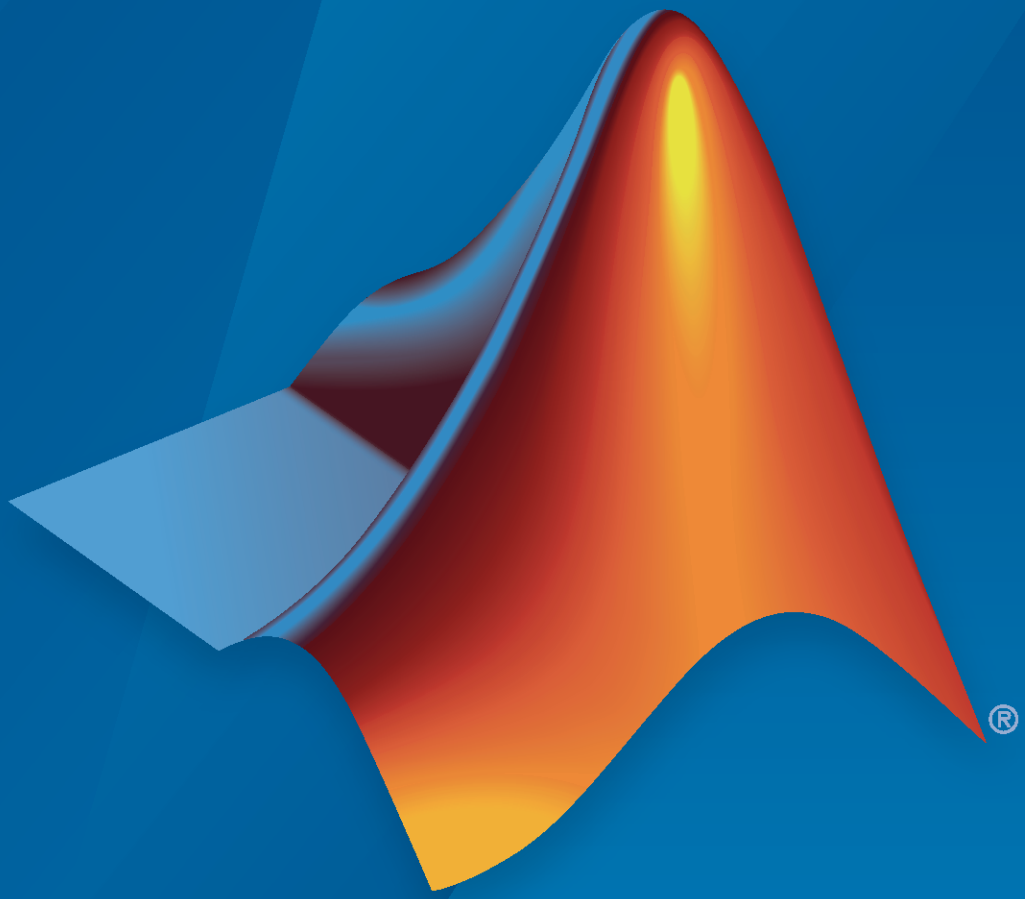
SerDes Toolbox™

User's Guide



MATLAB® & SIMULINK®

R2020a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

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

Phone: 508-647-7000

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

SerDes Toolbox™ User's Guide

Revision History

March 2019	Online only	New for Version 1.0 (Release 2019a)
September 2019	Online only	Revised for Version 1.1 (Release 2019b)
March 2020	Online only	Revised for Version 1.2 (Release 2020a)

Contents

1 Design and Simulate SerDes System Topics

2 Customize SerDes Systems Topics

3 Create and Customize IBIS-AMI Models Topics

4 Design and Simulate SerDes Systems Examples

5 Customize SerDes Systems

6 Customize IBIS-AMI Models

7 Industry Standard IBIS-AMI Models

Design and Simulate SerDes System Topics

Fundamentals of SerDes Systems

Modern high-speed electronic systems are characterized by increased data speed integrated circuits (ICs). The input/output performance remains the bottleneck that limits the overall performance of a high-speed system. Serial data transfer is the most efficient way of communicating large data quickly between computer chips on printed circuit boards through copper cables and through short, medium, and long length fiber optics.

Thus, many systems now aggregate and serialize multiple input/ output (I/O) signals for transmission across fiber and copper cables and PCBs at a higher data rate, recovering and de-serializing the individual signals on the receiving end. These SerDes (Serializer/De-Serializer) implementations employ additional silicon real estate to perform sophisticated equalization required for reliable signal transmission at very high data speeds. This approach helps maximize throughput at the system level.

SerDes design is a complex, iterative process that typically starts with a baseline SerDes system that demonstrates the feasibility of a design approach. This system also establishes budgets for the different parts of the serial channel and associated transmitter (TX) and receiver (RX) equalization circuitry. The data that describes the desired behavior of each of the equalization filters in both the transmitter and the receiver is then back-annotated in the behavioral models with the correlation with simulations or measurements. The final step is to implement the training algorithms and control loops that will be executed by the chip during startup and from time to time when the channel needs to be retrained.

The SerDes system is then compiled into IBIS-AMI (Input/Output Buffer Information Specifications — Algorithmic Model Interface) models.

There are six sections of a SerDes system:

- TX equalization — This becomes the IBIS-AMI dll for the transmitter.
- TX AnalogOut — This becomes the analog model of the transmitter. It is part of the IBIS model for TX, and is typically represented by the I-V and V-T characteristics curves in the `.ibs` file.
- Channel — This becomes the model of the physical channel, including the TX and RX package models.
- RX AnalogOut — This becomes the analog model of the receiver. It is part of the IBIS model for RX, and is typically represented by the I-V and V-T characteristics curves in the `.ibs` file.
- RX equalization — This becomes the IBIS-AMI dll for the receiver.
- Training algorithms and control loops — These become the on-chip microcode that is executed inside of the chip during startup and when the channel needs to be retrained.

See Also

More About

- “Design SerDes System and Export IBIS-AMI Model”

Clock and Data Recovery in SerDes System

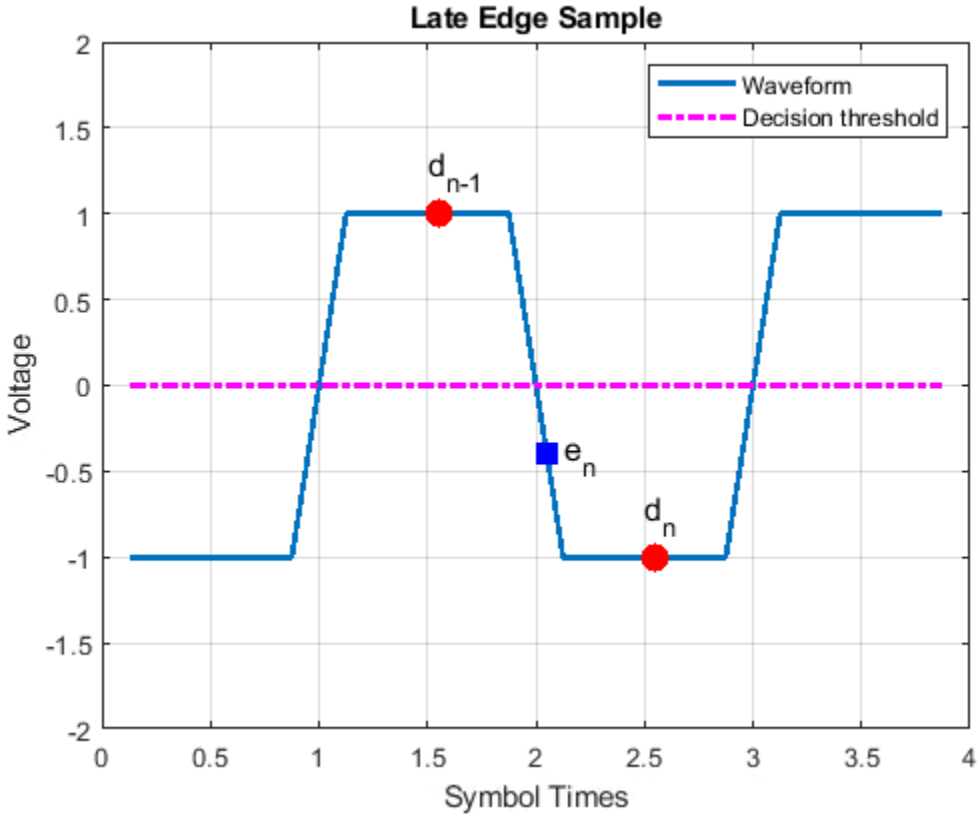
In this section...

