

**SerDes Toolbox™**

Reference



**MATLAB® & SIMULINK®**

R2023a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

## *SerDes Toolbox™ Reference*

© COPYRIGHT 2019–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## **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)
September 2020	Online only	Revised for Version 2.0 (Release 2020b)
March 2021	Online only	Revised for Version 2.1 (Release 2021a)
September 2021	Online only	Revised for Version 2.2 (Release 2021b)
March 2022	Online only	Revised for Version 2.3 (Release 2022a)
September 2022	Online only	Revised for Version 2.4 (Release 2022b)
March 2023	Online only	Revised for Version 3.0 (Release 2023a)

<b>1</b>	<b>Functions</b>
<b>2</b>	<b>System Objects</b>
<b>3</b>	<b>Blocks</b>
<b>4</b>	<b>Apps</b>
<b>5</b>	<b>Class</b>



# 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.

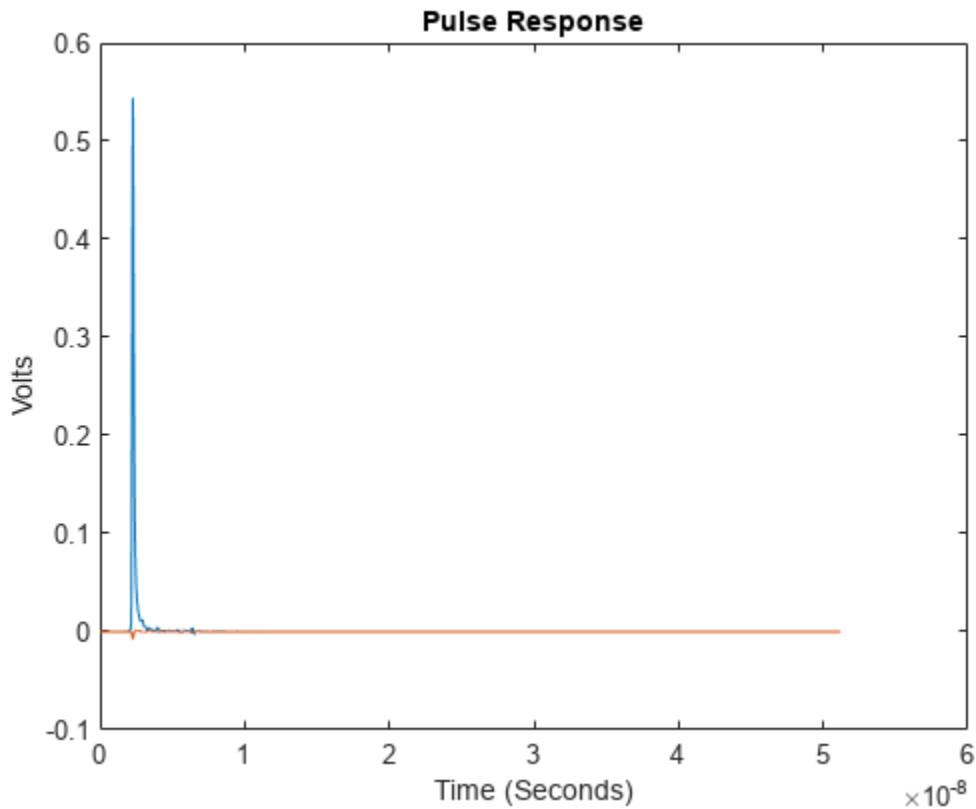
```
load('PulseResponseReflective100ps.mat');
```

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`

## Version History

Introduced in R2020a

## Extended Capabilities

### **C/C++ Code Generation**

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

### **See Also**

`impulse2step` | `pulse2impulse`



# 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.

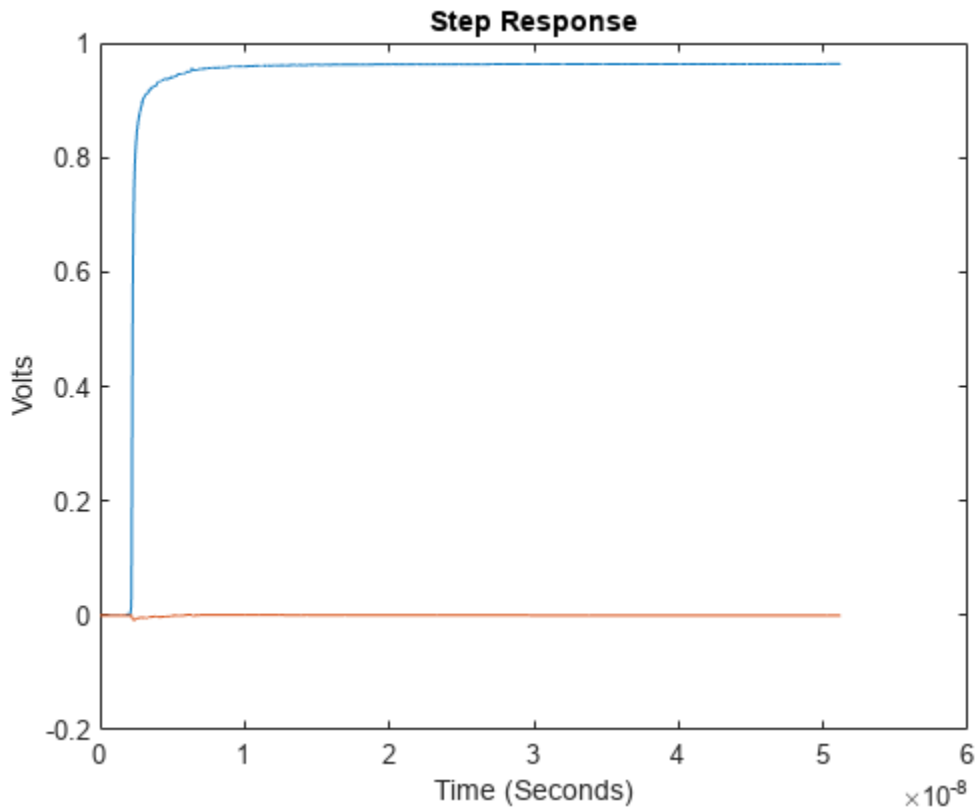
```
load('PulseResponseReflective100ps.mat');
```

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

## **Version History**

**Introduced in R2020a**

## **Extended Capabilities**

### **C/C++ Code Generation**

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

## **See Also**

[impulse2pulse](#) | [step2impulse](#)

## optPulseMetric

Pulse response metric for optimization routines

### Syntax

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

### Description

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

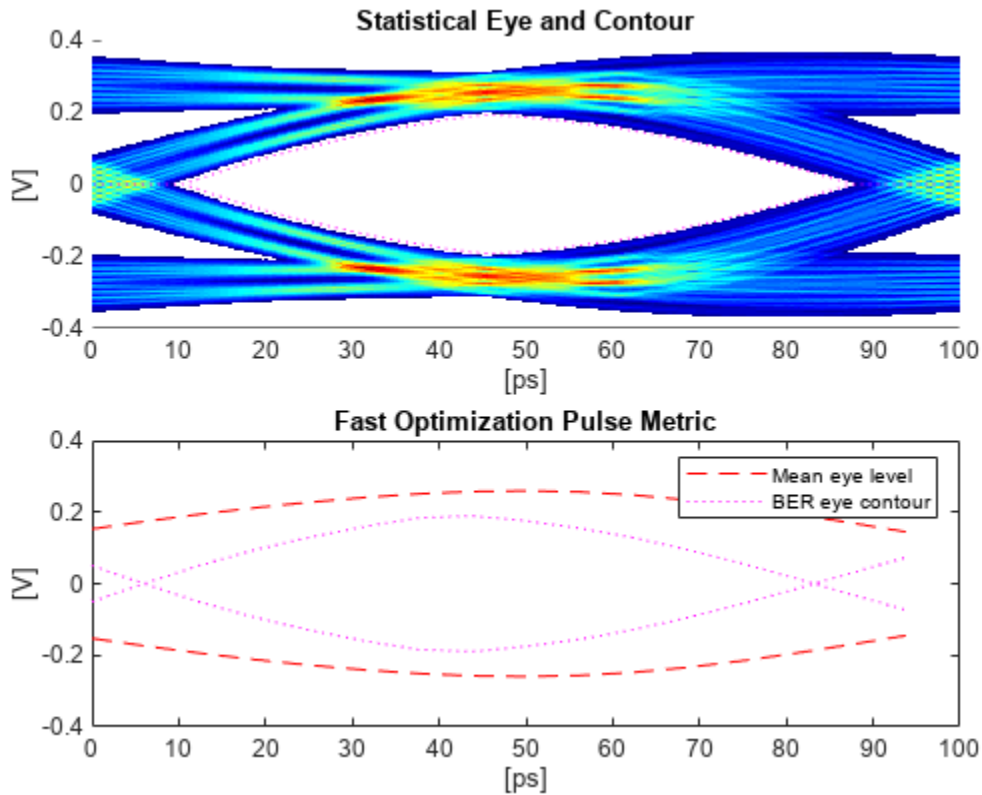
Calculate the statistical eye and the eye metrics.

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

    subplot(2,1,2)
    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:')
    legend('Mean eye level','BER eye contour')
    title('Fast Optimization Pulse Metric')
    ylabel('[V]')
    xlabel('[ps]')
    axis(ax)
```



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

### **N — Number of samples per symbol**

real positive scalar

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

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

### **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 — Pulse response metrics**

structure

Pulse response metrics, returned as a structure with these fields:

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:

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

`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.

## **Version History**

**Introduced in R2020a**

## **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™.



## prbs

Pseudorandom binary sequence

### Syntax

```
PRBS = prbs(0,N)
[PRBS,seed] = prbs(0,N)
[PRBS,seed] = prbs(0,N,seed)
PRBS = prbs(0,N,seed,reverse)
```

### Description

`PRBS = prbs(0,N)` calculates a pseudorandom binary sequence `P` of order `0` and length `N`.

`[PRBS,seed] = prbs(0,N)` calculates a pseudorandom binary sequence and the seed needed to continue the sequence.

`[PRBS,seed] = prbs(0,N,seed)` calculates a pseudorandom binary sequence and the seed needed to continue the sequence using the `seed` value.

`PRBS = prbs(0,N,seed,reverse)` uses the reverse linear-feedback shift register (LFSR) tap positions and returns a pseudorandom binary sequence pattern that is essentially the same as running the LFSR backward in time.

time.

### Examples

#### Create PRBS Pattern of Order 4

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

```
0 = 4;
N = 2^0-1;
pattern1 = prbs(0,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(0,1);
    for ii = 2:N
        [pattern2(ii),seed] = prbs(0,1,seed);
    end
```

Verify that both patterns are the same.

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

1

## Input Arguments

### **0 — 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

### **reverse — LFSR tap positions used to calculate PRBS pattern**

vector

The reverse linear-feedback shift register (LFSR) tap positions used to calculate a pseudorandom binary sequence pattern. The generated pattern is essentially the same as running the LFSR backward in time.

You can find the LFSR tap positions used by using the command `[~,~,tapPosition]=prbs(0,N)`.

Data Types: double

## Output Arguments

### **PRBS — Pseudorandom binary sequence pattern**

vector

Pseudorandom binary sequence pattern of order 0 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

## Version History

Introduced in R2020a

## Extended Capabilities

### **C/C++ Code Generation**

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

**See Also**

wave2pulse | pulse2wave

## step2impulse

Impulse response from step response

### Syntax

```
I = step2impulse(S,dt)
```

### Description

`I = step2impulse(S,dt)` converts a step response `S` to an impulse response `I`, given the uniform sample interval `dt`.

### Examples

#### Create Impulse Response from Step Response.

Load the step response column matrix from a file.

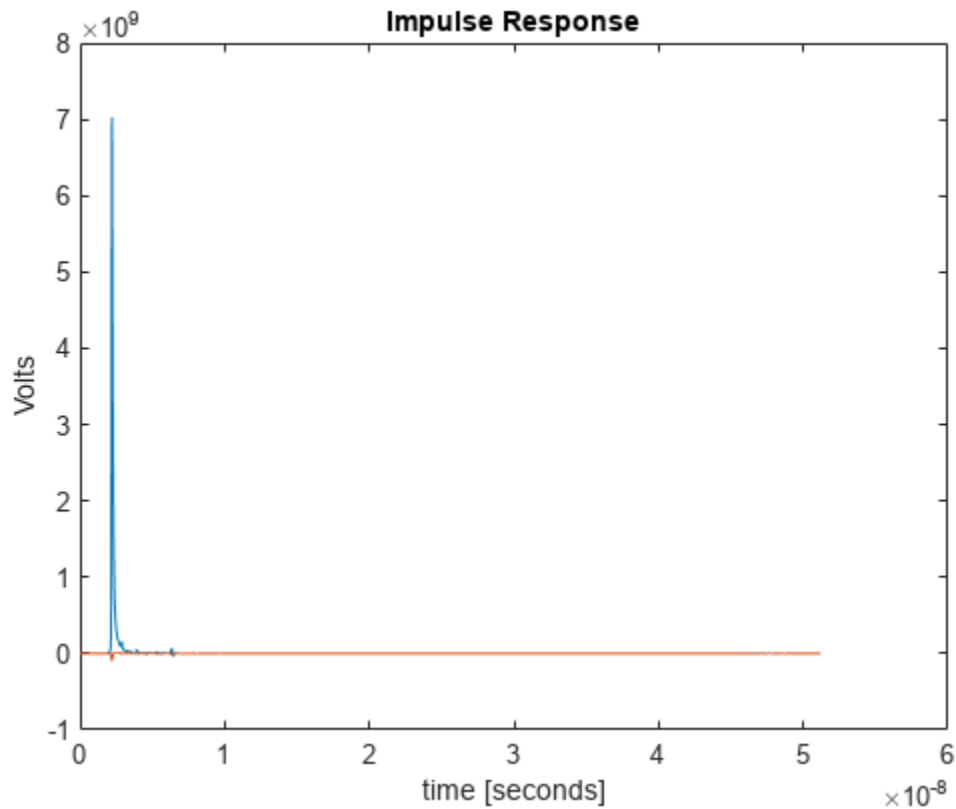
```
load('StepResponseReflective100ps.mat');
```

Calculate the impulse response from the step response.

```
impulse = step2impulse(step,dt);
```

Plot the step response.

```
plot(t(1:end-1),impulse)
xlabel('time [seconds]')
ylabel('Volts')
title('Impulse Response')
```



## Input Arguments

### **S** – Step response

column matrix

Input step response, specified 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

### **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. The

number of rows in the impulse response must be one less than the number of rows in the step response.

Data Types: double

## **Version History**

**Introduced in R2021b**

## **Extended Capabilities**

### **C/C++ Code Generation**

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

### **See Also**

impulse2step

# 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.

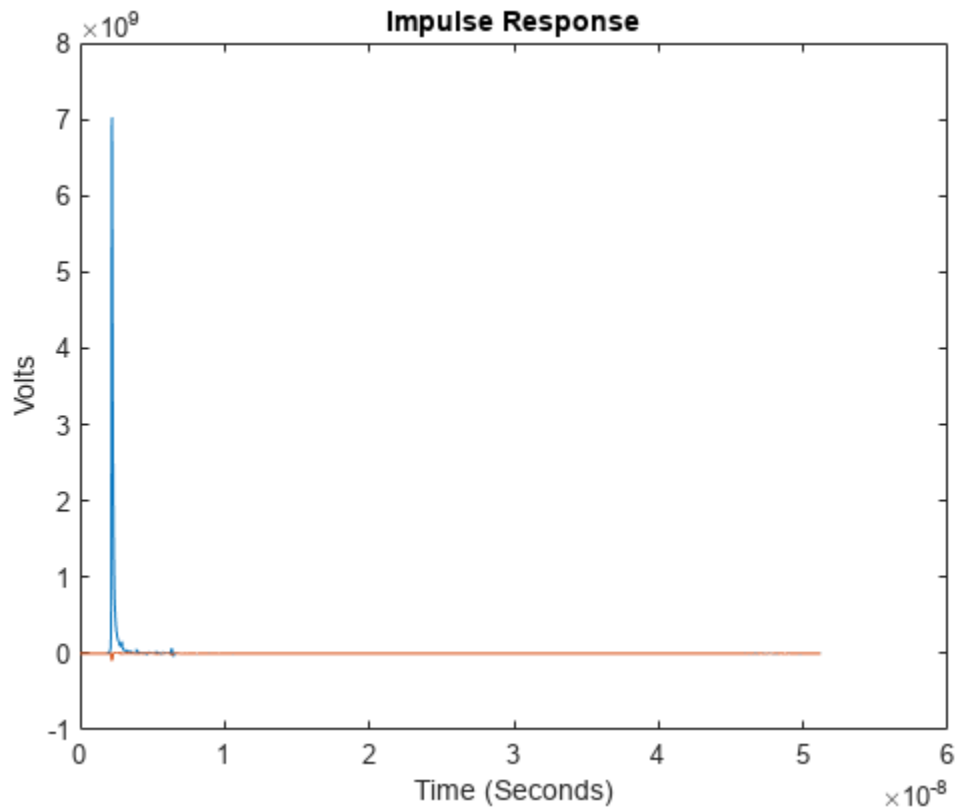
```
dt = SymbolTime/SamplesPerSymbol;
```

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`

## Version History

Introduced in R2020a

## Extended Capabilities

### **C/C++ Code Generation**

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

### **See Also**

`pulse2wave` | `impulse2pulse`

## 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.

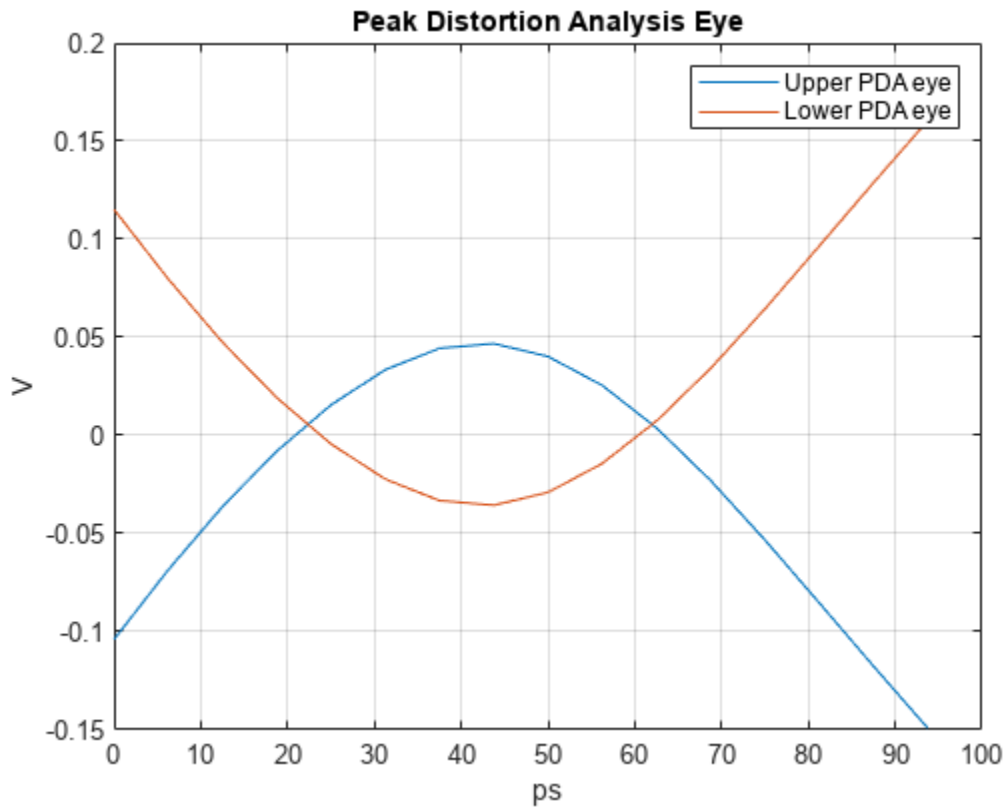
```
load('PulseResponseReflective100ps.mat');
```

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`

**Version History**

Introduced in R2020a

**See Also**

`pulse2stateye`

# pulse2stateye

Statistical eye from pulse response

## Syntax

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

## 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.

The function uses Hula-Hoop algorithm.

`[ ___,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.

`[S,VH,TH] = pulse2stateye(P,N,M, ClockIndex)` centers the resulting eye diagram at the clock index instead of using the Hula-Hoop algorithm.

## 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);
```

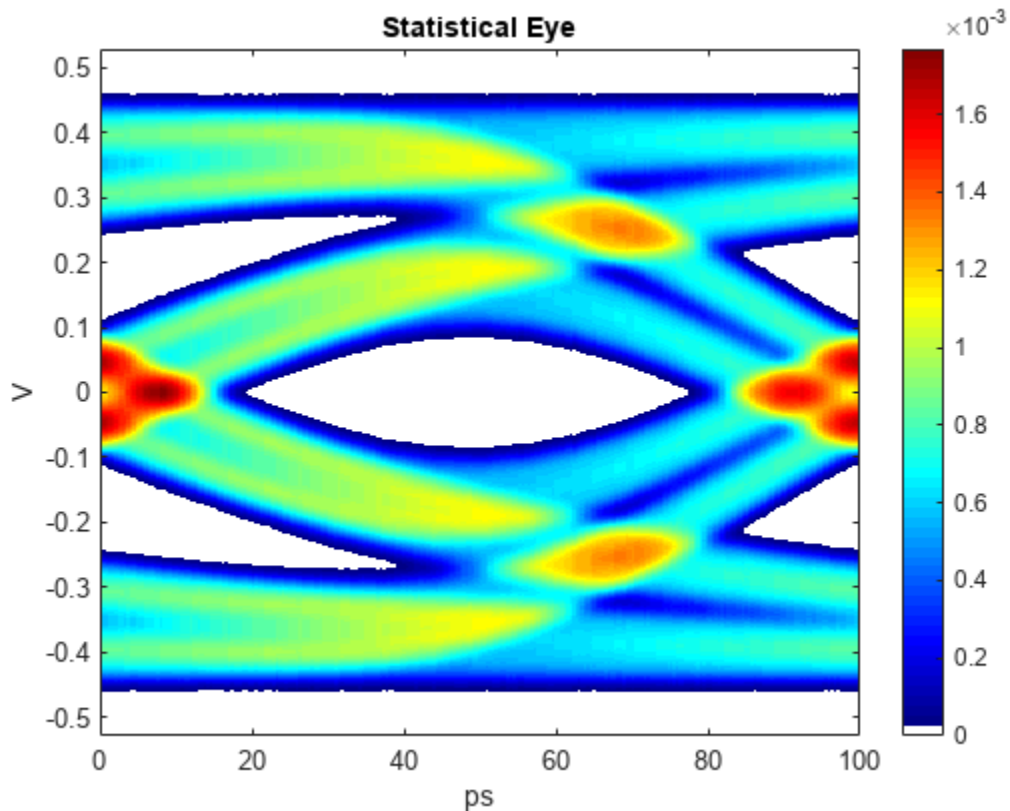
Visualize the statistical eye using a color map.

```
cmap = serdes.utilities.SignalIntegrityColorMap;
figure,
imagesc(th*SymbolTime*1e12,vh,stateye)
```

```

colormap(cmap)
colorbar
axis('xy')
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

### **ClockIndex — Index of pulse response where clock is centered**

scalar

Index of the pulse response where the clock is centered, specified as a scalar. When you specify the optional `ClockIndex` argument, the function centers the resulting eye diagram at the `ClockIndex`.

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

## **Version History**

**Introduced in R2020a**

### **See Also**

`pulse2pda`

## 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);
```

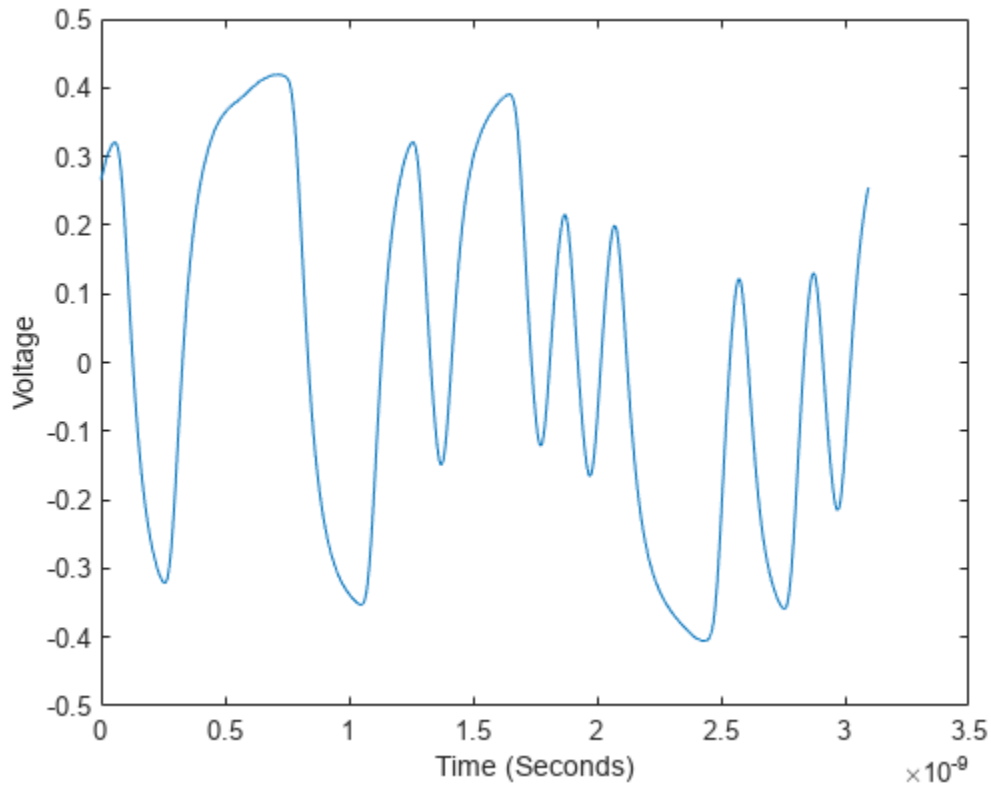
Create the symbol pattern of PRBS order 5.

```
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

## Version History

Introduced in R2020a

## Extended Capabilities

### **C/C++ Code Generation**

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

### **See Also**

[pulse2impulse](#) | [wave2pulse](#)

# 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  $(N \cdot K)$ , 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);
```

