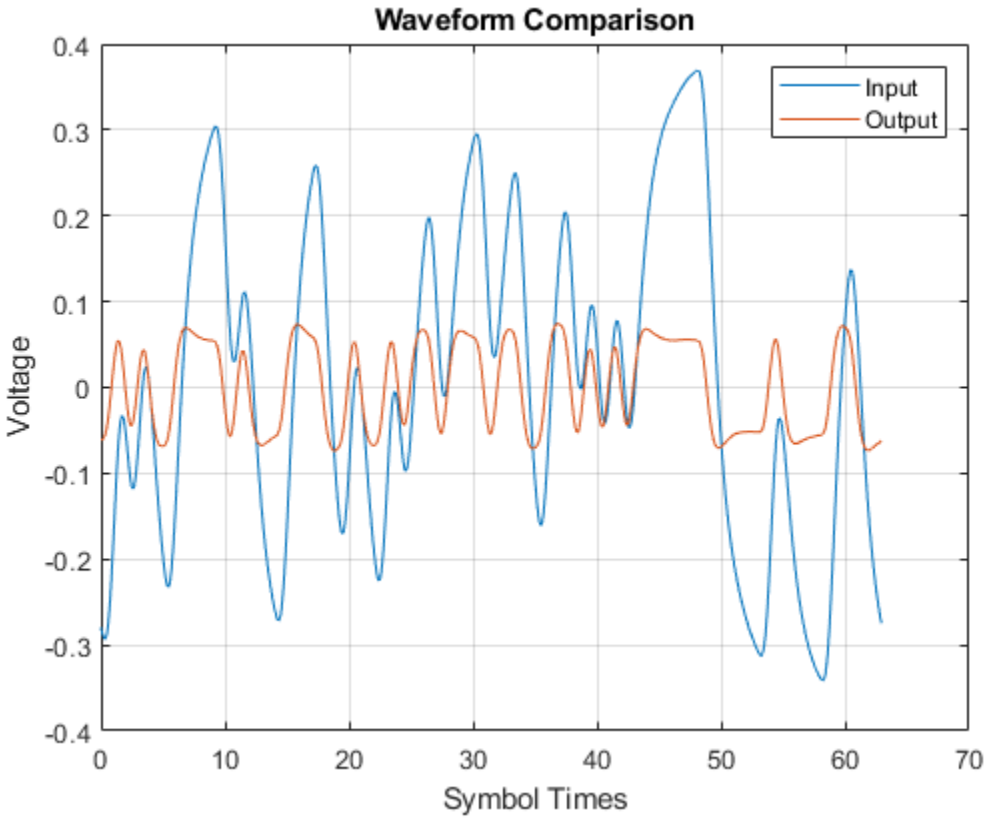
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

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

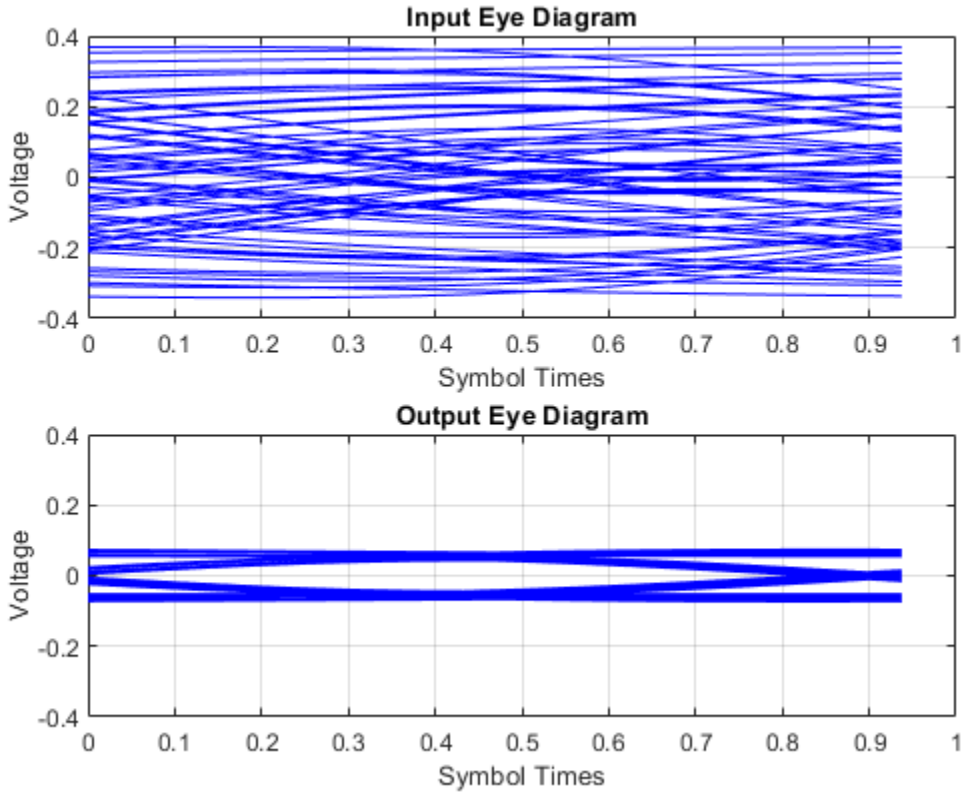
```
figure
plot(t,pulseIn,t,pulseOut)
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('Symbol Times'),ylabel('Voltage')
grid on
axis([47 60 -0.1 0.4])
```




```
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.

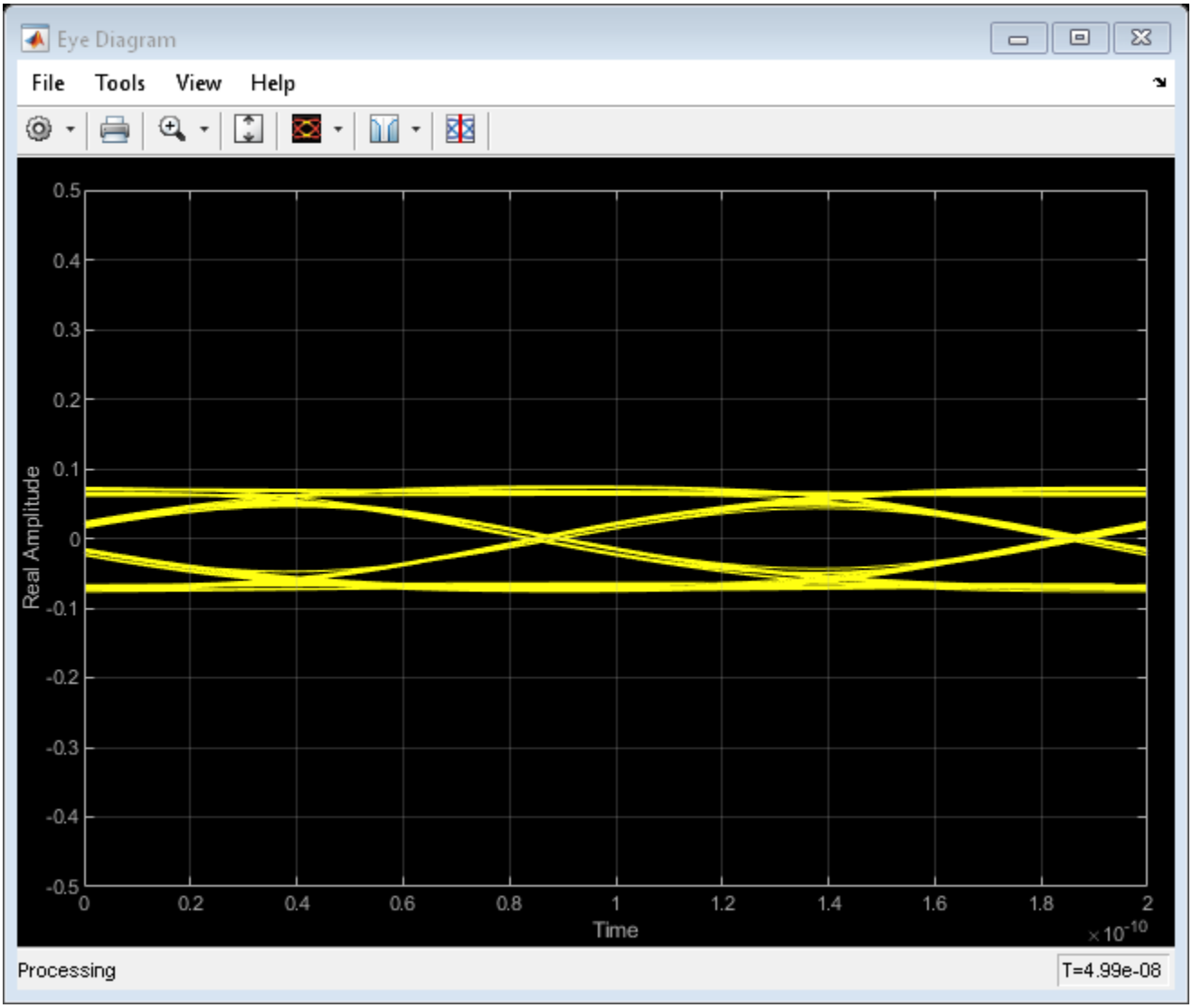
```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

Initialize PRBS generator.

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

Loop through one symbol at 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 CTLE
      for jj = 1:SamplesPerSymbol
          outwave(jj) = CTLE(y(jj));
      end

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



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 DFE modifies baseband signals to minimize the intersymbol interference (ISI) at the clock sampling times. The DFE samples data at each clock sample time and adjusts the amplitude of the waveform by a correction voltage.

For impulse response processing, the hula-hoop algorithm is used to find the clock sampling locations. The zero-forcing algorithm is then used to determine the *N* correction factors necessary to have no ISI at the *N* subsequent sampling locations, where *N* is the number of DFE taps.

For sample-by-sample processing, the clock recovery is accomplished by a first order phase tracking model. The bang-bang phase detector utilizes the unequalized edge samples and equalized data samples to determine the optimum sampling location. The DFE correction voltage for the *N*-th tap is adaptively found by finding a voltage that compensates for any correlation between two data samples spaced by *N* symbol times. This requires a data pattern that is uncorrelated with the channel ISI for correct adaptive behavior.

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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. 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.
`2`	adapt	The Init subsystem calls to the `serdes.DFECDR`. The `serdes.DFECDR` finds the optimum DFE tap values for the best eye height opening for statistical analysis. During time domain simulation, DFECDR uses the adapted values as the starting point and applies them to the input waveform. For more information about the Init subsystem, see “Statistical Analysis in SerDes Systems”

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. 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`

`MinimumTap` — Minimum value of adapted taps

`-1` (default) | real scalar | real-valued row vector

Minimum value of the adapted taps, specified as a real scalar or a real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the `TapWeights`.

Data Types: `double`

`MaximumTap` — Maximum value of adapted taps

`1` (default) | nonnegative real scalar | nonnegative real-valued row vector

Maximum value of the adapted taps, specified as a nonnegative real scalar or a nonnegative real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the `TapWeights`.

Data Types: `double`

`EqualizationGain` — Controls DFE tap weight update rate
`9.6e-5` (default) | positive real scalar

Controls DFE tap weight update rate, specified as a unitless nonnegative real 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) | nonnegative real scalar | nonnegative real-valued row vector

DFE adaptive step resolution, specified as a nonnegative real scalar or a nonnegative real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the `TapWeights`.

`EqualizationStep` specifies the minimum DFE tap change from one time step to the next to mimic hardware limitations. 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 greater than 4

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer greater than 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).

`Count` also controls the bandwidth of the CDR which is approximately calculated by using the equation:

$$\text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Early/late threshold count} \cdot \text{Step}}$$

Data Types: `double`

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

Clock phase resolution, specified as a real scalar in fraction of symbol time. `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 [−300, 300]

Reference clock offset impairment, specified as a real scalar in the range [−300, 300] in parts per million (ppm). `ReferenceOffset` is the deviation between transmitter oscillator frequency and receiver oscillator frequency.

Data Types: `double`

`Sensitivity` — Sampling latch metastability voltage
`0` (default) | real scalar

Sampling latch metastability 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:

- `'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.

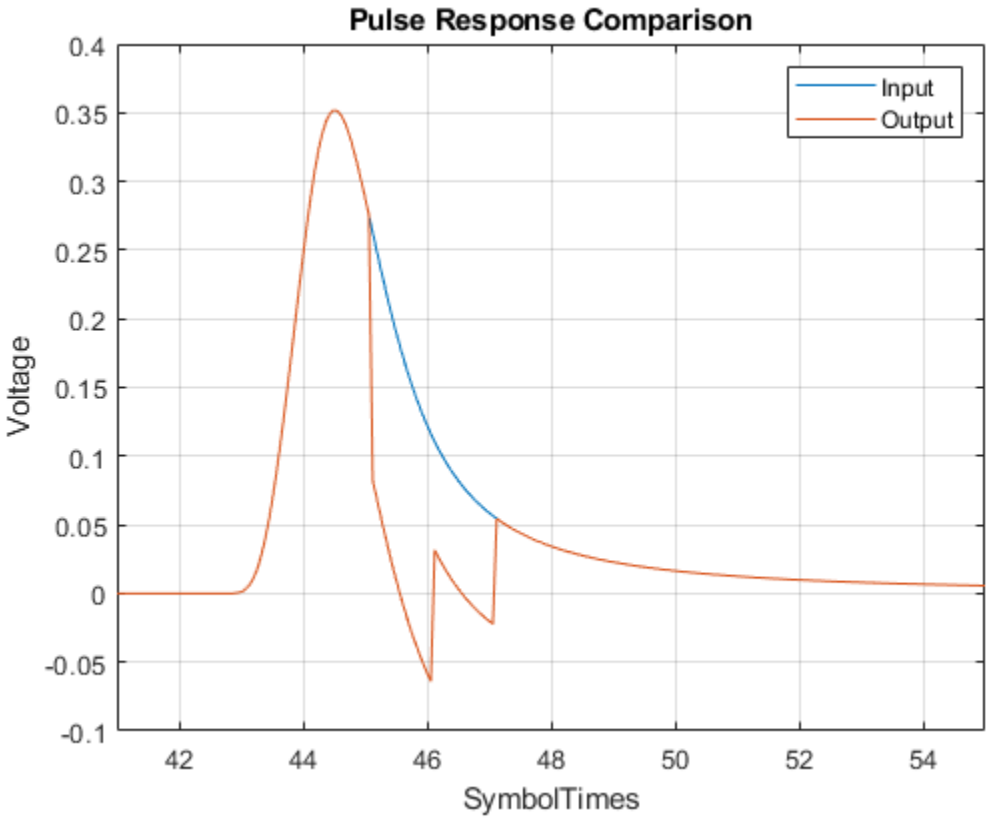
```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

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.

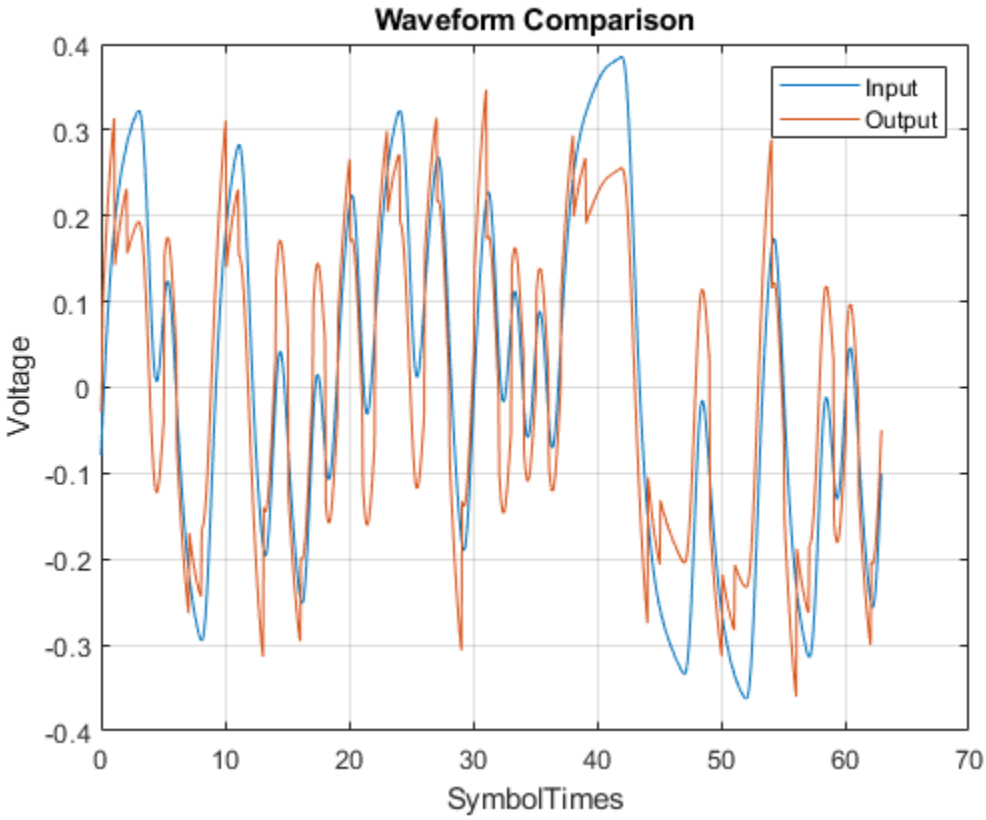
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

Plot the resulting waveforms.

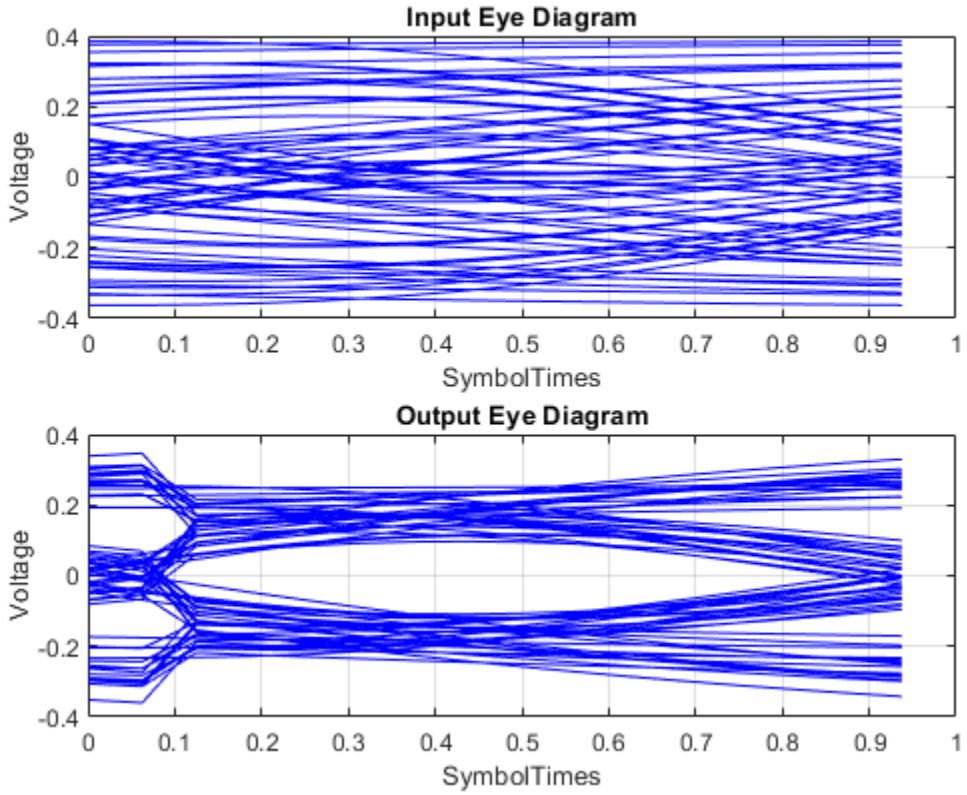
```
figure
plot(t,pulseIn,t,pulseOut)
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
axis([41 55 -0.1 0.4])
```




```
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 `20000` symbols.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 8;
dbloss = 14;
NumberOfDFETaps = 2;
prbsOrder = 12;
M = 20000;
```

Calculate the sample interval.

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

Create the DFECDR System object. Process the channel one sample at a time by setting the input waveforms to `'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-3);
```

Create the channel impulse response.

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

Create the eye diagram.