High-speed analog SerDes systems use clock and data recovery (CDR) circuitry to extract the proper time to correctly sample the incoming waveform. The CDR circuitry creates a clock signal that is aligned to the phase and to some extent the frequency of the transmitted signal. Phase tracking (first order CDR) is usually accomplished by using a nonlinear bang-bang or Alexander phase detector that drives a voltage-controlled oscillator (VCO). Frequency tracking (second order CDR) integrates any remaining phase errors and compensates for gross differences between the transmitter reference clock and the receiver reference clock. `serdes.CDR` and `serdes.DFECDR` use the first-order CDR algorithm.

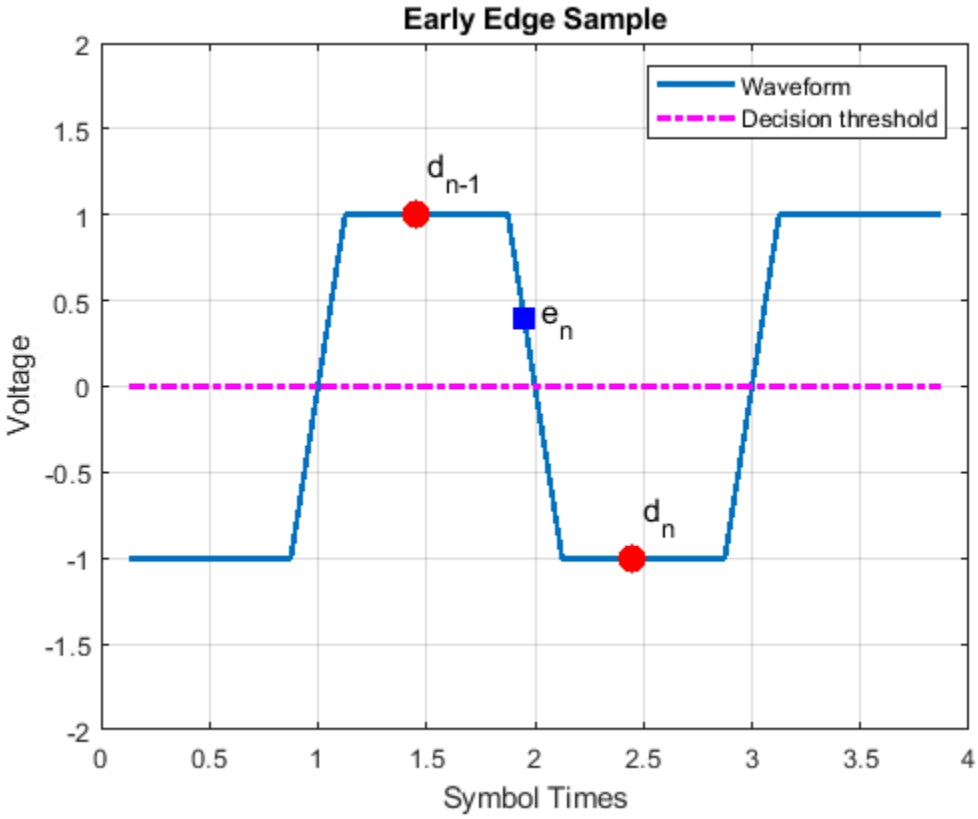
Phase Detector

The Alexander or bang-bang phase detector samples the received waveform at the edge and middle of each symbol. The edge sample (e_n) and data samples (d_{n-1} and d_n) are processed with some digital logic to determine if the edge sample, and thus the clock phase, is early or late. The edge sample, e_n, and data sample, d_n, are separated by half of a symbol time.

Consider the waveform where a data transition has occurred, and both e_n and d_n are below the decision threshold voltage. The binary values resolved from e_n and d_n match, which indicates the clock phase is late.



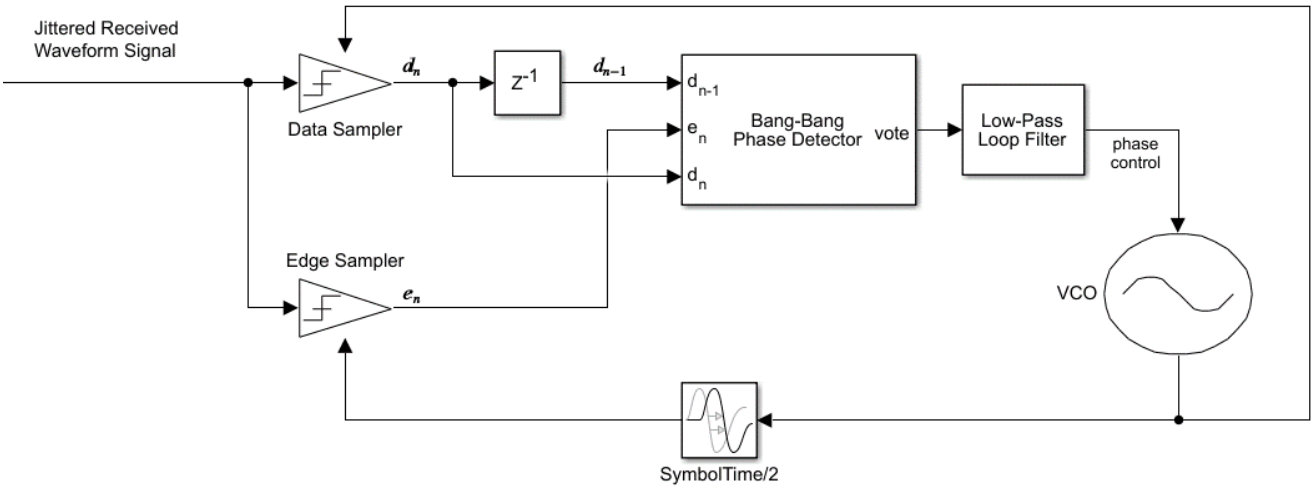
Similarly, when the binary values resolved from e_n and d_{n-1} match, the clock phase is early.



Representing the binary output of the sampler by ±1, the behavior of the phase detector is summarized here.

d_{n-1}	e_n	d_n	Action
-1	-1	1	Clock phase is early. Shift phase to the right.
1	1	-1	
-1	1	1	Clock phase is late. Shift phase to the left.
1	-1	-1	
-1	X	-1	No action is necessary.
1	X	1	

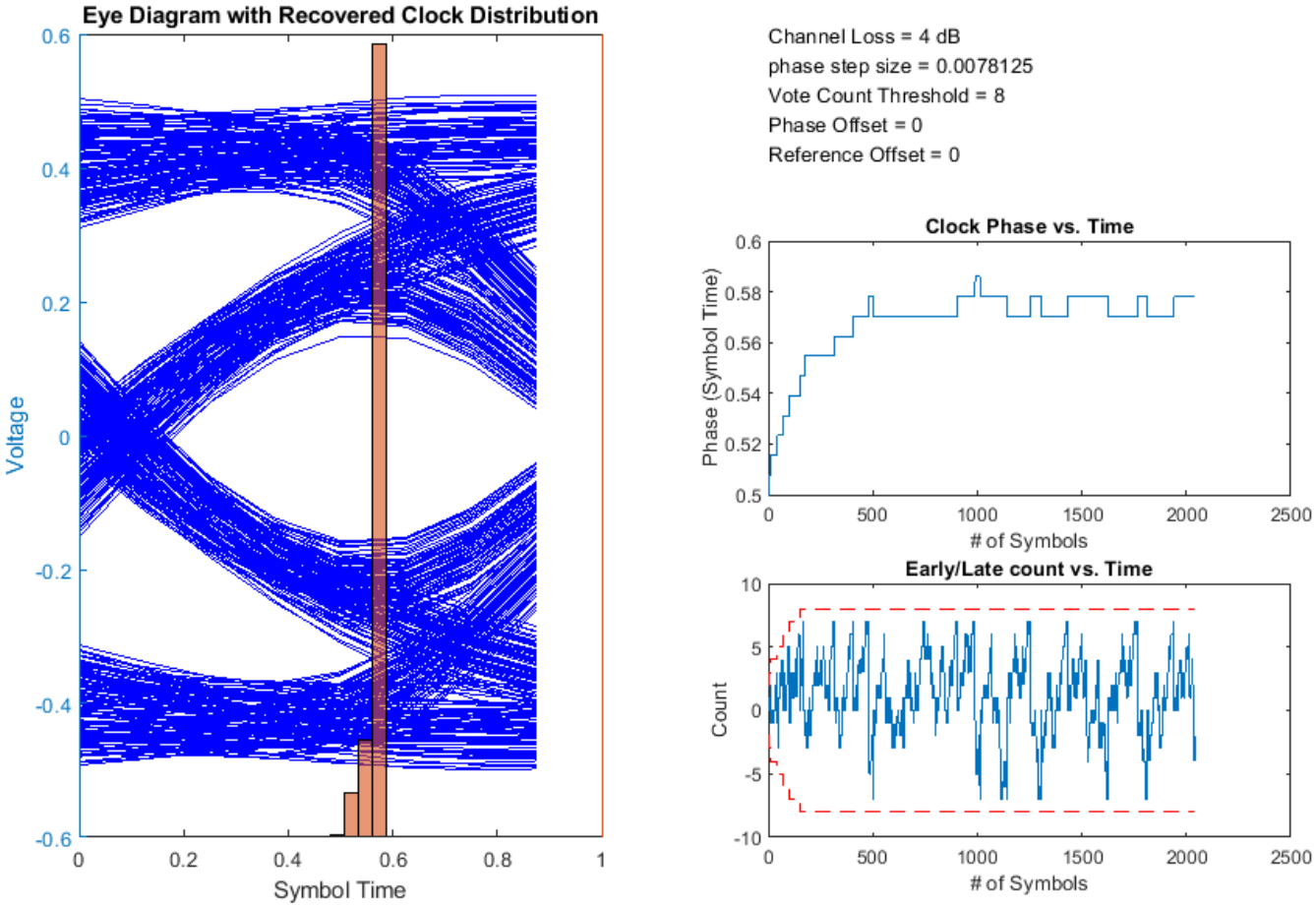
Driving the VCO directly from the phase detector output results in excessive clock jitter. To eliminate the jitter, the output of the phase detector is lowpass filtered by accumulating it in a vote. When the accumulated vote exceeds a specific count threshold, the phase of the VCO is incremented or decremented.