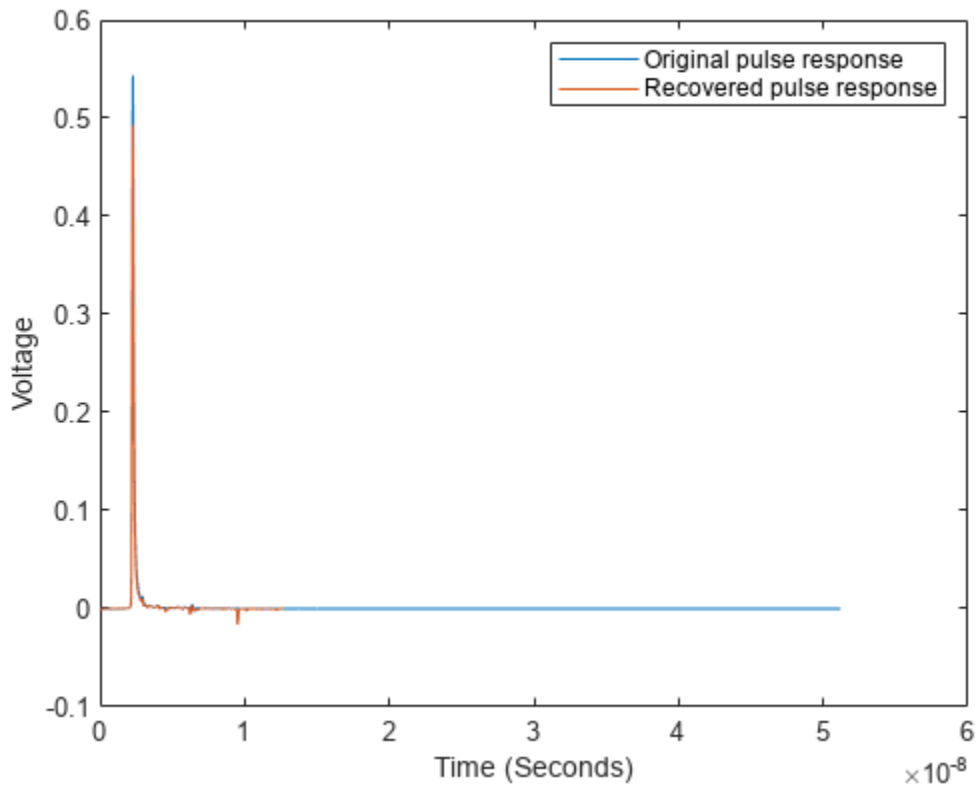
Recover the pulse response from the waveform `W2`.

```
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

**Version History**

Introduced in R2020a

**Extended Capabilities****C/C++ Code Generation**

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

**See Also**

pulse2wave



# System Objects

---

## serdes.AGC

**Package:** serdes

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?](#)

### 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](#).



**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) | 3 | 4 | 8 | 16

Modulation value that specifies the modulation scheme, the number of logic levels in the encoded signal, specified as 2, 3, 4, 8, or 16.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
3	Three-level pulse amplitude modulation (PAM3)
4	Four-level pulse amplitude modulation (PAM4)
8	Eight-level pulse amplitude modulation (PAM8)
16	Sixteen-level pulse amplitude modulation (PAM16)

**Note** According to IBIS BIRD (Buffer Issue Resolution Document) 213, IBIS-AMI models support any level of signaling from PAM2 (NRZ) to upwards, collectively known as PAMn. If your EDA tool supports it, you can export IBIS-AMI models supporting modulation schemes NRZ, PAM3, PAM4, PAM8, or PAM16.

---

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**

$y = \text{agc}(x)$

### **Description**

$y = \text{agc}(x)$

### **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.

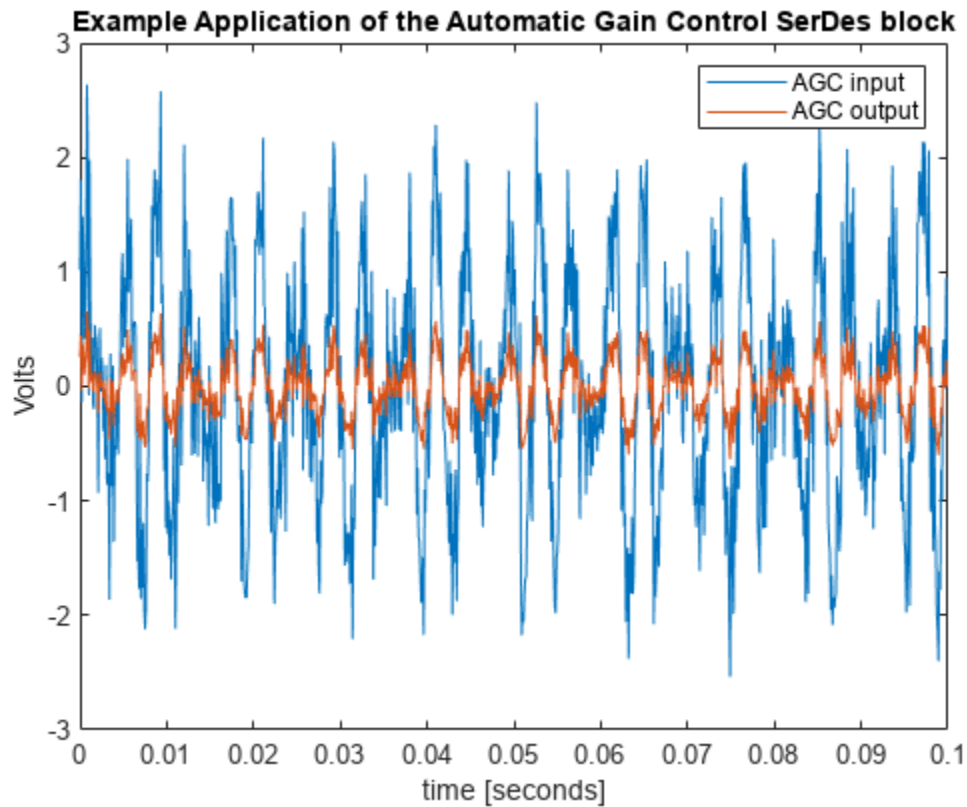
```
Fs = 10000;
L = 1000;
t = (0:L-1)/Fs;
x = sin(2*pi*250*t) + 0.75*cos(2*pi*340*t);           % Original signal
y = x + .5*randn(size(x));                          % Noisy signal
```

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');
```



## Version History

Introduced in R2019a

## References

- [1] *Design and Operation of Automatic Gain Control Loops for Receivers in Modern Communication Systems*, [https://www.analog.com/media/en/training-seminars/tutorials/42575412022953450461111812375Design\\_and\\_Operation\\_of\\_AGC\\_Loops.pdf](https://www.analog.com/media/en/training-seminars/tutorials/42575412022953450461111812375Design_and_Operation_of_AGC_Loops.pdf).

## 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

# serdes.CDR

**Package:** serdes

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?](#)

## 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](#).

**Main****Count — Early or late CDR count threshold to trigger phase update**16 (default) | real positive integer  $\geq 4$ 

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer  $\geq 4$ . 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  $\leq 300$ 

Reference clock offset impairment, specified as a real scalar  $\leq 300$  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 ( $\pm$ `Sensitivity`), there is a 50% probability of bit error..

Data Types: double

**PhaseDetector — Clock phase detector option**

BangBang (default) | BaudRateTypeA

Clock phase detector option used in the clock data recovery. You can choose between bang-bang (Alexander) or baud-rate type-A (Mueller-Muller).

**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) | 3 | 4 | 8 | 16

Modulation value that specifies the modulation scheme, the number of logic levels in the encoded signal, specified as 2, 3, 4, 8, or 16.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
3	Three-level pulse amplitude modulation (PAM3)
4	Four-level pulse amplitude modulation (PAM4)
8	Eight-level pulse amplitude modulation (PAM8)
16	Sixteen-level pulse amplitude modulation (PAM16)

**Note** According to IBIS BIRD (Buffer Issue Resolution Document) 213, IBIS-AMI models support any level of signaling from PAM2 (NRZ) to upwards, collectively known as PAMn. If your EDA tool supports it, you can export IBIS-AMI models supporting modulation schemes NRZ, PAM3, PAM4, PAM8, or PAM16.

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** $y = \text{cdr}(x)$ **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.

**Output Arguments****Phase — Relative recovered clock phase**

units of Symbol Time in the range [0,1]

Relative recovered clock phase, returned as a units of Symbol Time in the range [0,1].

**clkAMI — AMI clock bus**

structure

AMI clock bus, returned as a structure consisting of:

- ClockTime — Time taken to sample the data signal.
- clockValidOnRising — Valid clock time value on the rising edge of a signal

**Interior — Bus containing additional interior CDR signals**

structure

Bus containing additional interior CDR signals, returned as a structure consisting of:

- clockPhase — Relative clock phase in units of SymbolTime in the range of [0,1].
- symbolRecovered — Symbol recovered from data signal at ClockTime.
- voltageSample — Voltage observed from the data signal at ClockTime.
- PAM4Threshold — The estimated upper eye at PAM4 threshold.
- CDRedgeVoltage — The voltage observed from the data signal at ClockTime — SymbolTime/2.
- CDRCounter — The bang-bang CDR internal counter used to trigger samples.
- CDREarlyLateCounter — The bang-bang CDR accumulated (or filtered) signal used to trigger updated to the CDR phase.
- PAM4SymbolMiddleVoltage — Estimated PAM4 symbol voltage of the inner eye outer envelope to estimate PAM4 threshold.
- PAM4OuterVoltage — Estimated PAM4 outer envelope voltage to estimate PAM4 threshold.
- EyeHeightAbsAve — Estimates eye height.

**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
```

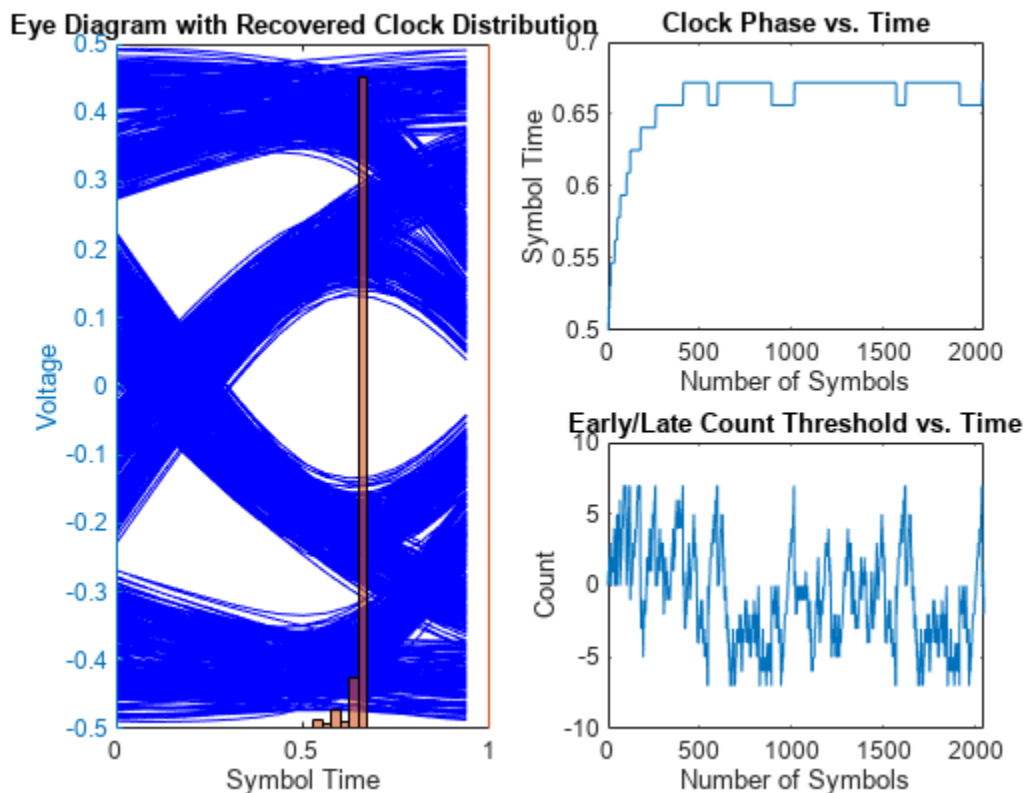
Plot the eye diagram with recovered clock distribution, clock phase vs. time, and early/late count threshold vs. time.

```
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')

```



## Version History

Introduced in R2019a

## 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**

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

## **Topics**

“Clock and Data Recovery in SerDes System”

## serdes.ChannelLoss

**Package:** serdes

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?](#)

## 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](#).

### 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 — 20%–80% 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.

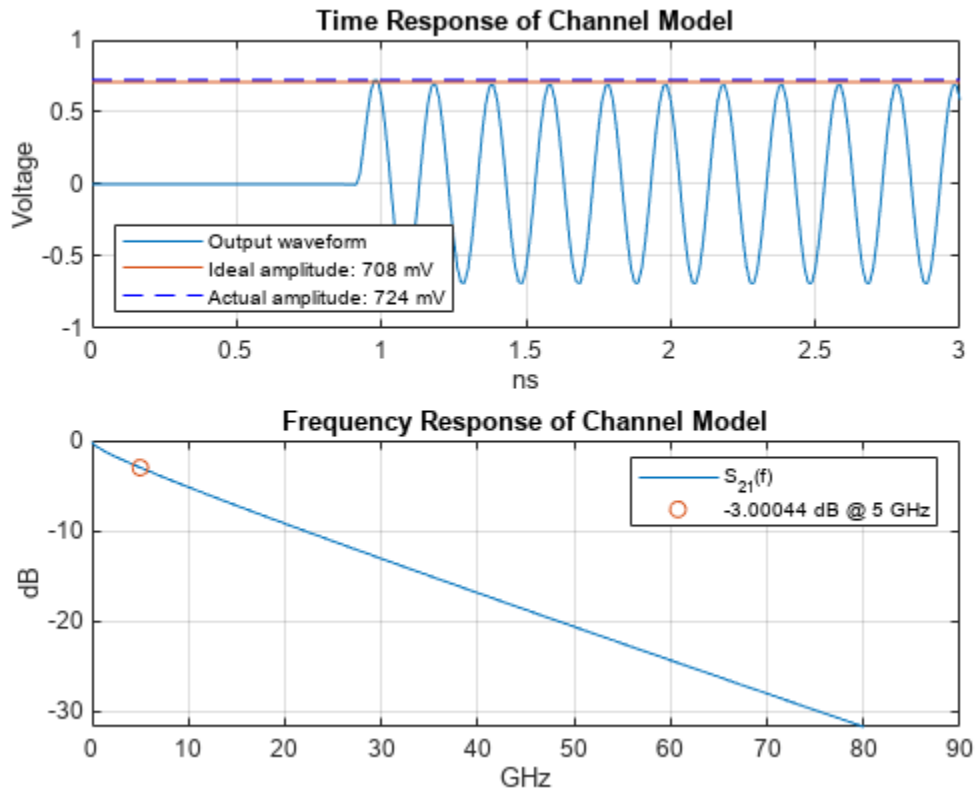
```
s21 = channelModel.s21;
f = (0:length(s21)-1)*channelModel.dF;
```

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')
```

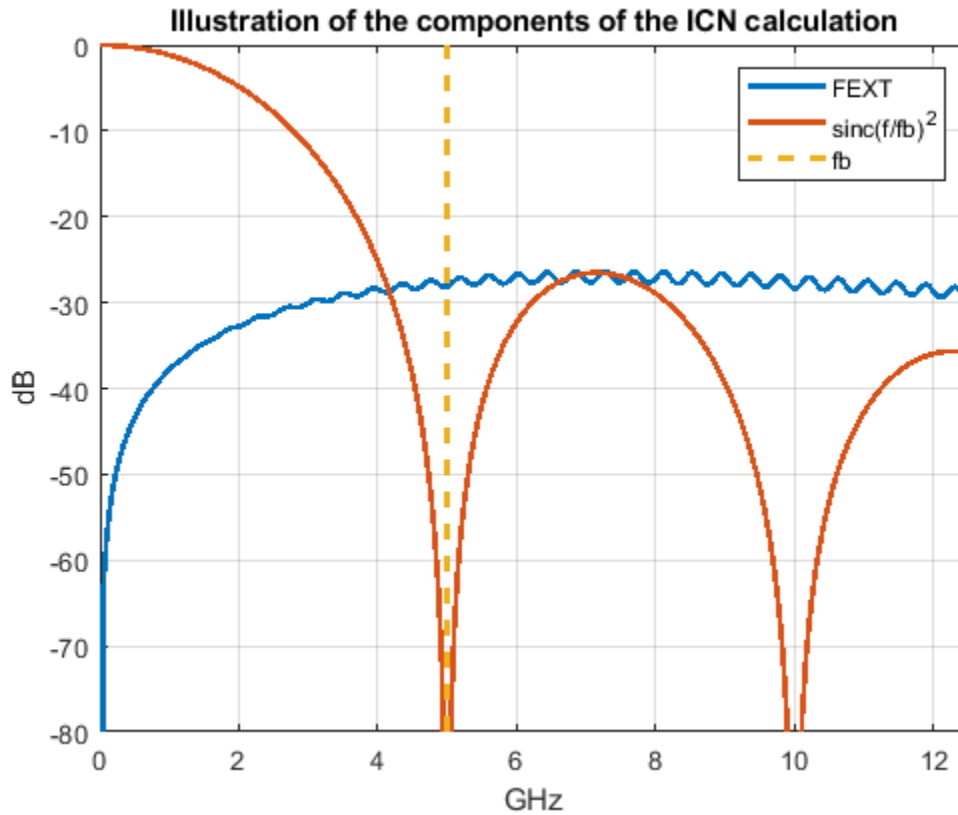


## More About

### 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.

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

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

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

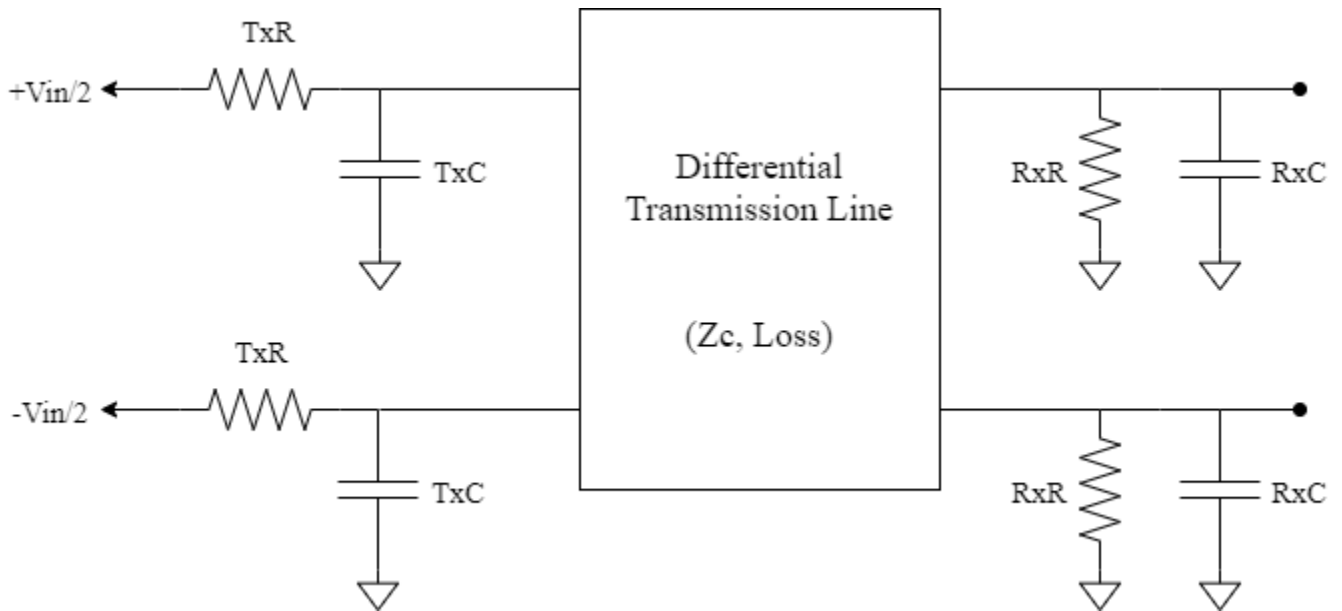
$$\sigma_x = \sqrt{\sigma_{fx}^2 + \sigma_{nx}^2}$$

## 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 an 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.

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

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

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

$$H_{\text{FEXT}}(f) = \mathcal{F}[I_{\text{FEXT}}(t)]$$

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

where ICN is the integrated crosstalk noise operator.

The sign of  $k_{\text{FEXT}}$  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  $S_{11}$ ).

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

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

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

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

## Version History

Introduced in R2019a

## 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](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"

## serdes.CTLE

**Package:** serdes

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?](#)

### 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](#).

**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 to maximize the performance metric as specified by the PerformanceCriteria property 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.

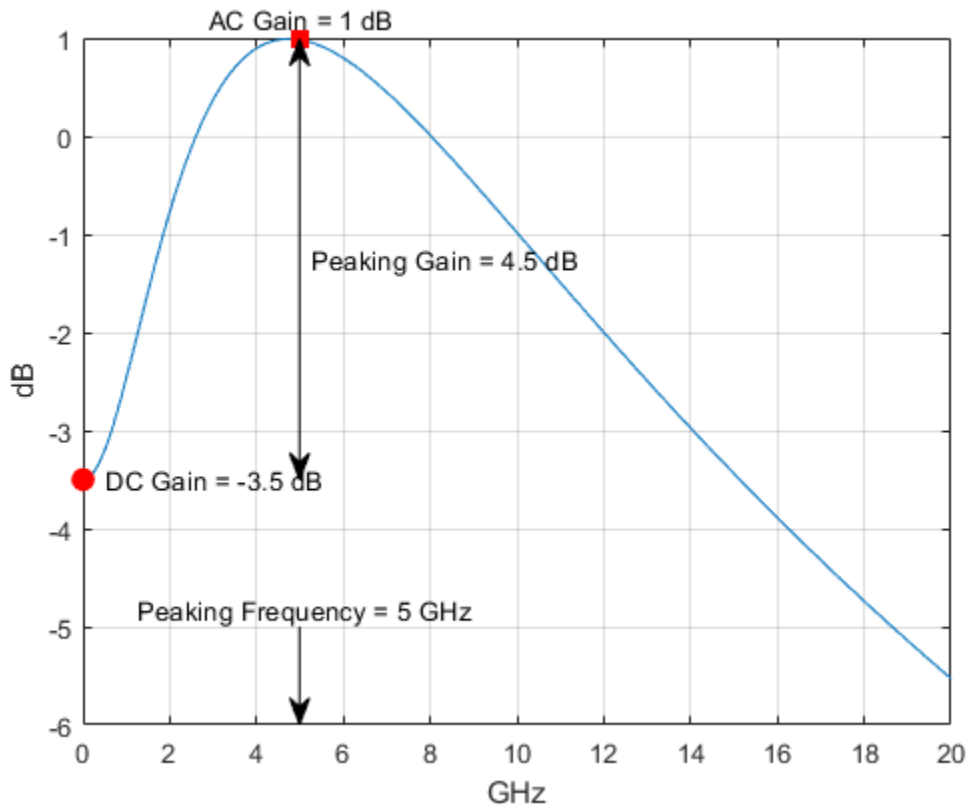
Data Types: double

**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. You can add more rows in the matrix to define additional configurations by using the ConfigSelect argument.

The serdes.CTLE System object only allows repeated poles or zeros when the FilterMethod is set to Cascaded. Complex poles or zeros must have conjugates. The number of poles must be greater than number of zeros for system stability. The System object ignores poles and zeros of 0 Hz and you can use them to zero-pad the matrix.

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

### PerformanceCriteria — Criterion used for CTLE optimization

SNR (default) | maxEyeHeight | maxMeanEyeHeight | maxCOM | eyeAreas | eyeWidth | centerEyeHeight | centerMeanEyeHeight | centerCOM

Criterion used for CTLE optimization when the `serdes.CTLE Mode` is set to `adapt`. The performance criteria is calculated using the `optPulseMetric` function.

Data Types: char

### FilterMethod — Method to calculate time domain filter coefficients

PartialFraction (default) | Cascaded

Method to calculate rational transfer function time domain filter coefficients:

- `PartialFraction` filter method uses a partial fraction expansion of the poles and zeros to directly calculate the filter coefficients. The `serdes.CTLE System` object does not allow repeated poles or zeros when the `FilterMethod` is set to `PartialFraction`.
- `Cascaded` filter method uses an approach that cascades together numerous shorter filters that collectively represent the specified behavior. This results in a more robust filter. The `serdes.CTLE System` object only allows repeated poles or zeros when the `FilterMethod` is set to `Cascaded`.

## 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.

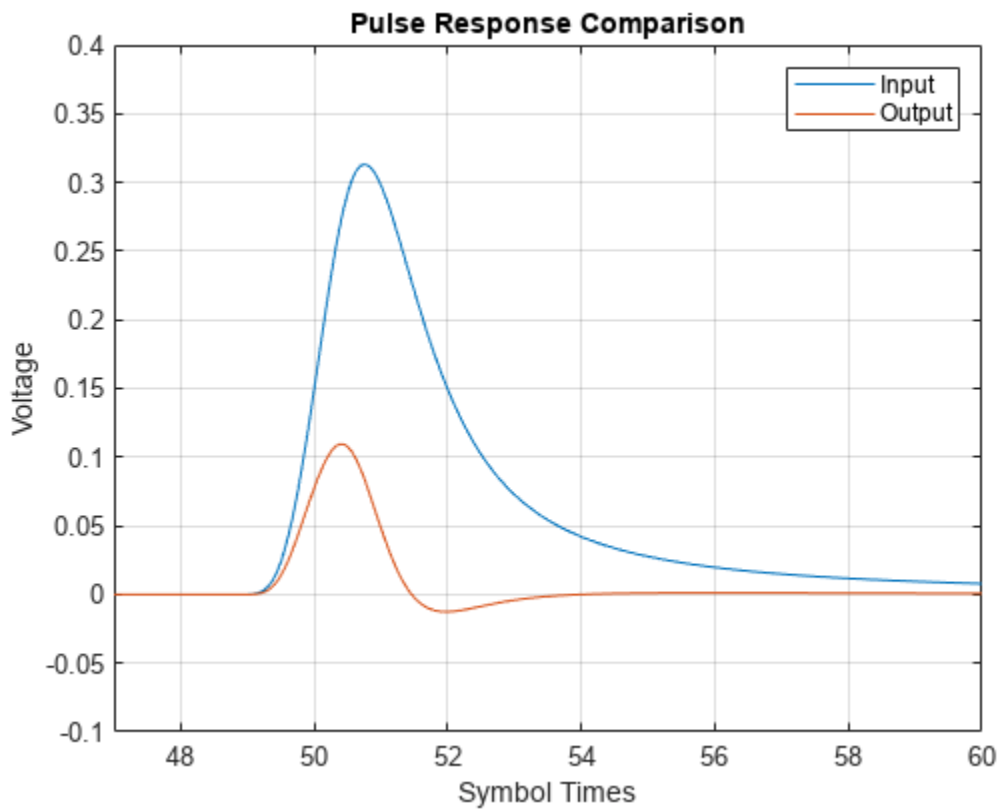
```
ord = 6;  
dataPattern = prbs(ord,2^ord-1)-0.5;  
  
pulseIn = impulse2pulse(impulseIn,SamplesPerSymbol,dt);  
waveIn = pulse2wave(pulseIn,dataPattern,SamplesPerSymbol);  
eyeIn = reshape(waveIn,SamplesPerSymbol,[]);  
  
pulseOut = impulse2pulse(impulseOut,SamplesPerSymbol,dt);  
waveOut = pulse2wave(pulseOut,dataPattern,SamplesPerSymbol);  
eyeOut = reshape(waveOut,SamplesPerSymbol,[]);
```

Create the time vectors.