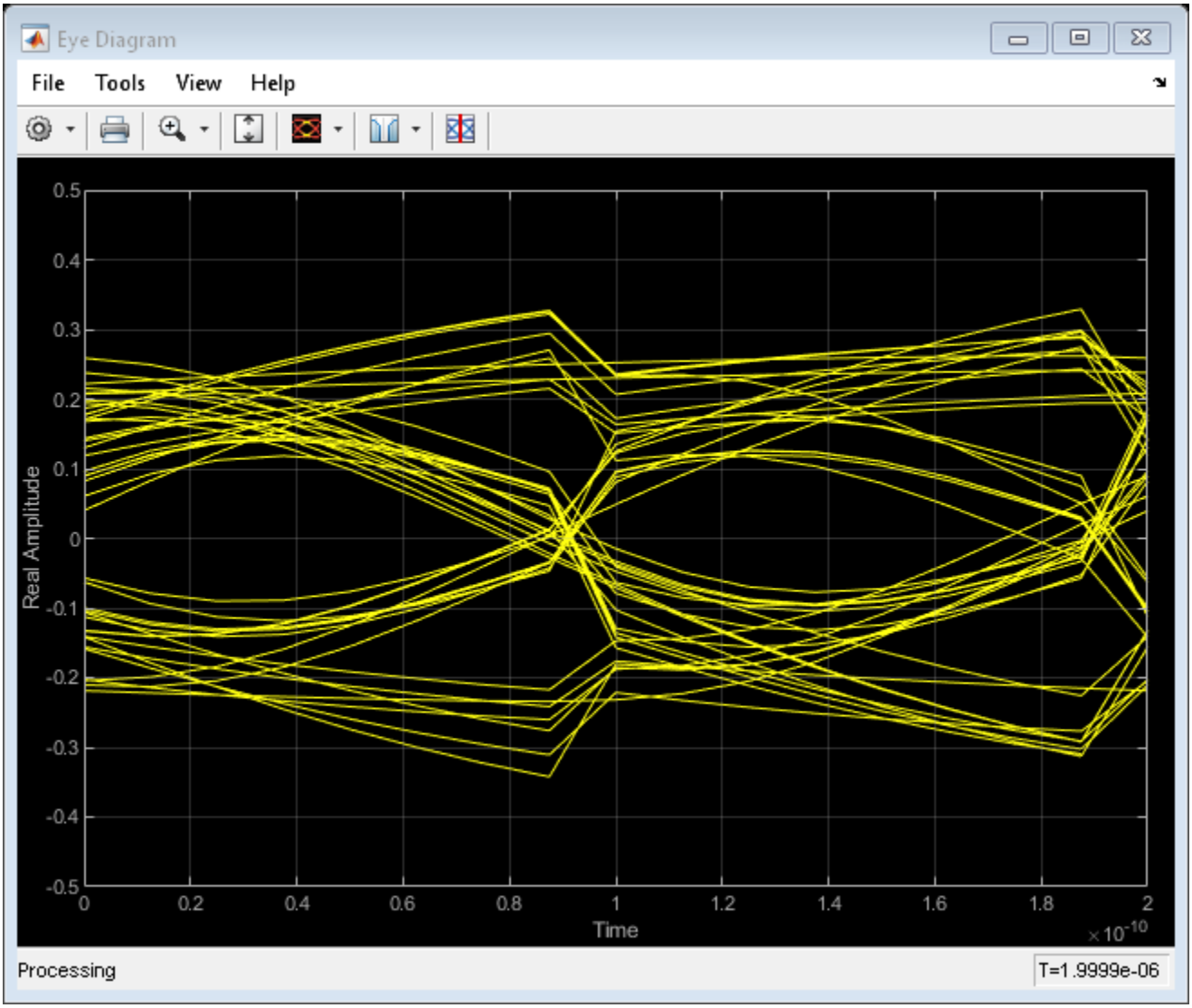
```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

Initialize the PRBS generator.

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

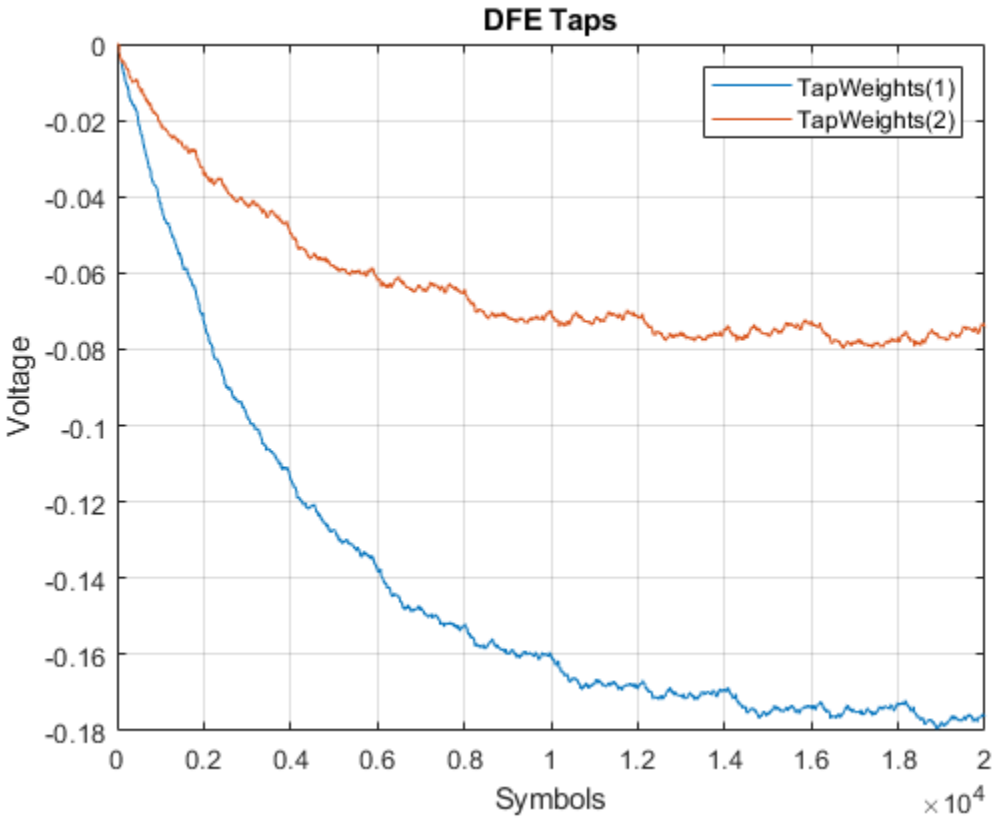
Generate the sample-by-sample eye diagram.

```
%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`

Topics

“Clock and Data Recovery in SerDes System”

Introduced in R2019a

serdes.FFE

Models a feed-forward equalizer

Description

The `serdes.FFE` System object applies a feed-forward equalizer (FFE) as a symbol-spaced finite-impulse response (FIR) filter. Apply the equalizer 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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `ffe = serdes.FFE('Mode',1)` returns an FFE object that applies specified tap weights to the 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).

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 and 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. Each vector element's value specifies the strength of the tap at that position. The tap with the largest magnitude is the main tap and therefore defines the number of pre- and post-cursor taps.

Data Types: `double`

`Normalize` — Normalize tap weights
`true` (default) | `false`

Normalize tap weight vectors, specified as `true` or `false`. When set to `true`, the object scales the `TapWeights` vector elements so that the sum of their absolute values is 1.

Data Types: `logical`

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 waveform type:

- `'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 the impulse response with FFE.

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

Convert the impulse responses to pulse, waveform and eye diagram data for visualization in later steps. Initialize the pseudorandom binary sequence (PRBS).

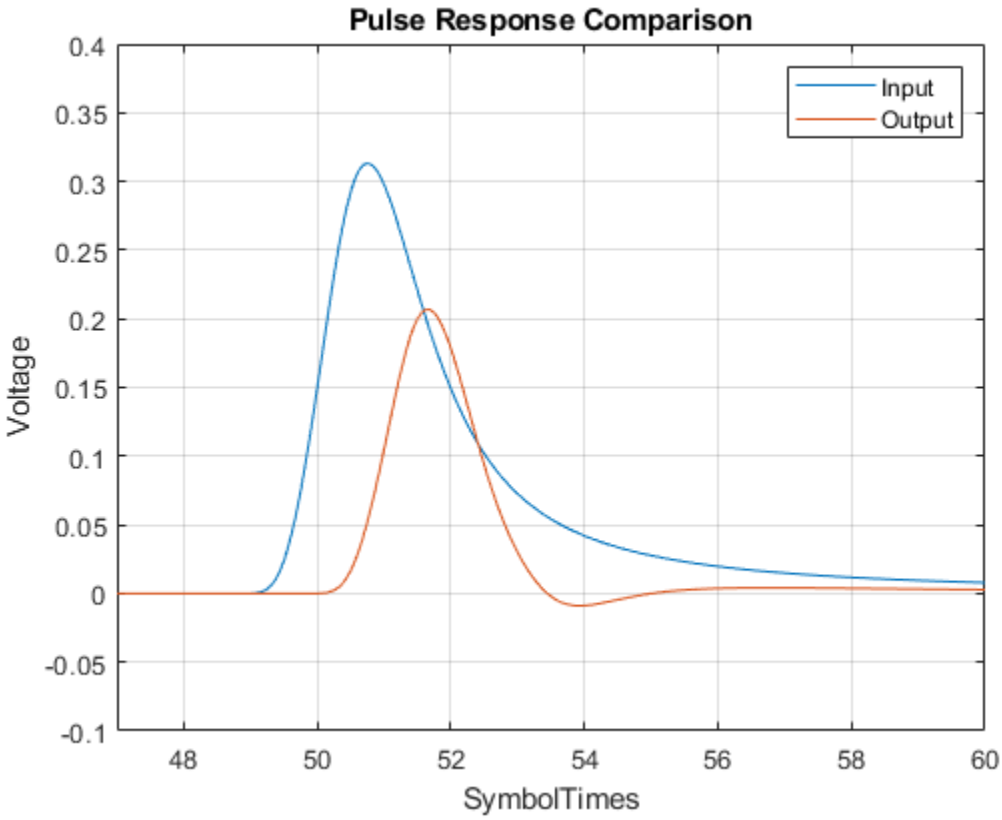
```
ord = 6;
dataPattern = prbs(ord,2^ord-1)-0.5;

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.

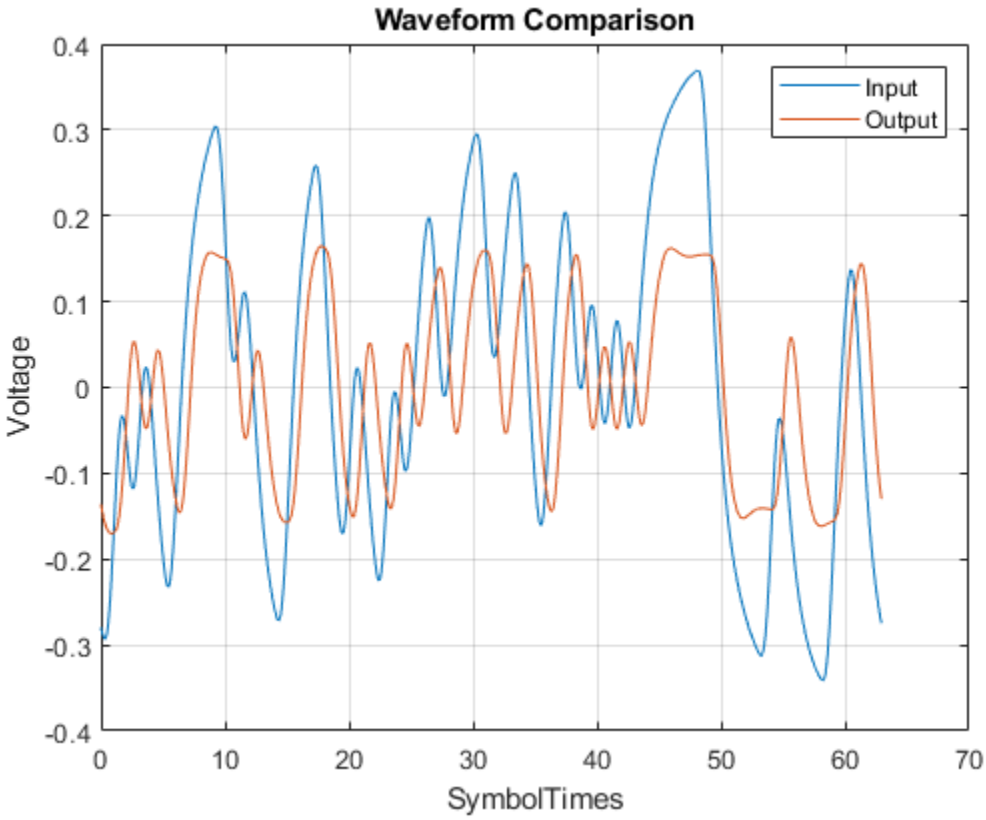
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;
teye = t(1:SamplesPerSymbol);
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

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

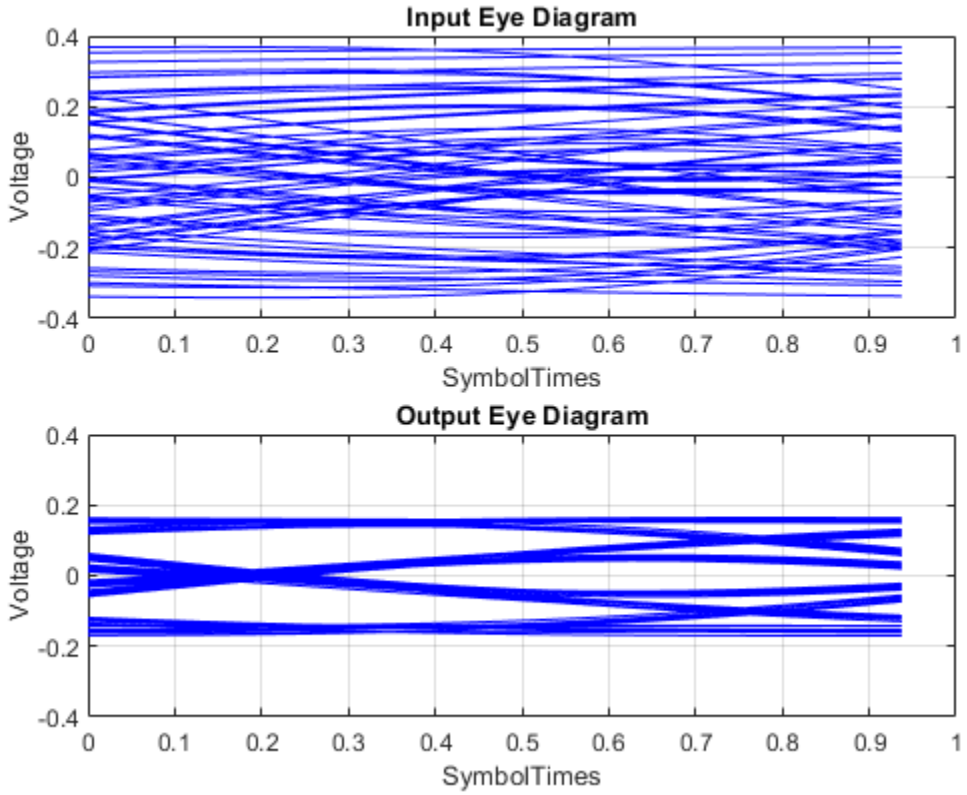
```
figure
plot(t,pulseIn,t,pulseOut)
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
axis([47 60 -0.1 0.4])
```




```
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.

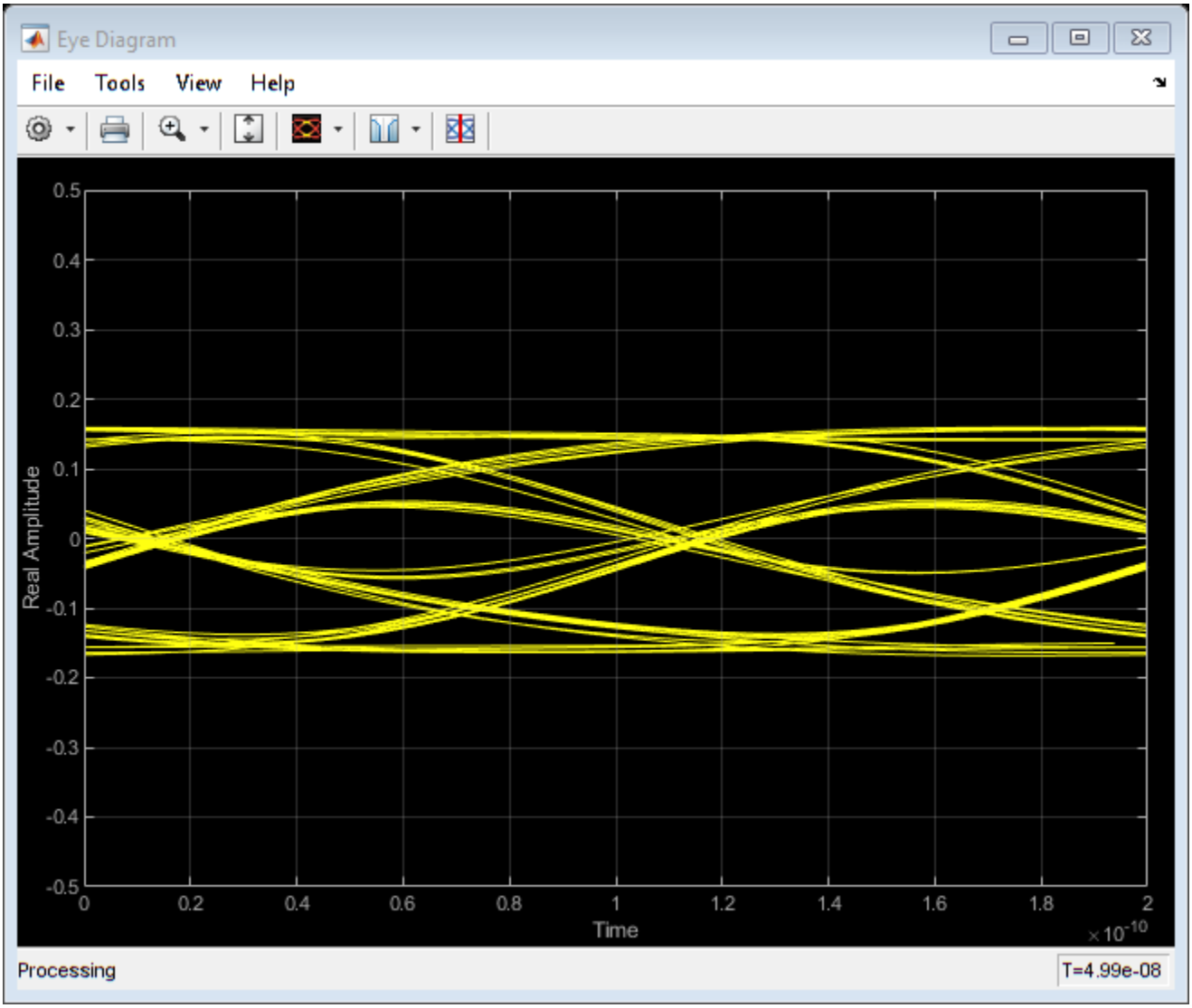
```
eyediagram = comm.EyeDiagram('SampleRate',1/dt,'SamplesPerSymbol',SamplesPerSymbol,...
      'YLimits',[-0.5 0.5]);
```