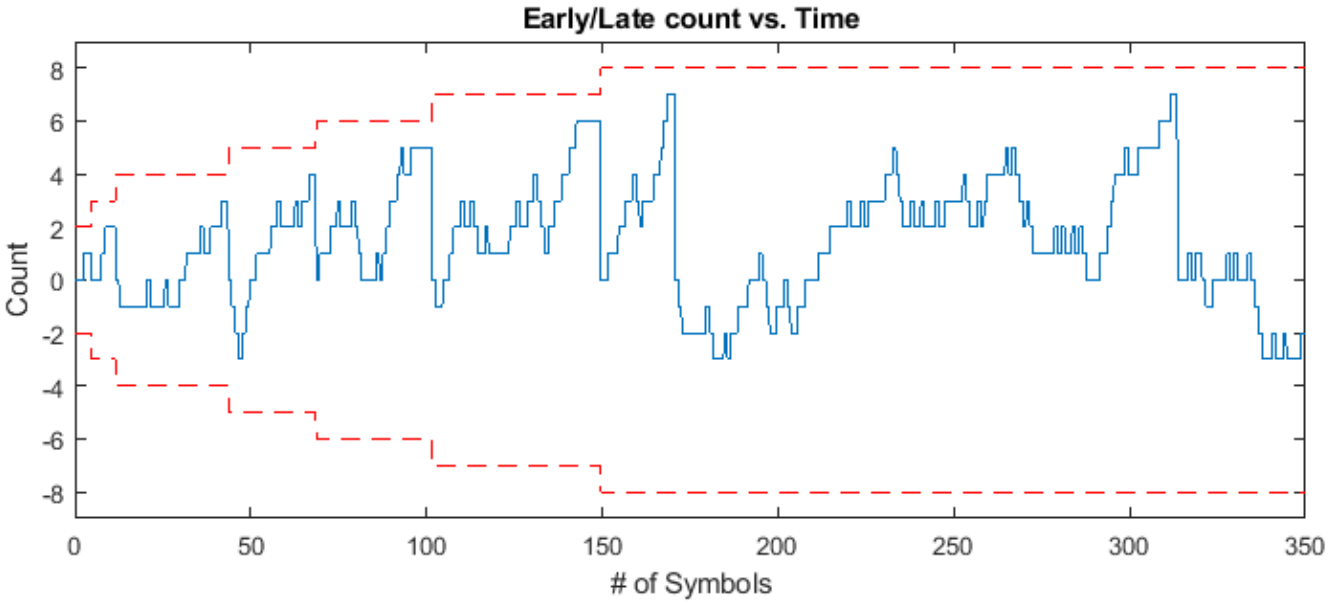
Recovering Clock Signal

Recover the clock signal from a repeating pseudorandom binary sequence (PRBS9) nonreturn to zero (NRZ) signal. Consider the channel has 4 dB loss, the phase step size is $\frac{1}{128}$, the vote count threshold is 8, and that there are no phase or reference offsets.

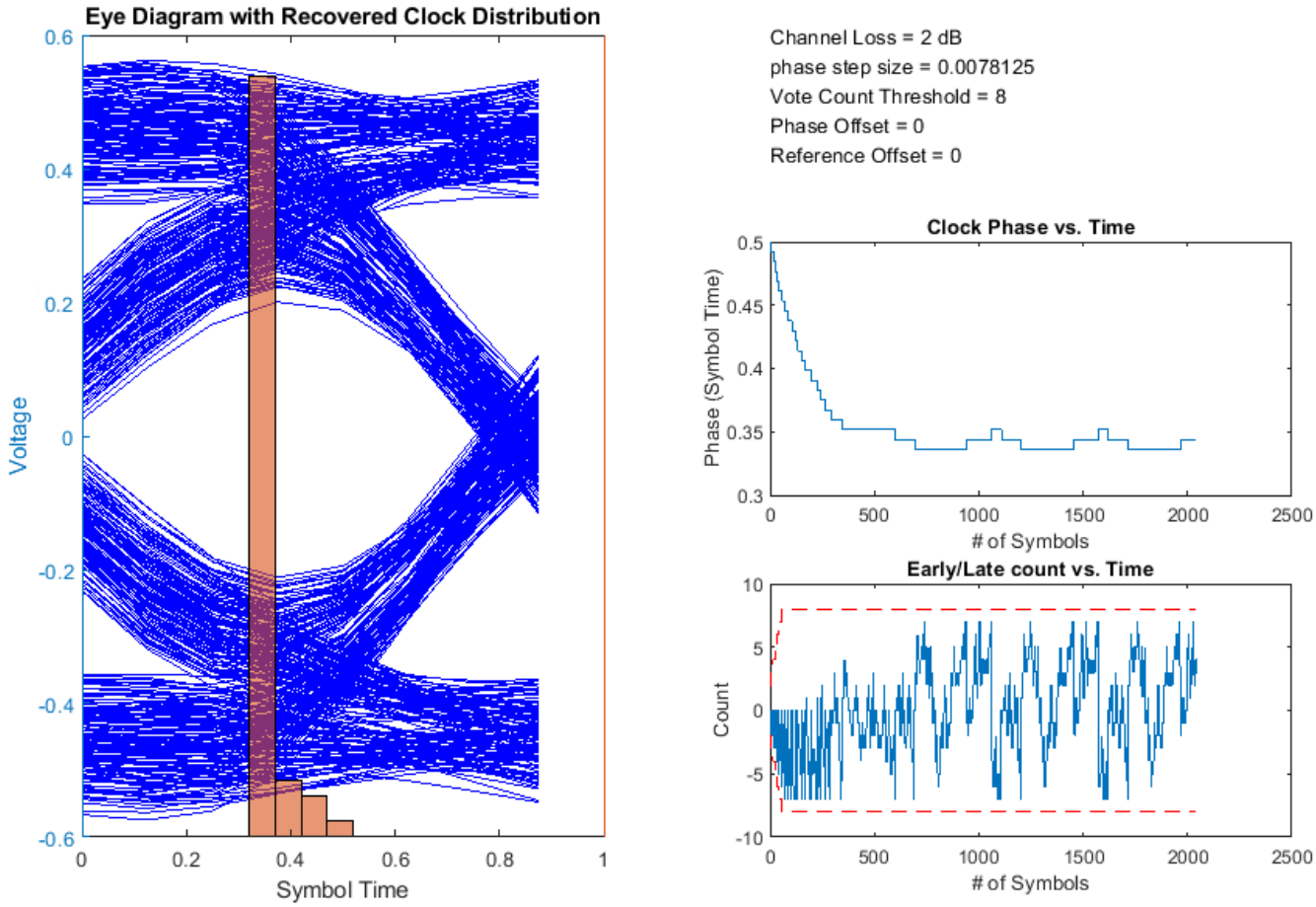
The baseline behavior is shown with the eye diagram and the resulting clock probability distribution function (PDF). The PDF is very near the center of the eye. The clock phase settles between a value of 0.5703 symbol time and 0.5781 symbol time. The dithering between the two values is a consequence of the nonlinear bang-bang phase detector and is the source of CDR hunting jitter. To reduce the magnitude of dithering, reduce the phase step size. To reduce the period of dithering, reduce the vote count threshold.