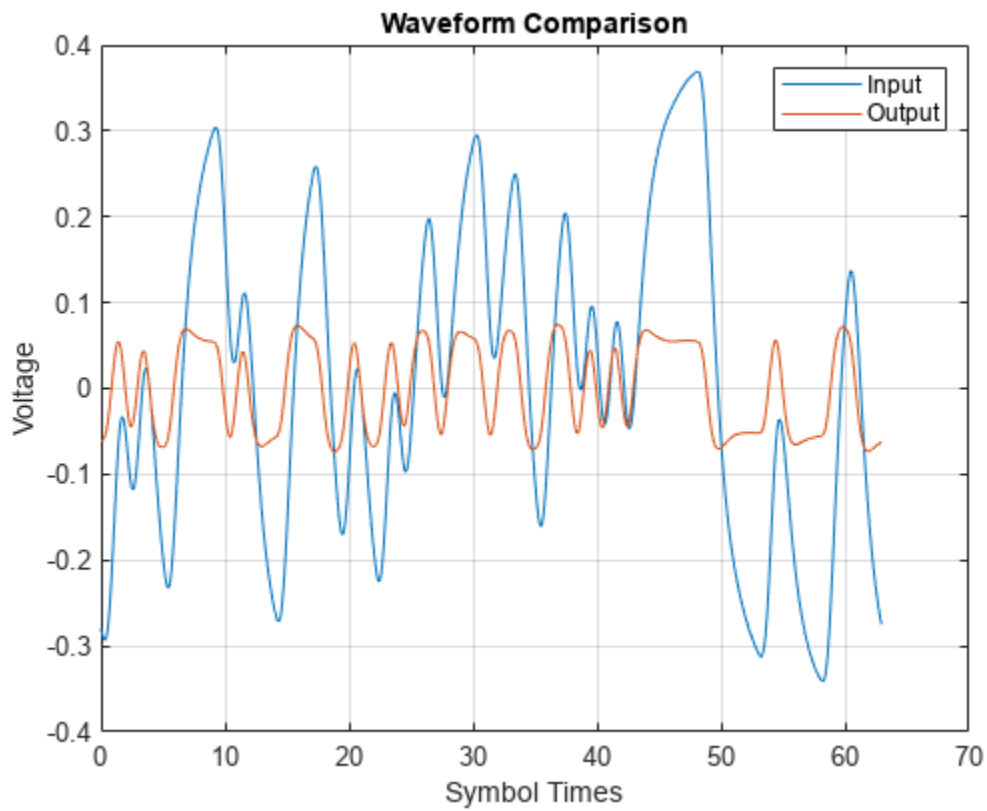
```
t = dt*(0:length(pulseOut)-1)/SymbolTime;  
teye = t(1:SamplesPerSymbol);  
t2 = dt*(0:length(waveOut)-1)/SymbolTime;
```

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

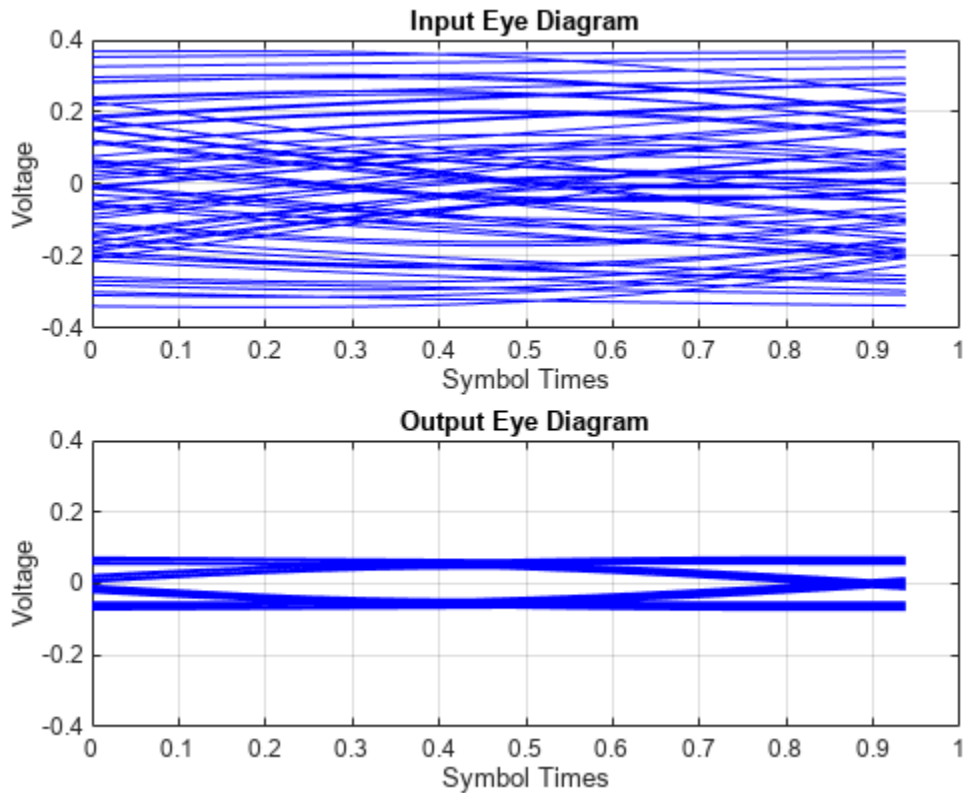
```
figure  
plot(t,pulseIn,t,pulseOut)  
legend('Input','Output')  
title('Pulse Response Comparison')  
xlabel('Symbol Times'),ylabel('Voltage')  
grid on  
axis([47 60 -0.1 0.4])
```



```
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);
```

Initialize PRBS generator.

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

Loop through one symbol at a time.

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

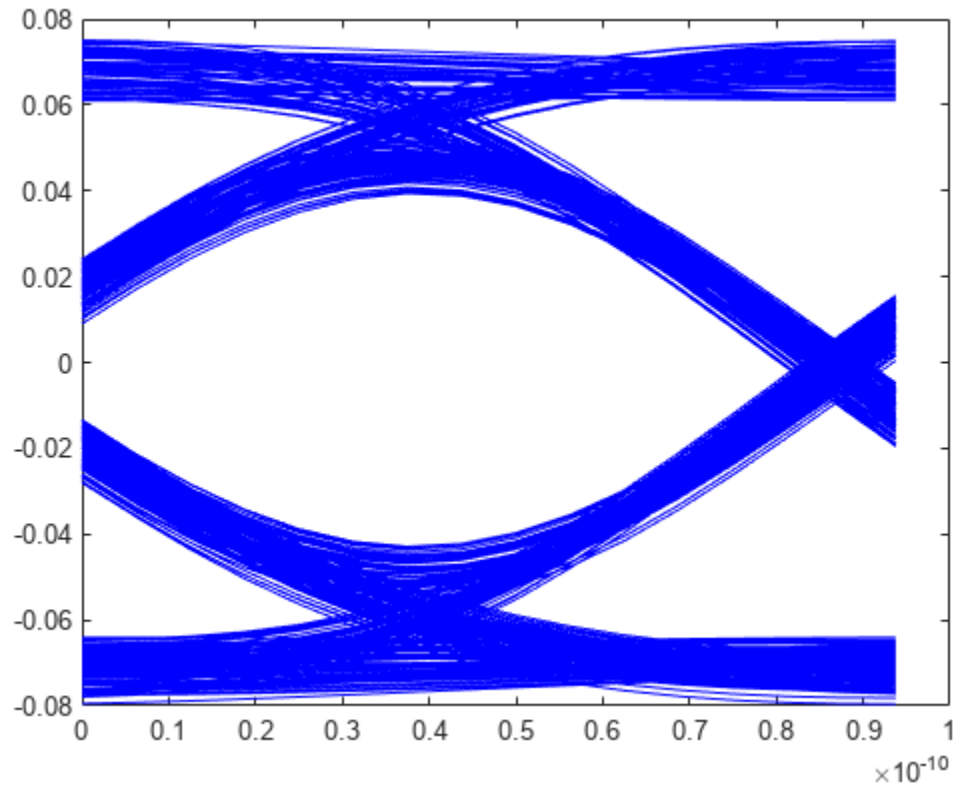
    % Convolve input waveform with channel
    y = channel(inSymbol);

    % Process one sample at a time through the CTLE
    for jj = 1:SamplesPerSymbol
        outWave((ii-1)*SamplesPerSymbol+jj) = CTLE(y(jj));
    end
end
```

After truncating the first 75 symbols to allow equalization, use the function reshape on the array of symbols to create the eye diagram.

```
foldedEye = reshape(outWave(75*SamplesPerSymbol+1:M*SamplesPerSymbol),SamplesPerSymbol,[]);
t = dt*(0:SamplesPerSymbol-1);
figure,plot(t,foldedEye,'b');
```





Copyright 2019 The MathWorks, Inc.

## Version History

Introduced in R2019a

## 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 | serdes.AGC | CTLE | DFECDR | serdes.DFECDR | SaturatingAmplifier | optPulseMetric

## serdes.DFE

**Package:** serdes

Minimize intersymbol interference (ISI) at clock sampling times

### Description

The `serdes.DFE` System object modifies a baseband signal to minimize the ISI at the clock sampling times. The decision feedback equalizer (DFE) samples data at each clock sample time and adjusts the amplitude of the waveform with a correction voltage.

To modify the input signal using a DFE:

- 1 Create the `serdes.DFE` 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?](#)

### Creation

#### Syntax

```
dfc = serdes.DFE  
dfc = serdes.DFE(Name=Value)
```

#### Description

`dfc = serdes.DFE` returns a DFE object that modifies an input waveform with the DFE to minimize the intersymbol interference (ISI).

`dfc = serdes.DFE(Name=Value)` sets properties using one or more name-value arguments. For example, `dfc = serdes.DFE(Mode=1)` returns a DFE object that applies the specified DFE 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](#).

#### Mode — DFE operating mode

2 (default) | 0 | 1

DFE operating mode, specified as 0, 1, or 2. The operating mode you select determines the DFE tap weights values that the object applies to the input waveform.

Mode Value	DFE Mode	DFE Operation
0	Off	The object bypasses DFE and the input waveform remains unchanged.
1	Fixed	The object applies the DFE tap weights specified in TapWeights to the input waveform.
2	Adapt	The object determines the optimum DFE tap weights and applies them to the input waveform.

Data Types: double

### TapWeights – Initial DFE tap weights

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

Initial DFE tap weights, specified as a row vector in volts. The length of the vector determines the number of taps. The vector elements signify the tap weights or tap strength. Set the tap weight to zero to initialize the tap.

Data Types: double

### MinimumTap – Minimum value of adapted tap weights

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

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

Data Types: double

### MaximumTap – Maximum value of adapted tap weights

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

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

Data Types: double

### EqualizationGain – Controls for update rate of tap weights

9.6e-5 (default) | positive real scalar

Controls for update rate of tap weights, specified as a unitless nonnegative real scalar. Increase the value of this property for the DFE adaptation to converge faster at the expense of more noise in the 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 the same value to all the DFE taps, or specify as a vector that has the same length as TapWeights.

**EqualizationStep** specifies the minimum change in DFE taps 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

**Taps2x — Multiply DFE tap weights by a factor of two**

false (default) | true

Multiply DFE tap weights by a factor of two, specified as true or false. Set this property to true

The output of the slicer in the `serdes.DFE System` object from the SerDes Toolbox is [-0.5 0.5]. But some industry applications require the slicer output to be [-1 1]. **Taps2x** allows you to quickly double the DFE tap weights to change the slicer reference.

**Advanced**

**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) | 3 | 4

Modulation scheme, specified as 2, 3 or 4.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
3	Three-level pulse amplitude modulation (PAM3)
4	Four-level pulse amplitude modulation (PAM4)

---

**Note** IBIS does not support a PAM3 modulation scheme, so you cannot export the System object to IBIS-AMI model if you set the modulation scheme to PAM3.

---

Data Types: double

**WaveType — Input waveform type**

'Sample' (default) | 'Impulse'

Input waveform type, specified as one of these:

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

Data Types: char

## Usage

### Syntax

```
y = dfe(x)
```

### Description

```
y = dfe(x)
```

### Input Arguments

#### **x** — Input baseband signal

scalar | vector

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

### Output Arguments

#### **y** — Modified channel output

scalar | vector

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

#### **TapWeights** — Estimated DFE tap weight values

vector

Estimated DFE tap weight values, returned 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

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

## Examples

## Equalize Baseband Signal Using DFE System Object

This example shows how to equalize a baseband signal using standalone DFE and CDR System objects. Use a symbol time of 80 ps and 8 samples per symbol. Set the channel loss to 8 dB, modulation scheme to NRZ, and number of DFE taps to 1.

```
N = 6000;           %Number of symbols to simulate
SymbolTime = 80e-12;
SamplesPerSymbol = 8;
ChannelLoss = 8;
Modulation = 2;    %Number of modulation levels
nDFEtaps = 1;     %Number of DFE taps
```

Define the system stimulus using a `serdes.Stimulus` System object.

```
dt = SymbolTime/SamplesPerSymbol; %sample interval
F = 1/SymbolTime/2; %Fundamental frequency
Stimulus = serdes.Stimulus(SampleInterval=dt,SymbolTime=SymbolTime,...
    Modulation=Modulation);
```

Define the channel model System object.

```
channelModel = serdes.ChannelLoss(Loss=ChannelLoss,dt=dt,...
    TargetFrequency= F);
```

Create the CDR System object.

```
CDR = serdes.CDR(SymbolTime=SymbolTime,...
    SampleInterval=dt,Step=1/64,...
    Modulation=Modulation);
```

Create the DFE System object.

```
DFE = serdes.DFE(SymbolTime=SymbolTime,...
    SampleInterval=dt,TapWeights=zeros(1,nDFEtaps),...
    EqualizationGain=2e-3,Modulation=Modulation);
```

Initialize the signals.

```
waveIn = zeros(N*SamplesPerSymbol,1);
waveOut = zeros(N*SamplesPerSymbol,1);
phase = zeros(size(waveIn));
tapsOut = zeros(N*SamplesPerSymbol,nDFEtaps);
```

Simulate the system with the CDR and DFE System objects.

```
for ii = 1:length(waveIn)
    %Create the input signal
    waveIn(ii) = channelModel(step(Stimulus));

    %Apply clock data recovery to the waveform
    [phase(ii),clkAMI,details]=CDR(waveIn(ii));

    %Pass the waveform, clock, and recovered data to the DFE
    [waveOut(ii),tapsOut(ii,:)] = DFE(waveIn(ii),clkAMI.clockValidOnRising,details.symbolRecoverer);
end
```

Visualize the waveform, eye diagram, CDR phase, and DFE tap weight adaptation.

```

t = dt*(0:length(waveIn)-1);
w = waveOut(end-50*SamplesPerSymbol+1:end);
M = reshape(w,SamplesPerSymbol,[]);

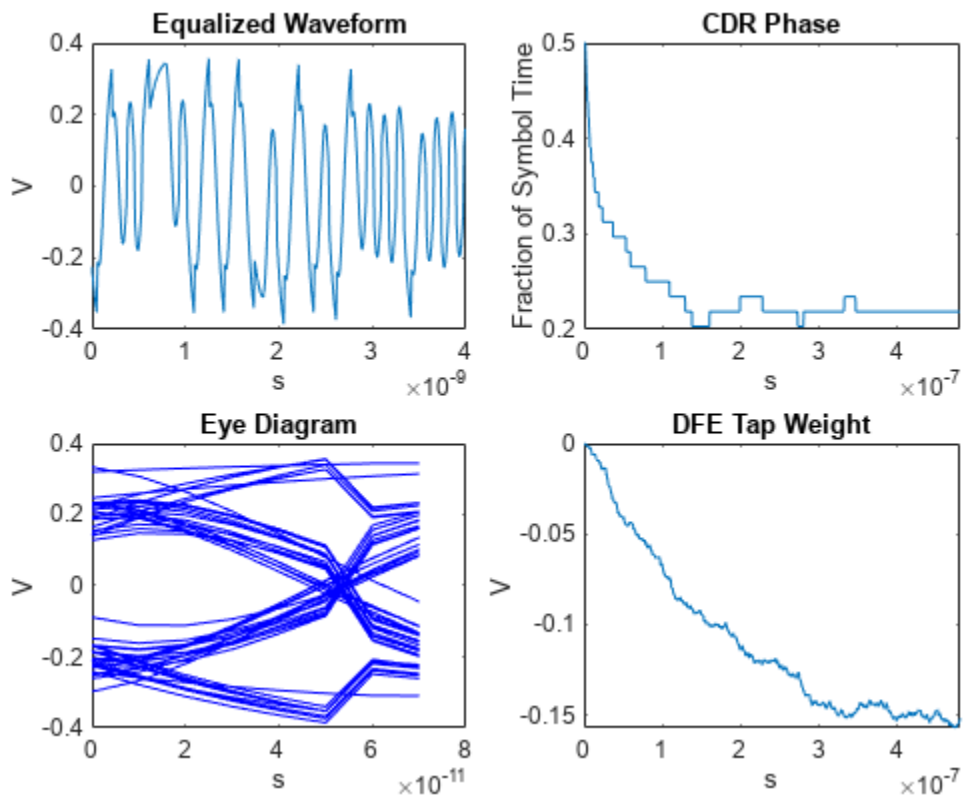
```

figure

```

subplot(221)
    plot(t(1:length(w)),w)
        xlabel('s')
        ylabel('V')
        title('Equalized Waveform')
subplot(222)
    plot(t,phase)
        xlabel('s')
        ylabel('Fraction of Symbol Time')
        title('CDR Phase')
subplot(223)
    plot(t(1:SamplesPerSymbol),M,'b')
        xlabel('s')
        ylabel('V')
        title('Eye Diagram')
subplot(224)
    plot(t,tapsOut)
        xlabel('s')
        ylabel('V')
        title('DFE Tap Weight')

```



### **More About**

#### **Adapt Operating Mode**

The Init subsystem calls to the `serdes.DFE`. The `serdes.DFE` finds the optimum DFE tap values for the best eye height opening for statistical analysis. During time domain simulation, DFE 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”

### **Version History**

**Introduced in R2023a**

#### **See Also**

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



## serdes.DFECDR

**Package:** serdes

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?](#)

## 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.

### DFE Properties

#### Mode — DFE operating mode

2 (default) | 0 | 1

DFE operating mode, specified as 0, 1, or 2. The operating mode you select determines the DFE tap weights values that the object applies to the input waveform.

Mode Value	DFE Mode	DFE Operation
0	Off	The object bypasses DFE and the input waveform remains unchanged.
1	Fixed	The object applies the DFE tap weights specified in <code>TapWeights</code> to the input waveform.
2	Adapt	The object determines the optimum DFE tap weights and applies them to the input waveform.

Data Types: double

#### TapWeights — Initial DFE tap weights

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

Initial DFE tap weights, specified as a row vector in volts. The length of the vector determines the number of taps. The vector elements signify the tap weights or tap strength. Set the tap weight to zero to initialize the tap.

Data Types: double

#### MinimumTap — Minimum value of adapted tap weights

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

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

Data Types: double

#### MaximumTap — Maximum value of adapted tap weights

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

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

Data Types: `double`

### **EqualizationGain — Controls for update rate of tap weights**

9.6e-5 (default) | positive real scalar

Controls for update rate of tap weights, specified as a unitless nonnegative real scalar. Increase the value of this property for the DFE adaptation to converge faster at the expense of more noise in the 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 the same value to all the DFE taps, or specify as a vector that has the same length as `TapWeights`.

`EqualizationStep` specifies the minimum change in DFE taps 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`

### **Taps2x — Multiply DFE tap weights by a factor of two**

false (default) | true

Multiply DFE tap weights by a factor of two, specified as true or false. Set this property to true to multiply the DFE tap weights by a factor of two.

The output of the slicer in the `serdes.DFECDR System` object from the SerDes Toolbox is [-0.5 0.5]. But some industry applications require the slicer output to be [-1 1]. `Taps2x` allows you to quickly double the DFE tap weights to change the slicer reference.

## **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 ( $\pm$ `Sensitivity`), there is a 50% probability of bit error.

Data Types: double

**PhaseDetector — Clock phase detector option**

BangBang (default) | BaudRateTypeA

Clock phase detector option used in the clock data recovery. You can choose between bang-bang (Alexander) or baud-rate type-A (Mueller-Muller).

**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) | 3 | 4

Modulation scheme, specified as 2, 3 or 4.

Modulation Value	Modulation Scheme
2	Non-return to zero (NRZ)
3	Three-level pulse amplitude modulation (PAM3)
4	Four-level pulse amplitude modulation (PAM4)

---

**Note** IBIS does not support a PAM3 modulation scheme, so you cannot export the System object to IBIS-AMI model if you set the modulation scheme to PAM3.

---

Data Types: double

### WaveType — Input waveform type

'Sample' (default) | 'Impulse'

Input waveform type, specified as one of these:

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

Data Types: char

## Usage

### Syntax

$y = \text{dfecdr}(x)$

### Description

$y = \text{dfecdr}(x)$

### Input Arguments

#### **x** — Input baseband signal

scalar | vector

Input baseband signal, specified as a scalar or vector. If you set the WaveType property to 'Sample', then the input signal is a sample-by-sample signal specified as a scalar. If you set the WaveType property 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.

### **TapWeights — Estimated DFE tap weight values**

vector

Estimated DFE tap weight values, returned as a vector.

### **Phase — Relative recovered clock phase**

units of SymbolTime in the range [0,1]

Relative recovered clock phase, returned as a units of SymbolTime in the range [0,1].

### **c1kAMI — AMI clock bus**

structure

AMI clock bus, returned as a structure consisting of:

- ClockTime — Time taken to sample the data signal.
- clockValidOnRising — Valid clock time value on the rising edge of a signal

### **Interior — Bus containing additional interior CDR signals**

structure

Bus containing additional interior CDR signals, returned as a structure consisting of:

- clockPhase — Relative clock phase in units of SymbolTime in the range of [0,1].
- symbolRecovered — Symbol recovered from data signal at ClockTime.
- voltageSample — Voltage observed from the data signal at ClockTime.
- PAM4Threshold — The estimated upper eye at PAM4 threshold.
- CDRedgeVoltage — The voltage observed from the data signal at ClockTime — SymbolTime/2.
- CDRCounter — The bang-bang CDR internal counter used to trigger samples.
- CDREarlyLateCounter — The bang-bang CDR accumulated (or filtered) signal used to trigger updated to the CDR phase.
- PAM4SymbolMiddleVoltage — Estimated PAM4 symbol voltage of the inner eye outer envelope to estimate PAM4 threshold.
- PAM4OuterVoltage — Estimated PAM4 outer envelope voltage to estimate PAM4 threshold.
- EyeHeightAbsAve — Estimates eye height.

## **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,[]);
```

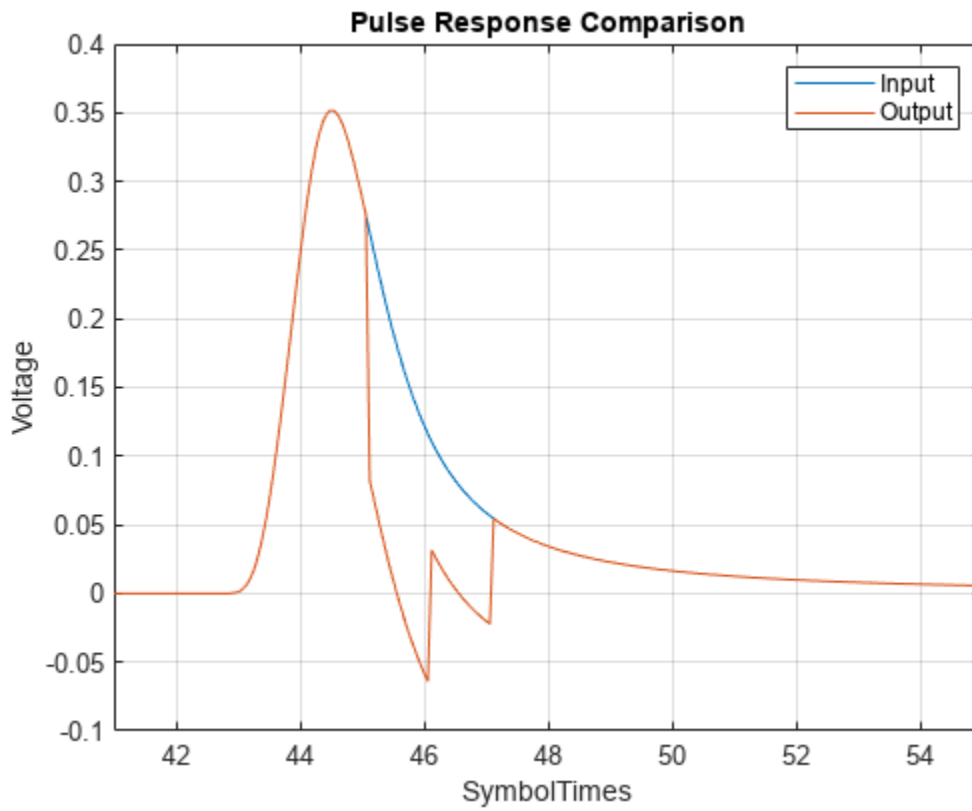
Create the time vectors.

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

Plot the resulting waveforms.