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
```



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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `SatAmp = serdes.PassThrough('Modulation',4)` returns a PassThrough object with PAM4 modulation scheme.

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.

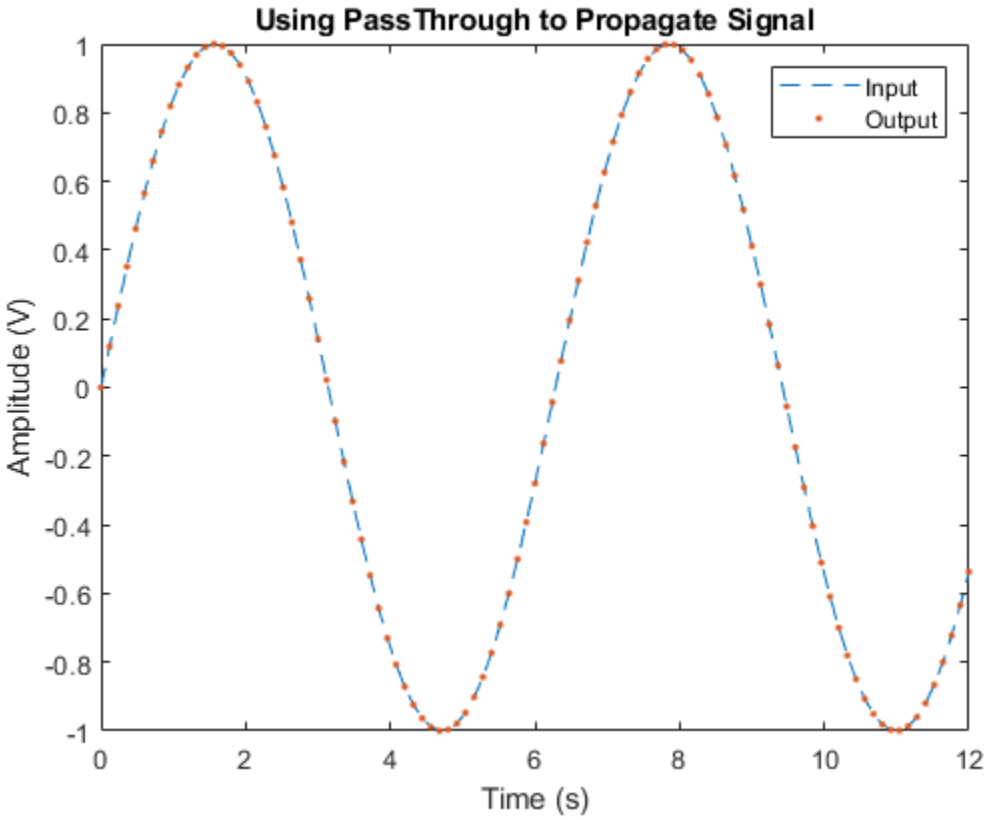
```
PT = serdes.PassThrough;
```

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 Propagate 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 versus voltage out response. The voltage in versus voltage out response is specified either by the soft clipping response defined by `Limit` and `Linear Gain` properties or by the `VinVout` property. `serdes.SaturatingAmplifier` System object applies memoryless nonlinearity to incoming waveform.

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 `1.2` V.

`SatAmp = serdes.SaturatingAmplifier(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `SatAmp = serdes.SaturatingAmplifier('Limit',5)` returns a SaturatingAmplifier object that limits the output waveform at 5 V.

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 and the input waveform remains unchanged.
`1`	On	`serdes.SaturatingAmplifier` scales the input waveform according to a voltage in versus 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:

- `'Limit and Linear Gain'` — Creates a soft clipping voltage in versus voltage out response with the values specified in the `Limit` and `Linear Gain` properties.
- `'VinVout'` — Generates output voltages corresponding to input voltage specified in the `VinVout` property. 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 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 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*-by-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

Clipping Input Waveform Using SaturatingAmplifier

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

Define an input sine wave with a frequency of `250` Hz.

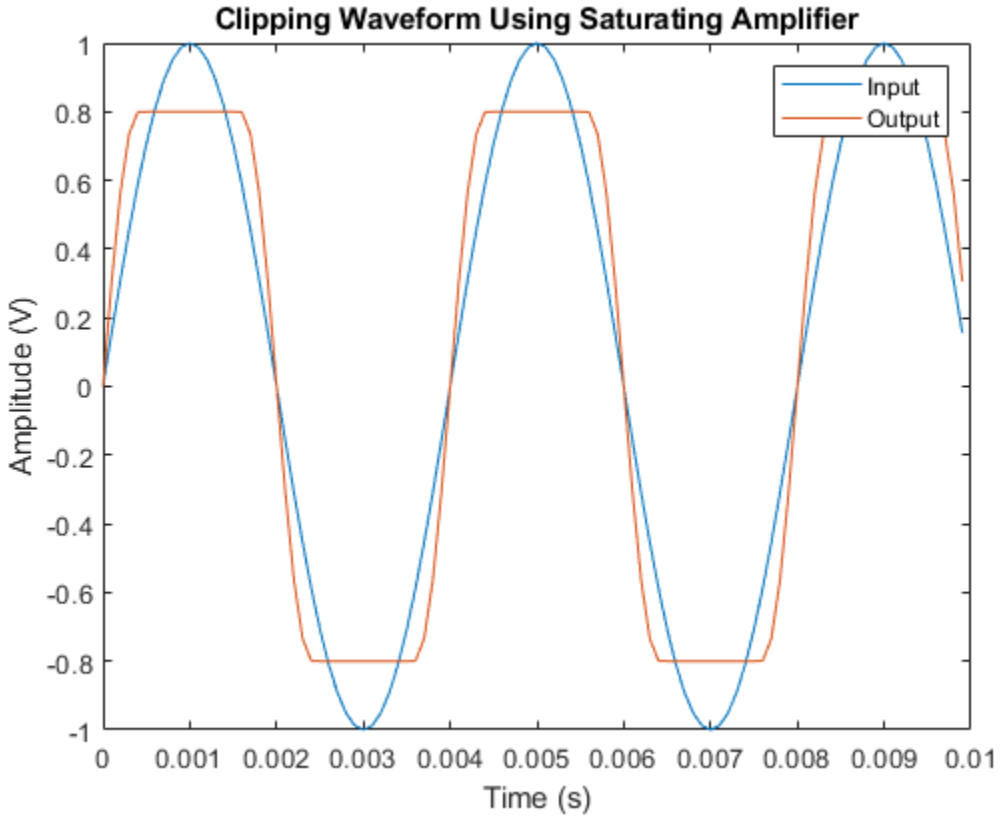
```
Fs = 10000;
L = 100;
t = (0:L-1)'/Fs;
x = sin(2*pi*250*t);
```

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)');
```



Define SaturatingAmplifier with VinVout Table

This example shows how to define a `serdes.SaturatingAmplifier` system object™ using the `VinVout` property.

Define an input sine wave with a frequency of `250 Hz.`

```
t = (0:99)/10000;
x = sin(2*pi*250*t);
```

Define the Voltage In/Voltage Out matrix.

```
M = [-0.6194   -0.8000
     -0.4129   -0.6954
     -0.2065   -0.3966
      0         0
      0.2065    0.3966
      0.4129    0.6954
      0.6194    0.8000];
```

Define the saturating amplifier with the `VinVout` table.

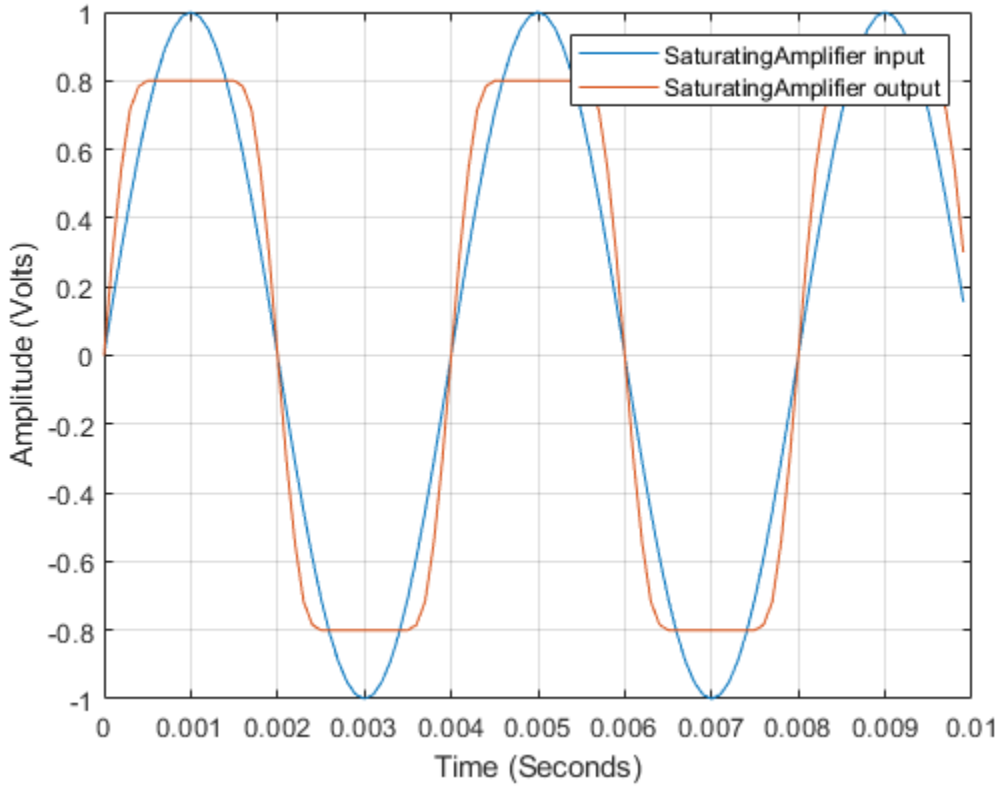
```
SatAmp = serdes.SaturatingAmplifier('Mode',1,'Specification','VinVout','VinVout',M);
```

Modify the input waveform with the saturating amplifier.