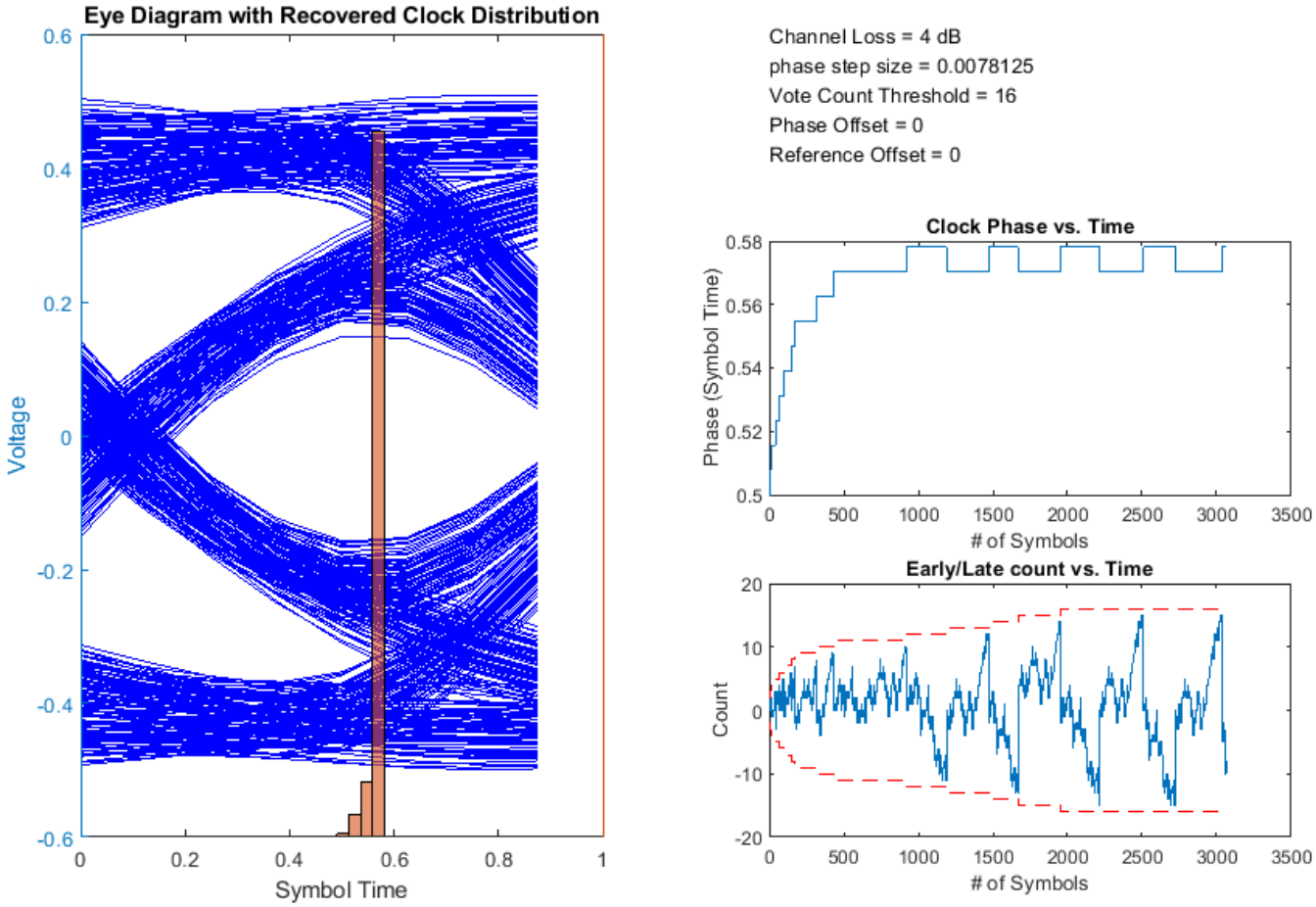
The output of the phase detector is accumulated in the early/late vote count. When the count exceeds the vote count threshold, the phase is incremented or decremented. To accelerate CDR convergence, the count threshold starts at 2, and each time the magnitude of the vote exceeds the threshold, the threshold is incremented until it reaches the maximum count. This figure shows the first 350 symbols of the early/late count (blue) and the threshold (dashed red line). Internal to the CDR block, the vote is incremented or decremented, checked against the threshold and then reset if necessary. The external vote value shown in figure below does not touch the threshold but is evident when the vote is reset to 0.



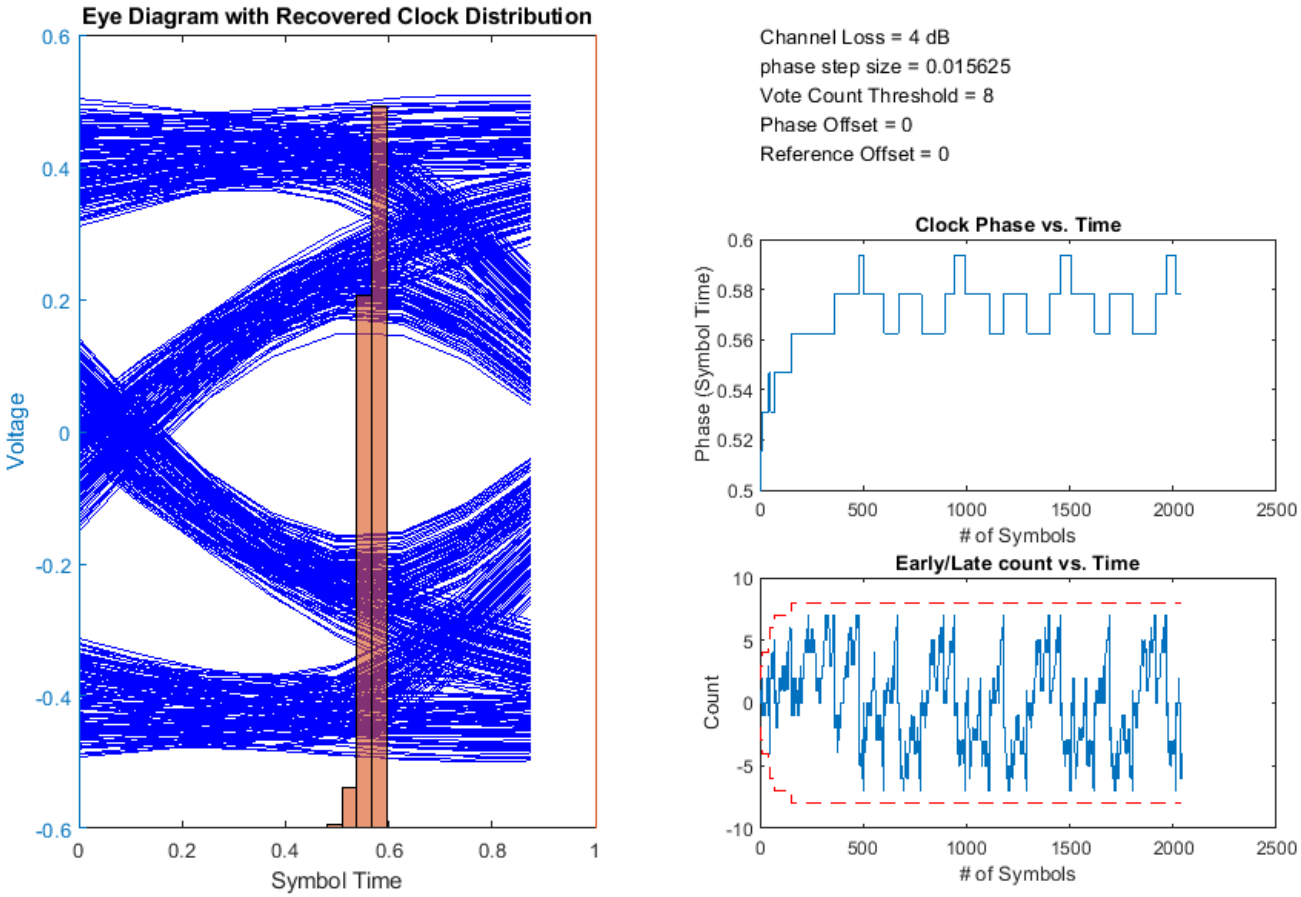
To show the clock converging to a different phase, change the channel loss to 2 dB. The clock phase now adapts to around 0.35 symbol time.