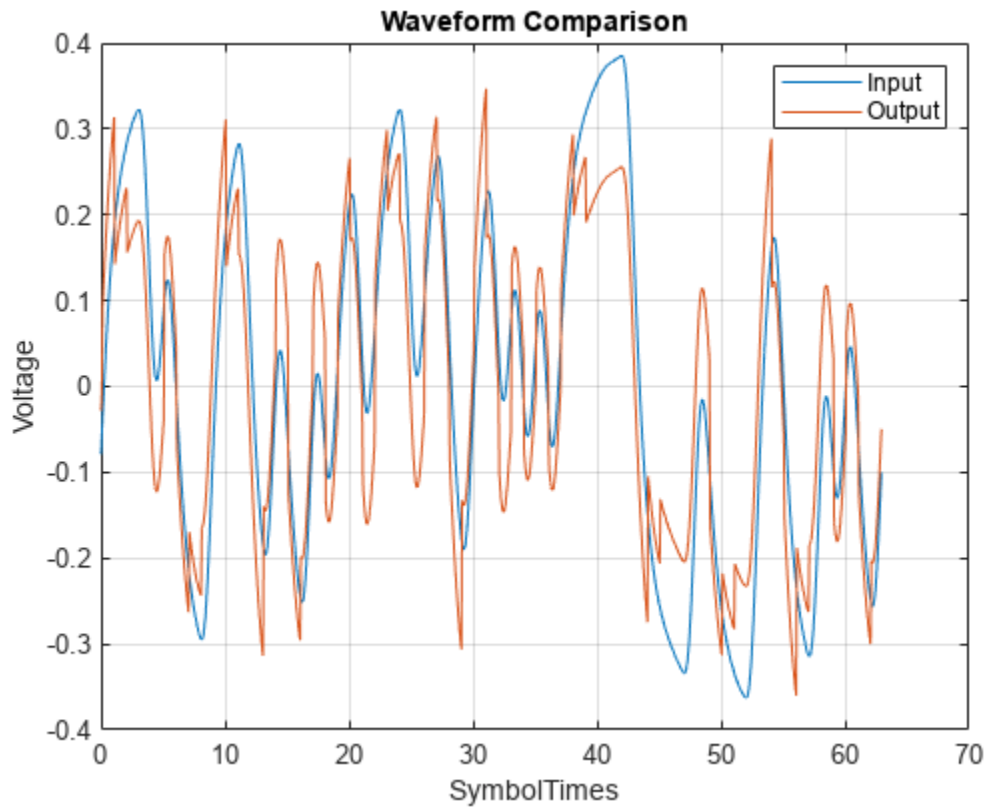
```
figure
plot(t,pulseIn,t,pulseOut)
```

```
legend('Input','Output')
title('Pulse Response Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
axis([41 55 -0.1 0.4])
```

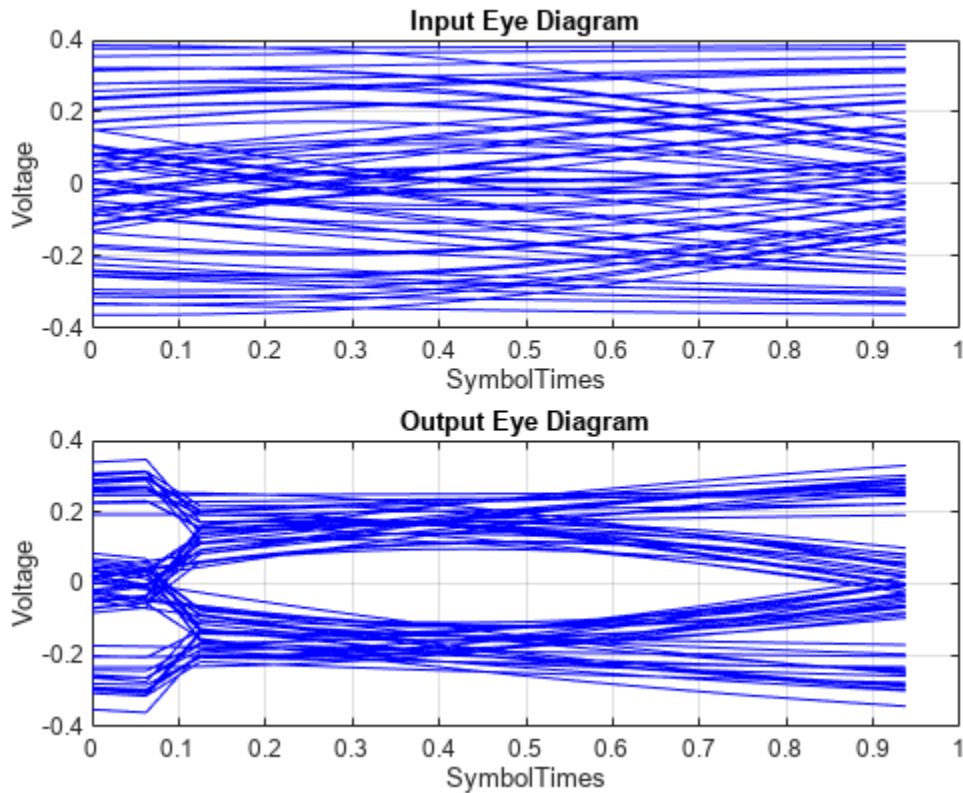


```
figure
plot(t2,waveIn,t2,waveOut)
legend('Input','Output')
title('Waveform Comparison')
xlabel('SymbolTimes'),ylabel('Voltage')
grid on
```





```
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);
```

Initialize the PRBS generator.

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

Generate the sample-by-sample eye diagram.

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

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

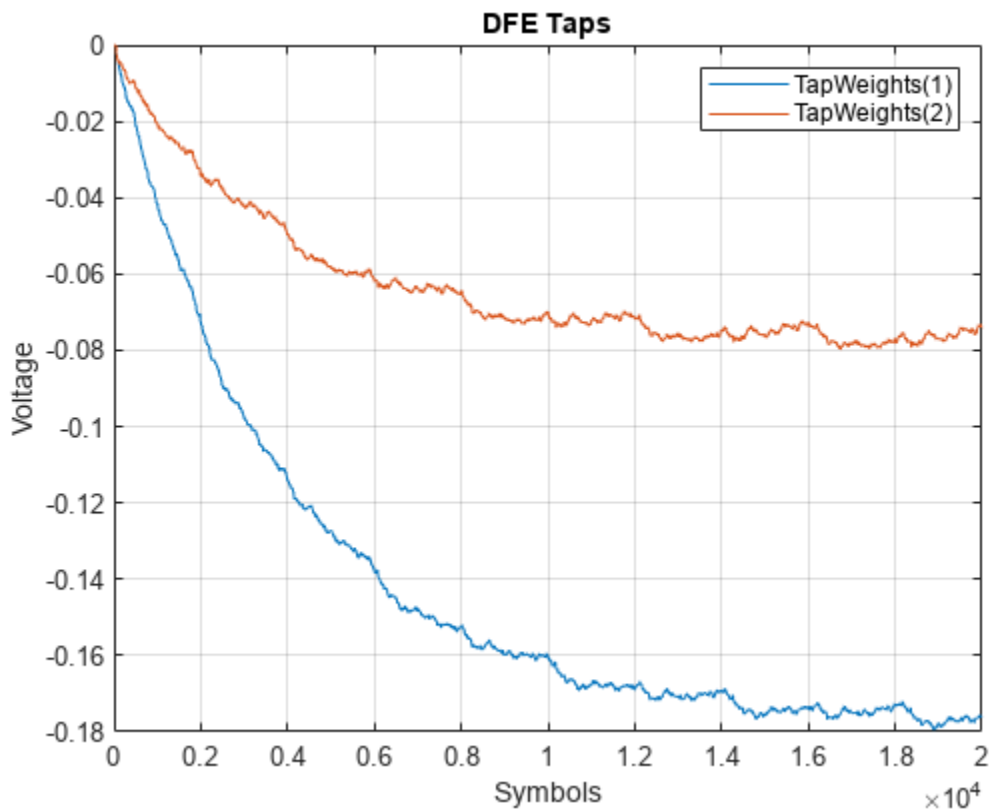
    %Convolve input waveform with channel
    y = channel(inSymbol);

    %Process one sample at a time through the DFE
    for jj = 1:SamplesPerSymbol
        [outWave((ii-1)*SamplesPerSymbol+jj),TapWeights] = DFE2(y(jj));
    end

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

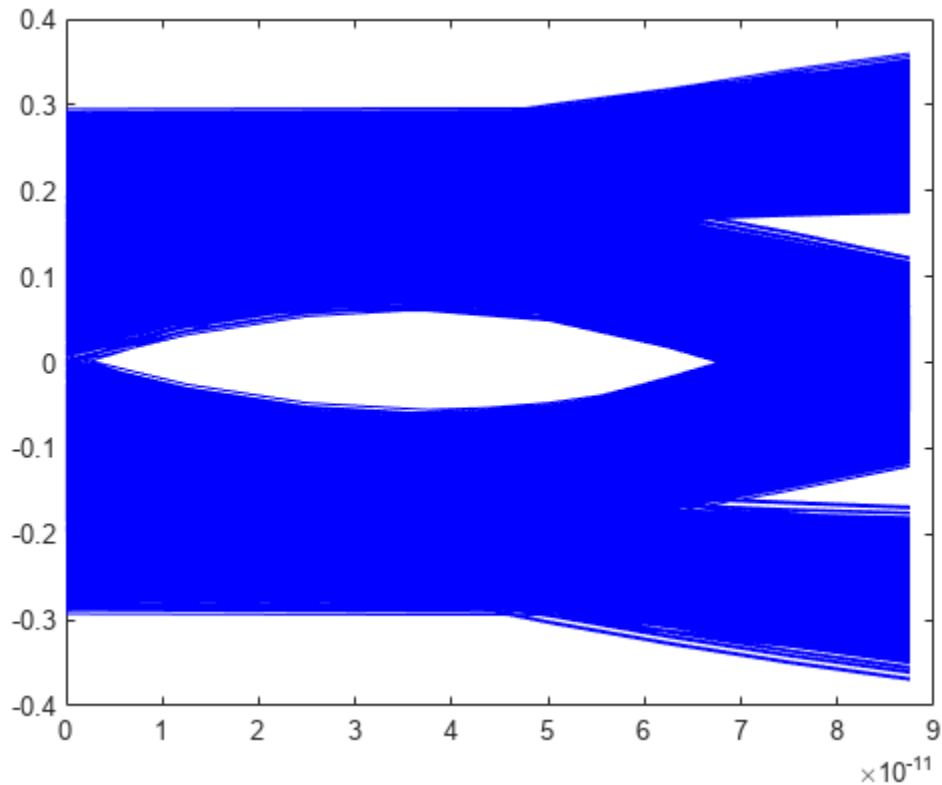
Plot the DFE adaptation history.

```
figure
plot(dfeTapWeightHistory)
grid on
legend('TapWeights(1)','TapWeights(2)')
xlabel('Symbols')
ylabel('Voltage')
title('DFE Taps')
```



You can observe from the plot that the DFE adaptation is approximately complete after the first 10000 symbols, so these can be truncated from the array. Then plot the eye diagram by applying the reshape function to the array of symbols.

```
foldedEye = reshape(outWave(10000*SamplesPerSymbol+1:M*SamplesPerSymbol),SamplesPerSymbol,[]);  
t = dt*(0:SamplesPerSymbol-1);  
figure,plot(t,foldedEye,'b');
```



## More About

### Adapt Operating Mode

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, DFE 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”

## Version History

Introduced in R2019a

## 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**

DFECCR | CTLE | CDR | serdes.CTLE | serdes.CDR

**Topics**

“Clock and Data Recovery in SerDes System”

# serdes.FFE

**Package:** serdes

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?](#)

## 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](#).

**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,[]);
```

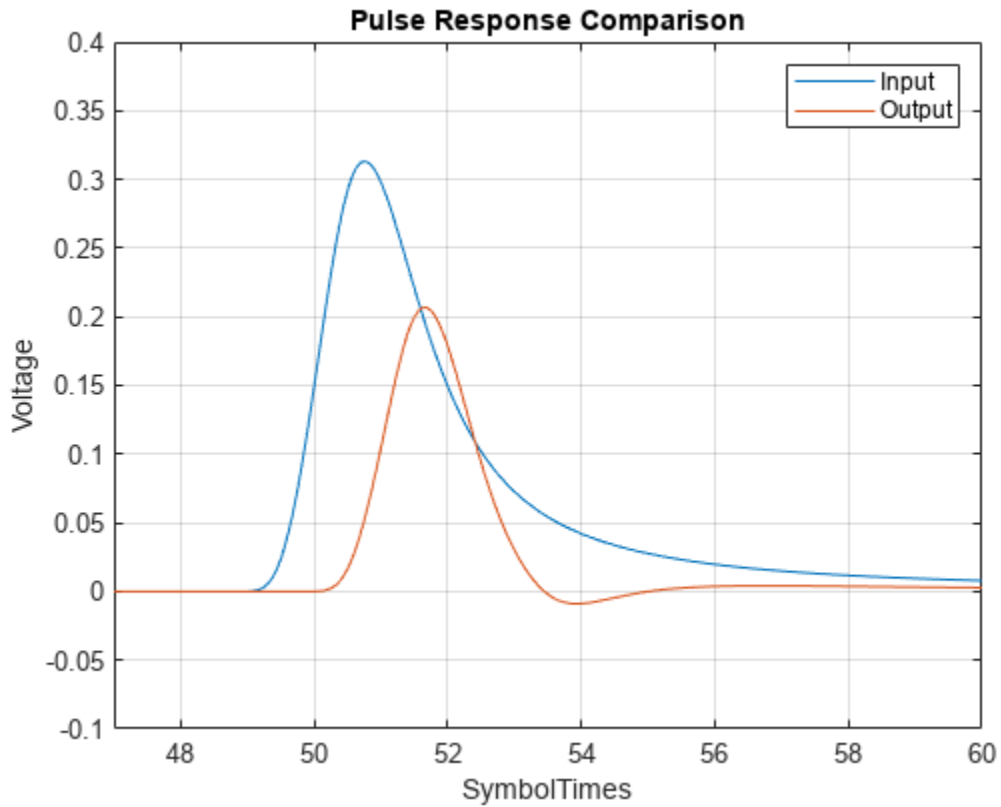
Create the time vectors.

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

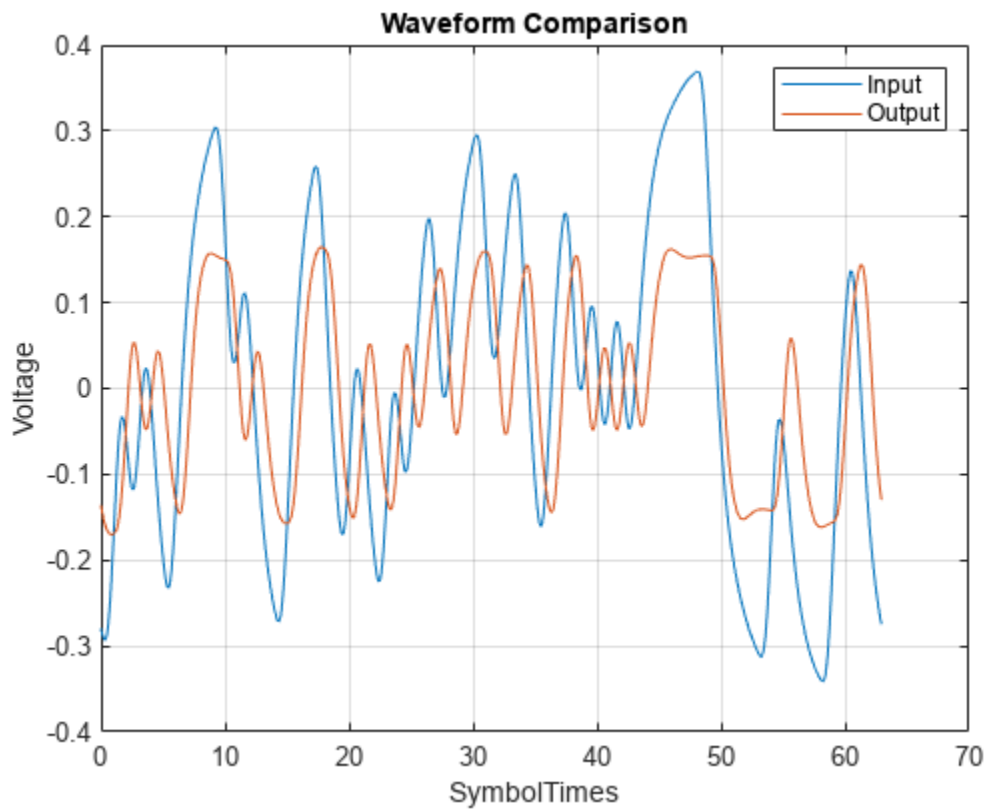
Plot the pulse response comparison, waveform comparison, and input and output eye diagrams.

```
figure  
plot(t,pulseIn,t,pulseOut)  
legend('Input','Output')  
title('Pulse Response Comparison')
```

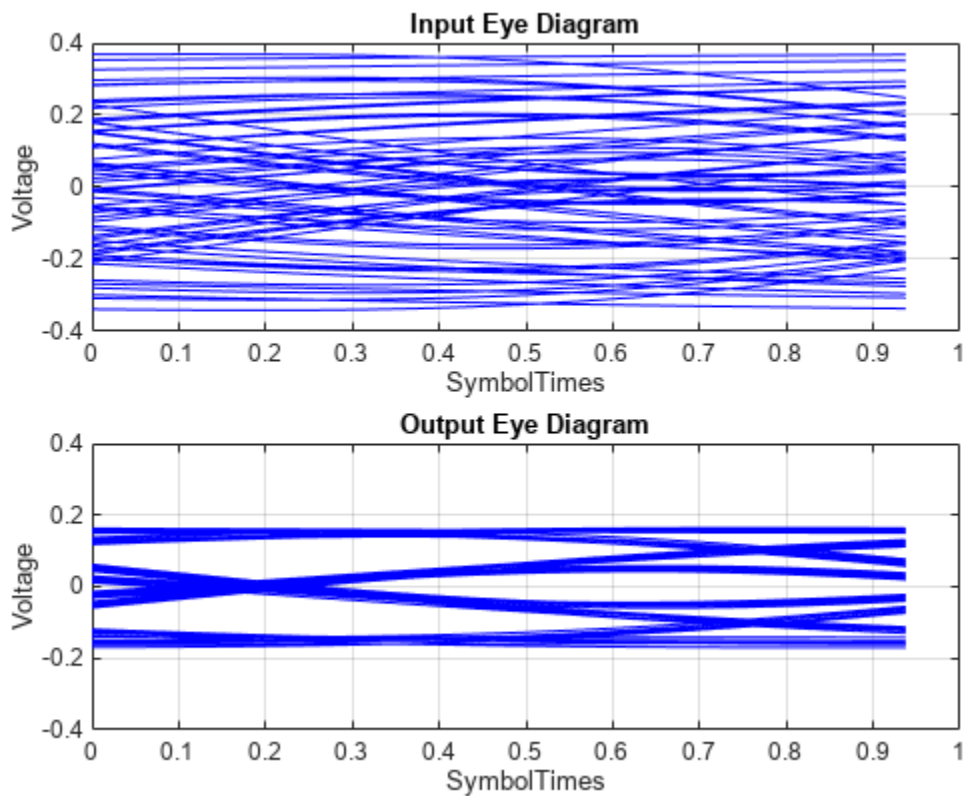
```
xlabel('SymbolTimes'),ylabel('Voltage')  
grid on  
axis([47 60 -0.1 0.4])
```



```
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);
```

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

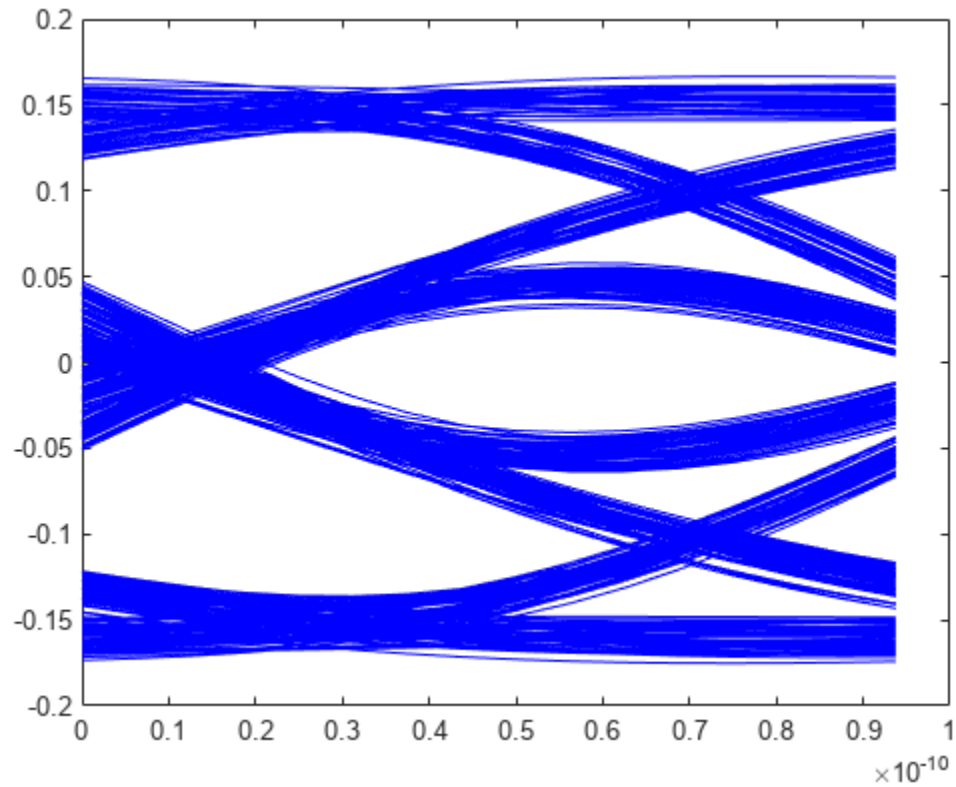
```
prbsOrder = 12;  
M = 500; %number of symbols to simulate  
[dataBit,prbsSeed]=prbs(prbsOrder,1);
```

Loop through one symbol at a time.

```
inSymbol = zeros(SamplesPerSymbol,1);  
outWave = zeros(SamplesPerSymbol*M,1);  
for ii = 1:M  
    %Get new symbol  
    [dataBit,prbsSeed]=prbs(prbsOrder,1,prbsSeed);  
    inSymbol(1:SamplesPerSymbol) = dataBit-0.5;  
  
    %convolve input waveform with channel  
    y = channel(inSymbol);  
  
    %process one sample at a time through the FFE  
    for jj = 1:SamplesPerSymbol  
        outWave((ii-1)*SamplesPerSymbol+jj) = FFE(y(jj));  
    end  
end
```

After truncating the first 75 symbols to allow equalization, use the function reshape on the array of symbols to create the eye diagram.

```
foldedEye = reshape(outWave(75*SamplesPerSymbol+1:M*SamplesPerSymbol),SamplesPerSymbol,[]);  
t = dt*(0:SamplesPerSymbol-1);  
figure,plot(t,foldedEye,'b');
```



## Version History

Introduced in R2019a

## 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

FFE | serdes.CTLE | CTLE

# serdes.PassThrough

**Package:** serdes

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?](#)

## 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](#).

### 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.

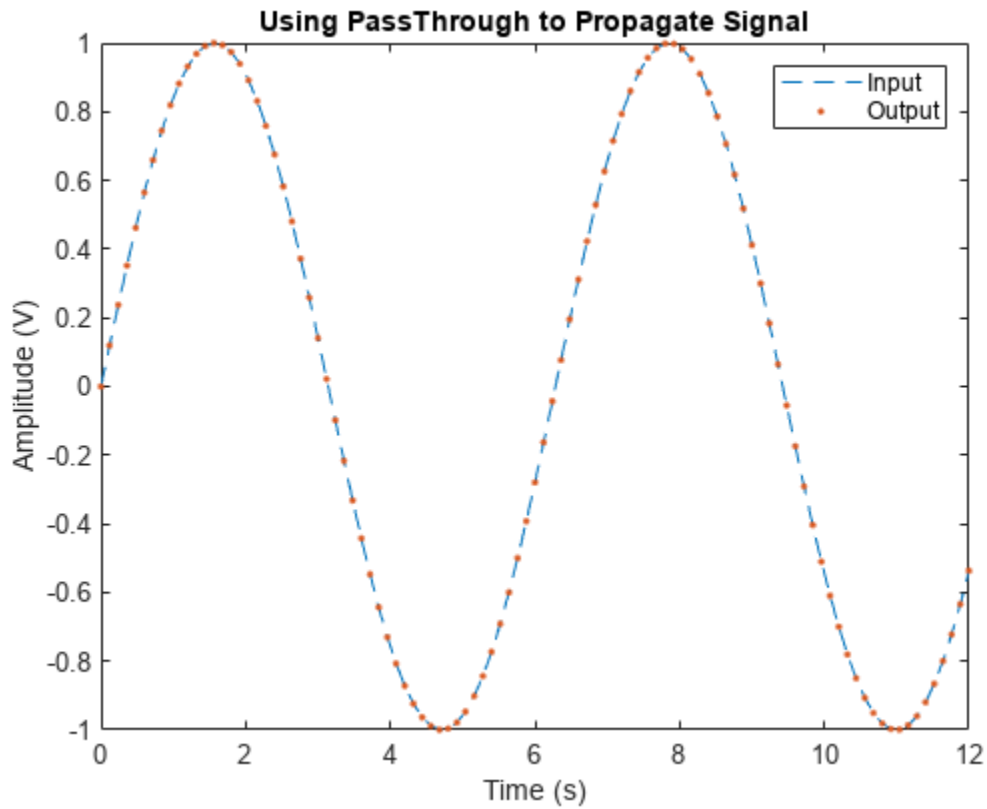
```
PT = serdes.PassThrough;
```

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
```

## Version History

Introduced in R2019a

## 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

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

# serdes.SaturatingAmplifier

**Package:** serdes

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?](#)

## 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](#).

**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** $N \times 2$  matrix

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

Data Types: double

**Usage****Syntax**

$$y = \text{SatAmp}(x)$$

### Description

$y = \text{SatAmp}(x)$

### 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.

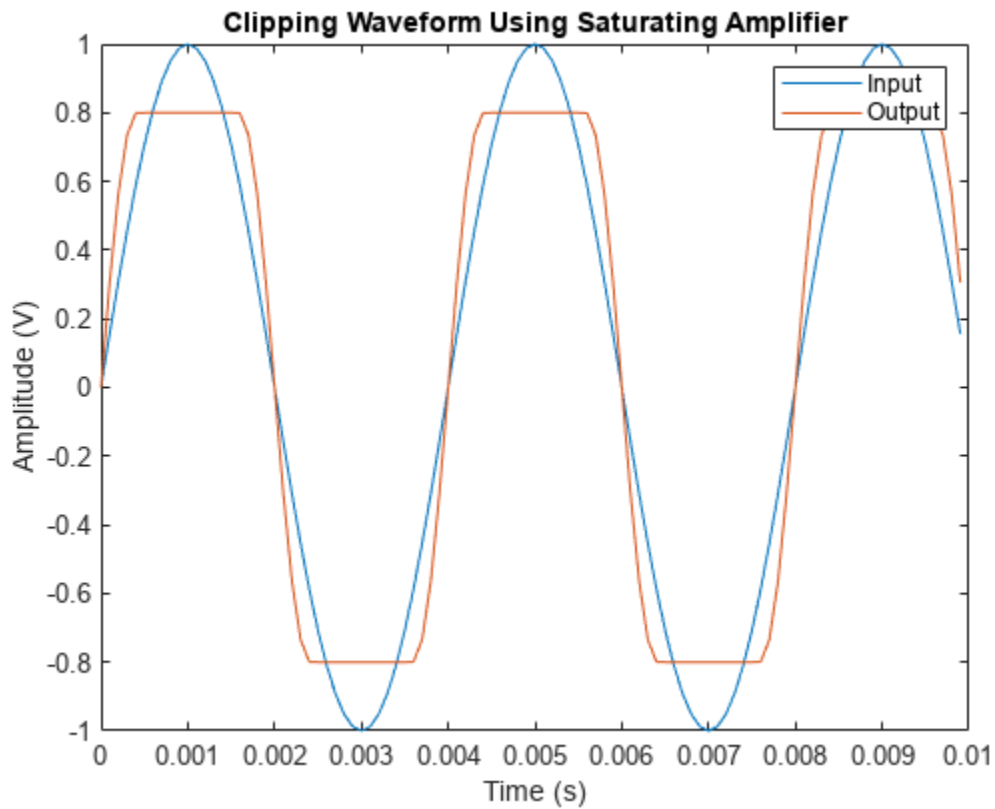
```
linearGain = 2;  
limit = 0.8;  
SaturatingAmplifier = serdes.SaturatingAmplifier('Mode',1,...  
        'Limit',limit,'LinearGain',linearGain);  
y = SaturatingAmplifier(x);
```

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.