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

Plot the input and modified output waveforms.

```
figure;
plot (t,x,t,y)
legend ('SaturatingAmplifier input','SaturatingAmplifier output');
grid on;
xlabel('Time (Seconds)');
ylabel('Amplitude (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 | 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)` sets properties using one or more name-value pairs. Enclose each property name in quotes. Unspecified properties have default values.

Example: `vga = serdes.VGA('Gain',5)` returns a VGA object with a multiplicative gain of 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`

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.

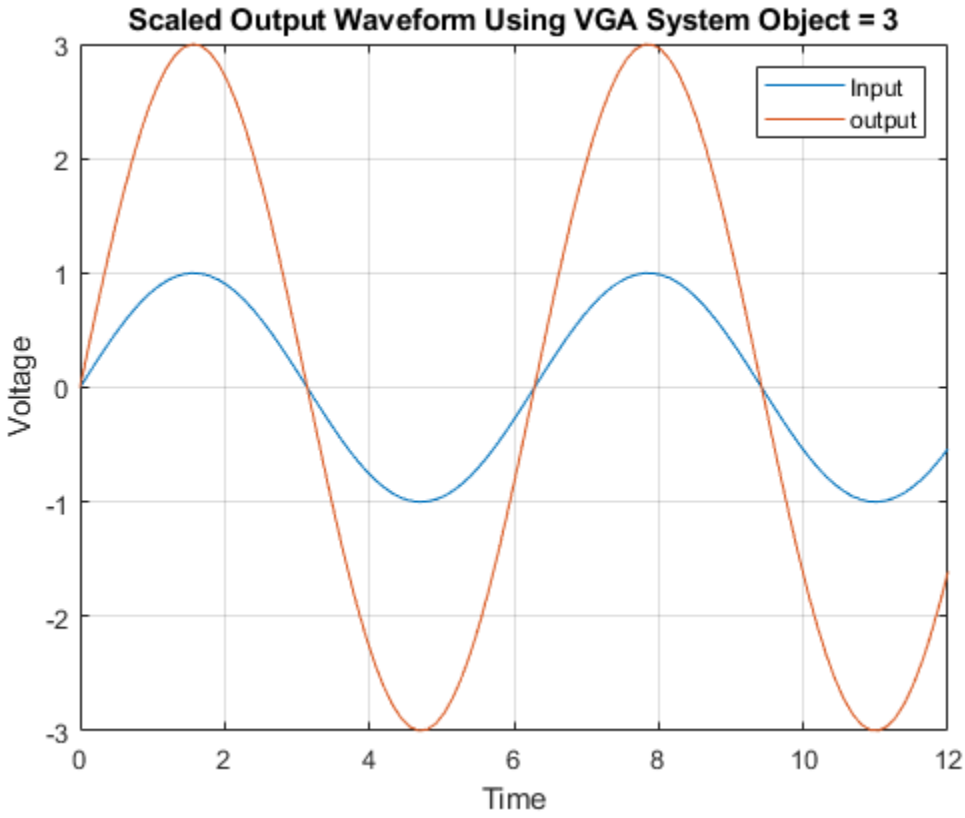
```
vga = serdes.VGA('Gain',3);
```

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

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. For more information, see “Analog Channel Loss in SerDes System”.

Ports

Input

`WaveIn` — Input signal

scalar | vector

Input signal, specified as a waveform.

Data Types: `double`

Output

`WaveOut` — Modified output data

scalar | vector

Modified output data that includes the effect of a lossy printed circuit board transmission line model according to the method outlined in [1].

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) | positive real scalar

Frequency for the desired channel loss, specified as a positive real scalar in hertz. 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 (Hz)**.
- Use `set_param(gcb,'TargetFrequency',value)` to set **Target frequency (Hz)** to a specific value.

Data Types: `double`

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

Data Types: `double`

`Impedance (Ohms)` — Channel characteristic impedance

positive real scalar

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

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

Data Types: `double`

`Impulse response` — User provided impulse response

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

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

You can use user specified impulse response to define your own crosstalk. If you decide to include crosstalk from your custom impulse response, you can specify upto six crosstalk stimuli as new columns in the impulse response.

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.

Data Types: `double`

`Impulse sample interval` — Sample interval of user provided impulse response

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

Sample interval of the user provided impulse response, specified as a positive real scalar in seconds.

Data Types: `double`

Analog Model

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

`50` (default) | nonnegative real scalar

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

Programmatic Use

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

Data Types: `double`

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

`100e-15` (default) | nonnegative real scalar

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

Programmatic Use

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

Data Types: `double`

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

`50` (default) | nonnegative real scalar

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

Programmatic Use

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

Data Types: `double`

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

`200e-15` (default) | nonnegative real scalar

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

Programmatic Use

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

Data Types: `double`

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

`10e-12` (default) | positive real scalar

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

Programmatic Use

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

Data Types: `double`

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

`1` (default) | positive real scalar

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

Programmatic Use

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

Data Types: `double`

Crosstalk

`Include crosstalk` — Include crosstalk in simulation

off (default) | on

Select to include crosstalk in the simulation. By default, this option is deselected.

Magnitude

`Specification` — Specify magnitude of near and far end aggressors
`100GBASE-CR4` (default) | `CEI-25G-LR` | `CEI-28G-SR` | `CEI-28G-VSR` | `Custom`

Specify the magnitude of the near and far end aggressors. You can choose to include maximum allowed crosstalk for specifications such as `100GBASE-CR4`, `CEI-25G-LR`, `CEI-28G-SR`, `CEI-28G-VSR`, or you can specify your own custom crosstalk integrated crosstalk noise (ICN) level.

Programmatic Use

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

`Far end crosstalk ICN (V)` — Desired integrated noise level of far end aggressor
`15e-3` (default) | nonnegative real scalar

Desired integrated crosstalk noise (ICN) level of the far end aggressor, specified as a nonnegative real scalar in volts. ICN specifies the strength of the crosstalk.

Dependencies

This parameter is only available when you choose `Custom` as crosstalk **Specification**.

Programmatic Use

- Use `get_param(gcb,'FEXTICN')` to view the current value of **Far end crosstalk ICN (V)**.
- Use `set_param(gcb,'FEXTICN',value)` to set **Far end crosstalk ICN (V)** to a specific value.

Data Types: `double`

`Near end crosstalk ICN (V)` — Desired integrated noise level of near end aggressor
`10e-3` (default) | nonnegative real scalar

Desired integrated crosstalk noise (ICN) level of the near end aggressor, specified as a nonnegative real scalar in volts. ICN specifies the strength of the crosstalk.

Dependencies

This parameter is only available when you choose `Custom` as crosstalk **Specification**.

Programmatic Use

- Use `get_param(gcb,'NEXTICN')` to view the current value of **Near end crosstalk ICN (V)**.
- Use `set_param(gcb,'NEXTICN',value)` to set **Near end crosstalk ICN (V)** to a specific value.

Data Types: `double`

FEXT Stimulus

`Symbol Time (s)` — Symbol time of far end crosstalk stimulus
`100e-12` (default) | positive real scalar

Symbol time of the far end crosstalk (FEXT) stimulus, specified as a positive real scalar in seconds.

Programmatic Use

- Use `get_param(gcb,'UIFEXT')` to view the current value of **Symbol Time (s)** in FEXT stimulus.
- Use `set_param(gcb,'UIFEXT',value)` to set **Symbol Time (s)** in FEXT stimulus to a specific value.

Data Types: `double`

`Delay (s)` — Delay offset of far end crosstalk stimulus

`0` (default) | nonnegative real scalar

Delay offset of the far end crosstalk (FEXT) stimulus, specified as a positive real scalar in seconds.

Programmatic Use

- Use `get_param(gcb,'DelayFEXT')` to view the current value of **Delay (s)** in FEXT stimulus.
- Use `set_param(gcb,'DelayFEXT',value)` to set **Delay (s)** in FEXT stimulus to a specific value.

Data Types: `double`

`Modulation` — Modulation levels of far end crosstalk stimulus

`NRZ` (default) | `PAM4`

Modulation levels of the far end crosstalk (FEXT) stimulus, specified between `NRZ` (2-level) and `PAM4` (4-level).

Programmatic Use

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

`PRBS Order` — PRBS order of far end crosstalk stimulus

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

Pseudorandom binary sequence (PRBS) order of the far end crosstalk (FEXT) stimulus.

Programmatic Use

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

Data Types: `double`

NEXT Stimulus

`Symbol Time (s)` — Symbol time of near end crosstalk stimulus

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

Symbol time of the near end crosstalk (NEXT) stimulus, specified as a positive real scalar in seconds.

Programmatic Use

- Use `get_param(gcb,'UINEXT')` to view the current value of **Symbol Time (s)** in NEXT stimulus.
- Use `set_param(gcb,'UINEXT',value)` to set **Symbol Time (s)** in NEXT stimulus to a specific value.

Data Types: `double`

`Delay (s)` — Delay offset of near end crosstalk stimulus

`0` (default) | nonnegative real scalar

Delay offset of the near end crosstalk (NEXT) stimulus, specified as a positive real scalar in seconds.

Programmatic Use

- Use `get_param(gcb,'DelayNEXT')` to view the current value of **Delay (s)** in NEXT stimulus.
- Use `set_param(gcb,'DelaynEXT',value)` to set **Delay (s)** in NEXT stimulus to a specific value.

Data Types: `double`

`Modulation` — Modulation levels of near end crosstalk stimulus

`NRZ` (default) | `PAM4`

Modulation levels of the near end crosstalk (NEXT) stimulus, specified between `NRZ` (2-level) and `PAM4` (4-level).

Programmatic Use

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

`PRBS Order` — PRBS order of near end crosstalk stimulus

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

Pseudorandom binary sequence (PRBS) order of the near end crosstalk (NEXT) stimulus.

Programmatic Use

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

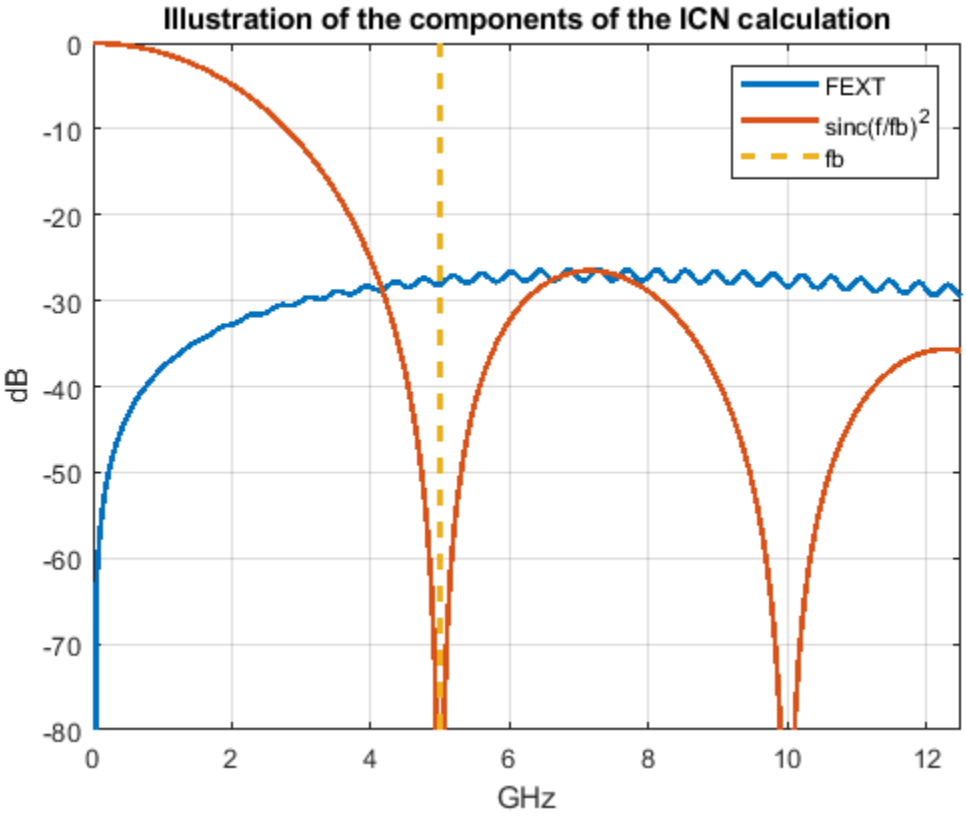
Data Types: `double`

More About

Integrated Crosstalk Noise (ICN)

ICN is a frequency domain metric where the crosstalk is multiplied by a weighting function and then numerically integrated from 50 MHz to the baud rate (f_b). If there are multiple aggressors, their power are summed together before combining with the weighting function.

The time domain signal does not excite all frequencies evenly. The power spectral density (PSD) of a baseband time domain excitation follows a sinc-squared type response. The weighting function mimics the excitation of the PSD and shapes the PSD by including the effects of the receiver bandwidth and the transmitter rise time.



The total ICN is calculated by root-sum-squaring the FEXT ICN and NEXT ICN values together.

$$W_{ft}(f) = \left(\frac{A_{ft}^2}{4f_b}\right)\mathrm{sinc}^2\left(\frac{f}{f_b}\right)\left[\frac{1}{1+\left(f/f_{ft}\right)^4}\right]\left[\frac{1}{1+\left(f/f_{ft}\right)^8}\right]$$

$$\sigma_{fx} = \left(2\Delta f\sum_n W_{ft}(f_n)10^{-MDFEXT_{loss}(f_n)/10}\right)^{1/2}$$

$$\sigma_{nx} = \left(2\Delta f\sum_n W_{ft}(f_n)10^{-MDNEXT_{loss}(f_n)/10}\right)^{1/2}$$

$$\sigma_x = \sqrt{\sigma_{fx}^2 + \sigma_{nx}^2}$$

Algorithms

Creating Far End Crosstalk

The impact felt on a victim line from a far end crosstalk aggressor is proportional to the rate of change of the aggressor waveform [2]. So, you can estimate the shape of a FEXT time domain signal with the derivative of the through response lossy impulse response.

$$I_{\mathrm{FEXT}}(t) = k_{\mathrm{FEXT}}\frac{dI(t)}{dt}$$

where, k_{FEXT} is a scale factor that scales the $I_{\mathrm{FEXT}}(t)$ so that it has user specified ICN value.

To calculate the ICN of the signal, transform the signal to frequency domain using Fourier transform.

$$H_{\mathrm{FEXT}}(f) = \mathscr{F}[I_{\mathrm{FEXT}}(t)]$$

The magnitude of the scale factor k_{FEXT} is: $k_{\mathrm{FEXT}}(f) = -\frac{ICN_{\mathrm{FEXT}}}{\mathbb{ICN}_{\mathrm{FEXT}}(H_{\mathrm{FEXT}}(f))}$,

where $\mathbb{ICN}$ is the integrated crosstalk noise operator.

The sign of k_{FEXT} is negative since in typical transmission lines in inhomogeneous dielectrics, the inducting coupling is generally greater than capacitive coupling. As a result, the forward crosstalk pulse has the opposite magnitude from the magnitude of the aggressor signal.