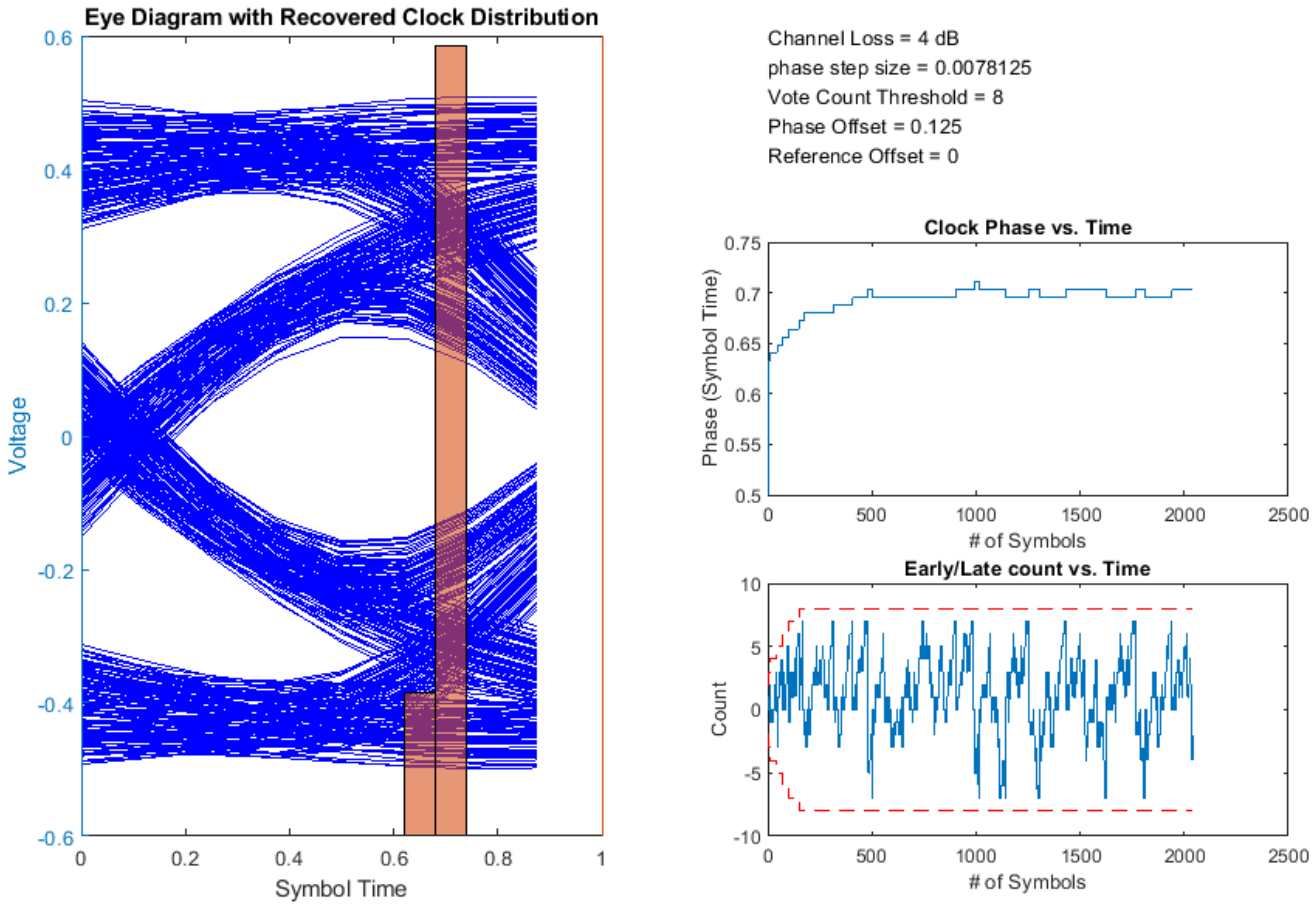
Increasing the vote count threshold to 16 results in a larger dithering period.



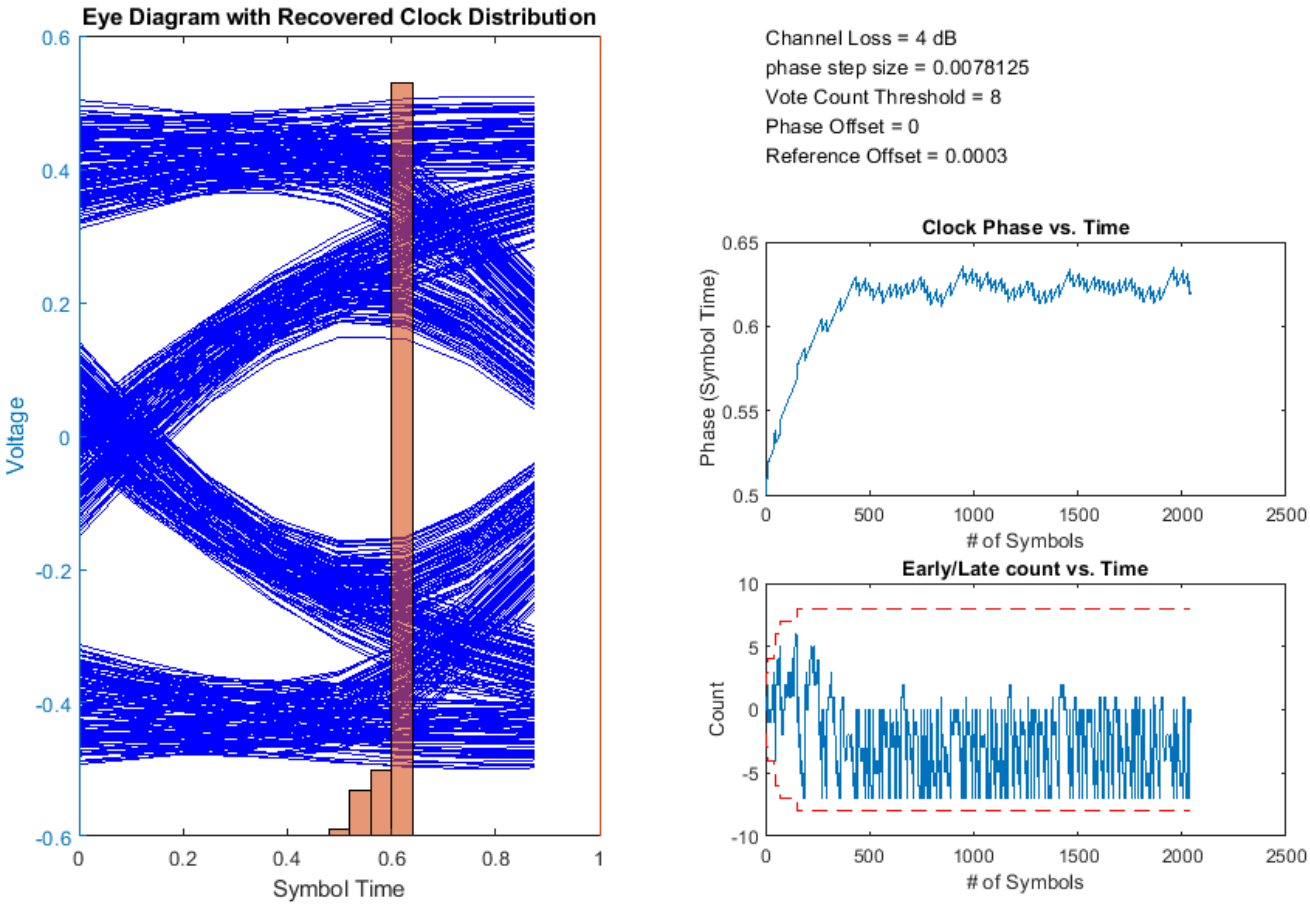
Increasing the phase step size to $\frac{1}{64}$ increases the dithering magnitude.



Manually shifting the data sampler location when the equalized eye does not display left/right symmetry can maximize the eye height. For example, shift the clock phase to the right by $\frac{1}{8}$ of a symbol time to shift the output clock phase from 0.57 symbol time to 0.7 symbol time.