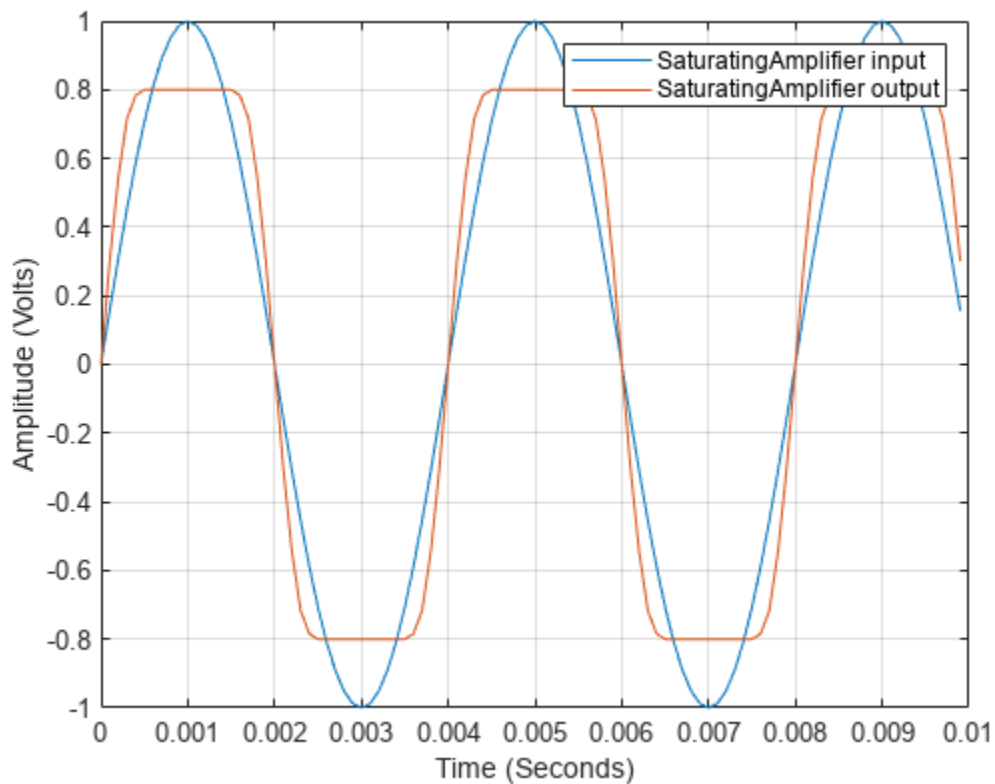
```
SatAmp = serdes.SaturatingAmplifier('Mode',1,'Specification','VinVout','VinVout',M);
```

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)');
```



## Version History

Introduced in R2019a

## 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 | serdes.AGC | SaturatingAmplifier | serdes.VGA | VGA

## serdes.Stimulus

**Package:** serdes

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

### Description

The `serdes.Stimulus` System object defines how to create the waveform and the number of symbols to simulate in a SerDes Toolbox model. You can directly specify a PAMn symbol pattern or define a binary to PAMn mapping scheme.

To create a multi-level pseudorandom binary sequence or custom data pattern:

- 1 Create the `serdes.Stimulus` 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?](#)

### Creation

#### Syntax

```
stimulus = serdes.Stimulus  
stimulus = serdes.Stimulus(Name=Value)
```

#### Description

`stimulus = serdes.Stimulus` returns a `Stimulus` object that outputs multi-level pseudorandom binary sequence or custom data pattern.

`stimulus = serdes.Stimulus(Name=Value)` sets properties using one or more name-value arguments. Unspecified properties have default values.

Example: `stimulus = serdes.Stimulus(Modulation=4,Delay=10e-12)` returns a `Stimulus` object that uses a four-level pulse amplitude modulation (PAM4) modulation scheme and has a delay of `10e-12` seconds.

### 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](#).

## Main

### Specification — Waveform creation method

'Random Symbols' (default) | 'Symbol Pattern' | 'Serial PRBS' | 'Binary Pattern' | 'Symbol Voltage Pattern' | 'Sampled Voltage' | 'Parallel PRBS' | 'PRBS' | 'PAMn' | 'Data Pattern'

Waveform creation method, specified as one of these values.

- 'Random Symbols' — Randomly generate symbols directly from a uniform distribution.
- 'Symbol Pattern' — Directly define the symbol pattern as an integer from 0 to (Number of modulation levels - 1).
- 'Serial PRBS' — Define a binary sequence and map to a higher order modulation symbol. The mapping sequence depends on the modulation scheme defined in the Configuration block.
- 'Binary Pattern' — Directly define the binary pattern and map to a higher order modulation symbol. The mapping sequence depends on the modulation scheme defined in the Configuration block.
- 'Symbol Voltage Pattern' — Directly define the voltages of a symbol spaced pattern.
- 'Sampled Voltage' — Directly define a sample interval spaced waveform.
- 'Parallel PRBS' — Utilize multiple PRBS streams to form PAM4, PAM8, PAM16, and PAM32 symbols. You can define up to 5 PRBS streams for PAM32. NRZ modulation scheme uses a single PRBS stream. Other modulation schemes use the random symbols waveform creation method.
- 'PRBS' — Generate a pseudorandom binary sequence (PRBS) based on the Seed. Same as 'Parallel PRBS'.
- 'PAMn' — Generate a pulse amplitude modulated (PAM) signal based on the PAMn Seed. Same as 'Random Symbols'.
- 'Data Pattern' — Specify a data pattern to modulate. Same as 'Sampled Voltage'.

---

**Note** 'PRBS', 'PAMn', and 'Data Pattern' options are being deprecated. Use 'Parallel PRBS',

'Random Symbols', and 'Sampled Voltage' instead, respectively.

---

**Tunable:** No

Data Types: char

### Modulation — Number of logic levels in signaling scheme

2 (default) | integer in the range of [2, 32]

Modulation value that specifies the modulation scheme, the number of logic levels in the encoded signal, specified as an integer in the range of [2, 32].

---

**Note** According to IBIS BIRD (Buffer Issue Resolution Document) 213, IBIS-AMI models support any level of signaling from PAM2 (NRZ) to upwards, collectively known as PAMn. If your EDA tool supports it, you can export IBIS-AMI models supporting modulation schemes NRZ, PAM3, PAM4, PAM8, PAM16, or PAM32.

---

---

**Note** This property is not enabled if `Specification` is set to 'Symbol Voltage Pattern', 'Sampled Voltage', or 'Data Pattern'.

---

**Tunable:** No

Data Types: double

**Delay – Delay offset for stimulus waveform**

0 (default) | nonnegative real scalar

Delay the stimulus waveform by modulo symbol time to slightly shift the eye diagram or recovered clock phase.

**Tunable:** No

Data Types: double

**Order – PRBS order**

7 (default) | numeric scalar | numeric vector

PRBS order, specified as a numeric scalar or a numeric vector of positive integers less than 100. Specify `Order` as a numeric vector when the value of `Modulation` is greater than 2. For higher order modulation, the first `Order` entry defines a PRBS stream for the least significant bit (LSB) and the last `Order` entry defines a PRBS stream for the most significant bit (MSB).

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS'.

Data Types: double

**Seed – PRBS seed**

{[0 1 0 1 0 1 0]} (default) | cell array of binary vectors

PRBS seed, specified as a cell array of cell array of binary vectors. The length of each binary vector is the same length as the corresponding `Order`.

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS'.

Data Types: cell

**MapToVoltage – PRBS mapping from binary stream(s) to voltages**

[-0.5 0.5] (default) | numeric real vector

Map the binary data stream(s) to symbol voltages, specified as a vector. The length of `MapToVoltage` needs to be the same as the number of logic levels defined by the `Modulation` property. The LSB PRBS stream, `p1`, and the MSB PRBS stream, `pn`, index `MapToVoltage` as follows:  $\text{Index} = p1 + 2^{1*p2} + 2^{2*p3} + \dots + 2^{(n-1)*pn}$ .

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Random Symbols', 'Parallel PRBS', or 'PRBS'.

Data Types: double

**DataPattern — Custom data pattern of symbol time spaced voltages**

`[-0.5000 0.5000 -0.5000 0.5000 0.5000 0.5000 -0.5000]` (default) | numeric vector

Data pattern of symbol voltages spaced at symbol time interval, specified as a numeric vector. This property is continually cycled through.

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Symbol Voltage Pattern'.

Data Types: double

**SymbolPattern — Custom pattern of symbol voltages spaced at symbol time interval**

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

Custom pattern of symbol voltages spaced at symbol time interval, specified as a numeric vector of zero based indices in the range of `[0, Number of modulation levels -1]`.

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Symbol Pattern'.

Data Types: double

**BinaryPattern — Custom pattern of 1s and 0s spaced at symbol interval**

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

Custom pattern of 1s and 0s spaced at the symbol interval, specified as a numeric or logical vector.

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Binary Pattern'.

Data Types: double

**SampledVoltage — Custom voltage pattern of sample interval spaced voltages**

`[0.5000 0.5000 0.5000 0.5000 -0.5000 -0.5000 -0.5000 -0.5000]` (default) | numeric vector | logical vector

Custom voltage pattern of sample interval spaced voltages, specified as a numeric vector.

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Sampled Voltage'.

Data Types: double

**PAMnSeed — Seed for the PAMn pattern generator**

1.4317e+09 (default) | nonnegative numeric scalar

Seed for the PAMn pattern generator, specified as a nonnegative number between 2 and  $2^{31} - 1$ .

**Tunable:** No

**Dependencies**

To enable this property, set `Specification` to 'Random Symbols'.

Data Types: double

**Jitter****Dj — Deterministic jitter**

0 (default) | nonnegative numeric scalar

Deterministic jitter, or uncorrelated bounded uniform jitter, specified as a nonnegative number. The value of `Dj` indicates half of the peak to peak variation of the deterministic jitter.

**Tunable:** No

Data Types: double

**DjUnit — Deterministic jitter unit**

'UI' (default) | 'Seconds'

Deterministic jitter unit, specified as 'Seconds' for seconds or 'UI' for unit interval.

**Tunable:** No

Data Types: char

**Rj — Random jitter**

0 (default) | nonnegative numeric scalar

Random jitter, specified as a nonnegative number. Random jitter is defined as the standard deviation of the white Gaussian Phase noise process.

**Tunable:** No

Data Types: double

**RjUnit — Random jitter unit**

'UI' (default) | 'Seconds'

Random jitter unit, specified as 'Seconds' for seconds or 'UI' for unit interval.

**Tunable:** No

Data Types: char

**DCD — Duty cycle distortion**

0 (default) | nonnegative numeric scalar

Duty cycle distortion, specified as a nonnegative number. The value of DCD indicates half of the peak-to-peak DCD, where DCD is defined as half of the difference in symbol duration between one symbol and the next.

**Tunable:** No

Data Types: double

**DCDUnit — DCD unit**

'UI' (default) | 'Seconds'

DCD unit, specified as 'Seconds' for seconds or 'UI' for unit interval.

**Tunable:** No

Data Types: char

**Sj — Sinusoidal or periodic jitter**

0 (default) | nonnegative numeric scalar

Sinusoidal or periodic jitter, specified as a number. The value of Sj indicates half of the peak to peak amplitude of sinusoidal or periodic jitter.

**Tunable:** No

Data Types: double

**SjUnit — Sinusoidal or periodic jitter unit**

'UI' (default) | 'Seconds'

Sinusoidal or periodic jitter unit, specified as 'Seconds' for seconds or 'UI' for unit interval.

Data Types: char

**SjFrequency — Sinusoidal or periodic jitter frequency**

0 (default) | nonnegative numeric scalar

Sinusoidal or periodic jitter frequency, specified as a nonnegative numeric scalar in Hz.

**Tunable:** No

Data Types: double

**JitterSeed — Random seed for jitter**

0 (default) | nonnegative numeric scalar

Random seed for the Dj and Rj jitter streams, specified as a nonnegative number.

**Tunable:** No

Data Types: double

**Advanced****OutputJitter — Enable optional jitter output**

0 (default) | 1

To enable jitter output, set `OutputJitter` to 1. By default, this property is set to 0.

**Tunable:** No

Data Types: `logical`

### **ReversePRBS — Reverse PRBS sequence in time**

0 (default) | 1

To reverse the PRBS sequence in time, set `ReversePRBS` to 1. This reverses the tap positions in the characteristic polynomial. By default, this property is set to 0.

**Tunable:** No

#### **Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS'.

Data Types: `logical`

### **InvertPRBS — Invert PRBS sequence in voltage**

0 (default) | 1

To invert the PRBS sequence in voltage, set `InvertPRBS` to 1. This flips the values of 0s and 1s. By default, this property is set to 0.

**Tunable:** No

#### **Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS'.

Data Types: `logical`

### **BinaryToPAMnMapping — Define mapping of binary payload to symbol voltage**

'Default' (default) | 'ETH\_100BASE\_T1' | 'USB4V2' | 'Uniform'

Define how to map the binary payload to a symbol voltage.

---

**Note** If you set `BinaryToPAMnMapping` to 'Uniform', you must define the properties `PayloadLength` and `MessageLength`.

---

**Tunable:** No

#### **Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS'.

Data Types: `double`

### **PayloadLength — Length of binary payload**

1 (default) | positive integer less than 50

Length of the binary payload, specified as a positive integer less than 50.

**Tunable:** No



**Dependencies**

To enable this property, set `Specification` to 'Serial PRBS', 'Parallel PRBS', or 'PRBS' and set `BinaryToPAMnMapping` to 'Uniform'.

Data Types: double

**MessageLength — Length of symbol message**

1 (default) | positive integer

Length of the symbol message, specified as a positive integer less than or equal to `PayloadLength`

**Tunable:** No

**Dependencies**

To enable this property, set `BinaryToPAMnMapping` to 'Uniform'.

Data Types: double

**SymbolTime — Time of single symbol duration**

1e-10 (default) | positive real scalar

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

**Tunable:** No

Data Types: double

**SampleInterval — Uniform time step of output waveform**

6.25e-12 (default) | positive real scalar

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

**Tunable:** No

Data Types: double

**Usage****Syntax**

```
val = stimulus
```

**Description**

```
val = stimulus
```

**Output Arguments****val — Sample of the PRBS from stimulus**

scalar

A single output sample of the PRBS each time stimulus is called. Equivalent to calling `step(stimulus)`.

## 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)
```

### Specific to `serdes.Stimulus`

`step` Run System object algorithm  
`clone` Create duplicate System object

### 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

### Generate Custom Stimulus Pattern

This example shows how to use the `serdes.Stimulus` object to generate a pseudorandom binary sequence (PRBS) and also modulate custom samples.

First, define the parameters of the sequences. Use a symbol time of 80 ps with 8 samples per symbol. Send 15 symbols, and use 0.1 as the deterministic jitter.

```
SymbolTime = 80e-12;  
SamplesPerSymbol = 8;  
NumberOfSymbols = 15;  
Dj = 0.1;
```

Calculate the sample interval.

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

Create a default Stimulus object that generates a PRBS and uses a NRZ modulation scheme.

```
stimulus1 = serdes.Stimulus(...  
    'SampleInterval',dt,...  
    'SymbolTime',SymbolTime,...  
    'Dj',Dj);
```

Create a Stimulus object that generates a PRBS with an order of 8.

```
stimulus2 = serdes.Stimulus(...  
    'SampleInterval',dt,...  
    'SymbolTime',SymbolTime,...  
    'Delay',dt*4,...  
    'Specification','PRBS',...  
    'Order',8,...  
    'Seed',{[0 1 0 0 0 0 0 0]},...  
    'Dj',Dj);
```

Create a Stimulus object that generates a PRBS and uses a PAM4 modulation scheme.

```
stimulus3 = serdes.Stimulus(...
'Specification','Parallel PRBS',...
'SampleInterval',dt,...
'SymbolTime',SymbolTime,...
'Modulation',4,...
'Delay',dt*4,...
'Order',[6 8],...
'MapToVoltage',2*[-1/2, 1/6, -1/6, 1/2],...
'Dj',Dj);
```

Create a Stimulus object that modulates a custom data pattern.

```
stimulus4 = serdes.Stimulus(...
'SampleInterval',dt,...
'SymbolTime',SymbolTime,...
'Specification','Data Pattern',...
'DataPattern',(-2:2)/2,...
'Dj',Dj);
```

Create a Stimulus object that modulates a sampled voltage.

```
stimulus5 = serdes.Stimulus(...
'SampleInterval',dt,...
'SymbolTime',SymbolTime,...
'Specification','Sampled Voltage',...
'SampledVoltage',[-1.5:0.25:1.5,1.25:-0.25:-1.25]);
```

Initialize the variables for plotting, where N is the number of samples, waveOut contains all of the samples, and t is the vector with the time associated with each sample.

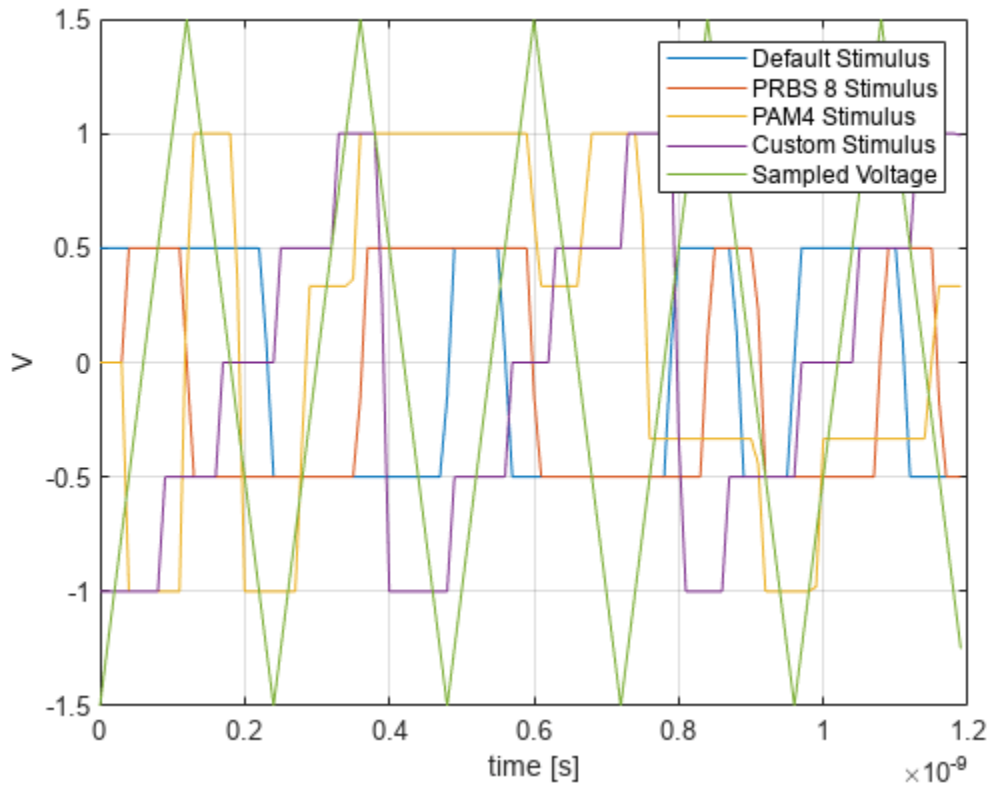
```
N = SamplesPerSymbol*NumberOfSymbols;
waveOut = zeros(N,5);
t = (0:N-1)*dt;
```

Then, use the step of each Stimulus object to sample the generated signals.

```
for ii = 1:N
    waveOut(ii,1) = step(stimulus1);
    waveOut(ii,2) = step(stimulus2);
    waveOut(ii,3) = step(stimulus3);
    waveOut(ii,4) = step(stimulus4);
    waveOut(ii,5) = step(stimulus5);
end
```

Finally, plot waveOut with a legend and labels on the axes.

```
figure
plot(t,waveOut)
xlabel('time [s]')
ylabel('V')
grid on
legend('Default Stimulus','PRBS 8 Stimulus','PAM4 Stimulus',...
'Custom Stimulus','Sampled Voltage')
```



## Tips

- If the PRBS waveform specification definition is incomplete, then extended defaults will be applied. For instance, if Modulation is 4 but Seed is incomplete or incorrect, a valid Seed value will be found and used.
- The PAMn waveform specification creates pulse amplitude modulation (PAM) signals for 2 to 16 levels. This random integer generator selects 16 bits from a PRBS31 data stream to form an integer between 0 and  $2^{16}-1$ . This integer is then linearly mapped to the range of  $[0.501, N + 0.499]$  which is then rounded to an integer in the range of 1 to N, where N is the desired number of PAM levels. This uniform integer then selects the appropriate symbol voltage from the `MapToVoltage` property.
- The injected jitter is accumulated as follows:

DJ:  $\text{jitter} = 2 * \text{Dj} * (\text{rand}(1) - 0.5);$

Rj:  $\text{jitter} = \text{Rj} * \text{randn}(1);$

DCD:  $\text{jitter} = \text{DCD} / 2 * (-1.0)^N$ , where N is the symbol index

Sj:  $\text{jitter} = \text{pSj} * \sin(N * T * 2 * \pi * \text{Sj Frequency})$ , where N is the symbol index, T is the symbol time.

Jitter is injected in the symbol to waveform conversion and therefore jitter cannot be injected when the Specification is 'Sampled Voltage'.

## **Version History**

**Introduced in R2021b**

### **See Also**

`serdes.ChannelLoss` | Stimulus | Analog Channel | **SerDes Designer**

## serdes.VGA

**Package:** serdes

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?](#)

### 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](#).

#### 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

$y = \text{vga}(x)$

### Description

$y = \text{vga}(x)$

### 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);
```

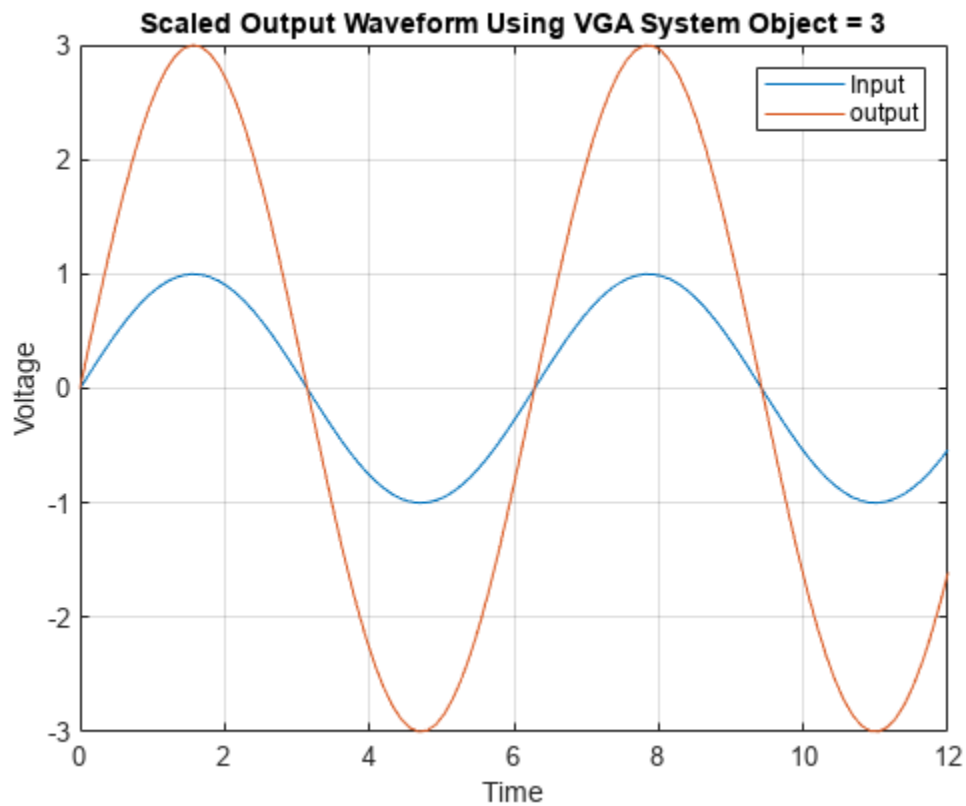
Process the input waveform with the VGA object.