Creating Near End Crosstalk

To calculate the near end crosstalk, note that the frequency domain NEXT response is similar in shape (not in magnitude) to the victim's return loss (S_{11} or S_{11}).

$$H_{\mathrm{NEXT}}(f) = k_{\mathrm{NEXT}} \cdot S_{11}(f)$$

Then the scale factor k_{NEXT} is: $k_{\mathrm{NEXT}} = -\frac{ICN_{\mathrm{NEXT}}}{\mathbb{ICN}(S_{11}(f))}$

And the time domain NEXT signal is derived from the inverse Fourier transform.

$$I_{\mathrm{NEXT}}(t) = \mathscr{F}^{-1}[k_{\mathrm{NEXT}} \cdot S_{11}(f)]$$

References

[1] IEEE 802.3bj-2014. "IEEE Standard for Ethernet Amendment 2: Physical Layer Specifications and Management Parameters for 100 Gb/s Operation Over Backplanes and Copper Cables." URL: https://standards.ieee.org/standard/802_3bj-2014.html.

[2] Stephen Hall and Howard Heck. *Advanced Signal Integrity for High-Speed Digital Designs*. Hoboken, NJ: Wiley Press, 2009.

See Also

Configuration | Stimulus

Topics
"Analog Channel Loss in SerDes System"

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

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

AGC operating mode:

- `Off` — AGC is bypassed, the input waveform remains unchanged.
- `On` — 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 (V)` — Desired RMS voltage of output waveform

`0.3` (default) | real scalar in the range [0.003, 10]

Desired RMS voltage of the output waveform, specified as a real scalar in the range [0.003, 10] in volts.

Programmatic Use

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

Data Types: `double`

`Maximum gain` — Maximum allowed AGC gain

`10` (default) | positive real scalar

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

Programmatic Use

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

Data Types: `double`

`Averaging length` — Averaging length for RMS calculation

`100` (default) | positive real scalar

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

Programmatic Use

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

Data Types: `double`

IBIS-AMI parameters

Mode — Include Mode parameter in IBIS-AMI model

on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

`Target RMS voltage` — Include Target RMS voltage parameter in IBIS-AMI model

on (default) | off

Select to include **Target RMS voltage** as a parameter in the IBIS-AMI file. If you deselect **Target RMS voltage**, it is removed from the AMI files, effectively hard-coding **Target RMS voltage** to its current value.

See Also

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

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 first order phase tracking CDR model. For more information, see “Clock and Data Recovery in SerDes System”..

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

`Phase offset (symbol time)` — Clock phase offset

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

Clock phase offset, specified as a real scalar in the range [0, 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 (symbol time)**.
- Use `set_param(gcb,'PhaseOffset',value)` to set CDR to a specific **Phase offset (symbol time)**.

Data Types: `double`

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

`0` (default) | real scalar in the range [0, 300]

Reference clock offset impairment, specified as a real scalar in the range [0, 300] in parts per million (ppm). **Reference offset (ppm)** 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 (ppm)**.
- Use `set_param(gcb,'ReferenceOffset',value)` to set CDR to a specific **Reference offset (ppm)**.

Data Types: `double`

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

`16` (default) | real positive integer ≥5

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer ≥5. 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).

Early/late count threshold also controls the bandwidth of the CDR which is approximately calculated by using the equation:

$$\text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Early/late threshold count} \cdot \text{Step}}$$

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**.

Data Types: `double`

`Step (symbol time)` — Clock phase resolution

`0.0078` (default) | real scalar

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

Programmatic Use

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

Data Types: `double`

`Sensitivity (V)` — Sampling latch metastability voltage

`0` (default) | real scalar

Sampling latch metastability voltage, specified as a real scalar in volts. If the data sample voltage lies within the region (±**Sensitivity (V)**), there is a 50% probability of bit error.

Programmatic Use

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

Data Types: `double`

IBIS-AMI parameters

`Phase Offset` — Include Phase Offset parameter in IBIS-AMI model
on (default) | off

Select to include **Phase Offset** as a parameter in the IBIS-AMI file. If you deselect **Phase Offset**, it is removed from the AMI files, effectively hard-coding **Phase Offset** to its current value.

`Reference offset` — Include Reference offset parameter in IBIS-AMI model
on (default) | off

Select to include **Reference offset** as a parameter in the IBIS-AMI file. If you deselect **Reference offset**, it is removed from the AMI files, effectively hard-coding **Reference offset** to its current value.

See Also

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

Topics
“Clock and Data Recovery in SerDes System”

Introduced in R2019a

Configuration

Configure system wide settings in 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 generation of IBIS and AMI models 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 (s)**.
- Use `set_param(gcb,'SymbolTime',value)` to set **Symbol time (s)** to a specific value.

Data Types: `double`

`Samples per symbol` — Data points per symbol

16 (default) | 8 | 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.

Data Types: `double`

`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 read-only and is derived from **Symbol time (s)** and **Samples per symbol**.

Programmatic Use

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

Data Types: `double`

`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.

Data Types: `double`

`Modulation` — Modulation scheme

`'NRZ'` (default) | `'PAM4'`

Number of logic levels in the modulation scheme:

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

Data Types: `char`

`Signaling` — How signal is transmitted through wires

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

How the incoming signal is transmitted through wires:

- `'Differential'` — Transmit the incoming signal using a differential pair of signals. The receiver responds to the difference between the two signals.
- `'Single-ended'` — 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 affects the generated IBIS files. Voltage levels in Simulink do not change when changing the signaling type. **Signaling** also affects the **Impedance** of Analog Channel when the **Channel model** is `Loss 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.

Data Types: `char`

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

on (default) | off

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 model configuration (Tx and Rx, I/O, redriver, retimer), IBIS settings (model names, corner percentage), and AMI model settings (model type, 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, additional tap structure, and jitter and noise profiles using the **AMI-Tx** and **AMI-Rx** tabs. For more information, see “Manage IBIS-AMI Parameters”.

See Also

Analog Channel | Stimulus

Topics
“Managing AMI Parameters”
“Customizing SerDes Toolbox Datapath Control Signals”
“Manage IBIS-AMI Parameters”

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. The filter is a real one-zero two-pole (1z/2p) filter, unless you define the gain-pole-zero (GPZ) matrix.

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

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

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 waveform vector, then the Init subsystem calls to the CTLE System object. The CTLE System object determines the CTLE transfer function for the best eye height opening and applies the transfer function to the input waveform for time domain simulation. This optimized transfer function is used by the CTLE for

entire time domain simulation. For more information about the Init subsystem, see “Statistical Analysis in SerDes Systems”

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**.

`Configuration select` — Select which member of transfer function to apply in fixed mode

`0` (default) | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8`

Select which transfer function configuration to apply in CTLE 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 (dB)** and **Peaking gain (dB)** vectors to be applied to the filter transfer function.

If CTLE **Mode** is set to `Adapt` and the input is an impulse response vector or a waveform vector, **Configuration select** is automatically calculated to determine the best eye height opening. To view the value of the **Configuration select** parameter, choose **Add Plots > Report** in the **SerDes Designer** app.

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.

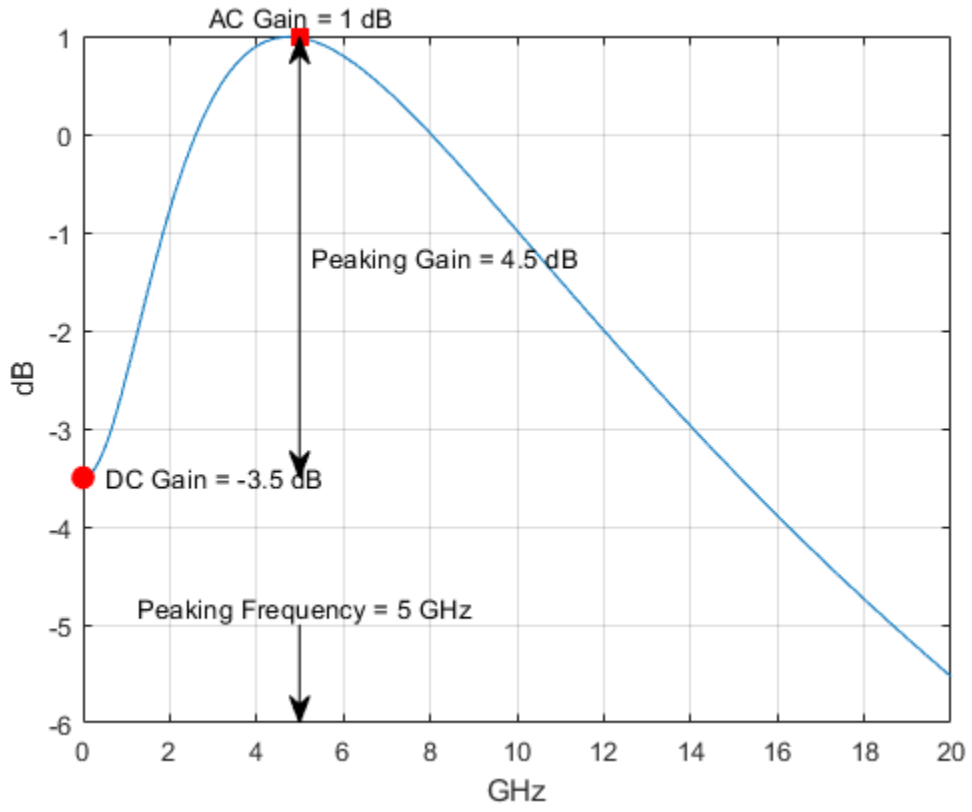
Data Types: `double`

`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. There are five inputs which can be used to define the CTLE transfer function family: “DC gain (dB)” on page 3-0 , “Peaking gain (dB)” on page 3-0 , “AC gain (dB)” on page 3-0 , “Peaking frequency (Hz)” on page 3-0 , and “Gain pole zero matrix” on page 3-0 .

You can define the CTLE response from any two of the three gains and peaking frequency or you can define the GPZ matrix for the CTLE.



- Select `'DC Gain and Peaking Gain'` to specify CTLE response from **DC gain (dB)**, **Peaking gain (dB)**, and **Peaking frequency (Hz)**.
- Select `'DC Gain and AC Gain'` to specify CTLE response is from **DC gain (dB)**, **AC gain (dB)**, and **Peaking frequency (Hz)**.
- Select `'AC Gain and Peaking Gain'` to specify CTLE response from **AC gain (dB)**, **Peaking gain (dB)**, and **Peaking frequency (Hz)**.
- Select `'GPZ Matrix'` to specify CTLE response is 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 scalar, it is converted to match the length of **Peaking gain (dB)**, **AC gain (dB)**, and **Peaking frequency (Hz)** by scalar expansion. If specified as a vector, the vector length must be the same as the vectors in **Peaking gain (dB)**, **AC gain (dB)**, and **Peaking frequency (Hz)**.

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 (dB)**.
- Use `set_param(gcb,'DCGain',value)` to set **DC gain (dB)** 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 (dB)** is the difference between **AC gain (dB)** and **DC gain (dB)** for the CTLE transfer function. If specified as a scalar, it is converted to match the length of **DC gain (dB)**, **AC gain (dB)**, and **Peaking frequency (Hz)** by scalar expansion. If specified as a vector, the vector length must be the same as the vectors in **DC gain (dB)**, **AC gain (dB)**, and **Peaking frequency (Hz)**.

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 (dB)**.