You can also inject a small amount of reference clock frequency offset impairment to implement a more realistic CDR.



References

[1] Sonntag, J. L. and Stonick, J. "A Digital Clock and Data Recovery Architecture for Multi-Gigabit/s Binary Links." *IEEE Journal of Solid-State Circuits*, 2006.

[2] Razavi, B. "Challenges in the design high-speed clock and data recovery circuits." *IEEE Communications Magazine*, 2002.

See Also

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

Analog Channel Loss in SerDes System

In this section...

Limiting factors in high-speed data transmission includes cross talk, attenuation, and reflection noise. The Analog Channel block and `serdes.ChannelLoss` System object™ parameterize a channel model that represents a lossy transmission line typical in high-speed SerDes application. The loss model is constructed either from a parameterized channel loss model or from an impulse response from another source.

Loss Model from Channel Loss Metric

A discrete time, band-limited analog impulse response characterizes the `serdes.ChannelLoss` System object. It represents the response of a system to an impulse response vector with an impulse magnitude of $\frac{1}{dt}$, where dt is the sample interval.

To calculate the impulse response, `serdes.ChannelLoss` first calculates the S-parameter component S21 according to channel loss at frequencies ranging from 0 to $f_{\max}$, maximum frequency of interest, where $f_{\max} = \frac{1}{dt}$. This is done by determining the loss at the target frequency, and then linearly extrapolating required channel length to achieve target channel loss. Then transmitter and receiver termination S-parameter are then calculated according to the equations 93A-17 and 93A-18 from the IEEE 802.3bj-2014 specifications [1].

After calculating S21, the System object adds the negative frequency data points based on the expected even symmetry of the real components of S21 and the odd symmetry of the imaginary components of S21 of the frequency response. The impulse response is calculated from the inverse Fourier transform of S21. Finally, the impulse response is resampled so that the sample interval is dt.

Loss Model from Impulse Response

To construct a loss model from an impulse response vector, input the impulse response vector from another source. You can also define the impulse sample interval. Changing the symbol time and number of samples per symbol changes the data rate of the SerDes system.

Introducing Cross Talk

You can include crosstalk in your simulation from the **SerDes Designer** app, or using the Analog Channel block in Simulink®. If the parameterized channel loss model is used, you can specify the strength of the near and far end crosstalk aggressors according to specification standards or you can specify your own custom integrated crosstalk noise (ICN) levels. If a custom impulse response is used, then up to 6 additional columns can be used to represent the crosstalk impulse response. For more information, see Analog Channel and `serdes.ChannelLoss`.

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.

See Also

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

Manage IBIS-AMI Parameters

You can manage the IBIS-AMI parameters by opening the SerDes IBIS-AMI Manager dialog box from the Configuration block.

Contents of IBIS File

The **IBIS** tab in the SerDes IBIS-AMI Manager dialog box defines the content of the IBIS file. Set the parameters used to define the IBIS file in the AnalogOut and AnalogIn blocks in the **SerDes Designer** app and in the **IBIS** tab in the SerDes IBIS-AMI Manager.

From the transmitter side in the AnalogOut block:

- **Voltage (V)** — Typical value of voltage range in the IBS file.
- **R (Ohms)** — Slope of the typical pull-up and pull-down IV curves in the IBS file.
- **C (pF)** — Typical value of the C_comp in the IBS file.

From the receiver side in the AnalogIn block:

- **Voltage (V)** — Typical value of voltage range in the IBS file.
- **R (Ohms)** — Slope of the typical ground clamp IV curve in the IBS file.
- **C (pF)** — Typical value of the C_comp in the IBS file.

You can only enter the typical values for these parameters. You can define the **Tx and Rx corner percentage** in the **Export** tab of the SerDes IBIS-AMI Manager dialog box. The minimum and maximum values are generated by subtracting or adding to the typical value its fractional corner percentage.

The performance of an input/output (I/O) buffer is a function of process, voltage, and temperature (PVT). There are 27 PVT corners. IBIS supports three model corners: Typ, Min, and Max. When generating the IBIS file, the **Voltage (V)**, **R (Ohms)**, and **C (pF)** values are used for the Typ corner.

- Min refers to the slow/weak corner. It groups slow process, low voltage, and high temperature. The voltage and resistance are decreased and the capacitance is increased for the Min corner.
- Max refers to the fast/strong corner. It groups fast process, high voltage, and low temperature. The voltage and resistance are increased and the capacitance is decreased for the Max corner.

You can also specify the IBIS-AMI model in the **Export** tab of the SerDes IBIS-AMI Manager dialog box as single I/O, redriver or retimer. Selecting these model configurations changes the contents of the IBIS file.

- If you select **I/O** as the model configuration, the IBIS model is reconfigured to a single model of ModelType I/O.
- If you select **Retimer** or **Redriver** as the model configuration, the components of the IBIS file is updated to include the repeater pins.

Contents of AMI File

The **AMI - Tx** and **AMI - Rx** tabs in the SerDes IBIS-AMI Manager dialog box define the content of the AMI file. They contain the required and commonly used reserved AMI parameters. You can also define the model-specific parameters for the relevant blocks.

There are five `Reserved_Parameters` included in every AMI file generated by the SerDes Toolbox:

- **AMI_Version** — IBIS version supported by the model
- **Init_Returns_Impulse** — whether the model supports statistical simulation or not
- **GetWave_Exists** — whether the model supports time-domain simulation or not.
- **Max_Init_Aggressors** — the number of crosstalk aggressors supported by the model
- **Modulation** — the modulation scheme of the model.

If you select **Retimer** or **Redriver** as the model configuration in the **Export** tab of the SerDes IBIS-AMI Manager dialog box, an additional `Reserved_Parameter` **Repeater_Type** is added to the **AMI - Rx** tab. This parameter specifies the type of the repeater.

You can also define and modify the parameters of individual transmitter and receiver blocks. From the `Model_Specific` parameters, you can add new AMI parameters to specific blocks. For more information, see “Managing AMI Parameters” on page 6-2.

You can also add a new tap structure to the equalizer blocks. These additional taps are included both in the Simulink model and the exported IBIS-AMI models. The taps enable you to adjust equalization, especially when you build your custom blocks from scratch.

You can also include standard-compliant transmitter and receiver jitter and noise parameters to the `Reserved_Parameter` section of the AMI file using the **Reserved Parameters...** button. The jitter and noise parameters are only used in EDA tools. Simulink ignores these parameters.

Debugging AMI Files in EDA

To enable debugging the AMI files in EDA tools, in the **AMI-Tx** or **AMI-Rx** tab, click the **Reserved Parameters...** button and select **DLL_ID** parameter. **DLL_ID** is a standard IBIS-AMI parameter that appears as a `Reserved_Parameter`. It also enables the **AMI_Debug** parameter as a `Model_Specific` parameter.

Set **Enable** value to `true` to output debug files. You can improve performance by setting **Enable** value to `false` and not output any debug files, but still have the option to turn on debugging in the EDA tools if necessary. Use **Start_Time** to define the simulation time at which debug output generation begins.