```
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))
```





## Version History

Introduced in R2019a

## 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

VGA | AGC | serdes.AGC



# Blocks

---

# Analog Channel

Construct loss model from channel loss metric or impulse response



**Libraries:**  
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)** — 20%–80% 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.

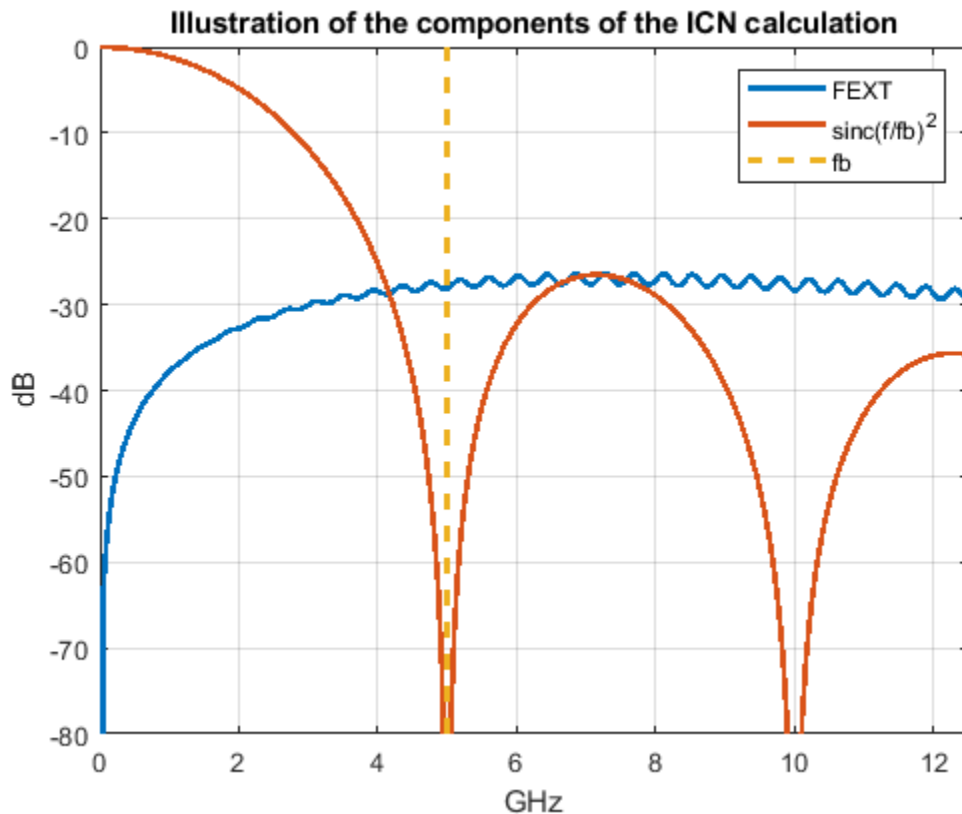
Data Types: double

## 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.

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

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

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

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

## 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.

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

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

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

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

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

where  $\mathbb{I}CN$  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  $S_{11}$ ).

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

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

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

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

## Version History

Introduced in R2019a

## 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](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

Stimulus | Configuration

## Topics

"Analog Channel Loss in SerDes System"

## AGC

Automatically adjusts gain to maintain output waveform amplitude



**Libraries:**  
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.

## **Version History**

**Introduced in R2019a**

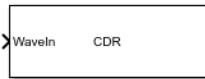
### **See Also**

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



# CDR

Models a clock data recovery circuit



**Libraries:**  
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  $\geq 5$

Early or late CDR count threshold to trigger a phase update, specified as a unitless real positive integer  $\geq 5$ . 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 ( $\pm$ **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.

## More About

### Phase Detector Model

You can select which phase detector model the block uses in the clock recovery: bang-bang (Alexander) or baud-rate type-A (Mueller-Muller). To view the phase detector model used in Simulink, you need to look under the mask of the block and double click the CDR System object to open the block parameter dialog box.

If the **SerDes Designer** exports a CDR block to Simulink that uses bang-bang phase detector model, the app automatically defines the clock position. If you change the phase detector option to baud-rate type-A Simulink, you need to manually add the reserved parameter `Rx_Decision_Time`. To add the parameter, open the AMI-Rx tab of the SerDes IBIS-AMI Manager dialog box. Adding reserved parameter also requires refreshing the Init function. For more information, see “Define Clock Position in Statistical Eye”.

If the **SerDes Designer** exports a CDR block to Simulink that uses baud-rate type-A phase detector model, the app automatically uses the reserved AMI parameter `Rx_Decision_Time` in the AMI-Rx tab of the SerDes IBIS-AMI Manager dialog box to define the clock position.

## Version History

Introduced in R2019a

### See Also

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

### Topics

“Clock and Data Recovery in SerDes System”

## Configuration

Configure system wide settings in SerDes system model



**Libraries:**  
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.

### Limitations

- Non-inlined S-functions are not supported for IBIS-AMI model export. If you have a Simulink Coder™ or Embedded Coder® license, you can convert your S-functions to inlined to support IBIS-AMI model export.

### 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) | PAM3 | PAM4 | PAM8 | PAM16

Number of logic levels in the modulation scheme:

- Select NRZ if the modulation scheme has two logic levels.
- Select PAM3 if the modulation scheme has three logic levels.
- Select PAM4 if the modulation scheme has four logic levels.
- Select PAM8 if the modulation scheme has eight logic levels.
- Select PAM16 if the modulation scheme has sixteen 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. The results derived from the equalized impulse response includes the pulse response, waveform, statistical eye, and statistical eye metrics (eye height, eye width, eye area, channel operating margin, and vertical eye closure).

**Plot time domain analysis after simulation** — Plot time analysis after simulation  
on (default) | off

Select to plot the time domain analysis results after the simulation is run. By default, this option is selected. The result includes time domain eye with clock ticks histogram and time domain eye metrics (eye height, eye width, eye area, channel operating margin, vertical eye closure, and minimum BER).

The time domain eye is derived from the output of the receiver blocks and the clock ticks generated by the DFECDR, CDR block, or IBIS-AMI clock\_times block. To generate a meaningful time domain eye, the maximum value of the **Bits to ignore** parameter for Tx and Rx in the **Export** tab of the SerDes IBIS-AMI Manager dialog box is used.

**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 Contents of IBIS and AMI files”.

**Open Signal Integrity Link** — Open Signal Integrity Link to Signal Integrity Toolbox™  
button

Click to open the Signal Integrity Link dialog box. The Signal Integrity Link dialog box is used to test the IBIS-AMI SerDes models in the **Serial Link Designer** or **Parallel Link Designer** app from the Signal Integrity Toolbox. You can transfer the data required to reproduce a **Serial Link Designer** or **Parallel Link Designer** test case back to Simulink for debugging and refinement.

Using the Signal Integrity Link, you can:

- Create a **Serial Link Designer** project.
- Create a **Parallel Link Designer** project.

- Import **Serial Link Designer** or **Parallel Link Designer** simulation data into Simulink workspace.
- Update **Serial Link Designer** or **Parallel Link Designer** with new data from Simulink workspace.

If the linked **Serial Link Designer** or **Parallel Link Designer** project resides in a different directory from the SerDes Simulink model file, you need to save the full path to locate it. For more information, see “Signal Integrity Link”.

## More About

### Eye Metrics

You can view the performance metrics of statistical and time domain eye by plotting the statistical and time domain results.

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 levels of PAMn, given by the equation: $\text{Linearity} = \frac{\text{Minimum amplitude of the different eye levels}}{\text{Maximum amplitude of the different eye levels}}$
COM	Channel operating margin, given by the equation: $\text{COM} = 20\log_{10}\left(\frac{\text{Mean eye height}}{\text{Mean eye height} - \text{Inner eye height}}\right)$
VEC	Vertical eye closure, given by the equation: $\text{VEC} = \frac{\text{Mean eye height}}{\text{Inner eye height}}$
Minimum BER	Minimum BER possible based on number of symbols and ignored bits set in the SerDes IBIS-AMI Manager dialog box, given by the equation: $\text{Minimum BER} = \frac{1}{(\text{Number of symbols} - \text{Ignored bits})}$  Minimum BER should be greater than or equal to target BER.

**Note** Mean eye height is defined as the height of the eye between the densest part of eye histogram below and above the inner eye.

### Generating Meaningful Time Domain Eye

The time domain eye represents the results of the time domain analysis derived from the output of the receiver and, if generated, the clock ticks. To generate a meaningful time domain eye, it is important to ignore the results from a preliminary number of bits from the transmitter and receiver while the equalization algorithms are adapting to find the optimum configuration. You can set the number of bits to ignore by setting the Tx and Rx **Bits to Ignore** parameter in the **Export** tab of the SerDes IBIS-AMI Manager dialog box. For time domain analysis, the maximum value of the two **Bits to Ignore** parameters is ignored. Depending on how many bits are ignored, the eye and associated metrics of the time domain eye might change.

## **Version History**

**Introduced in R2019a**

### **See Also**

Analog Channel | Stimulus

### **Topics**

“Managing AMI Parameters”

“Customizing SerDes Toolbox Datapath Control Signals”

“Manage Contents of IBIS and AMI files”



# CTLE

Models continuous time linear equalizer (CTLE)



**Libraries:**  
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. The time domain filter coefficients can be calculated from one large partial fraction or many cascaded sections.

## 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 to maximize the SNR 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.

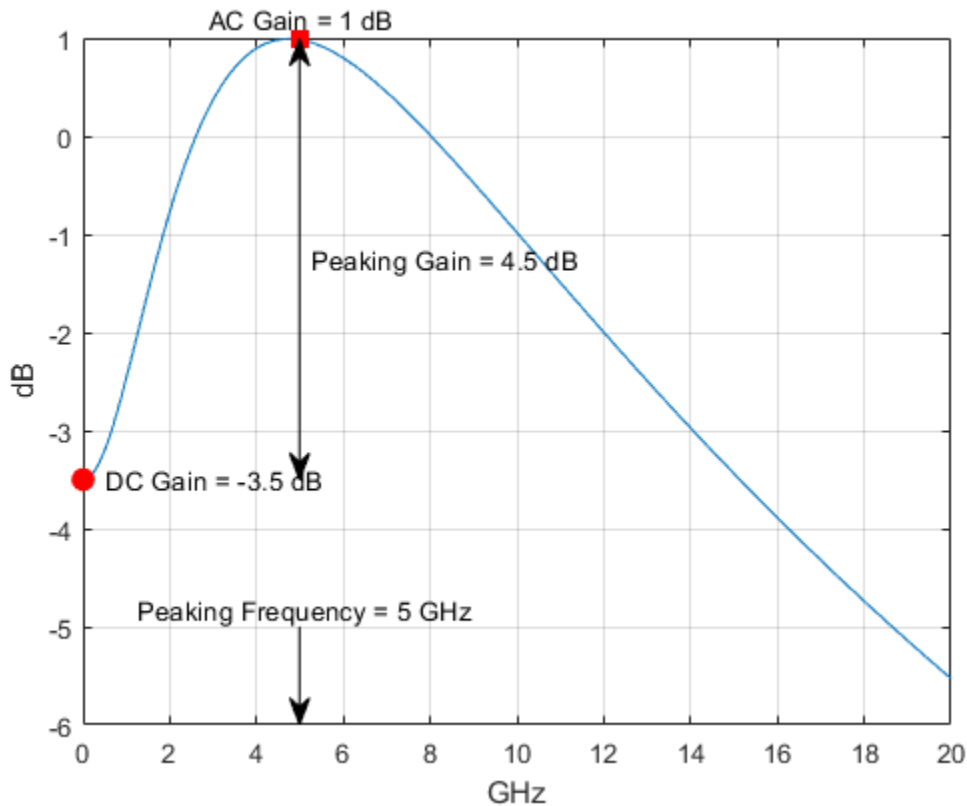
Data Types: double

**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.

The block only allows repeated poles or zeros when it uses a cascaded filter. To change the filter method the block uses, look under the mask and double click the CTLE System object. This launches block parameters dialog box, and you can find the **Filter method** parameter in the **Advanced** tab.

Complex poles or zeros must have conjugates. The number of poles must be greater than number of zeros for system stability. The block ignores poles and zeros of 0 Hz and you can use them to zero-pad the matrix.

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

#### Visualize Response — Plot CTLE response button

Click to plot the CTLE response. The resulting plot shows CTLE transfer function family curves for different CTLE configurations.

#### Launch CTLE Fitter App — Launch CTLE Fitter app button

Click to launch the CTLE fitter app. For more information, see **CTLE Fitter**.

**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.

## Version History

Introduced in R2019a

### See Also

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

# DFE/CDR

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



**Libraries:**  
SerDes Toolbox / Datapath Blocks

## Description

The DFE/CDR 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 object 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”.

### 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

**2x tap weights** — Multiply DFE tap weights by a factor of two  
off (default) | on

Select to multiply the DFE tap weights by a factor of two.

The output of the slicer in the DFECDR block from the SerDes Toolbox is [-0.5 0.5]. But some industry applications require the slicer output to be [-1 1]. **2x tap weights** allows you to quickly double the DFE tap weights to change the slicer reference.

## 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  $\geq 5$

Early or late CDR count threshold to trigger a phase update, specified as a unitless positive real integer  $\geq 5$ . 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 DFECCR **Early/late count threshold** value.
- Use `set_param(gcb, 'Count', value)` to set DFECCR 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 DFECCR **Step (symbol time)** value.
- Use `set_param(gcb, 'ClockStep', value)` to set DFECCR 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 ( $\pm$ **Sensitivity (V)**), there is a 50% probability of bit error.

**Programmatic Use**

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

## More About

### Phase Detector Model

You can select which phase detector model the block uses in the clock recovery: bang-bang (Alexander) or baud-rate type-A (Mueller-Muller). To view the phase detector model used in Simulink, you need to look under the mask of the block and double click the DFECDR System object to open the block parameter dialog box.

If the **SerDes Designer** exports a DFECDR block to Simulink that uses bang-bang phase detector model, the app automatically defines the clock position. If you change the phase detector option to baud-rate type-A Simulink, you need to manually add the reserved parameter `Rx_Decision_Time`. To add the parameter, open the AMI-Rx tab of the SerDes IBIS-AMI Manager dialog box. Adding reserved parameter also requires refreshing the Init function. For more information, see “Define Clock Position in Statistical Eye”.

If the **SerDes Designer** exports a DFECDR block to Simulink that uses baud-rate type-A phase detector model, the app automatically uses the reserved AMI parameter `Rx_Decision_Time` in the AMI-Rx tab of the SerDes IBIS-AMI Manager dialog box to define the clock position.

## Version History

**Introduced in R2019a**

### See Also

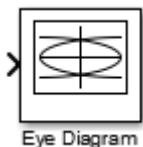
`serdes.DFECDR` | `serdes.CDR` | `CDR`

### Topics

“Clock and Data Recovery in SerDes System”

# Eye Diagram

Display eye diagram of time-domain signal



## Libraries:

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

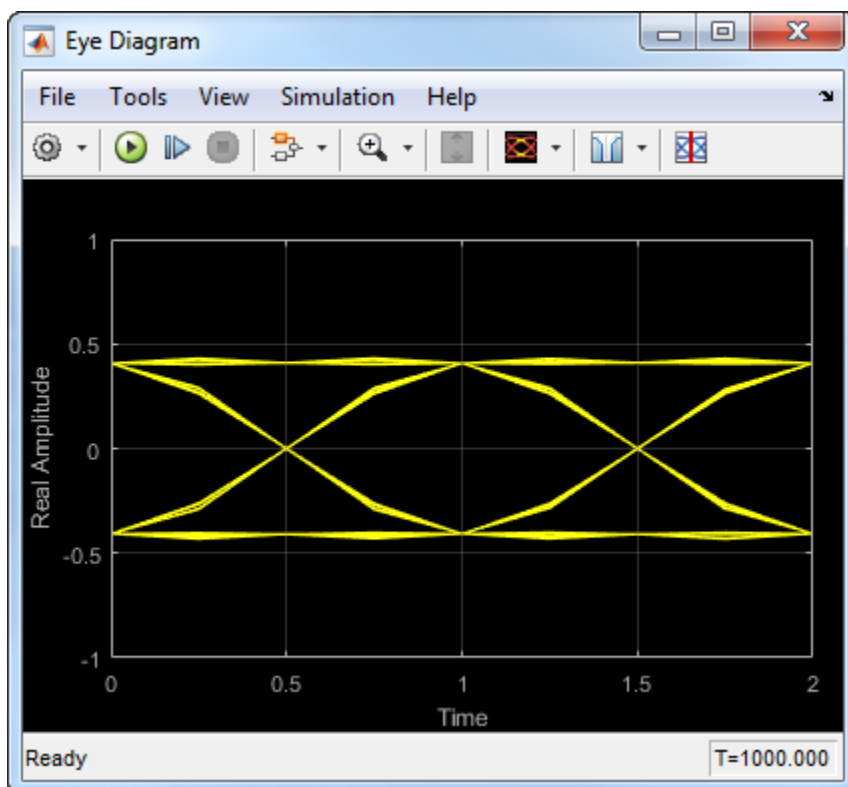
## Description

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. .

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 MATLAB Handle Graphics® plot objects.

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

**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

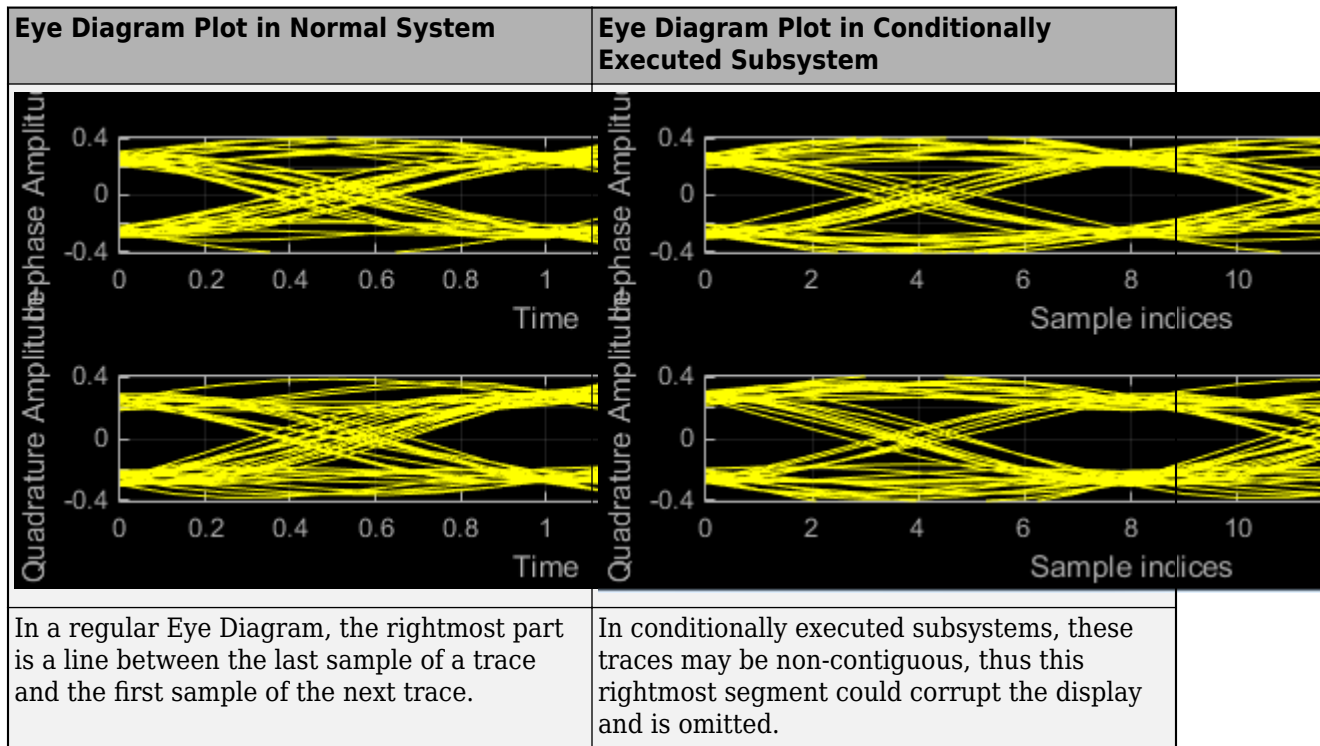
<b>Data Types</b>	Boolean   double   enumerated   fixed point   integer   single
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	no
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

## 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:

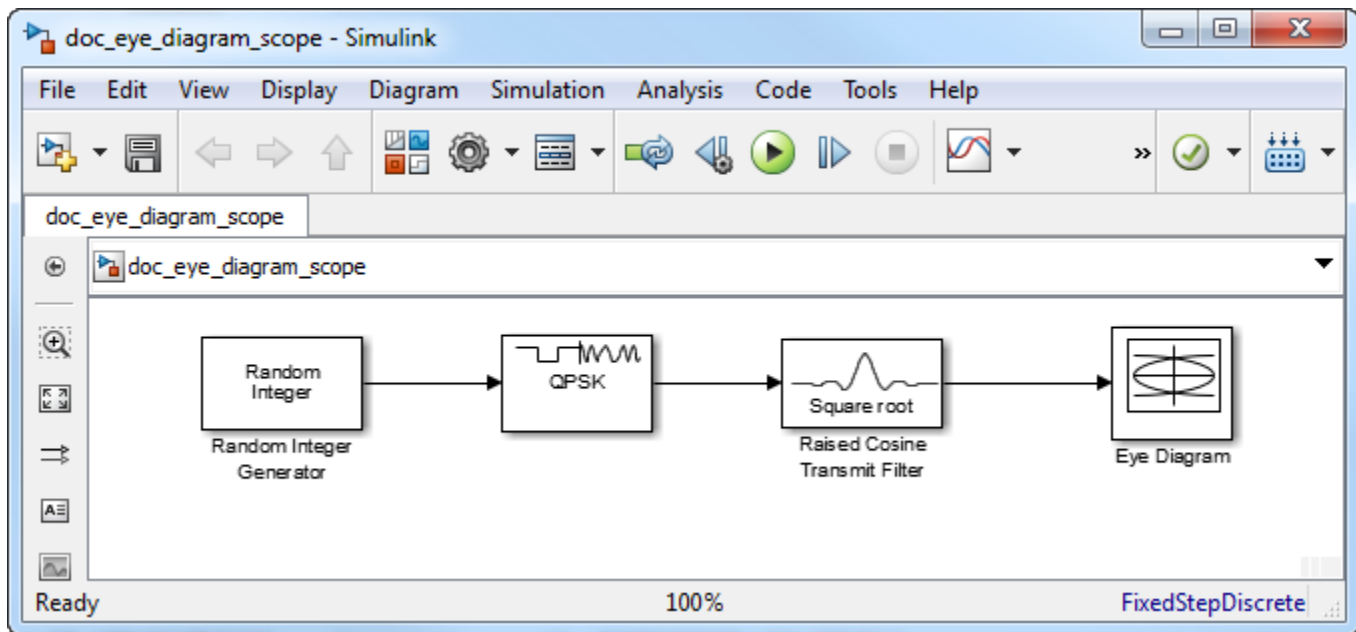
- Input size must be an integer multiple of `SamplesPerSymbol * SymbolsPerTrace`
- Sample offset must be zero
- The rightmost part of the display is intentionally omitted. This figure compares typical eye diagram display when placed in a normal system versus one placed in a conditionally executed subsystem.



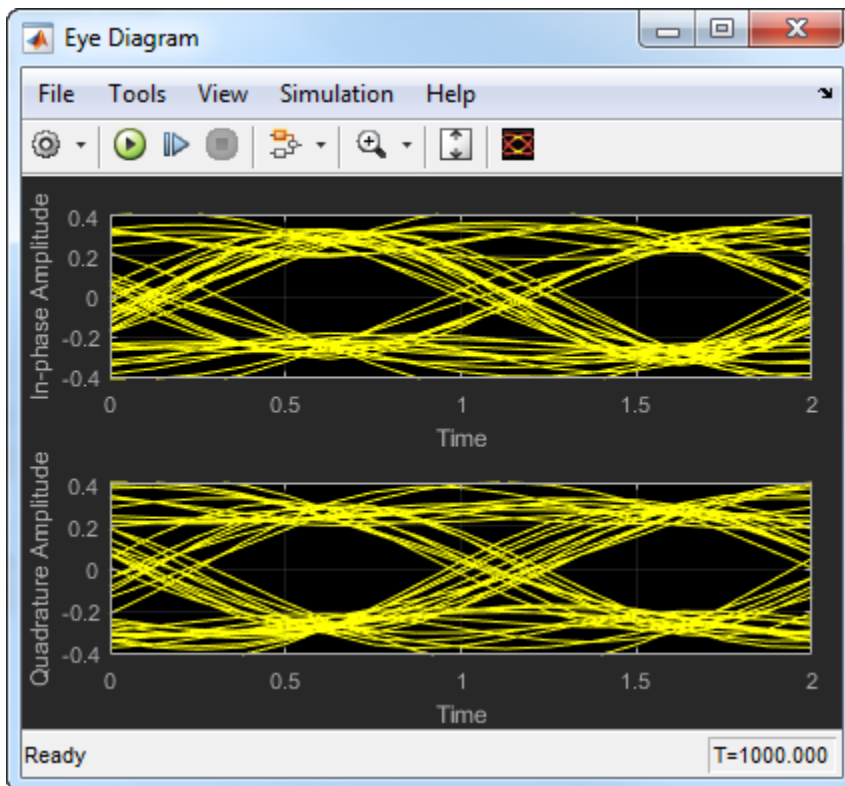
### View Eye Diagram

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

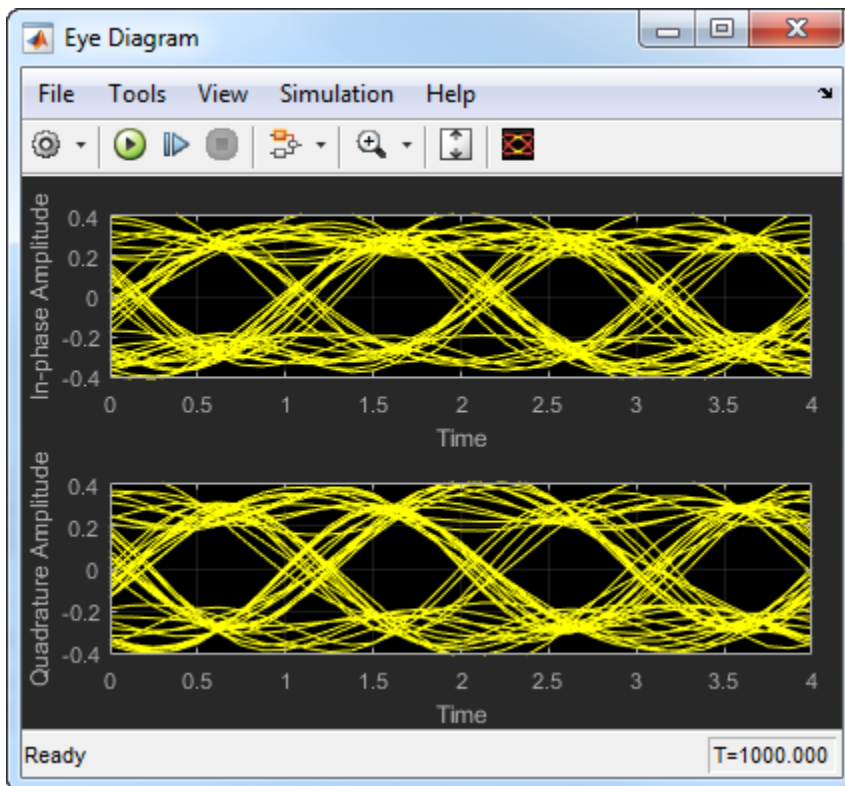
Build the `doc_eye_diagram_scope` model as shown in this figure.



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.



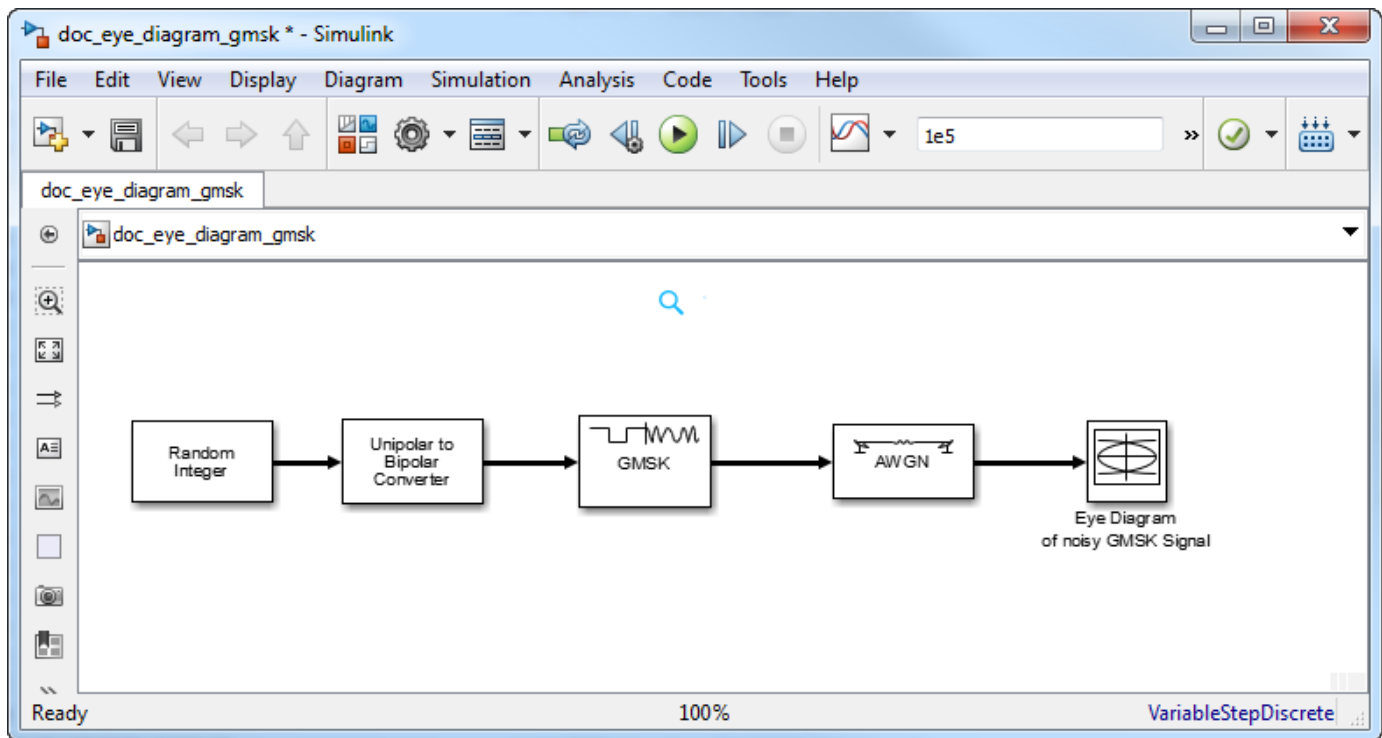
Try changing the Raised Cosine Transmit Filter parameters or changing additional Eye Diagram parameters to see their effects on the eye diagram.

### Histogram Plots

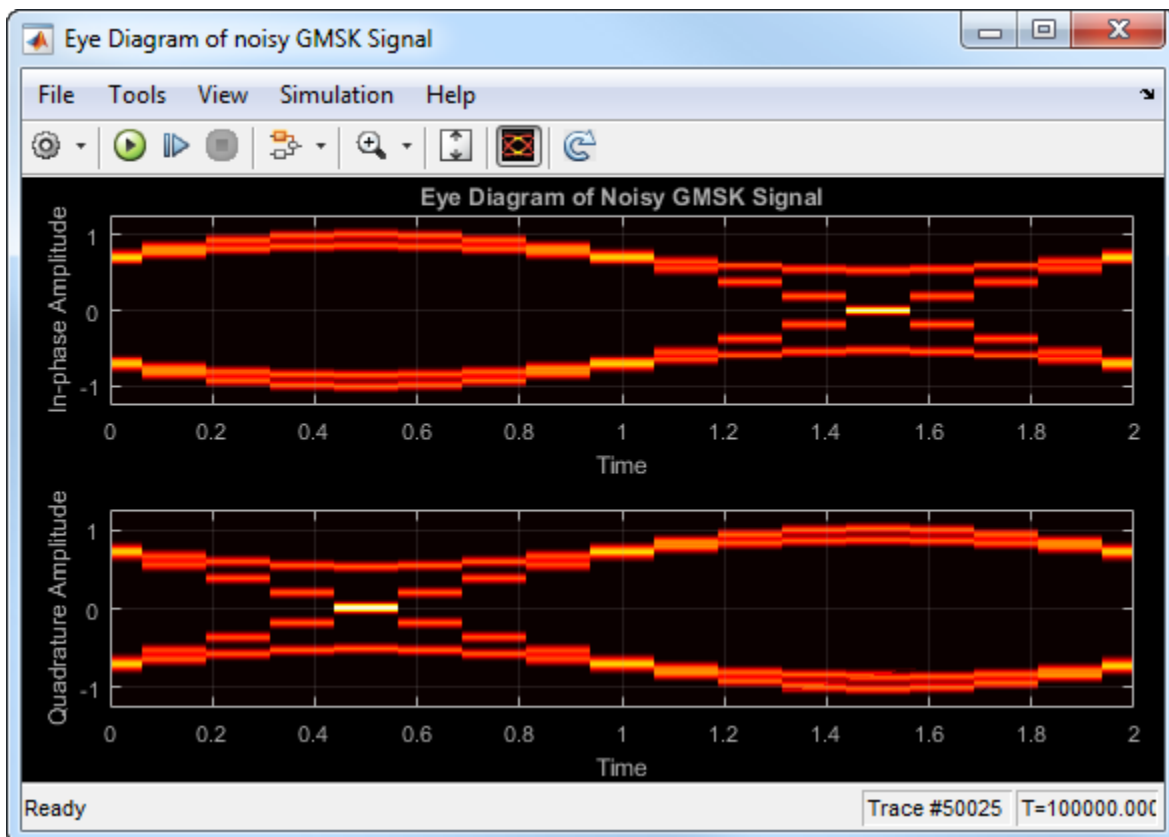
Display histogram plots of a noisy GMSK signal.

Build the `doc_eye_diagram_gmsk` model as shown in this figure.



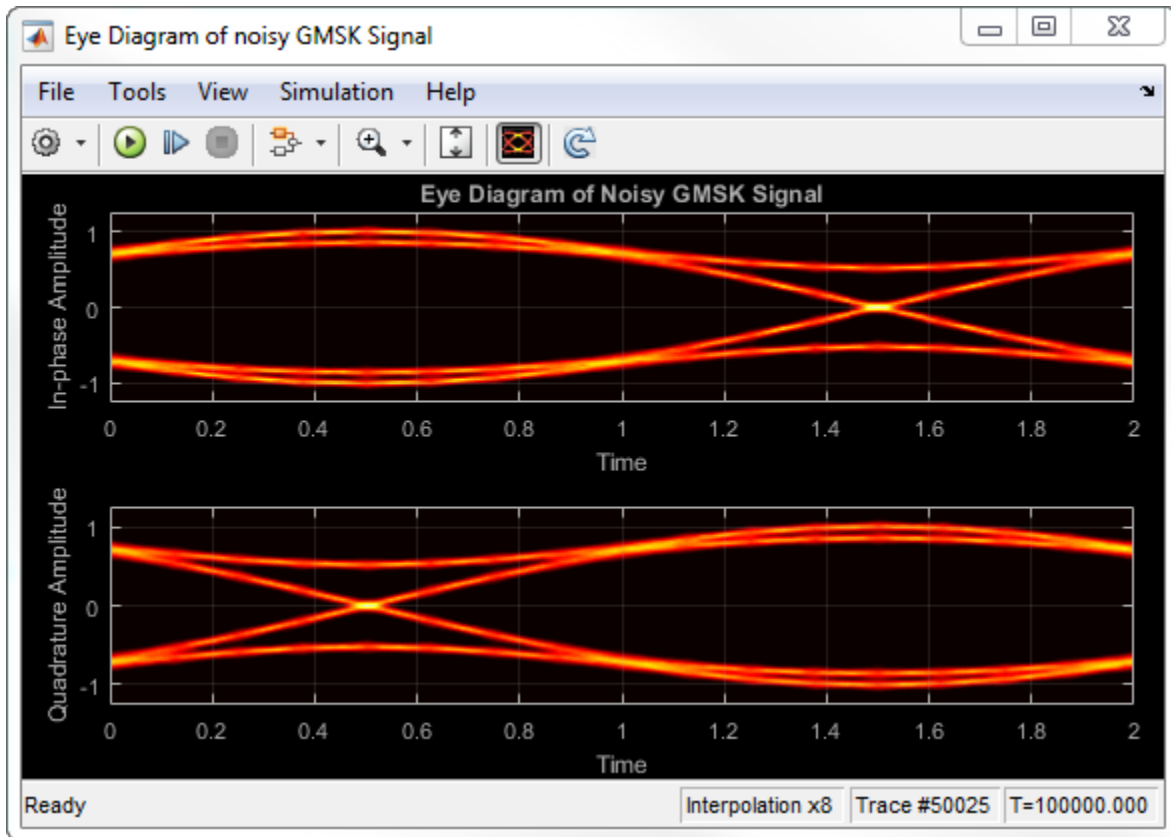


Run the model. The eye diagram is configured to show a histogram without interpolation.



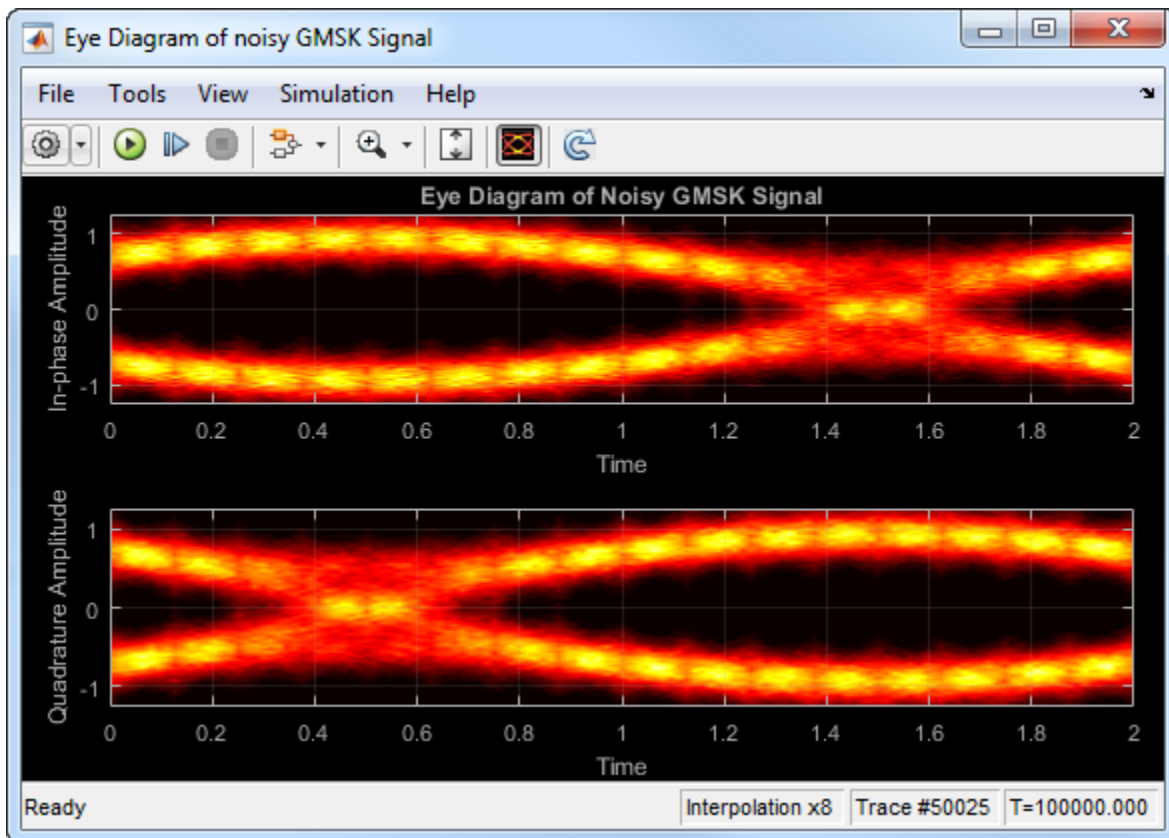
The lack of interpolation results in a plot having piecewise-continuous behavior.

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



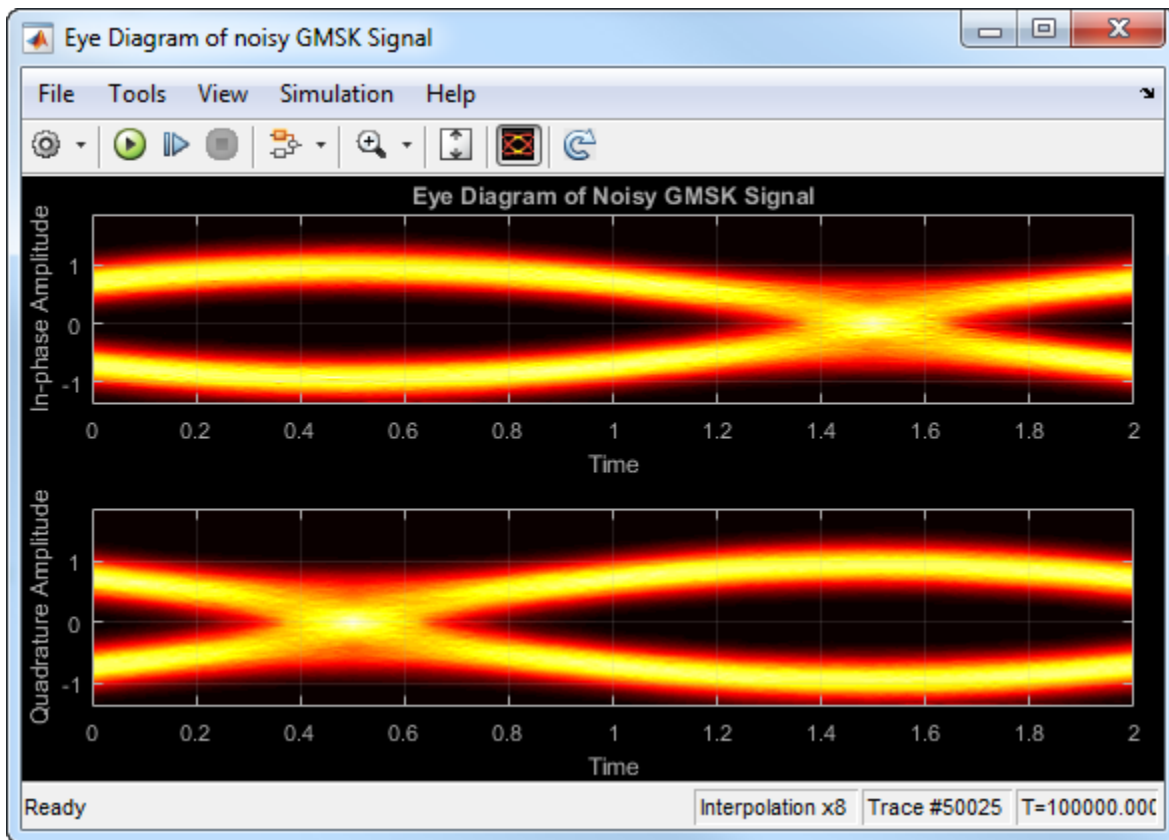
The interpolation smooths the eye diagram.

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.

## Version History

Introduced in R2014a

## 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.

# FFE

Models a feed-forward equalizer



**Libraries:**  
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.

#### Visualize Response — Plot FFE FIR filter response

button

Click to plot the response of the FFE FIR (finite impulse response) filter.

#### 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.

## Version History

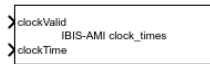
Introduced in R2019a

### See Also

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

# IBIS-AMI clock\_times

Recover SerDes clock time values from custom DFECDR and CDR



**Libraries:**  
SerDes Toolbox / Utilities

## Description

The IBIS-AMI clock\_times block recovers the clock time values of a SerDes system when you use a custom DFECDR or CDR block instead of the ones provided with the SerDes Toolbox. The block recovers the clock time values from the AMI\_GetWave function and outputs them to the generated IBIS-AMI models.

The output of the IBIS-AMI clock\_times block is two global variables, `extern double* Rx_clock_times_buf` and `extern int Rx_clock_times_idx`. The block only creates the variables. You must add them to your custom code to make them work. Together, the variables implement the `clock_times` vector as defined in the IBIS 7.0 specifications.

## Ports

### Input

**clockValid** — Update input at clockTime port  
scalar | vector

Update the input at **clockTime** port, specified as a scalar or vector. The value at the **clockValid** port results in only one clock times output per sample.

Each rising edge of the input signal at the **clockValid** port updates the S function that passes through the **clockTime** port. Usually this happens once per unit interval (UI).

Data Types: `double` | `int8`

**clockTime** — Clock time data  
array

Clock time data, specified as an array. Clock time data is passed to the AMI wrapper code using an S function.

The time value at the **clockTime** port must be half UI before the actual data sample times.

Data Types: `double`

## Parameters

**Enable IBIS-AMI Offset** — Offset clock time values by half UI  
on (default) | off

Offset clock time values by half UI, according to IBIS-AMI specifications.

## **More About**

### **Clock times vector**

The EDA tool allocates the clock times vector. The vector is referenced at the first call by the AMI\_GetWave function. The clock times are exactly half the bit time before the input data signal is sampled. The algorithmic model returns nonnegative clock time values. After the last valid clock tick in the clock times vector, a -1 denotes the end of the vector. The clock ticks represented by clock times vector is strictly monotonic.

To sample the output waveform, each valid value of the clock times vector is added to half the bit time (half the sample time for PAM4).

## **Version History**

**Introduced in R2020b**

## **References**

[1] IBIS 7.0 Specification, [https://ibis.org/ver7.0/ver7\\_0.pdf](https://ibis.org/ver7.0/ver7_0.pdf).

## **See Also**

DFECDR | CDR



# PassThrough

Propagates baseband signal without modification



**Libraries:**  
SerDes Toolbox / Datapath Blocks

## Description

The PassThrough block passes the input signal without any modification. This block is used as a placeholder 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

## Version History

**Introduced in R2019a**

## See Also

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

## Topics

“Managing AMI Parameters”

# SaturatingAmplifier

Models a saturation amplifier



## Libraries:

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*×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`

**Visualize Response** — Plot input vs output voltage response

button

Click to plot the input vs. output voltage response of the Saturating Amplifier block.

#### **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.

## **Version History**

**Introduced in R2019a**

### **See Also**

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

# Stimulus

Set waveform generation method and number of symbols to simulate in SerDes model



**Libraries:**  
SerDes Toolbox / Utilities

## Description

The Stimulus defines how to create the waveform and the number of symbols to simulate in a SerDes Toolbox model. You can directly specify a PAMn symbol pattern or define a binary to PAMn mapping scheme.

## 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

**Number of symbols** — Length of stimulus pattern used for simulation  
20000 (default) | positive real integer

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

### Programmatic Use

**Block parameter:** NumberOfSymbols

**Type:** character vector

**Values:** positive real integer

**Default:** 20000

**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.

**Recommended simulation stop time** defines the minimum time the simulation must run for meaningful results. It is calculated by multiplying the **Sample Interval** from the Configuration block with the **Number of symbols**.

**Waveform creation method** — Define how stimulus waveform is created

Random Symbols (default) | Serial PRBS | Symbol Pattern | Binary Pattern | Symbol Voltage Pattern | Sampled Voltage | Parallel PRBS

Define how the stimulus waveform is created:

- **Random Symbols** — Randomly generate symbols directly from a uniform distribution.
- **Serial PRBS** — Define a binary sequence and map to a higher order modulation symbol. The mapping sequence depends on the modulation scheme defined in the Configuration block.
- **Symbol Pattern** — Directly define the symbol pattern as an integer from 0 to (Number of modulation levels - 1).
- **Binary Pattern** — Directly define the binary pattern and map to a higher order modulation symbol. The mapping sequence depends on the modulation scheme defined in the Configuration block.
- **Symbol Voltage Pattern** — Directly define the voltages of a symbol spaced pattern.
- **Sampled Voltage** — Directly define a sample interval spaced waveform.
- **Parallel PRBS** — Utilize multiple PRBS streams to form PAM4, PAM8, PAM16, and PAM32 symbols. You can define up to 5 PRBS streams for PAM32. NRZ modulation scheme uses a single PRBS stream. Other modulation schemes use the random symbols waveform creation method.

**Programmatic Use**

**Block parameter:** CreationMethod

**Type:** character vector

**Values:** Random Symbols | Serial PRBS | Symbol Pattern | Binary Pattern | Symbol Voltage Pattern | Sampled Voltage | Parallel PRBS

**Default:** Random Symbols

**Random symbols seed** — Seed for generating random symbol waveform  
1431655765 (default) | positive real scalar

Seed for generating the random symbol waveform, specified as a positive real scalar.

**Dependencies**

- To enable this parameter, set **Waveform creation method** parameter to Random Symbols.

**Programmatic Use**

**Block parameter:** soPAMnSeed

**Type:** character vector

**Values:** positive real scalar

**Default:** 1431655765

**PRBS order** — Order of pseudorandom binary sequence  
11 (default) | 7 | 9 | 13 | 15 | 20 | 23 | 31 | 47

Order of the pseudorandom binary sequence while creating the stimulus waveform using serial PRBS or parallel PRBS method.

**Dependencies**

To enable this parameter, set **Waveform creation method** parameter to Serial PRBS or Parallel PRBS.

**Programmatic Use**

**Block parameter:** PRBS

**Type:** character vector

**Values:** 7 | 9 | 11 | 13 | 15 | 20 | 23 | 31 | 47

**Default:** 11

**PRBS seed** — Seed of pseudorandom binary sequence

1 (default) | integer in the range [1,  $2^{\text{Order}-1}$ ]

Seed for generating the pseudorandom binary sequence, specified as an integer in the range of [1,  $2^{\text{Order}-1}$ ].

**Dependencies**

To enable this parameter, set **Waveform creation method** parameter to Serial PRBS or Parallel PRBS.

**Programmatic Use**

**Block parameter:** PRBSSeed

**Type:** character vector

**Values:** integer in the range [1,  $2^{\text{Order}-1}$ ]

**Default:** 1

**Reverse PRBS in time** — Reverse PRBS in time by reversing tap positions of characteristic polynomial

off (default) | on

Reverse the PRBS in time (along x-axis) by reversing the tap positions of the characteristic polynomial.

**Dependencies**

To enable this parameter, set **Waveform creation method** parameter to Serial PRBS or Parallel PRBS.

**Invert PRBS in voltage** — Invert PRBS in voltage by flipping the values of 0 and 1

off (default) | on

Invert the PRBS in voltage (along y-axis) by converting 0s to 1s and 1s to 0s.

**Dependencies**

To enable this parameter, set **Waveform creation method** parameter to Serial PRBS or Parallel PRBS.

**Symbol pattern** — Vector of symbol integers spaced at symbol time interval

[0 0 1 0 0 1 1 0 1 0 1 1 1 1 0] (default) | vector with element values in the range [0, (Number of modulation levels-1)]

Vector of the symbol integers spaced at the symbol time interval. The elements of the vector must have values in the range [0, (Number of modulation levels-1)]

**Dependencies**

To enable this parameter, set **Waveform creation method** parameter to Symbol Pattern.

**Programmatic Use**

**Block parameter:** SymbolPattern

**Type:** character vector

**Values:** vector with element values in the range [0, (Number of modulation levels-1)]

**Default:** [0 0 1 0 0 1 1 0 1 0 1 1 1 1 0]

**Binary pattern** — Vector of 1s and 0s spaced at sample time interval

[0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0] (default) | vector with binary elements

Vector of binary elements (1s and 0s) spaced at the symbol time interval.

#### Dependencies

To enable this parameter, set **Waveform creation method** parameter to Binary Pattern.

#### Programmatic Use

**Block parameter:** BinaryPattern

**Type:** character vector

**Values:** vector with binary elements

**Default:** [0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0]

**Symbol voltage pattern** — Vector of symbol voltages spaced at symbol time interval

$1/2*[1 1 1 1 -1 -1 -1 -1]$  (default) | vector with decimal values

Vector of symbol voltages spaced at the symbol time interval.

#### Dependencies

To enable this parameter, set **Waveform creation method** parameter to Symbol Voltage Pattern.

#### Programmatic Use

**Block parameter:** SymbolVoltageDataPattern

**Type:** character vector

**Values:** vector with decimal values

**Default:**  $1/2*[1 1 1 1 -1 -1 -1 -1]$

**Sampled voltage pattern** — Vector of voltages spaced at sample interval

[zeros(1, (SymbolTime/SampleInterval)), ones(1, (SymbolTime/SampleInterval))]-0.5 (default) | real valued vector

Vector of voltages spaced at the sample interval.

#### Dependencies

To enable this parameter, set **Waveform creation method** parameter to Sampled Voltage.

#### Programmatic Use

**Block parameter:** CustomStimulus

**Type:** character vector

**Values:** real valued vector

**Default:** [zeros(1, (SymbolTime/SampleInterval)), ones(1, (SymbolTime/SampleInterval))]-0.5

**Sub symbol time delay (s)** — Delay stimulus waveform by modulo symbol time

0 (default) | nonnegative real scalar

Delay the stimulus waveform by modulo symbol time to slightly shift the eye diagram or recovered clock phase.

The modulo operation finds the remainder after division.



**Programmatic Use****Block parameter:** `soDelay`**Type:** character vector**Values:** nonnegative real scalar**Default:** 0**Version History****Introduced in R2019a****See Also**Analog Channel | Configuration | `serdes.Stimulus`

## VGA

Models a variable gain amplifier



**Libraries:**  
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.

## Version History

Introduced in R2019a

### See Also

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



# Apps

---

## CTLE Fitter

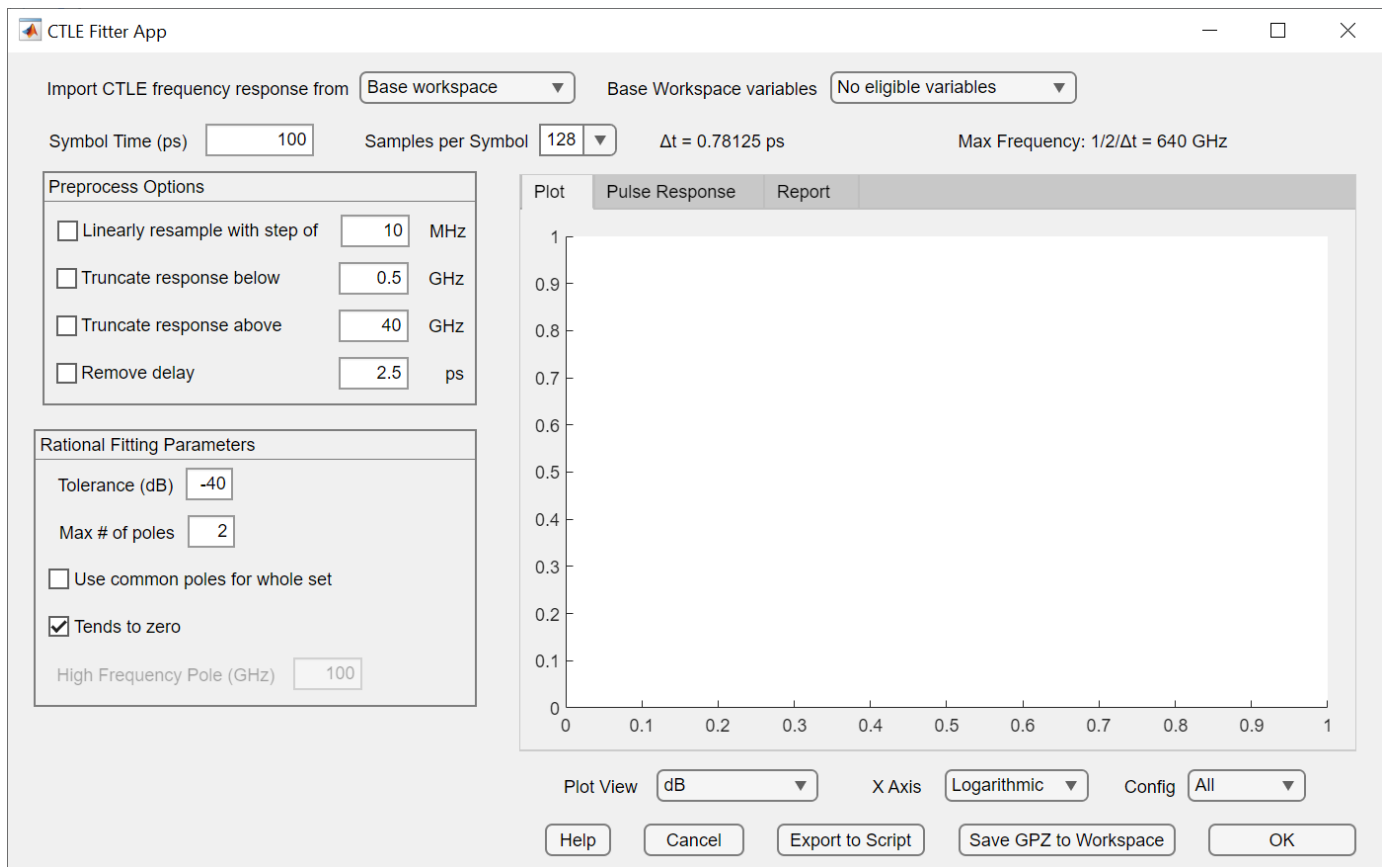
Fit poles and zeros to CTLE transfer functions

### Description

The **CTLE Fitter** app fits poles, zeros, and gains to CTLE transfer functions. The app also allows an organized approach to preprocess CTLE data sets information. The app then fits the parameters using the rational function from the RF Toolbox™.

Using the app, you can:

- Easily fit poles and zeros to frequency domain CTLE transfer functions.
- Easily input GPZ matrix to the block.
- Export the script to base workspace.



### Open the CTLE Fitter App

- MATLAB command prompt: Enter `ctlefitter`.
- **SerDes Designer** app: Open the CTLE block parameters dialog box and click the **Launch CTLE Fitter** button.

- Simulink model: Open the CTLE block parameters dialog box and click the **Launch CTLE Fitter** button.

## Examples

- “Find Zeros, Poles, and Gains for CTLE from Transfer Function”

## Programmatic Use

ctlefitter opens a new session of the **CTLE Fitter** app.

## More About

### Importing Data

You can import data to the **CTLE Fitter** app in two ways:

- **From CSV file** — The data in the CSV file must have specific headers and data arrangement.

freq	<data_1>	<data_2>	<data_1>	<data_2>	<data_1>	<data_2>
<value>	<value>	<value>	<value>	<value>	<value>	<value>
<value>	<value>	<value>	<value>	<value>	<value>	<value>

<data\_1> and <data\_2> column pairs can only be one of the following options:

- real and imag,
- db and phaseDeg,
- db and phaseRad,
- mag and phaseDeg,
- mag and phaseRad.

The CSV file must contain the first three columns. The subsequent column pairs are optional. The first column must contain frequencies. The subsequent column pairs can only have one set of data format. For example:

freq	db	phaseDeg	db	phaseDeg
<value>	<value>	<value>	<value>	<value>

- **From base workspace** — You can create the base workspace variables by reading the data from a CSV file.