- Use `set_param(gcb,'PeakingGain',value)` to set **Peaking gain (dB)** 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 scalar, it is converted to match the length of **DC gain (dB)**, **Peaking gain (dB)**, and **Peaking frequency (Hz)** by scalar expansion. If specified as a vector, the vector length be the same as the vectors in **DC gain (dB)**, **Peaking gain (dB)**, and **Peaking frequency (Hz)**.

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 (dB)**.
- Use `set_param(gcb,'ACGain',value)` to set **AC gain (dB)** to a specific value.

Data Types: `double`

`Peaking frequency (Hz)` — 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 GHz. If specified as a scalar, it is converted to match the length of **DC gain (dB)**, **AC gain**

(dB), and **Peaking gain (dB)** by scalar expansion. If specified as a vector, the vector length must be the same as the vectors in **DC gain (dB)**, **AC gain (dB)**, and **Peaking gain (dB)**.

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 (Hz)**.
- Use `set_param(gcb,'PeakingFrequency',value)` to set **Peaking frequency (Hz)** 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)** (dB) in the first column and then poles and zeros in alternating columns. The poles and zeros are specified in Hz. Additional rows in the matrix define additional configurations, which can be selected using the **Configuration Select** parameter.

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 matrix**.
- Use `set_param(gcb,'GPZ',value)` to set **Gain pole zero matrix** to a specific value.

Data Types: `double`
Complex Number Support: Yes

IBIS-AMI parameters

`Mode` — Include Mode parameter in IBIS-AMI model

on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

`Configuration select` — Include Configuration select parameter in IBIS-AMI model

on (default) | off

Select to include **Configuration select** as a parameter in the IBIS-AMI file. If you deselect **Configuration select**, it is removed from the AMI files, effectively hard-coding **Configuration select** to its current value.

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 DFE modifies baseband signals to minimize the intersymbol interference (ISI) at the clock sampling times. The DFE samples data at each clock sample time and adjusts the amplitude of the waveform by a correction voltage.

For impulse response processing, the hula-hoop algorithm is used to find the clock sampling locations. The zero-forcing algorithm is then used to determine the N correction factors necessary to have no ISI at the N subsequent sampling locations, where N is the number of DFE taps.

For sample-by-sample processing, the clock recovery is accomplished by a first order phase tracking model. The bang-bang phase detector utilizes the unequalized edge samples and equalized data samples to determine the optimum sampling location. The DFE correction voltage for the N-th tap is adaptively found by finding a voltage that compensates for any correlation between two data samples spaced by N symbol times. This requires a data pattern that is uncorrelated with the channel ISI for correct adaptive behavior.

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

DFE

`Mode` — DFE operating mode

`Adapt` (default) | `Off` | `Fixed`

DFE operating mode:

- `Off` — DFECDR is bypassed and the input waveform remains unchanged.
- `Fixed` — DFECDR applies the input DFE tap weights specified in **Initial tap weights (V)** to the input waveform.
- `Adapt` — The Init subsystem calls to the DFECDR System object. The DFECDR System objectfinds the optimum DFE tap values for the best eye height opening for statistical analysis. During time domain simulation, DFECDR uses the adapted values as the starting point and applies them to the input waveform. For more information about the Init subsystem, see “Statistical Analysis in SerDes Systems”.

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**.

`Initial tap weights (V)` — Initial DFE tap weights

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

Initial DFE tap weights, specified as a row vector in volts. The length of the vector specifies the number of DFE 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.

You can use a valid MATLAB expression to evaluate the **Initial tap weights (V)** row vector.

Example: `set_param(gcb,'TapWeights',"zeros(1,100)")` creates a DFE with 100 taps.

Programmatic Use

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

Data Types: `double`

`Adaptive gain` — Controls DFE tap weight update rate

`9.6e-5` (default) | positive real scalar

Controls DFE tap weight update rate, specified as a unitless positive real 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**.

Data Types: `double`

`Adaptive step size (V)` — DFE adaptive step resolution

`1e-06` (default) | nonnegative real scalar | nonnegative real-valued row vector

DFE adaptive step resolution, specified as a nonnegative real scalar or a nonnegative real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the **Initial tap weights (V)**.

Adaptive step size (V) specifies the minimum DFE tap change from one time step to the next to mimic hardware limitations. Setting **Adaptive step size (V)** 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 (V)** value.
- Use `set_param(gcb,'EqualizationStep',value)` to set DFECDR to a specific value of **Adaptive step size (V)**.

Data Types: `double`

`Minimum DFE tap value (V)` — Minimum value of adapted taps

`-1` (default) | real scalar | real-valued row vector

Minimum value of the adapted taps, specified as a real scalar or a real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the **Initial tap weights (V)**.

Programmatic Use

- Use `get_param(gcb,'MinimumTap')` to view the current DFECDR **Minimum DFE tap value (V)** value.
- Use `set_param(gcb,'MinimumTap',value)` to set DFECDR to a specific value of **Minimum DFE tap value (V)**.

Data Types: `double`

`Maximum DFE tap value (V)` — Maximum value of adapted taps

`1` (default) | nonnegative real scalar | nonnegative real-valued row vector

Maximum value of the adapted taps, specified as a nonnegative real scalar or a nonnegative real-valued row vector in volts. Specify as a scalar to apply to all the DFE taps or as a vector that has the same length as the **Initial tap weights (V)**.

Programmatic Use

- Use `get_param(gcb,'MaximumTap')` to view the current DFECDR **Maximum DFE tap value (V)** value.
- Use `set_param(gcb,'MaximumTap',value)` to set DFECDR to a specific value of **Maximum DFE tap value (V)**.

Data Types: `double`

CDR

`Phase offset (symbol time)` — Manual clock phase offset

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

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

Programmatic Use

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

Data Types: `double`

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

`0` (default) | real scalar in the range [−300, 300]

Reference clock offset impairment, specified as a real scalar in the range [−300, 300] in parts per million (ppm). **Reference offset (ppm)** 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 (ppm)** value.
- Use `set_param(gcb,'ReferenceOffset',value)` to set DFECDR to a specific value of **Reference offset (ppm)**.

Data Types: `double`

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

`16` (default) | positive real integer ≥5

Early or late CDR count threshold to trigger a phase update, specified as a unitless positive real integer ≥5. 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).

Early/late count threshold also controls the bandwidth of the CDR, which is approximately calculated by using the equation:

$$\text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Early/late threshold count} \cdot \text{Step}}$$

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**.

Data Types: `double`

`Step (symbol time)` — Clock phase resolution

`0.0078` (default) | real scalar

Clock phase resolution of the recovered clock, specified as a real scalar in fraction of symbol time. **Step (symbol time)** is the inverse of the number of phase adjustments in the CDR. If the CDR has 128 steps of phase adjustment, the **Step (symbol time)** value is 1/128.

Programmatic Use

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

Data Types: `double`

`Sensitivity (V)` — Sampling latch metastability voltage

`0` (default) | real scalar

Sampling latch metastability voltage, specified as a real scalar in volts. If the data sample voltage lies within the region of (±**Sensitivity (V)**), there is a 50% probability of bit error.

Programmatic Use

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

Data Types: `double`

IBIS-AMI parameters

`Mode` — Include Mode parameter in IBIS-AMI model

on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

`Tap weights` — Include Tap weights parameter in IBIS-AMI model

on (default) | off

Select to include **Tap weights** as a parameter in the IBIS-AMI file. If you deselect **Tap weights**, it is removed from the AMI files, effectively hard-coding **Tap weights** to its current value.

`Phase Offset` — Include Phase Offset parameter in IBIS-AMI model

on (default) | off

Select to include **Phase Offset** as a parameter in the IBIS-AMI file. If you deselect **Phase Offset**, it is removed from the AMI files, effectively hard-coding **Phase Offset** to its current value.

`Reference offset` — Include Reference offset parameter in IBIS-AMI model

on (default) | off

Select to include **Reference offset** as a parameter in the IBIS-AMI file. If you deselect **Reference offset**, it is removed from the AMI files, effectively hard-coding **Reference offset** to its current value.

See Also

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

Topics

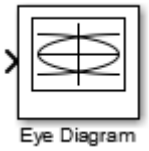
“Clock and Data Recovery in SerDes System”

Introduced in R2019a

Eye Diagram Scope

Display eye diagram of time-domain signal

Library: Communications Toolbox / Comm Sinks
Communications Toolbox HDL Support / Comm Sinks
Mixed-Signal Blockset / Utilities
SerDes Toolbox / Utilities

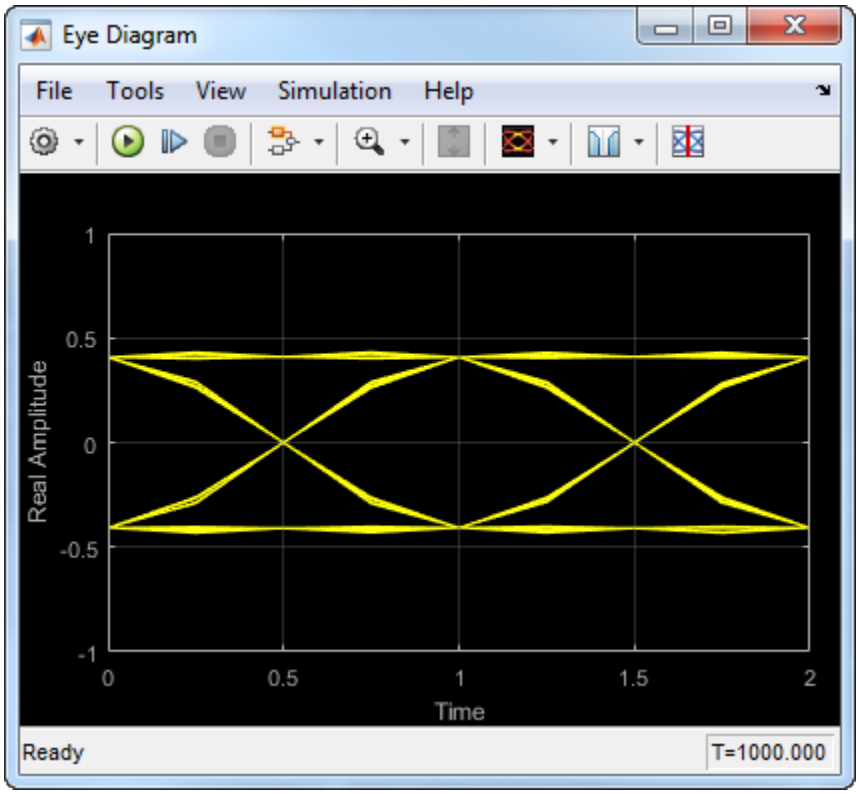


Description

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. For more information, see .

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.

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.



Ports

Input

`In` — Input signal
scalar | column vector

Input signal, specified as a scalar or column vector.

Data Types: `double`

Parameters

Main Tab

`Display mode` — Display mode
`Line plot` (default) | `2D color Histogram`

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

Tunable: Yes

`Enable measurements` — Enable measurements
`off` (default) | `on`

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

`Show horizontal (jitter) histogram` — Display jitter histogram
`off` (default) | `on`

Select this radio button to display the jitter histogram. This can also be accessed by using the histogram button drop down on the toolbar.

Dependencies

This parameter is available when **Display mode** is `2D color histogram` and **Enable measurements** is selected.

`Show vertical (noise) histogram` — Display noise histogram
`off` (default) | `on`

Select this radio button to display the noise histogram. This can also be accessed by using the histogram button drop down on the toolbar.

Dependencies