See Also

Configuration

More About

- “Managing AMI Parameters” on page 6-2

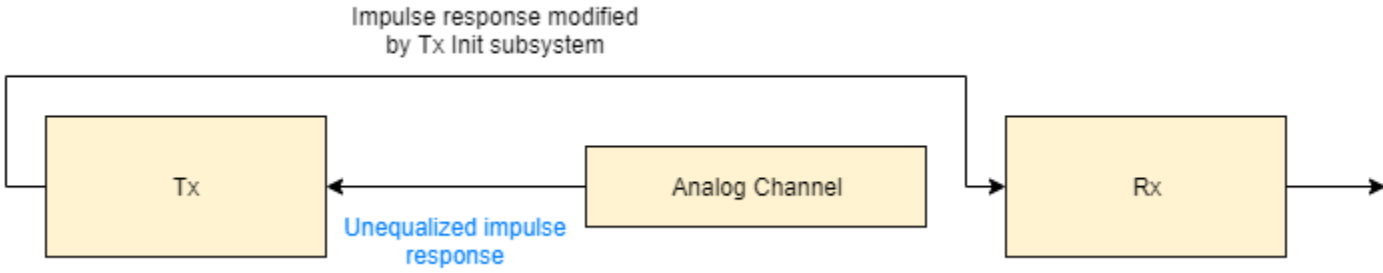
External Websites

- https://ibis.org

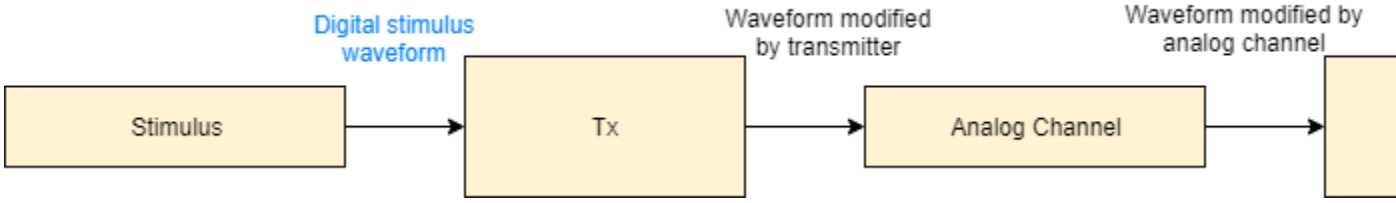
Statistical Analysis in SerDes Systems

A SerDes system simulation involves a transmitter (Tx) and a receiver (Rx) connected by a passive analog channel. There are two distinct phases to a SerDes system simulation: statistical analysis and time-domain analysis. Statistical analysis (also known as analytical, linear time-invariant, or Init analysis) is based on impulse responses enabling fast analysis and adaptation of equalization algorithms. Time-domain analysis (also known as empirical, bit-by-bit or GetWave analysis) is a waveform-based implementation of equalization algorithms that can optionally include nonlinear effects.

The reference flow of statistical analysis differs from time-domain analysis. During a statistical analysis simulation, an impulse response is generated. The impulse response represents the combined response of the transmitter's analog output, the channel, and the receiver's analog front end. The impulse response of the channel is modified by the transmitter model's statistical functions. The modified impulse response from the transmitter output is then further modified by the receiver model's statistical functions. The simulation is then completed using the final modified impulse response which represents the behavior of both AMI models combined with the analog channel.



During a time-domain simulation, a digital stimulus waveform is passed to the transmitter model's time-domain function. This modified time-domain waveform is then convolved with the analog channel impulse response used in the statistical simulation. The output of this convolution is then passed to the receiver model's time-domain function. The modified output of the receiver becomes the simulation waveform at the receiver latch.



In SerDes Toolbox, the Init subsystem within both the Tx and Rx blocks uses an Initialize Function Simulink block. The Initialize Function block contains a MATLAB® function to handle the statistical analysis of an impulse response vector. The impulse response vector is generated by the Analog Channel block.

The MATLAB code within the Init subsystems mimics the architecture of Simulink time-domain simulation by initializing and setting up the library blocks from the SerDes Toolbox that implement equalization algorithms. Each subsystem then processes the impulse response vector through one or more System objects representing the corresponding blocks.

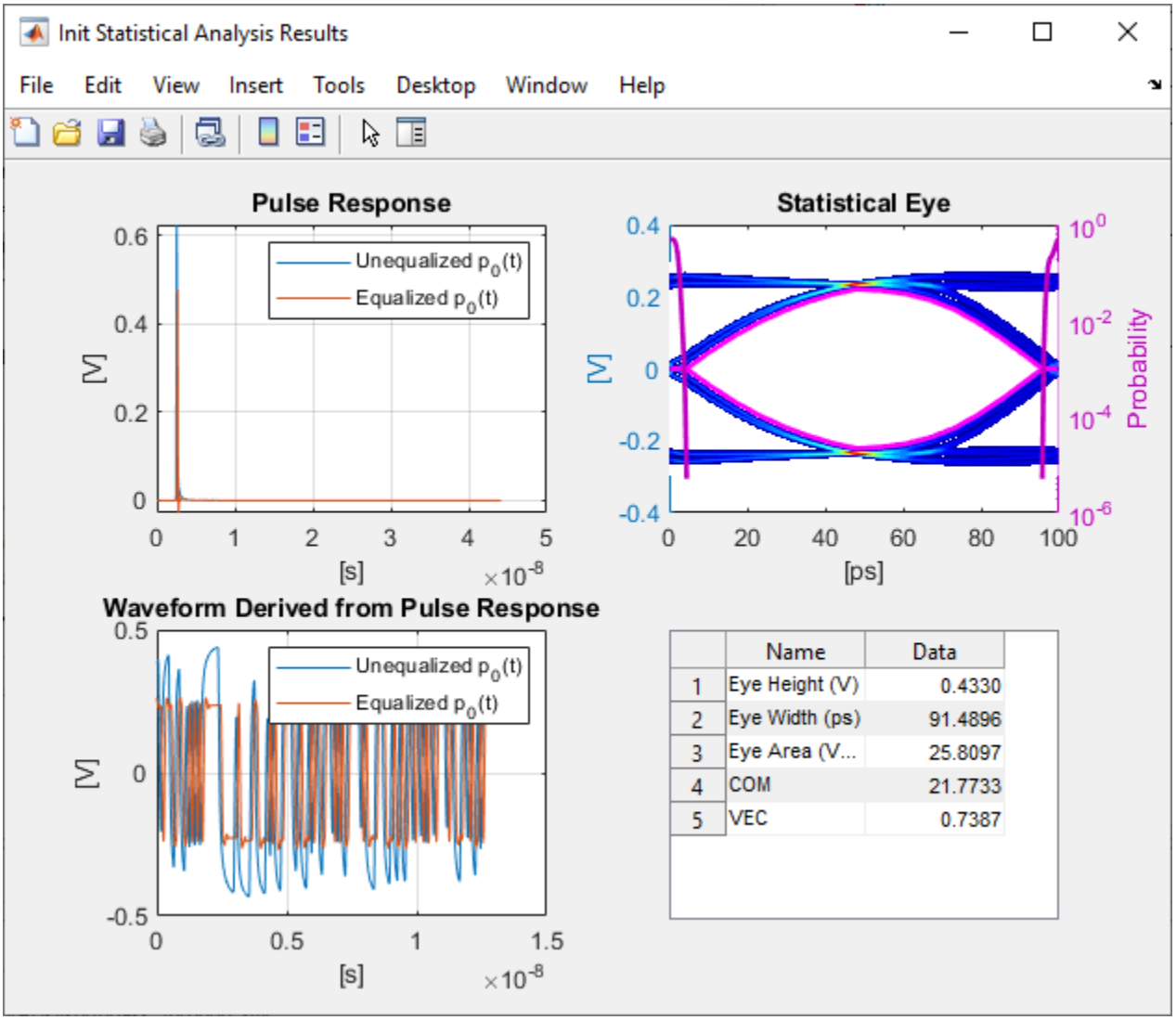
Additionally, an Init subsystem can adapt or optimize the equalization algorithms and then apply the modified algorithms to the impulse response. The output of an Init subsystem is an adapted impulse response. If the Init subsystem adapts the equalization algorithms, it can also output the modified equalization settings as AMI parameters. These modified equalization parameters can also be passed to the time-domain analysis as an optimal setting or to provide a starting point for faster time-domain adaptation.

Init Subsystem Workflow

In a Simulink model of a SerDes system, there are two Init subsystems, one on the transmitter side (Tx block) and one on the receiver side (Rx block). During statistical analysis, the impulse response of the analog channel is first equalized by the Init subsystem inside the Tx block based on the System object properties. The modified impulse response is then fed as an input to the Rx block. The Init system inside the Rx block further equalizes the impulse response and produces the final output.

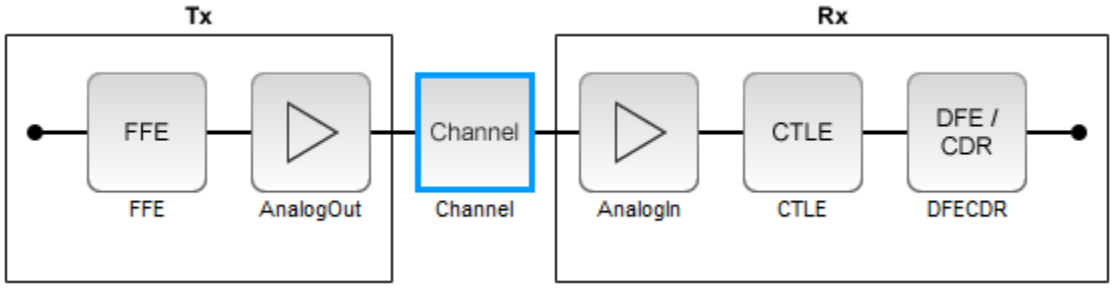
The System objects corresponding to the Tx and Rx blocks modify the impulse response in the same order as they were received. If there are multiple self-adapting System objects in a Tx or Rx block, each System object finds the best setting for the impulse response and modifies it before sending it to the next System object.