```
fn = 'ExampleCTLEDatadBPhaseDeg.csv';
[f,H]=ctlefit.readcsv(fn);
MyVar = [f,H];
```

## Version History

Introduced in R2022a

## **See Also**

### **Apps**

**SerDes Designer**

### **Functions**

`serdes.CTLE | rational`

### **Topics**

“Find Zeros, Poles, and Gains for CTLE from Transfer Function”



# SerDes Designer

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

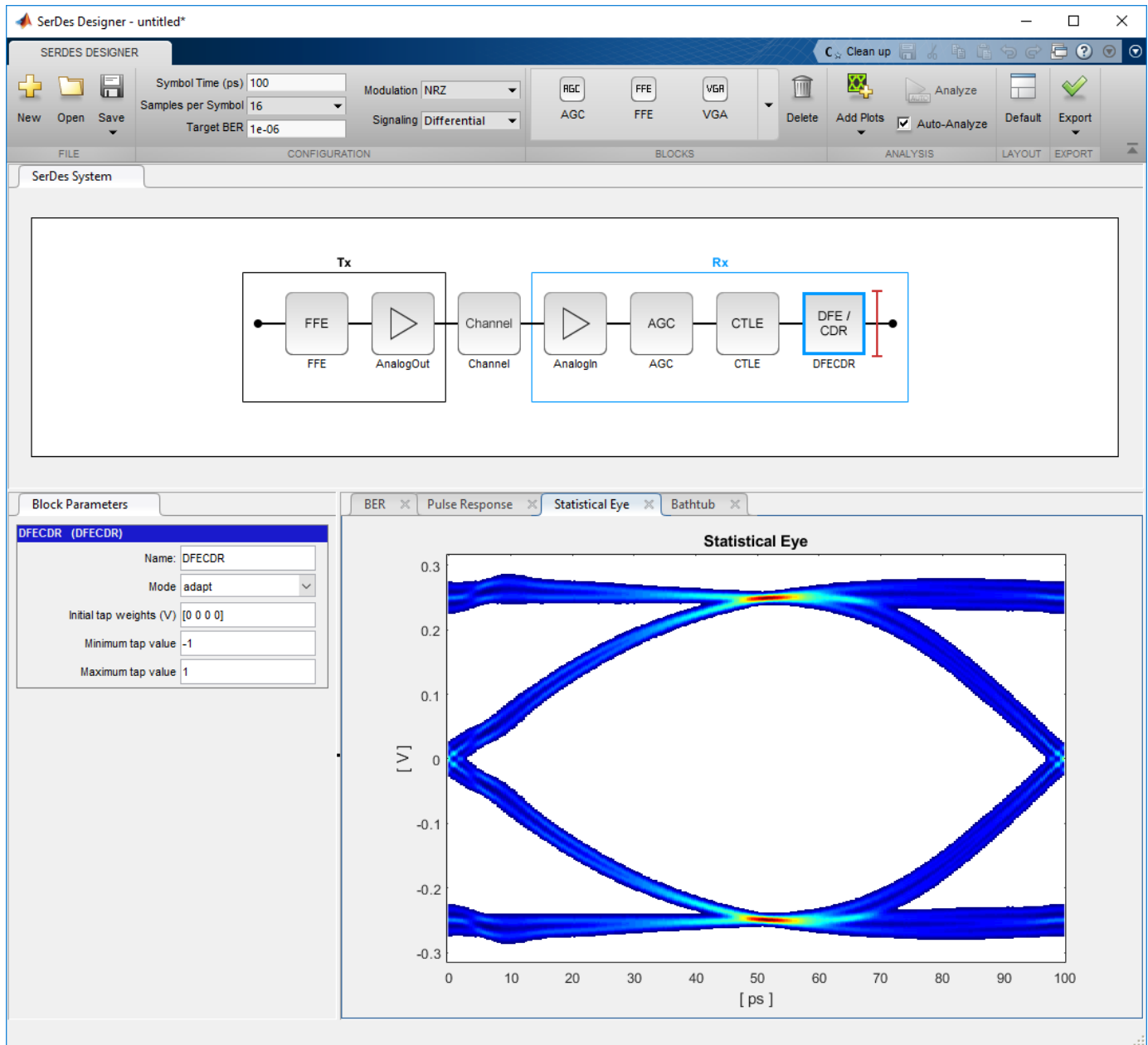
## Description

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, PAM3, PAM4, PAM8, and PAM16.
Signalling	Differential	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 levels of PAMn, given by the equation: $\text{Linearity} = \frac{\text{Minimum amplitude of the different eye levels}}{\text{Maximum amplitude of the different eye levels}}$
COM	Channel operating margin, given by the equation: $\text{COM} = 20\log_{10}\left(\frac{\text{Mean eye height}}{\text{Mean eye height} - \text{Inner eye height}}\right)$
VEC	Vertical eye closure, given by the equation: $\text{VEC} = \frac{\text{Mean eye height}}{\text{Inner eye height}}$

---

**Note** In a PAM $n$  modulation, there are  $(n-1)$  inner eyes. Each eye generates its own COM and VEC values. The app reports the minimum of the generated COM values and the maximum of the generated VEC values.

---

**Note** According to IBIS BIRD (Buffer Issue Resolution Document) 213, IBIS-AMI models support any level of signaling from PAM2 (NRZ) to upwards, collectively known as PAM $n$ . If your EDA tool supports it, you can export IBIS-AMI models supporting modulation schemes NRZ, PAM3, PAM4, PAM8, or PAM16.

---

## Jitter Analysis

You can include jitter impairment in analysis during link and equalization design. This helps you perform trade-off between different equalization schemes which have different contributions to the total jitter. You can also calculate eye margin during link analysis and include jitter values in the IBIS-AMI models.

From the app, you can define the clock mode, transmitter jitter parameters, and receiver jitter parameters. The clock probability density function (PDF) is included on the BER plot. For more information about jitter parameters, see “Jitter Analysis in SerDes Systems”.

If you export your SerDes model to Simulink, the jitter values set in the app are used to populate the SerDes IBIS-AMI tree.

## Exporting SerDes System

From the app toolbar, 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.

## Import S-Parameters for Channel Model

If you set your **Channel model** as Impulse response, you can import S-Parameters from a Touchstone file to use as your channel model by clicking the **Import S-Parameter** model. From the S-Parameter Fitter dialog box that opens, you can set the maximum number of poles, error tolerance, transmitter amplitude, rise time and RTF factor, and the resistor and capacitor values. You can also plot the impulse response before selecting it for channel model.

## Select Clock Recovery Model

To successfully send and receive data between a SerDes transmitter and receiver, you need to ensure that the receiver samples the voltage waveform at the correct time instants to recover the message. Typically, the transmitter and receiver have independent oscillators which set the operating

frequency of the system within a specified tolerance of each other. But the receiver still needs to recover the clock phase and frequency from the waveform signal itself. A clock and data recovery function is accomplished by utilizing a phase detector algorithm and filtering the phase error information to recover the signal phase and frequency.

You can select which phase detector model the app uses in the clock recovery: bang-bang (Alexander) or baud-rate type-A (Mueller-Muller) using the CDR or DFECDR block.

The bang-bang phase detector is typically found in analog-based SerDes systems. In these systems, an edge of the eye sampler and a data sampler at the middle of the eye are used together to determine if the data sample is early or late. A sampler circuit that can determine if the input voltage is greater than or less than a voltage threshold.

The baud-rate type-A phase detector is found in ADC-based SerDes systems. In these systems, the voltage sample, determined with several bits of precision, and the resolved symbol is used to determine if the sample is early or late.

## **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 a license for Simulink Coder or Embedded Coder, you can keep your C files during .dll file generation. Otherwise, your C files will be deleted during the .dll file generation.
  - To learn about compatibility issues on different Linux platforms, see “Linux Version Compatibilities”.
- 

## **Version History**

**Introduced in R2019a**

### **See Also**

#### **Blocks**

FFE | CTLE | DFECDR | CDR | AGC | VGA | SaturatingAmplifier | PassThrough

#### **Objects**

serdes.FFE | serdes.CTLE | serdes.DFECDR | serdes.CDR | serdes.AGC | serdes.VGA | serdes.SaturatingAmplifier | serdes.PassThrough

#### **Topics**

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

#### **External Websites**

Supported and Compatible Compilers

# S-Parameter Fitter

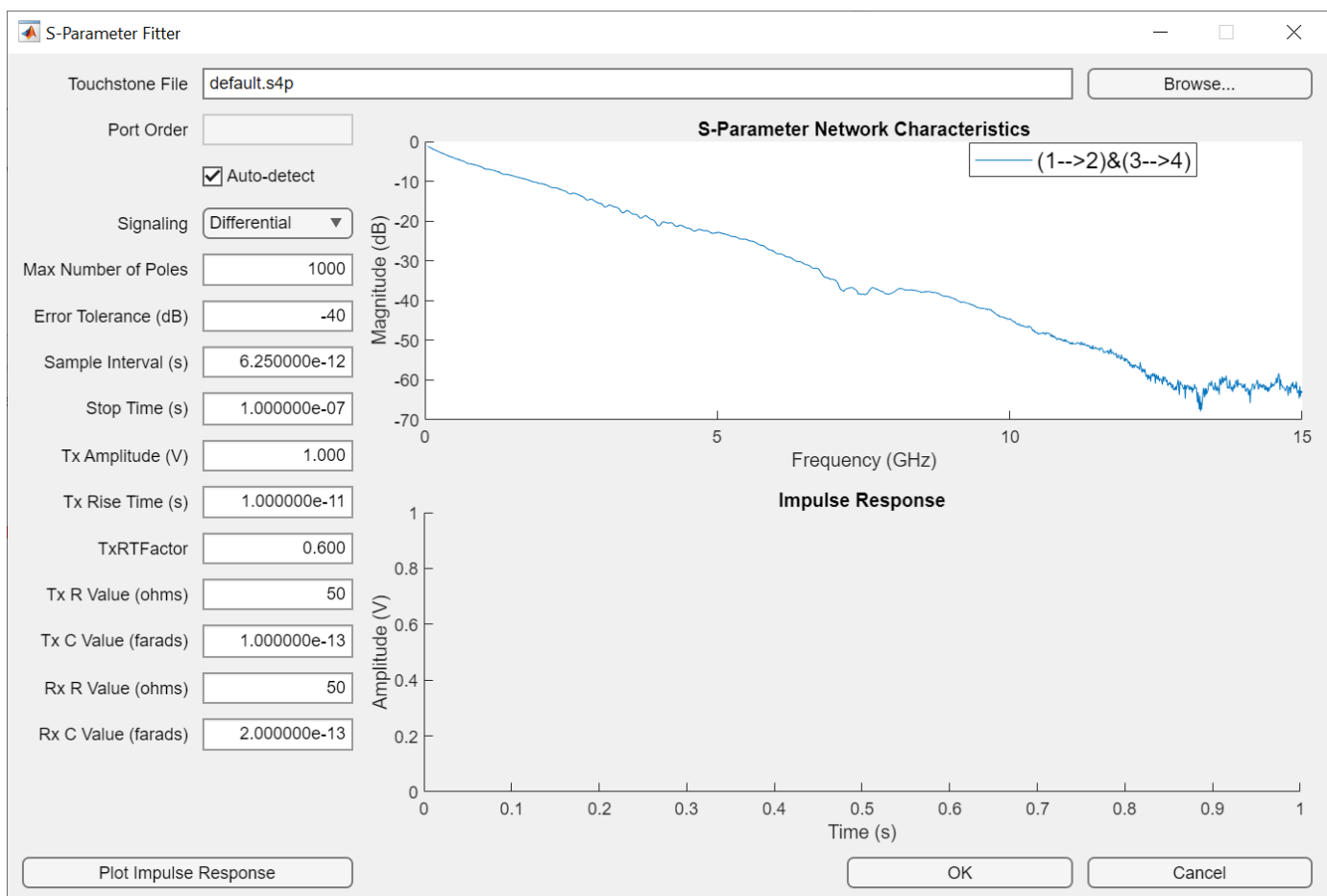
Convert S-Parameter network to impulse response

## Description

The **S-Parameter Fitter** app creates a `SParameterChannel` object which combines a scattering parameter (S-Parameter) model of a baseband communication channel with an analog transmitter and receiver into a single S-Parameter, with adjustable port ordering. There must be at least 4 ports. The app then calculates an impulse response of the S-Parameter using the `rational` function from the RF Toolbox.

Using the app, you can:

- Set the maximum number of poles, error tolerance, transmitter amplitude, rise time and RTF factor, and the resistor and capacitor values.
- Plot the impulse response before selecting it for channel model.



## Open the S-Parameter Fitter App

- MATLAB command prompt: Enter `sParameterFitter`.
- **SerDes Designer** app: Open the Analog Channel block parameters dialog box and click the **Import S-Parameter** button.
- Simulink model: Open the Analog Channel block parameters dialog box and click the **Import S-Parameter Touch File...** button.

## Examples

- “Convert Scattering Parameter to Impulse Response for SerDes System”

## Programmatic Use

`sParameterFitter` opens a new session of the **S-Parameter Fitter** app.

## Version History

Introduced in R2021b

## See Also

### Apps

**SerDes Designer**

### Functions

`SParameterChannel` | `rational`

### Topics

“Convert Scattering Parameter to Impulse Response for SerDes System”



# Class

---

# SParameterChannel

Convert S-parameter to impulse response

## Description

The `SParameterChannel` object combines the scattering parameter (S-parameter) model of a baseband communication channel with an analog transmitter and receiver into a single S-parameter with adjustable port ordering. The object then calculates the impulse response of the S-Parameter using the `rational` function. The `SParameterChannel` object supports `.sNp` Touchstone files, where  $N$  represents the number of ports.

## Creation

### Syntax