This parameter is available when **Display mode** is `2D color histogram` and **Enable measurements** is selected.

`Do not show horizontal or vertical histogram` — Do not show horizontal or vertical histogram
`on` (default) | `off`

Select this radio button to display neither the histogram noise nor the histogram jitter.

Dependencies

This parameter is available when **Display mode** is `2D color histogram` and **Enable measurements** is selected.

`Show horizontal bathtub curve` — Show horizontal bathtub curve

`off` (default) | `on`

Select this check box to display the horizontal bathtub curve. This can also be accessed by using the bathtub curve button on the toolbar.

Dependencies

This parameter is available when **Enable measurements** is selected.

`Show vertical bathtub curve` — Show vertical bathtub curve

`off` (default) | `on`

Select this check box to display the vertical bathtub curve. This can also be accessed by using the bathtub curve button on the toolbar.

Dependencies

This parameter is available when **Enable measurements** is selected.

`Eye diagram to display` — Eye diagram to display

`Real only` (default) | `Real and imaginary`

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`.

Tunable: Yes

`Color fading` — Color fading

`off` (default) | `on`

Select this check box to fade the points in the display as the interval of time after they are first plotted increases.

Tunable: Yes

Dependencies

This parameter is available only when the **Display mode** is `Line plot`.

`Samples per symbol` — Samples per symbol

8 (default) | positive integer

Number of samples per symbol, specified as a positive integer. Use with **Symbols per trace** to determine the number of samples per trace.

Tunable: Yes

`Sample offset` — Sample offset

0 (default) | nonnegative integer

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.

Tunable: Yes

`Symbols per trace` — Symbols per trace
2 (default) | positive integer

Number of symbols plotted per trace, specified as a positive integer.

Tunable: Yes

`Traces to display` — Number of traces to display
40 (default) | positive integer

Number of traces plotted, specified as a positive integer.

Tunable: Yes

Dependencies

This parameter is available only when the **Display mode** is `Line plot`

Axes Tab

`Title` — Title label
None (default)

Label that appears above the eye diagram plot.

Tunable: Yes

`Show grid` — Toggle scope grid
`on` (default) | `off`

Toggle this check box to turn the grid on and off.

Tunable: Yes

`Y-limits (Minimum)` — Lower limit of *y*-axis
`-1.1` (default) | scalar

Minimum value of the *y*-axis.

Tunable: Yes

`Y-limits (Maximum)` — Upper limit of *y*-axis
`1.1` (default) | scalar

Maximum value of the *y*-axis.

Tunable: Yes

`Real axis label` — Real axis label
`Real Amplitude` (default)

Text that the scope displays along the real axis.

Tunable: Yes

`Imaginary axis label` — Imaginary axis label
`Imaginary Amplitude` (default)

Text that the scope displays along the imaginary axis.

Tunable: Yes

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
`None` (default) | `Input interpolation` | `Histogram interpolation`

Oversampling method, specified as `None`, `Input interpolation`, or `Histogram interpolation`.

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.

Tunable: Yes

`Color scale` — Color scale
`Linear` (default) | `Logarithmic`

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.

Tunable: Yes

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, histograms and bathtub curves` — Export measurements, histograms and bathtub curves
`Off` (default) | `off`

Select this check box export the eye diagram measurements to the MATLAB® workspace.

Tunable: Yes

`Variable name` — Variable name
`EyeData` (default)

Specify the name of the variable to which the eye diagram measurements are saved. The data is saved as a structure having these fields:

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

Tunable: Yes

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**.

`Figure color` — Figure color
`black` (default)

Specify the background color of the scope figure.

`Axes colors` — Axes colors
`black` | `gray` (default)

Specify the fill and line colors for the axes.

`Line` — Line style, thickness and color for line plots
`continuous` | `0.5` | `yellow` (default)

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

Dependencies

This parameter is available only when the **Display mode** is `Line plot`.

`Marker` — Data point marker
`None` (default) | ...

Data point marker for the selected signal, specified as one of the choices in this table data point markers. This parameter is similar to the `Marker` property for MATLABHandle Graphics® plot objects.

Specifier	Marker Type
none	No marker (default)
○	Circle
□	Square
×	Cross

Specifier	Marker Type
.	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)

Dependencies

This parameter is available only when the **Display mode** is `Line plot`.

`Colormap` — Colormap for histograms

`Hot` (default) | `Parula` | `Jet` | `HSV` | `Cool` | `SpringSummer` | `Autumn` | `Winter` | `Gray` | `Bone` | `Copper` | `Pink` | `Lines` | `Custom`

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`. If you select `Custom`, a dialog box pops up from which you can enter code to specify your own colormap.

Dependencies

This parameter is available only when the **Display mode** is `2D color histogram`.

Measurement Settings Pane

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: Yes

`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.

Tunable: Yes

`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.

Tunable: Yes

`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.

Tunable: Yes

`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: Yes

`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.

Tunable: Yes

Dependencies

To enable this parameter, select **Show horizontal bathtub curve**, **Show vertical bathtub curve**, or both.

`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.

Block Characteristics

Data Types	`Boolean` | `double` | `enumerated` | `fixed point` | `integer` | `single`
Direct Feedthrough	`no`

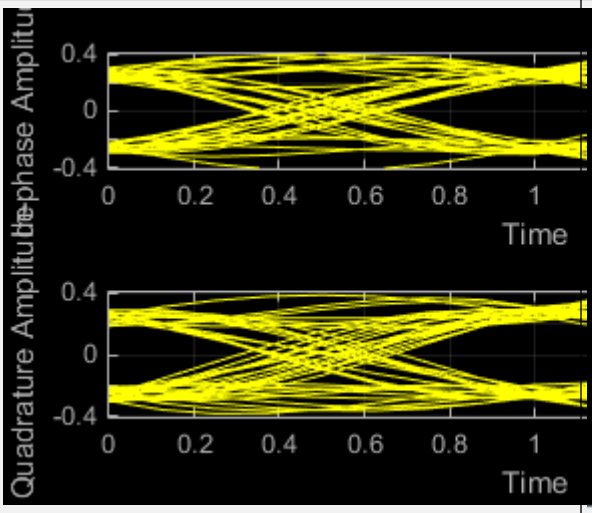
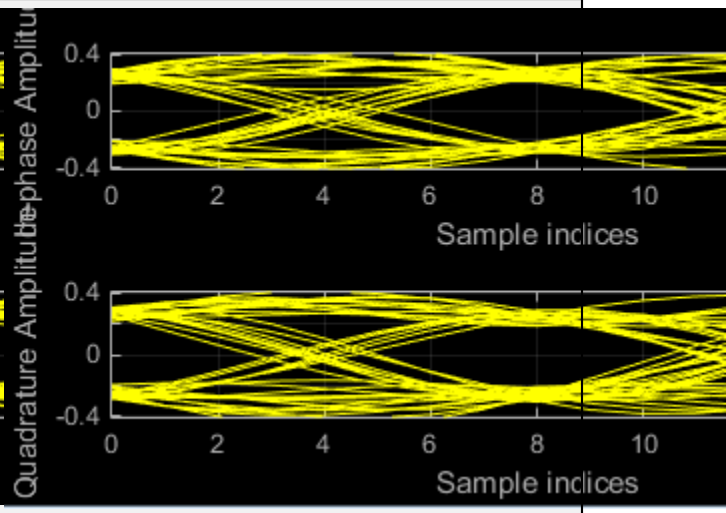
Multidimensional Signals	no
Variable-Size Signals	no
Zero-Crossing Detection	no

More About

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.

Measurements

Measurements assume that the eye diagram object has valid data. A valid eye diagram has two distinct eye crossing points and two distinct eye levels.

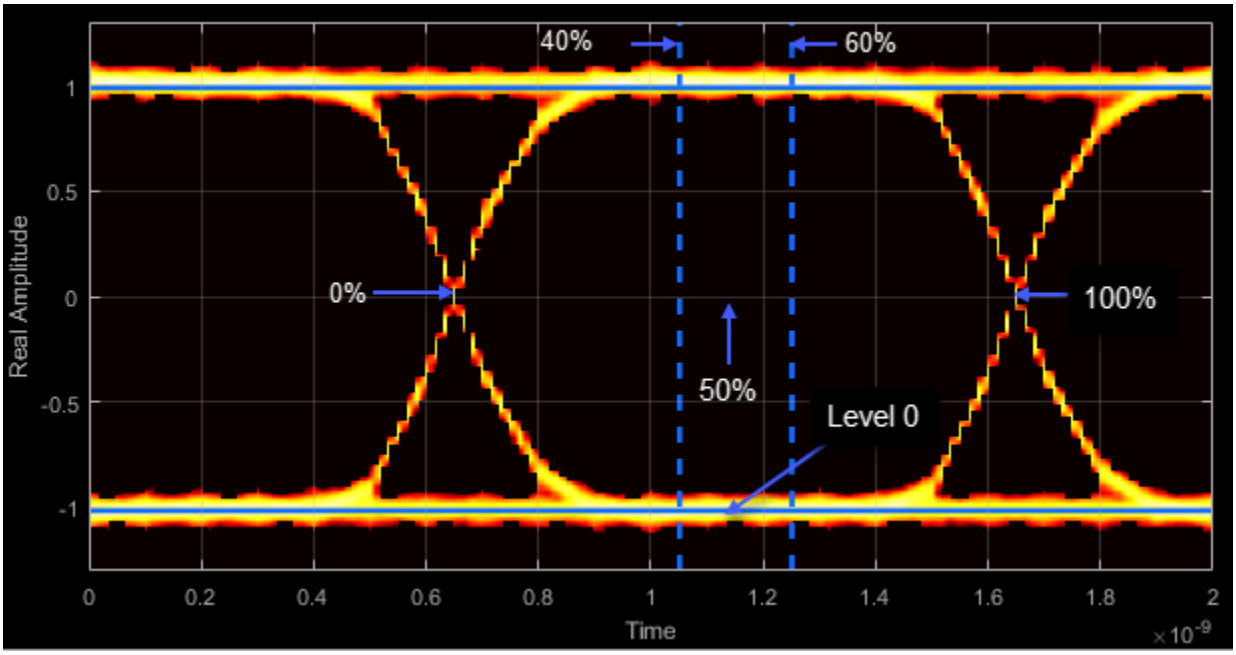
To open the measurements pane, click on the **Eye Measurements** button or select **Tools > Measurements > Eye Measurements** from the toolbar menu.

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 $2\times T_s$ seconds (where T_s 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 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.

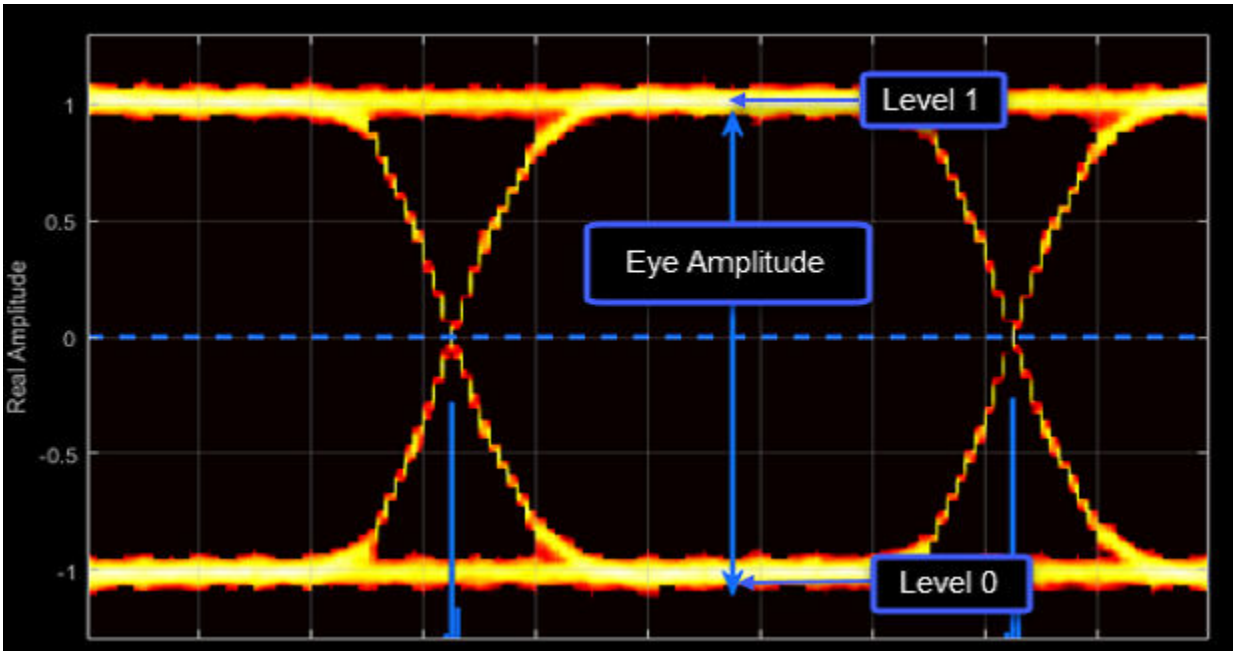
Eye Levels - Amplitude level used to represent data bits

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. For example, when the EyeLevelBoundaries property is set to `[40 60]`, that is, 40% and 60% of the symbol duration, the eye levels are calculated by estimating the mean value of the vertical histogram in this window marked by the eye level boundaries.



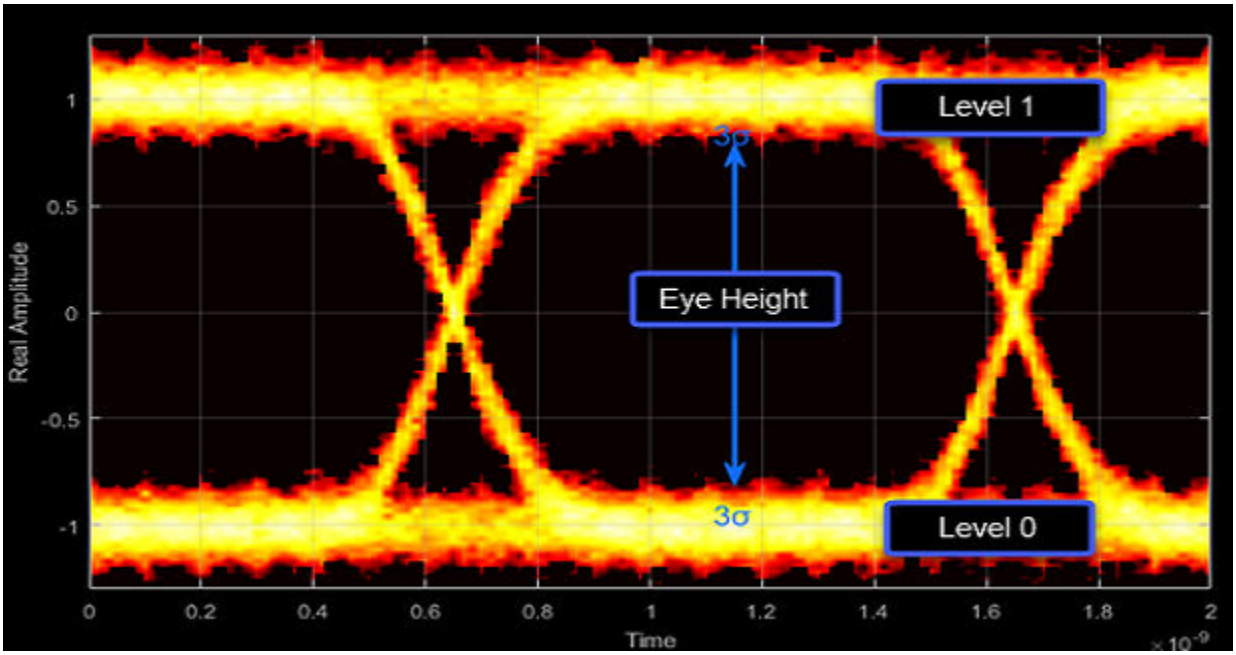
Eye Amplitude - Distance between eye levels

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



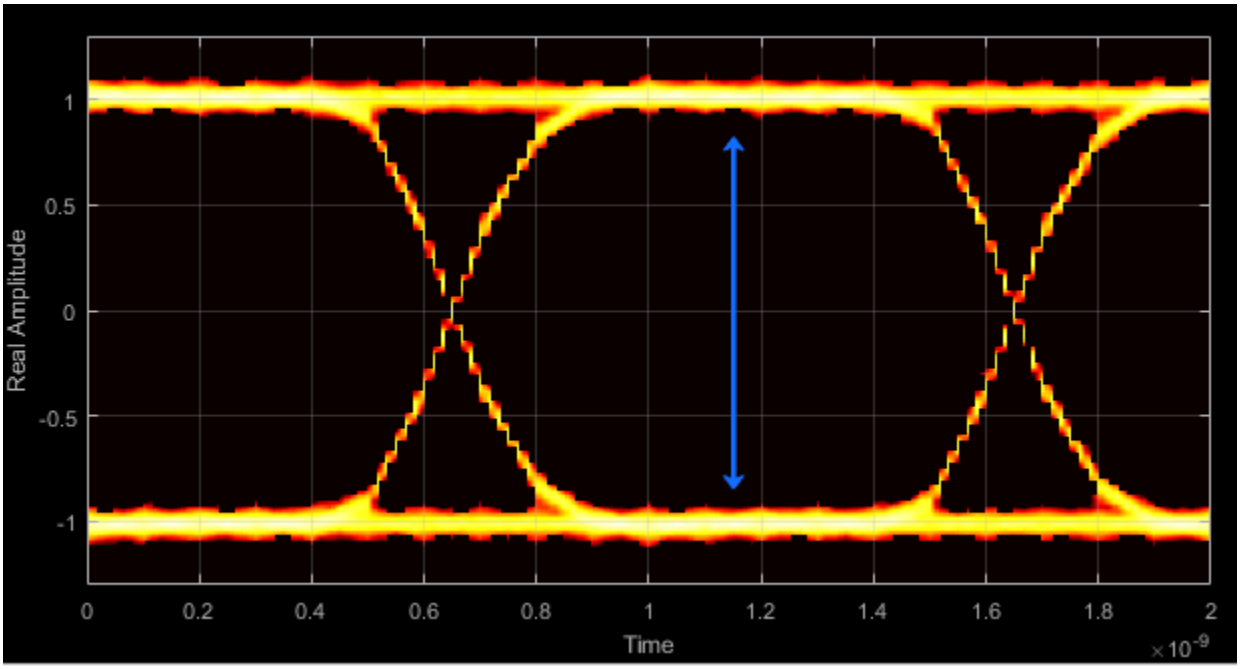
Eye Height - Statistical minimum distance between eye levels

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 BERThreshold property. 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:

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

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.

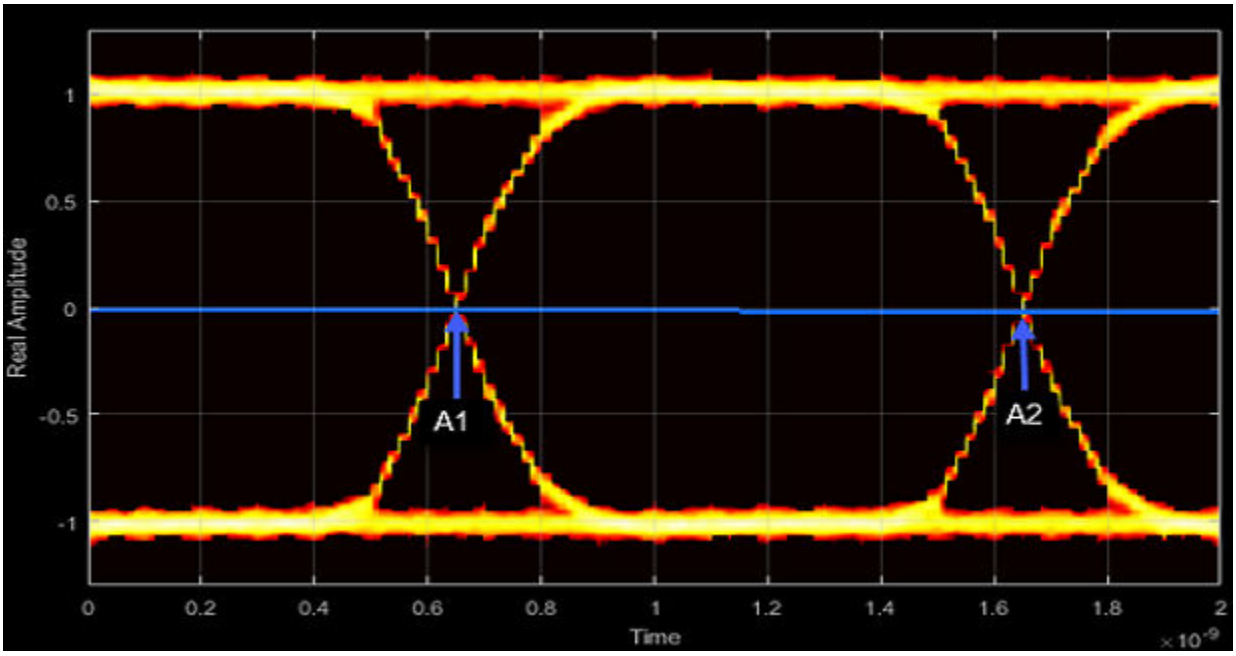
Q Factor - Quality factor

The Q factor is the quality factor and 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

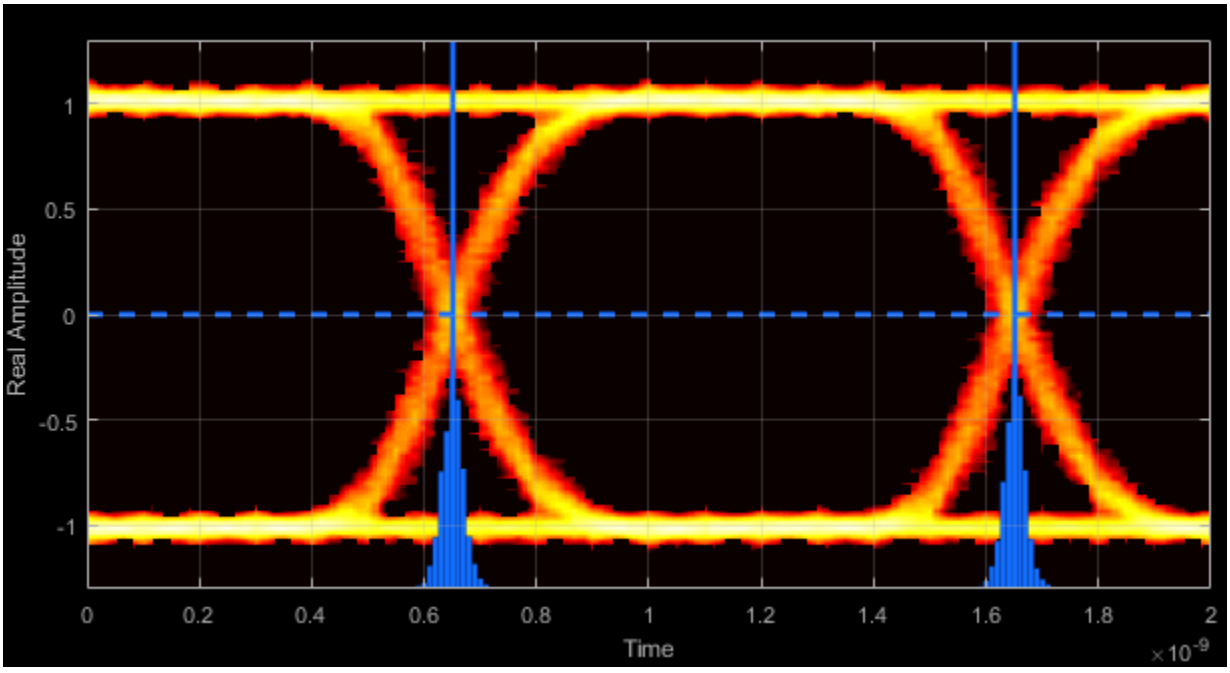
The crossing levels are the amplitude levels at which the eye crossings occur.

The level at which the input signal crosses the amplitude value is specified by the DecisionBoundary property.



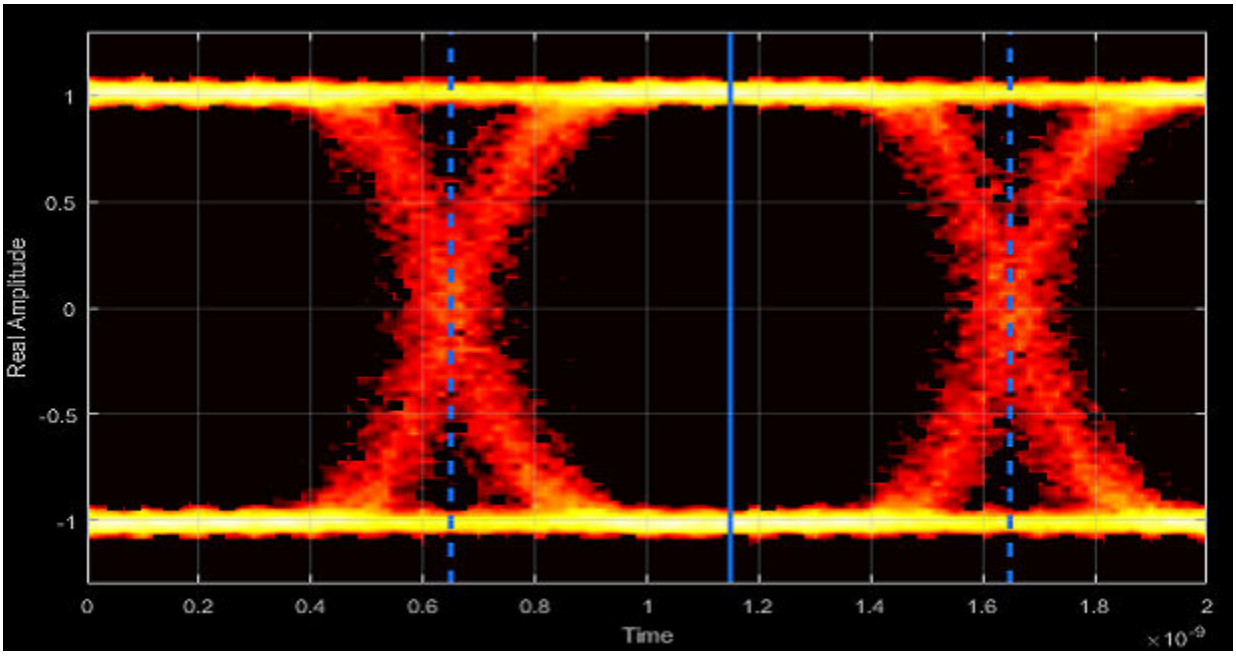
Crossing Times - Times for which crossings occur

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



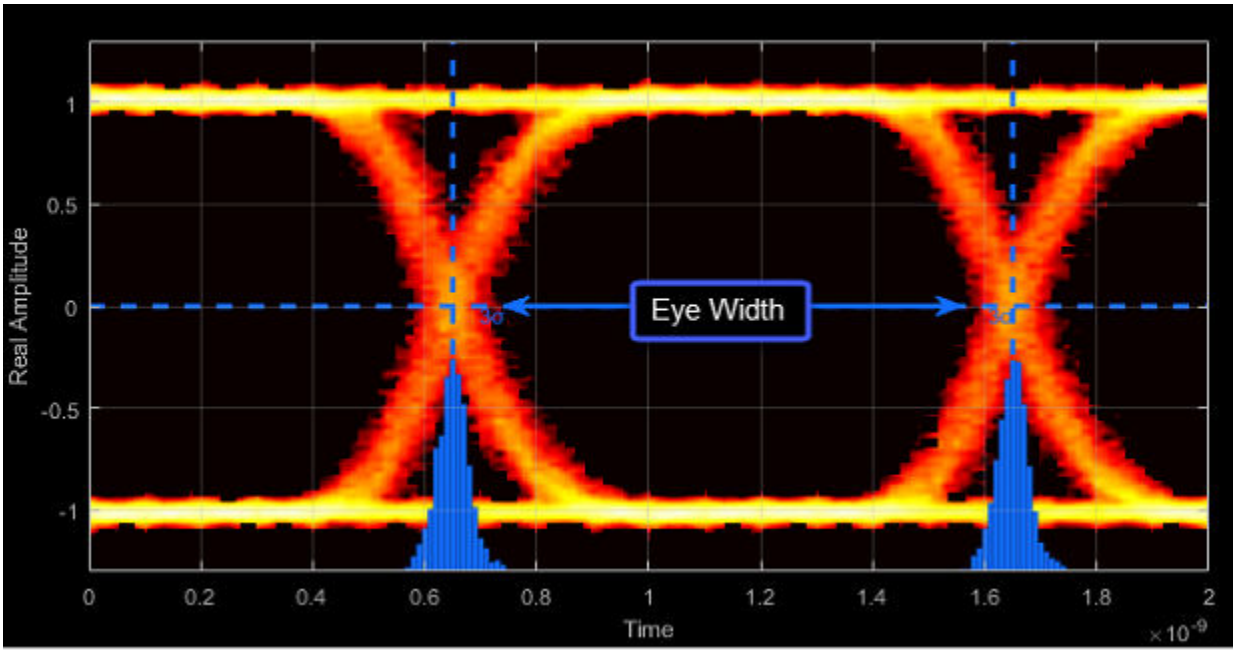
Eye Delay - Mean time between eye crossings

Eye delay is the midpoint between the two crossing times.



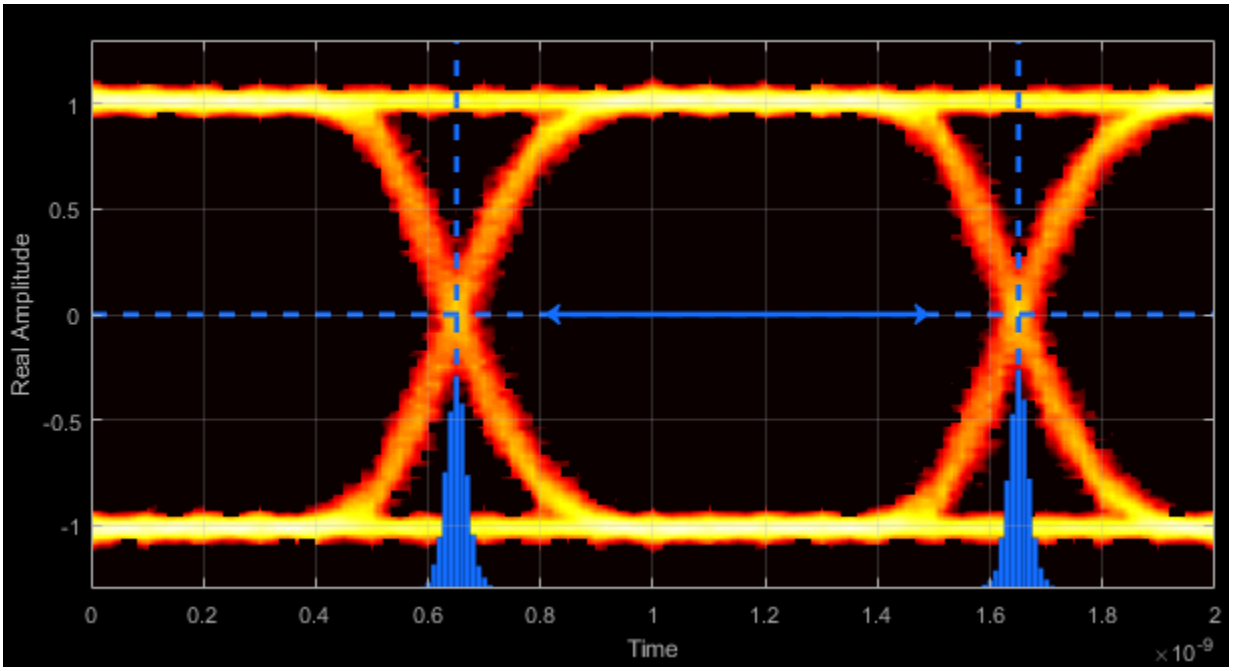
Eye Width - Statistical minimum time between eye crossings

Eye width is the horizontal distance between μ + 3σ of the left crossing time and μ – 3σ of the right crossing time. μ is the mean of the jitter histogram, and σ is the standard deviation.



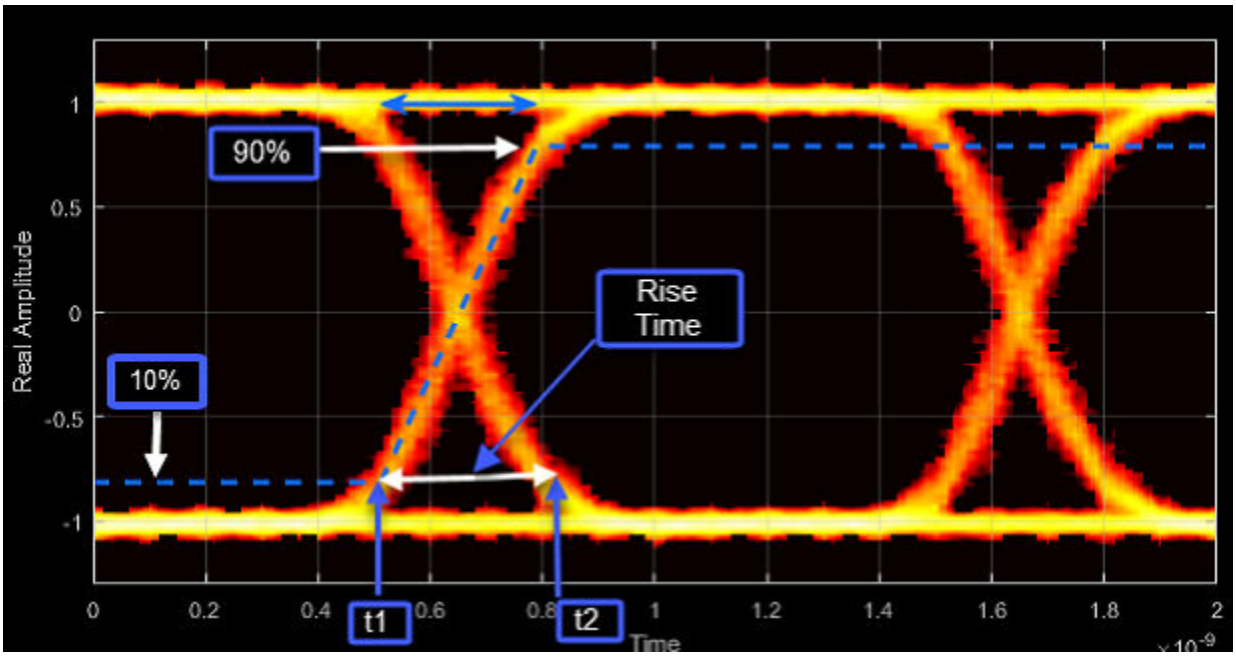
Horizontal Opening - Time between BER threshold points

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



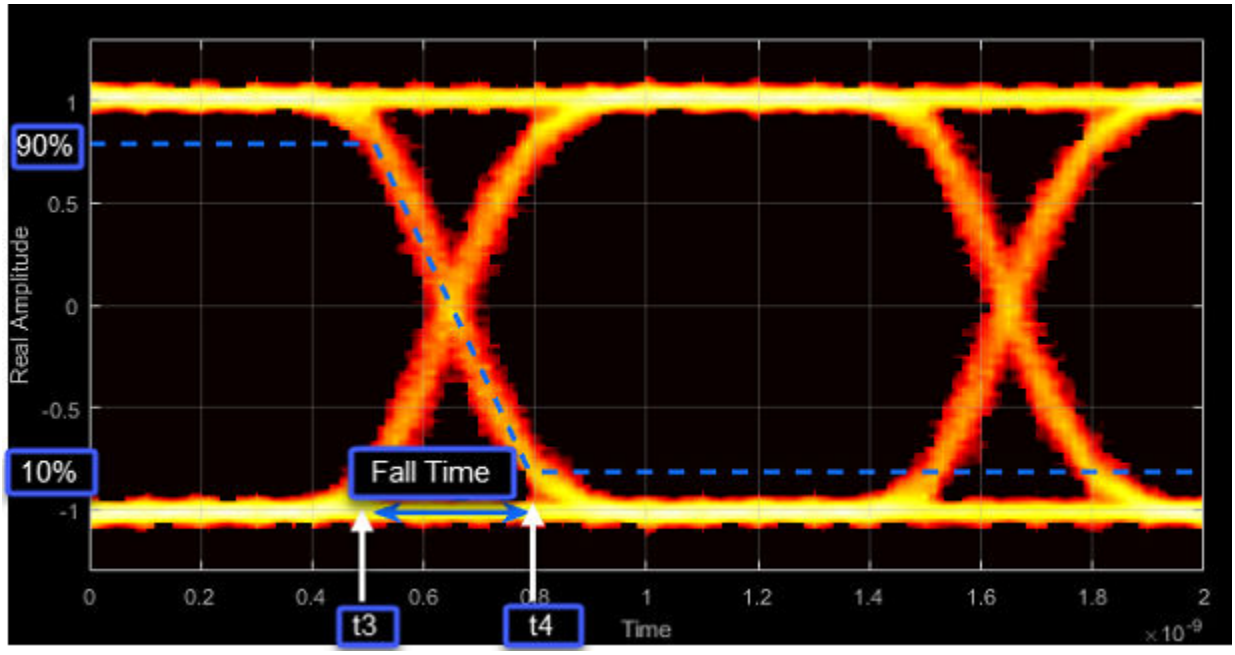
Rise Time - Time to transition from low to high

Rise time is the mean time between the low and high rise/fall 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

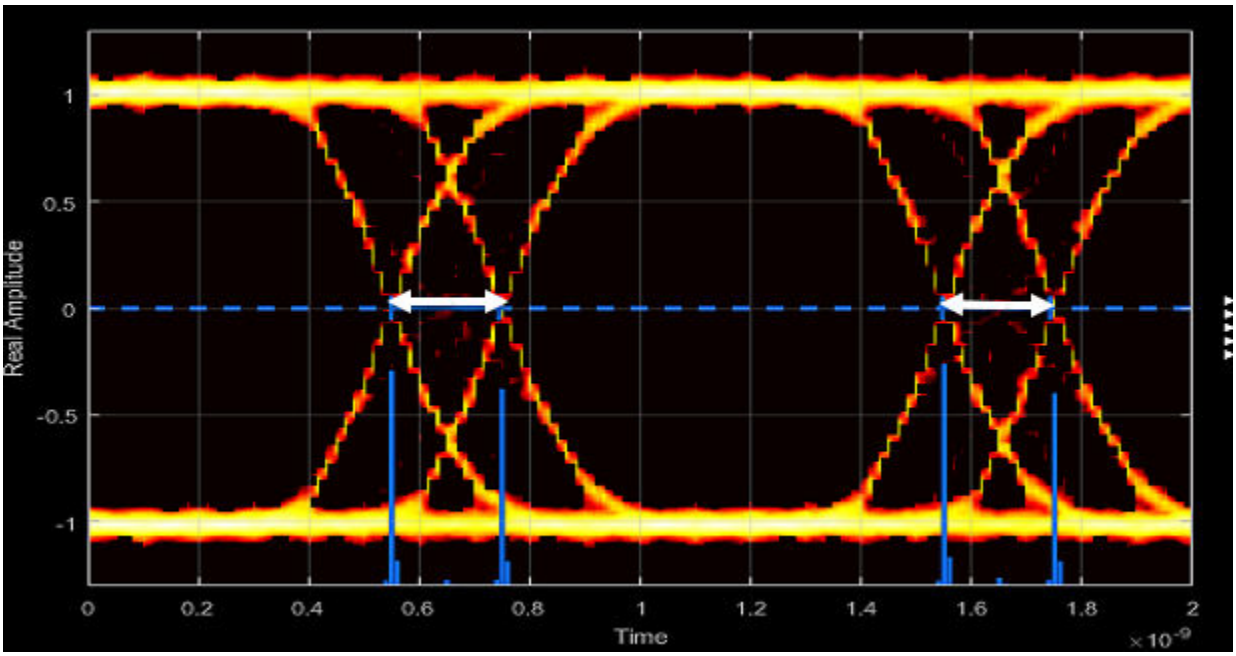
Fall time is the mean time between the high and low rise/fall thresholds defined in the eye diagram. The default thresholds are 10% and 90% of the eye amplitude.



Deterministic Jitter - Deterministic deviation from ideal signal timing

Jitter is the deviation of a signal's timing event from its intended (ideal) occurrence in time . Jitter can be represented with a dual-Dirac model. A dual-Dirac model assumes that the jitter has two components: deterministic jitter (DJ) and random jitter (RJ).

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

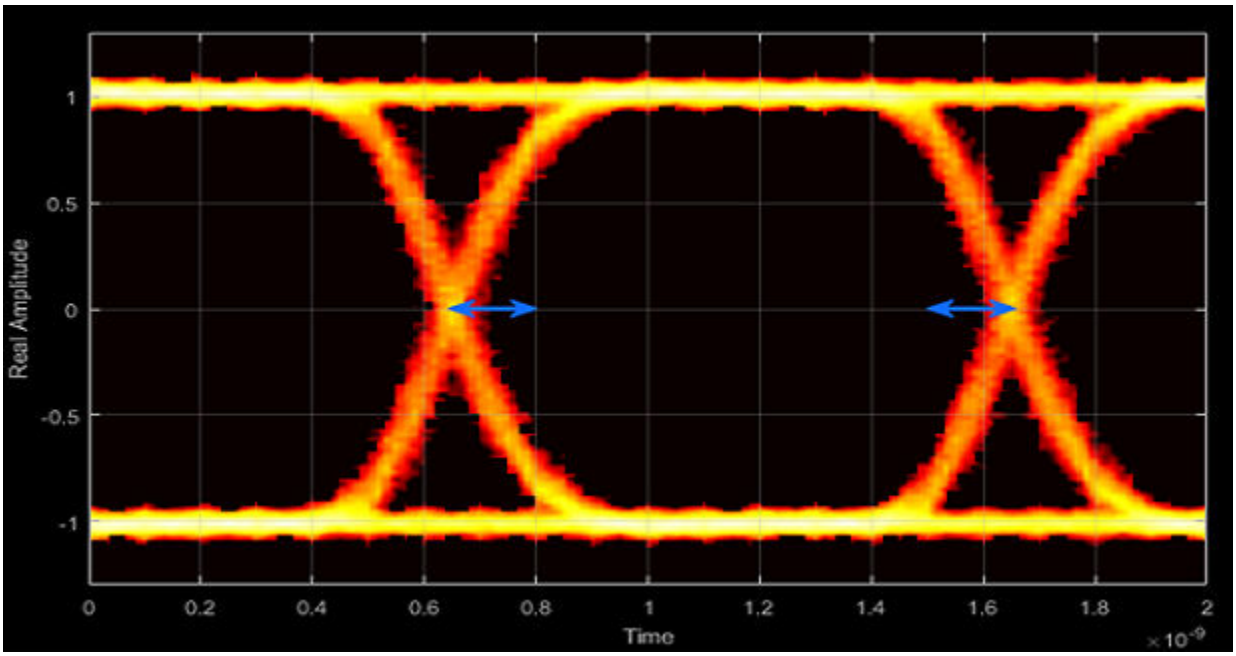
RJ is the Gaussian unbounded jitter component. The random component of jitter is modeled as a zero-mean Gaussian random variable with a specified standard-deviation of σ. The RJ is computed as:

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

where

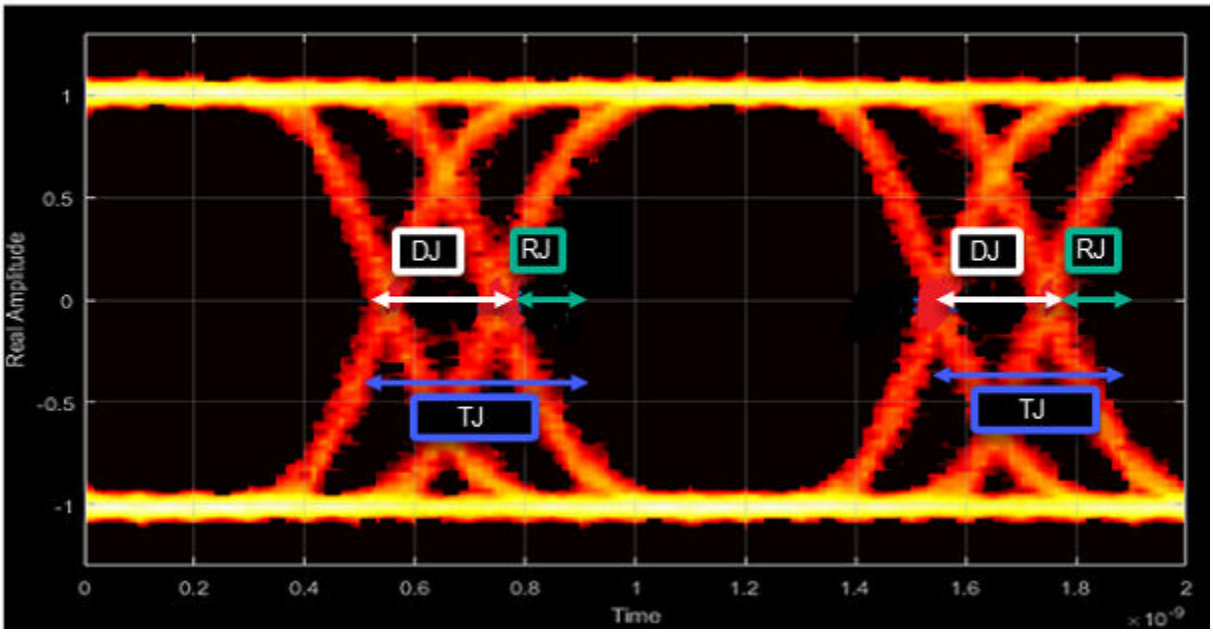
$$Q = \sqrt{2}\mathrm{erfc}^{-1}\left(2\frac{BER}{\rho}\right).$$

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.

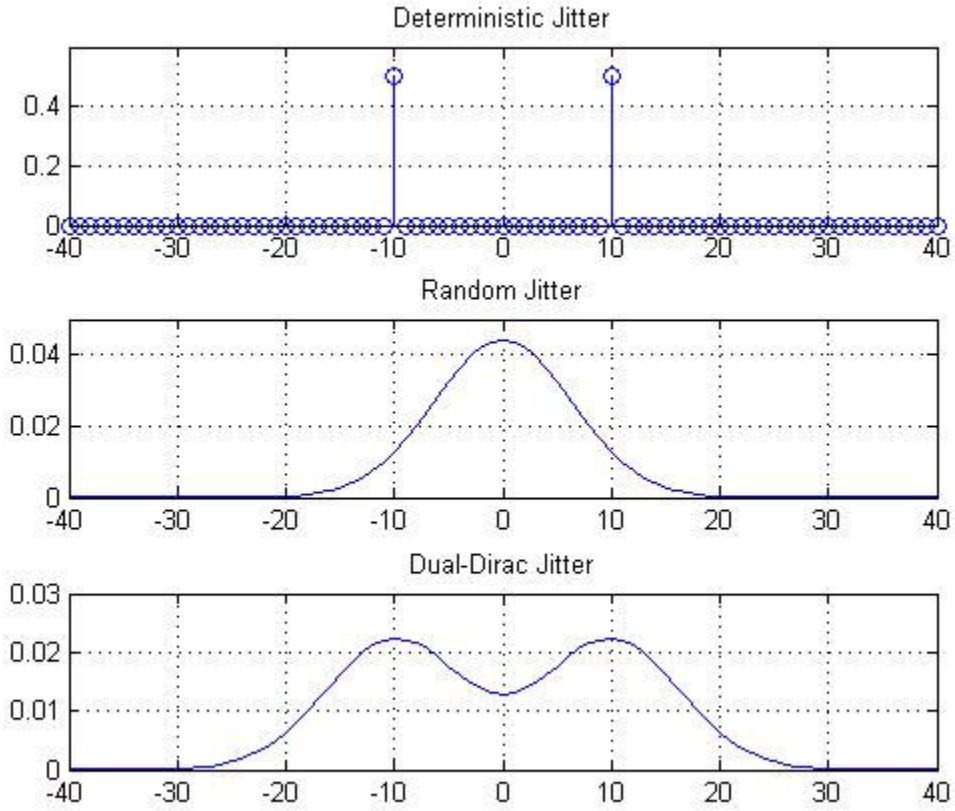


Total Jitter - Deviation from ideal signal timing

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

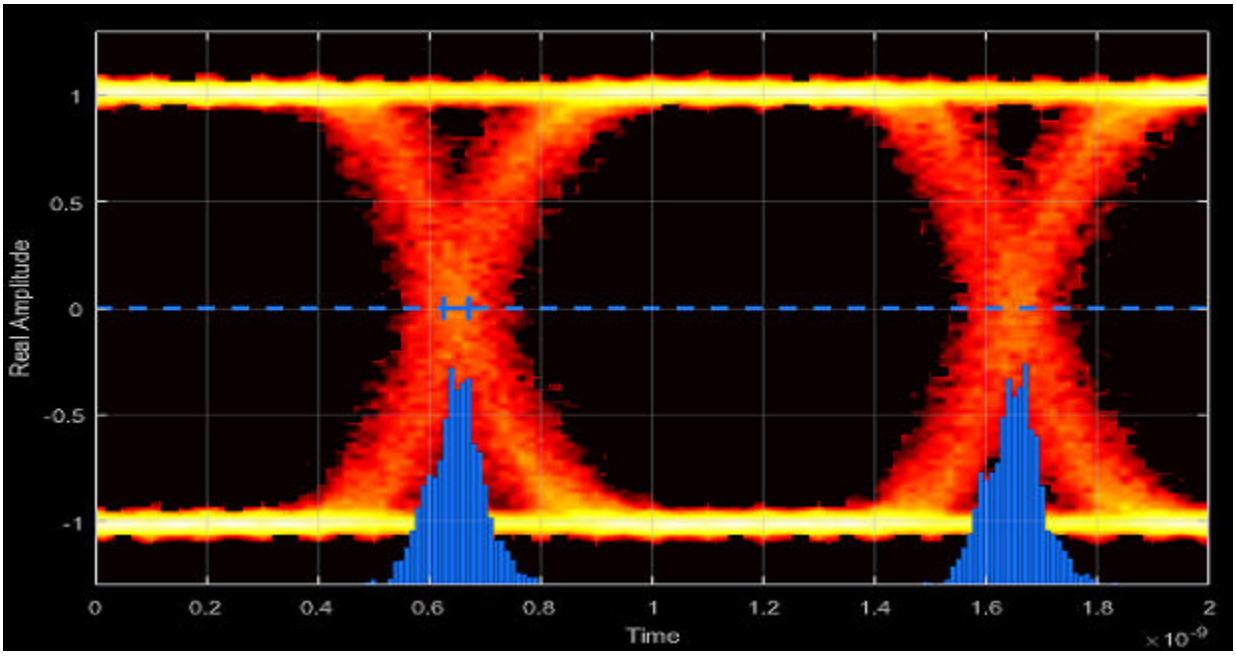


The total jitter PDF is the convolution of the DJ PDF and the RJ PDF.



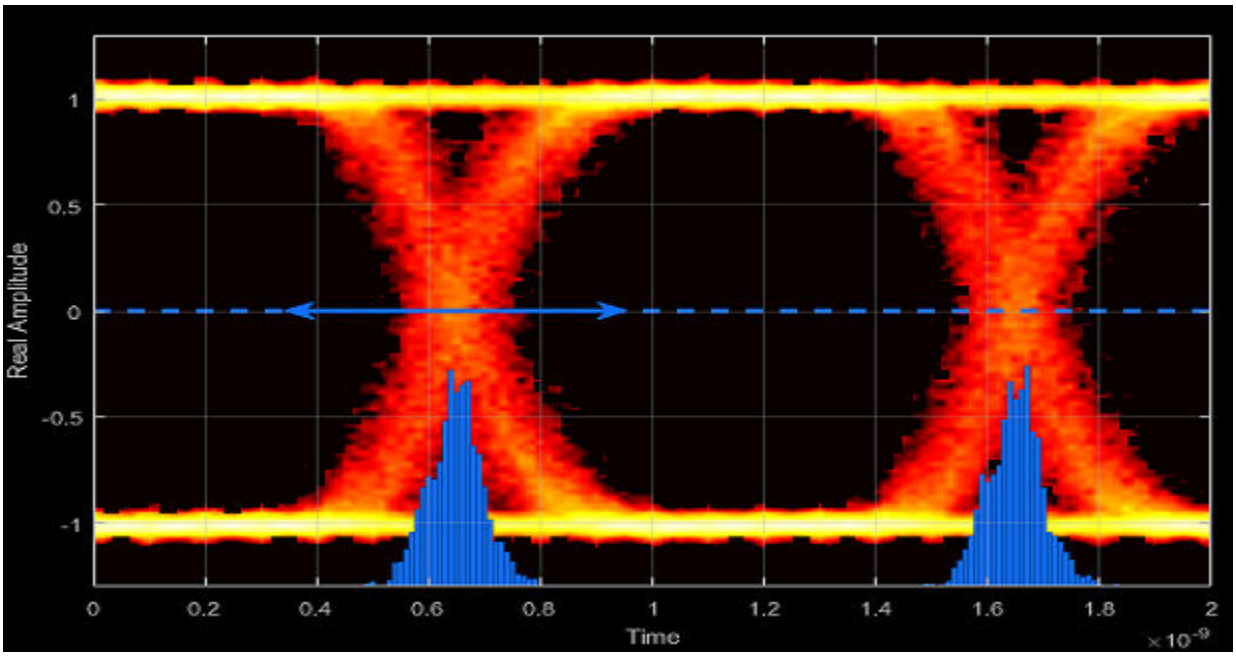
RMS Jitter - Standard deviation of jitter

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.

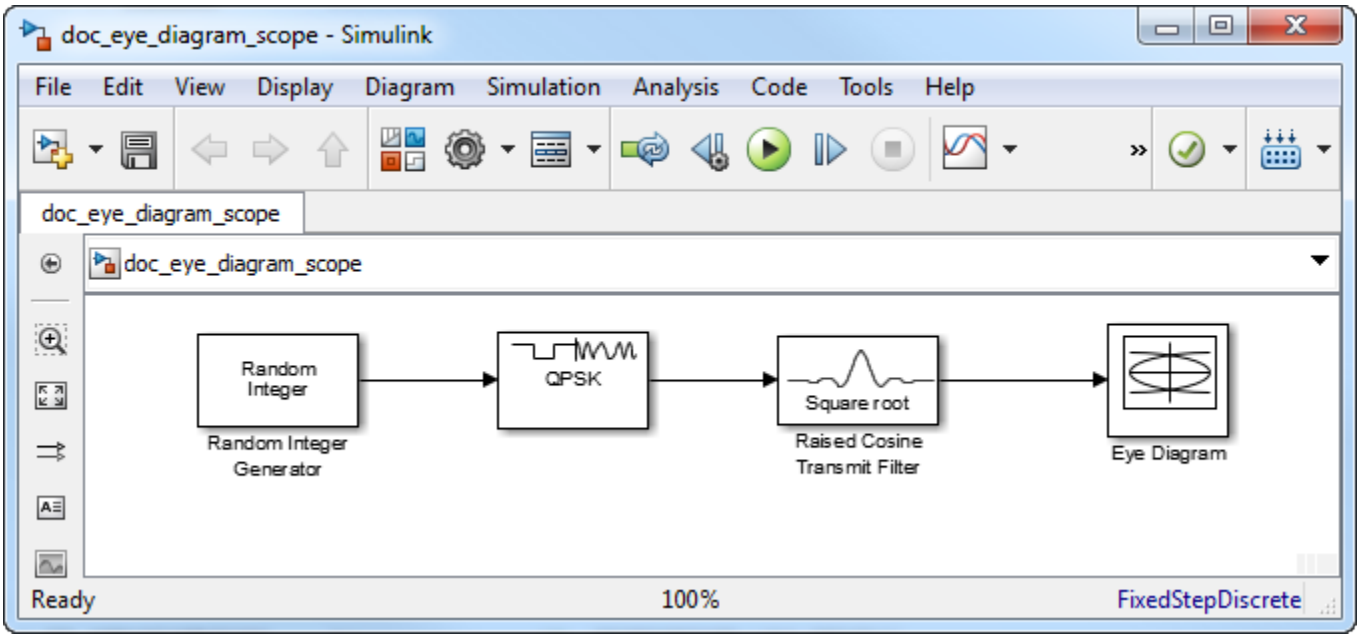


View Eye Diagram

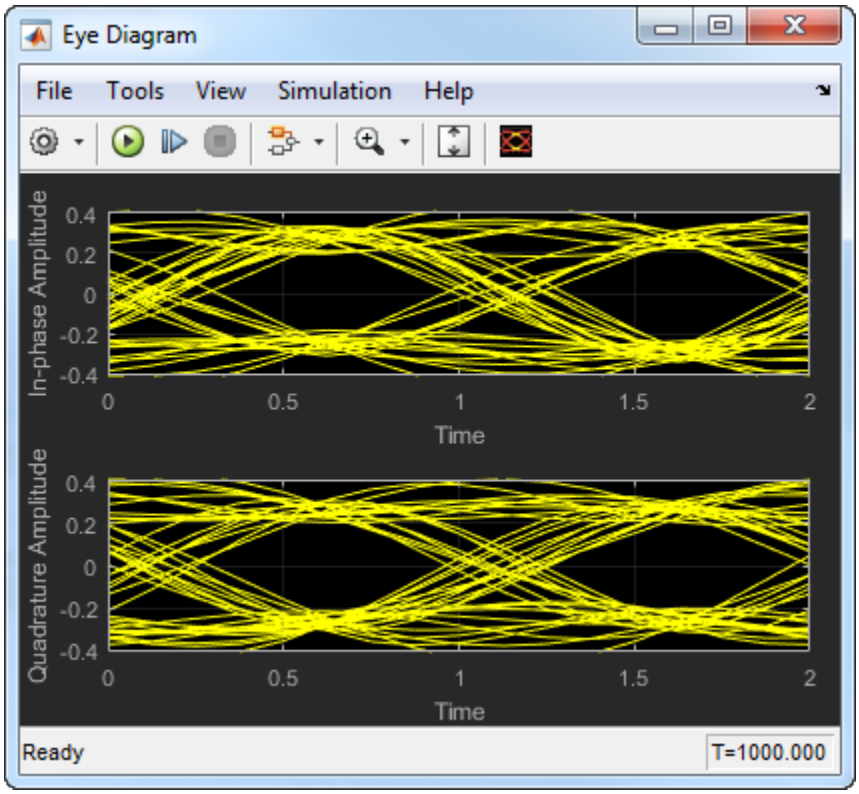
Display the eye diagram of a filtered QPSK signal using the Eye Diagram block.

Load the doc_eye_diagram_scope model from the MATLAB command prompt.

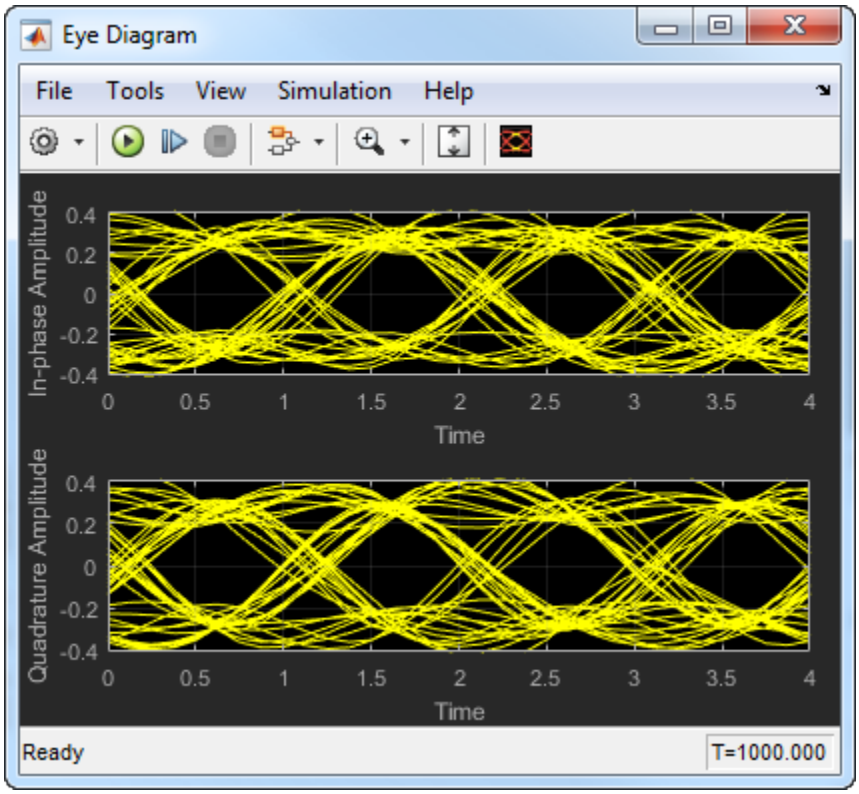
```
doc_eye_diagram_scope
```



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.



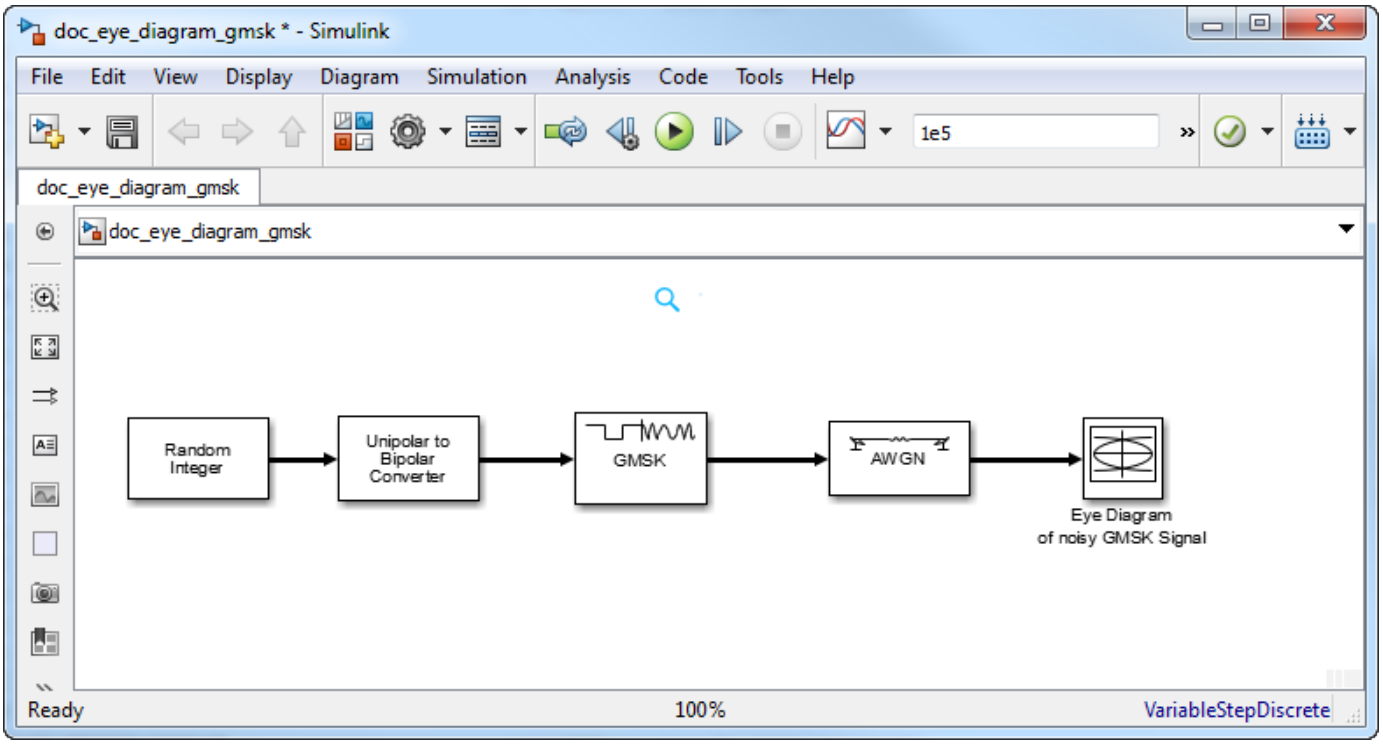
Try changing the Raised Cosine Transmit Filter parameters or changing additional Eye Diagram parameters to see their effects on the eye diagram.

Histogram Plots

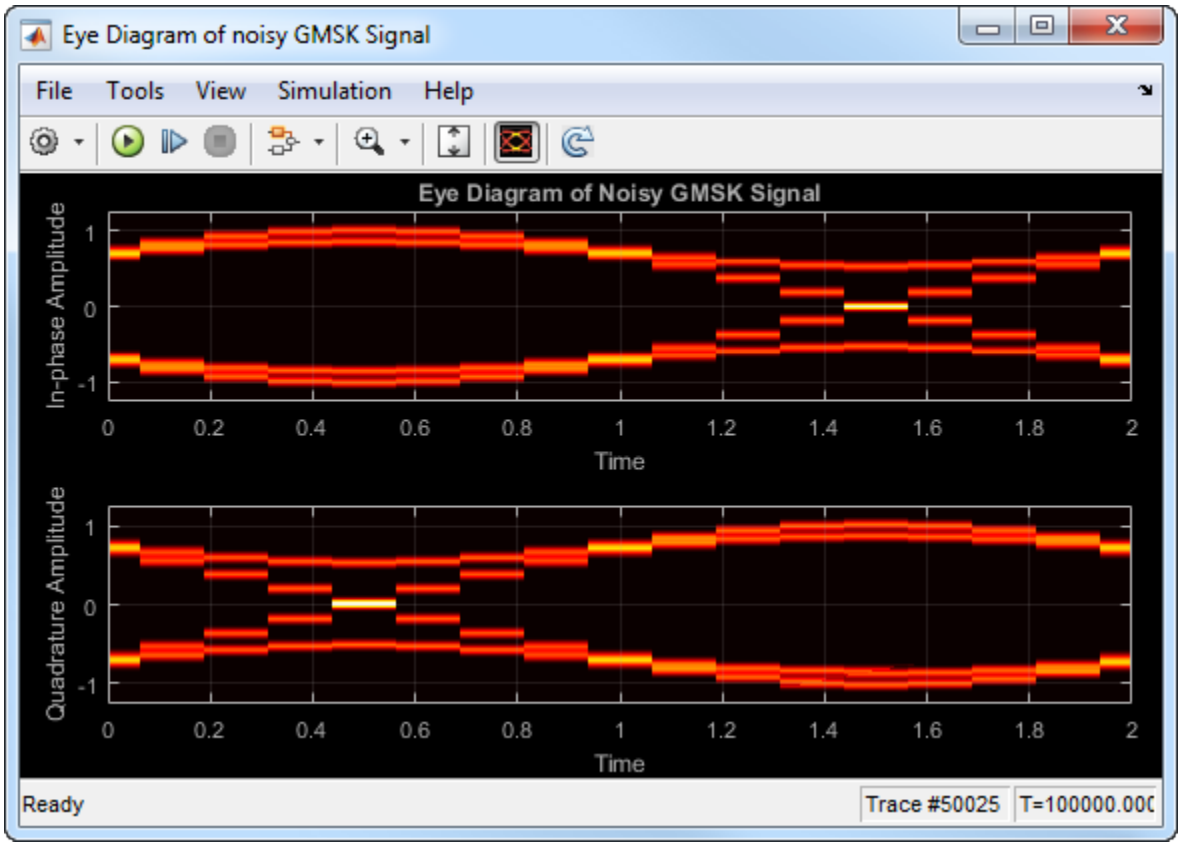
Display histogram plots of a noisy GMSK signal.

Load the doc_eye_diagram_gmsk model from the MATLAB command prompt.