```
obj = SParameterChannel(filename)
```

### Description

`obj = SParameterChannel(filename)` creates an S-parameter channel object `obj` by importing data from the Touchstone file specified by `filename`.

### Input Arguments

#### **filename** — Touchstone data file

character vector | string scalar

Touchstone 1.0 data file that contains network parameter data, specified as a character vector or string scalar. `filename` can be the name of a file on the MATLAB path or the full path to a file.

---

**Note** The data file must be formatted as a Touchstone 1.0 file. Touchstone 2.0 is not supported.

---

## Properties

#### **FileName** — Name of Touchstone data

`default.s4p` (default) | character vector | string scalar

The name of the Touchstone data file from where you want to import the S-parameter data, specified as a character vector or string scalar.

Data Types: `char` | `string`

#### **PortOrder** — Port order for S-parameter

`[]` (default) | character vector

Port order for the S-parameter, specified as a character vector. If left blank, the function automatically detects the through paths and the re-orders the ports in the order `[in1 out1 in2 out2]`.

Example: [1 2 3 4]

Data Types: char

**MaxNumberOfPoles — Maximum number of poles to use to fit rational model**

1000 (default) | positive scalar integer

Maximum number of poles to use to fit a rational model, specified as a positive scalar integer.

Data Types: double

**ErrorTolerance — Desired error tolerance for rational model fit**

-40 (default) | real scalar

Desired error tolerance for rational model fit, specified as a real scalar in dB.

Data Types: double

**SampleInterval — Sample interval of impulse response**

6.2500e-12 (default) | positive real scalar

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

Data Types: double

**StopTime — Desired duration of impulse response**

1.0000e-07 (default) | positive real scalar

Desired duration of the impulse response, specified as a positive real scalar in seconds. The total number of samples in the impulse response is calculated by dividing the StopTime property by SampleInterval property.

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 — Capacitance of transmitter analog model**

1.0000e-13 (default) | nonnegative real scalar

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

Data Types: double

**TxAmplitude — Amplitude of stimulus input**

1 (default) | real scalar

Amplitude of the stimulus input to the transmitter analog model, specified as a real scalar.

Data Types: double

**TxRiseTime — 20%-80% rise time of stimulus input**

1.0000e-11 (default) | positive real scalar

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

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**

2.0000e-13 (default) | nonnegative real scalar

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

Data Types: double

#### **PortOrderUsed — Port order used for S-parameter channel analysis**

vector

This property is read-only.

Port order used for the S-parameter channel analysis, returned as a vector. If you do not define the port order using `PortOrder` property, the through paths are automatically detected and reported here.

Data Types: double

#### **ImpulseResponse — System impulse response**

vector

This property is read-only.

Impulse response of the system defined by the S-parameter channel object, returned as a vector.

Data Types: double

#### **t — Time vector**

vector

This property is read-only.

Time vector corresponding to the system impulse responses, returned as a vector.

Data Types: double

#### **f — S-parameter frequencies**

positive real valued vector

This property is read-only.

S-parameter frequencies, specified as a positive real-valued vector. The frequencies are sorted from the smallest to the largest value.

Data Types: double

**TransferFunction — System transfer function**

vector

This property is read-only.

Transfer function of the system defined by the S-parameter channel object, returned as a vector.

Data Types: double

**RationalResults — Output statistics**

object

This property is read-only.

Output statistics from `rational` (RF Toolbox), returned as an object.

Data Types: double

**RiseTimeUncertainty — Uncertainty due to rise time discretization**

negative real scalar

This property is read-only.

Uncertainty due to rise time discretization, returned as a negative real scalar.

Data Types: double

**Examples****Impulse Response from S-Parameter**

Create an S-parameter channel object.

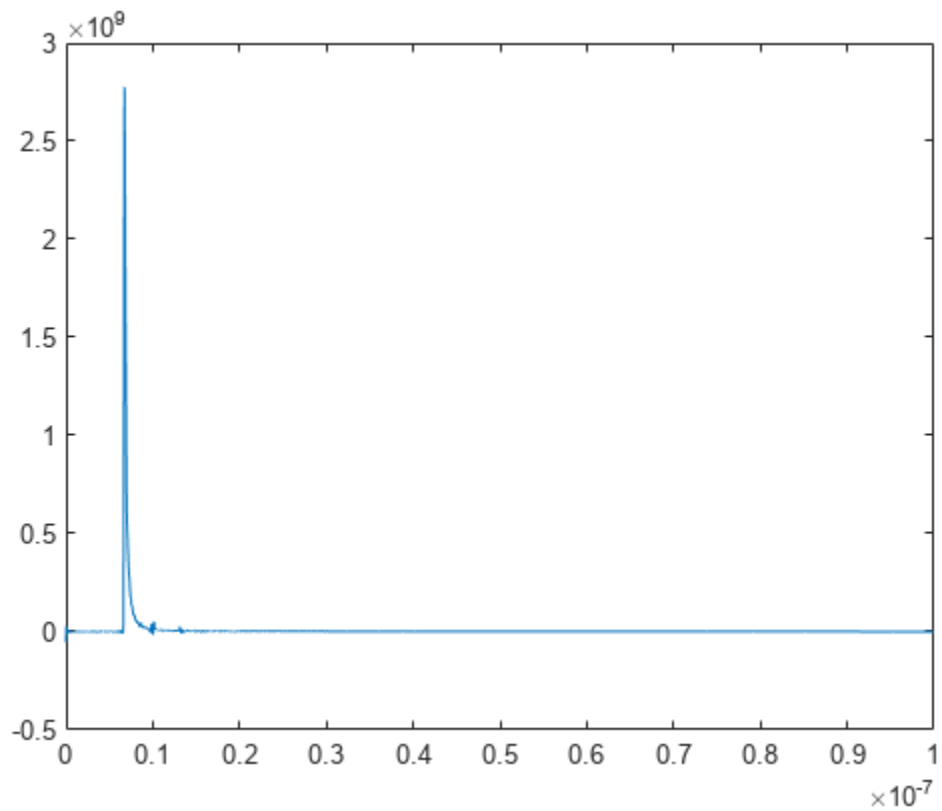
```
obj = SParameterChannel('FileName','default.s4p');
```

Extract the time vectors and impulse responses.

```
t = obj.t;  
impulse = obj.ImpulseResponse;
```

Plot the impulse response.

```
plot(t,impulse);
```



## Version History

Introduced in R2021a

## See Also

rational | Analog Channel | serdes.ChannelLoss

## Topics

“Convert Scattering Parameter to Impulse Response for SerDes System”