```
doc_eye_diagram_gmsk
```

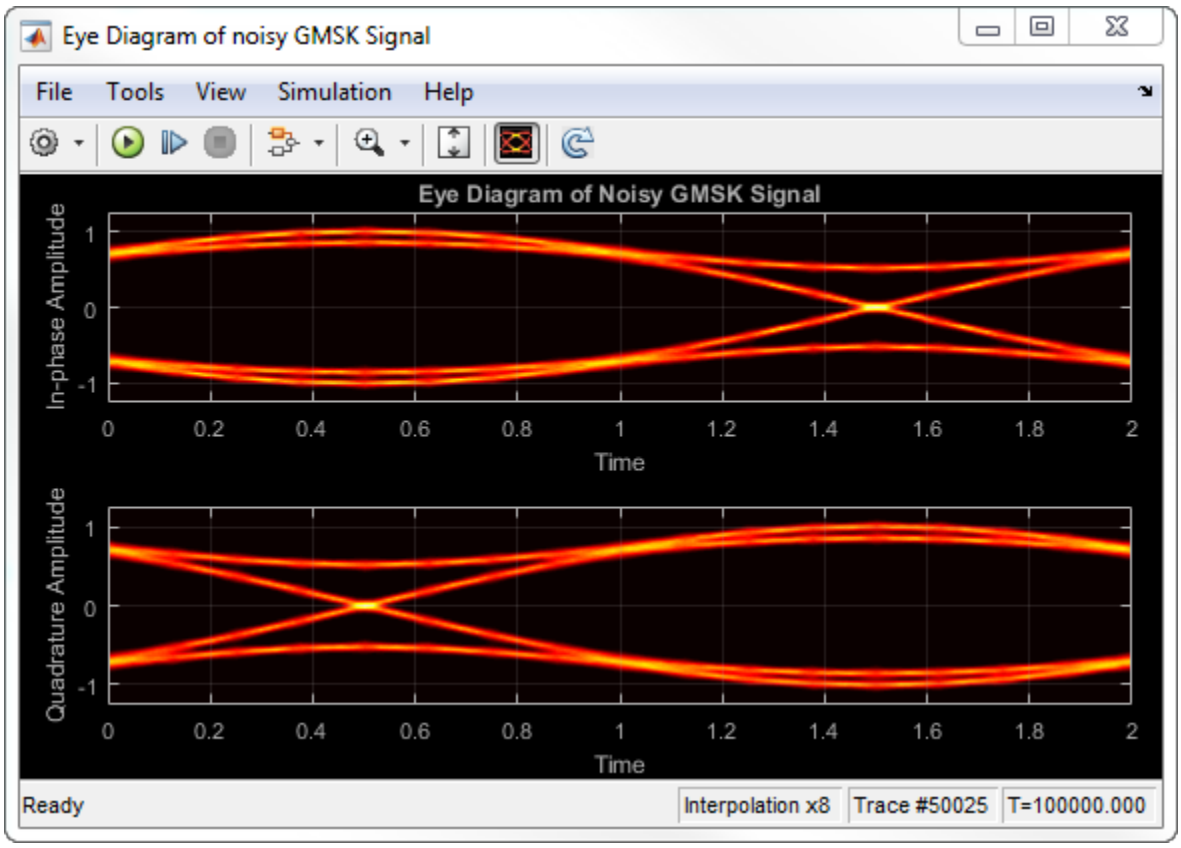


Run the model. The eye diagram is configured to show a histogram without interpolation.



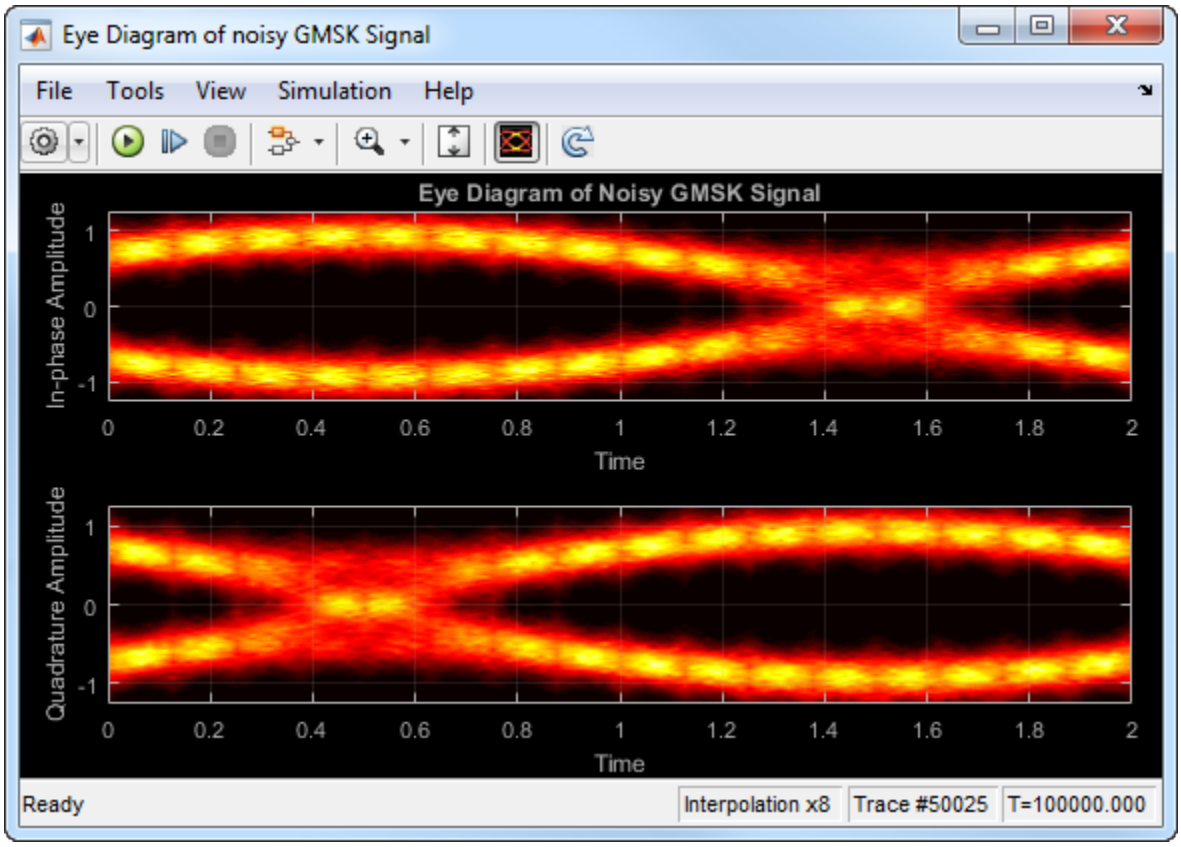
The lack of interpolation results in a plot having piecewise-continuous behavior.

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



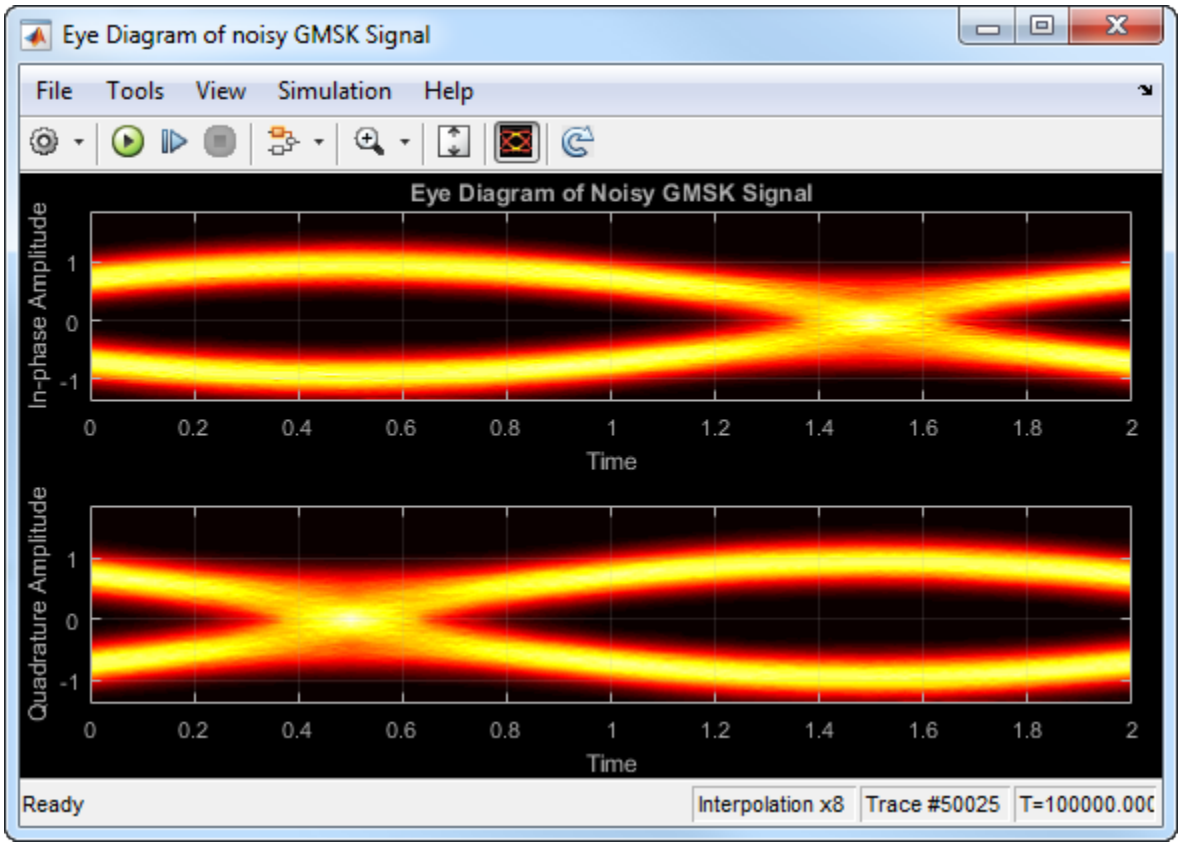
The interpolation smooths the eye diagram.

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.

Programmatic Configuration

You can programmatically configure the scope properties with callbacks or within scripts by using a scope configuration object as describe in .

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

Objects

Topics

Introduced in R2014a

FFE

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

Description

The FFE block applies a feed-forward equalizer (FFE) as a symbol-spaced finite-impulse response (FIR) filter to a sample-by-sample input signal or an impulse response vector input signal. This filtering reduces 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

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

FFE operating mode:

- `Off` — FFE is bypassed and the input waveform remains unchanged.
- `Fixed` — FFE applies the 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 volts. 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-cursor taps.

You can use a valid MATLAB expression to evaluate the **Initial tap weights (V)** row vector.

Example: `set_param(gcb,'TapWeights',"zeros[(1,4) 1 zeros(1,15)]")` creates an FFE with 20 taps where the fifth tap is the main tap.

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.

Data Types: `double`

`Normalize` — Normalize tap weights

on (default) | fff

Select to normalize tap weight vectors so that the sum of the absolute values of the **Tap weights** vector elements is one.

IBIS-AMI parameters

`Mode` — Include Mode parameter in IBIS-AMI model

on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

`Tap weights` — Include Tap weights parameter in IBIS-AMI model

on (default) | off

Select to include **Tap weights** as a parameter in the IBIS-AMI file. If you deselect **Tap weights**, it is removed from the AMI files, effectively hard-coding **Tap weights** to its current value.

See Also

CTLE | `serdes.CTLE` | `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`

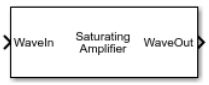
See Also
CTLE | DFECDR | FFE | `serdes.CTLE` | `serdes.DFECDR` | `serdes.FFE` | `serdes.PassThrough`

Topics
“Managing AMI Parameters”

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. The SaturatingAmplifier block applies memoryless nonlinearity to incoming waveform.

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

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

Amplifier operating mode:

- `Off` — SaturatingAmplifier is bypassed and 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:

- `'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**.

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.

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.

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.

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.

Data Types: `double`

`VinVout` — Input and corresponding output voltage response table

N×2 matrix

Input and corresponding output voltage response table, specified as an $N \times 2$ matrix in volts.

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.

Data Types: `double`

IBIS-AMI parameters

`Mode` — Include Mode parameter in IBIS-AMI model

on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

See Also

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

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 editable 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.

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

off (default) | on

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`

`Recommended simulation stop time (s)` — Minimum time simulation must run for meaningful results
`2e-7` (default) | positive real scalar

Minimum time the simulation must run for meaningful results, specified as a positive real scalar in seconds. **Recommended simulation stop time (s)** is calculated by multiplying the **Sample Interval** from the Configuration block with the **Number of symbols**.

Data Types: `double`

`Use recommended simulation stop time` — Set recommended simulation stop time as simulation stop time
on (default) | off

Select to directly set the **Recommended simulation stop time** as the simulation stop time. By default, this option is selected.

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

Data Types: `double`

Parameters

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

VGA operating mode:

- `Off` — VGA is bypassed and 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**.

`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

`Mode` — Include Mode parameter in IBIS-AMI model
on (default) | off

Select to include **Mode** as a parameter in the IBIS-AMI file. If you deselect **Mode**, it is removed from the AMI files, effectively hard-coding **Mode** to its current value.

`Gain` — Include Gain parameter in IBIS-AMI model
on (default) | off

Select to include **Gain** as a parameter in the IBIS-AMI file. If you deselect **Gain**, it is removed from the AMI files, effectively hard-coding **Gain** to its current value.

See Also

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

Introduced in R2019a

Apps

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 block to set up the transmitter.

Use the AnalogIn block 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

Configuring Analog Channel

Use the Channel block to set up the model of the analog channel. The Channel block constructs a loss model using a channel loss metric or an impulse response.

You can also introduce crosstalk in your simulations. You can specify the maximum allowed crosstalk for specifications such as `100GBASE-CR4`, `CEI-25G-LR`, `CEI-28G-SR`, `CEI-28G-VSR`. Or you can specify a custom integrated crosstalk noise (ICN) level.

Statistical Analysis

From the **SerDes Designer** app toolstrip, go to **Analysis** tab and select **Add Plots** to perform statistical (Init) analysis. By default, the app selects the **Auto-Analyze** button and automatically updates the plot results every time you make a change in the SerDes system. To update the plot at your preference, clear the **Auto-Analyze** button and update the plot by clicking the **Analyze** button.

You can view these plots using the app:

- Pulse Response
- Impulse Response
- Statistical Eye
- PRBS (pseudorandom binary sequence) Waveform
- Contours
- Bathtub
- BER (bit error rate)
- CTLE Transfer Function

You can also view important performance metrics of the SerDes system by selecting **Report** on the **Add Plots** tab.

Performance Metric	Description
Eye Height (V)	Eye height at the center of the BER contour
Eye Width (ps)	Eye width of the BER contour
Eye Area (V*ps)	Area inside the BER contour eye
Eye Linearity	Measure of the variance of amplitude separation among different PAM4 levels, given by the equation: $\text{Linearity} = \frac{\text{Minimum amplitude of the three eye levels}}{\text{Maximum amplitude of the three eye levels}}$
COM	Channel operating margin, given by the equation: $\text{COM} = 20\log_{10}\left(\frac{\text{Mean eye height}}{\text{Mean eye height - Inner eye height}}\right)$
VEC	Vertical eye closure, given by the equation: $\text{VEC} = \frac{\text{Mean eye height}}{\text{Inner eye height}}$

Note In a PAM4 modulation, there are three inner eyes. Each eye generates its own COM and VEC values. The app reports the minimum of the three COM values and the maximum of the three VEC values.

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**

The Simulink model of the SerDes system contains a Configuration block that contains the system level settings, a Stimulus block to use in time-domain simulation and an Eye Diagram Scope to view the statistical eye.

The Tx and Rx blocks are automatically generated by the **SerDes Designer** app. The equalization blocks are placed inside them to allow IBIS-AMI model generation.

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. For a list of supported compilers, see Supported and Compatible Compilers.
- 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”

External Websites
Supported and Compatible Compilers

Introduced in R2019a