The final equalized impulse response is used to derive the pulse response, statistical eye, and the waveforms.



SerDes System Using Init Subsystem

To understand how an Init subsystem handles statistical analysis in a SerDes system, create a SerDes system using the **SerDes Designer** App. The SerDes system contains an FFE block on the Tx side and CTLE and DFECDR blocks on the Rx side. Use the default settings for each block.



Export the SerDes system to a Simulink model. In Simulink, double-click the Tx block to open the Init block. Then double-click the Init block to open the Block Parameters dialog box. Click the **Show Init** button to open the code pertaining to the Init function of the transmitter.

The Init function first reshapes the impulse response vector of the analog channel into a 2-D matrix. The first column in the 2-D matrix represents the analog channel impulse response (victim). The subsequent columns (if any are present) represent the crosstalk (aggressors).

```
%% Impulse response formatting
% Size ImpulseOut by setting it equal to ImpulseIn
ImpulseOut = ImpulseIn;
% Reshape ImpulseIn vector into a 2D matrix using RowSize and Aggressors called LocalImpulse
LocalImpulse = zeros(RowSize,Aggressors+1);
AggressorPosition = 1;
for RowPosition = 1:RowSize:RowSize*(Aggressors+1)
    LocalImpulse(:,AggressorPosition) = ImpulseIn(RowPosition:RowSize-1+RowPosition)';
    AggressorPosition = AggressorPosition+1;
end
```

Then the Init function initializes the System objects that represent the blocks on the Tx side and sets up the simulation and AMI parameters and the block properties. In this SerDes system, there is only one block on the Tx side, FFE.

```
%% Instantiate and setup system objects
% Create instance of serdes.FFE for FFE
FFEInit = serdes.FFE('WaveType', 'Impulse');
% Setup simulation parameters
FFEInit.SymbolTime = SymbolTime;
FFEInit.SampleInterval = SampleInterval;
% Setup FFE In and InOut AMI parameters
FFEInit.Mode = FFEParameter.Mode;
FFEInit.TapWeights = FFEParameter.TapWeights;
% Setup FFE block properties
FFEInit.Normalize = true;
```

The channel impulse response is then processed by the System object on the Tx side.

```
%% Impulse response processing via system objects
% Return impulse response for serdes.FFE instance
LocalImpulse = FFEInit(LocalImpulse);
```

The modified impulse response in 2-D matrix form is reshaped back into an impulse response vector and sent to the Rx side for further equalization.

```
%% Impulse response reformating
% Reshape LocalImpulse matrix into a vector using RowSize and Aggressors
ImpulseOut(1:RowSize*(Aggressors+1)) = LocalImpulse;
```

Similarly, if you look at the Rx Init code, you can see that the Rx Init function first reshapes the output of the Tx Init function into a 2-D matrix.

Then the Init function initializes the System objects that represent the blocks on the Rx side and sets up the simulation and AMI parameters and the block properties. In this case, there are two blocks on the Rx side, CTLE and DFECDR.

```
%% Instantiate and setup system objects
% Create instance of serdes.CTLE for CTLE
```

```
CTLEInit = serdes.CTLE('WaveType', 'Impulse');
% Setup simulation parameters
CTLEInit.SymbolTime = SymbolTime;
CTLEInit.SampleInterval = SampleInterval;
% Setup CTLE In and InOut AMI parameters
CTLEInit.Mode = CTLEParameter.Mode;
CTLEInit.ConfigSelect = CTLEParameter.ConfigSelect;
% Setup CTLE block properties
CTLEInit.Specification = 'DC Gain and Peaking Gain';
CTLEInit.DCGain = [0 -1 -2 -3 -4 -5 -6 -7 -8];
CTLEInit.ACGain = 0;
CTLEInit.PeakingGain = [0 1 2 3 4 5 6 7 8];
CTLEInit.PeakingFrequency = 5000000000;
CTLEInit.GPZ = [0 -23771428571 -10492857142 -13092857142;-1 -17603571428 -7914982142 -1334464285
-2 -17935714285 -6845464285 -13596428571;-3 -15321428571 -5574642857 -13848214285;...
-4 -15600000000 -4960100000 -14100000000;-5 -15878571428 -4435821428 -14351785714;...
-6 -16157142857 -3981285714 -14603571428;-7 -16435714285 -3581089285 -14855357142;...
-8 -16714285714 -3227142857 -15107142857];
% Create instance of serdes.DFECDR for DFECDR
DFECDRInit = serdes.DFECDR('WaveType', 'Impulse');
% Setup simulation parameters
DFECDRInit.SymbolTime = SymbolTime;
DFECDRInit.SampleInterval = SampleInterval;
DFECDRInit.Modulation = Modulation;
% Setup DFECDR In and InOut AMI parameters
DFECDRInit.ReferenceOffset = DFECDRParameter.ReferenceOffset;
DFECDRInit.PhaseOffset = DFECDRParameter.PhaseOffset;
DFECDRInit.Mode = DFECDRParameter.Mode;
DFECDRInit.TapWeights = DFECDRParameter.TapWeights;
% Setup DFECDR block properties
DFECDRInit.EqualizationGain = 9.6e-05;
DFECDRInit.EqualizationStep = 1e-06;
DFECDRInit.MinimumTap = -1;
DFECDRInit.MaximumTap = 1;
DFECDRInit.Count = 16;
DFECDRInit.ClockStep = 0.0078;
DFECDRInit.Sensitivity = 0;
```

The impulse response that was previously modified by the System objects on the Tx side is then further modified by the System objects on the Rx side.

```
%% Impulse response processing via system objects
% Return impulse response and any Out or InOut AMI parameters for serdes.CTLE instance
[LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
% Return impulse response and any Out or InOut AMI parameters for serdes.DFECDR instance
[LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
```

The final equalized impulse response in 2-D matrix form is reshaped back into an impulse response vector.

Each Init function also contains a section, Custom user code area, where you can customize your own code.

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)

% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

For more information on how you can use the Custom user code area, see “Customizing Datapath Building Blocks” on page 5-11 and “Implement Custom CTLE in SerDes Toolbox PassThrough Block” on page 5-20.

See Also

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

More About

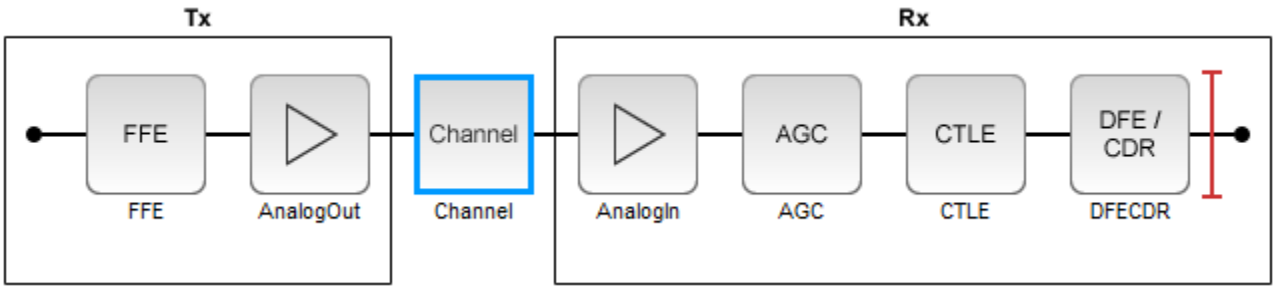
- “Customizing Datapath Building Blocks” on page 5-11
- “Implement Custom CTLE in SerDes Toolbox PassThrough Block” on page 5-20
- “Managing AMI Parameters” on page 6-2
- “Customizing SerDes Toolbox Datapath Control Signals” on page 5-2

Customize SerDes Systems Topics

Customize SerDes System in MATLAB

Open the **SerDes Designer** app. In the **CONFIGURATION** tab of the app toolstrip, set **Symbol Time (ps)** to 125 and **Target BER** to `1e-12`.

In a new blank canvas, add an FFE block to the **Tx** side. Add an AGC, a CTLE and a DFECDR block to the **Rx** side.



Select the channel block. Set **Channel loss (dB)** to 13.

From the **EXPORT** tab of the app toolstrip, select **Generate MATLAB code for SerDes System**. A MATLAB script open that represents the command line interface to the SerDes system.

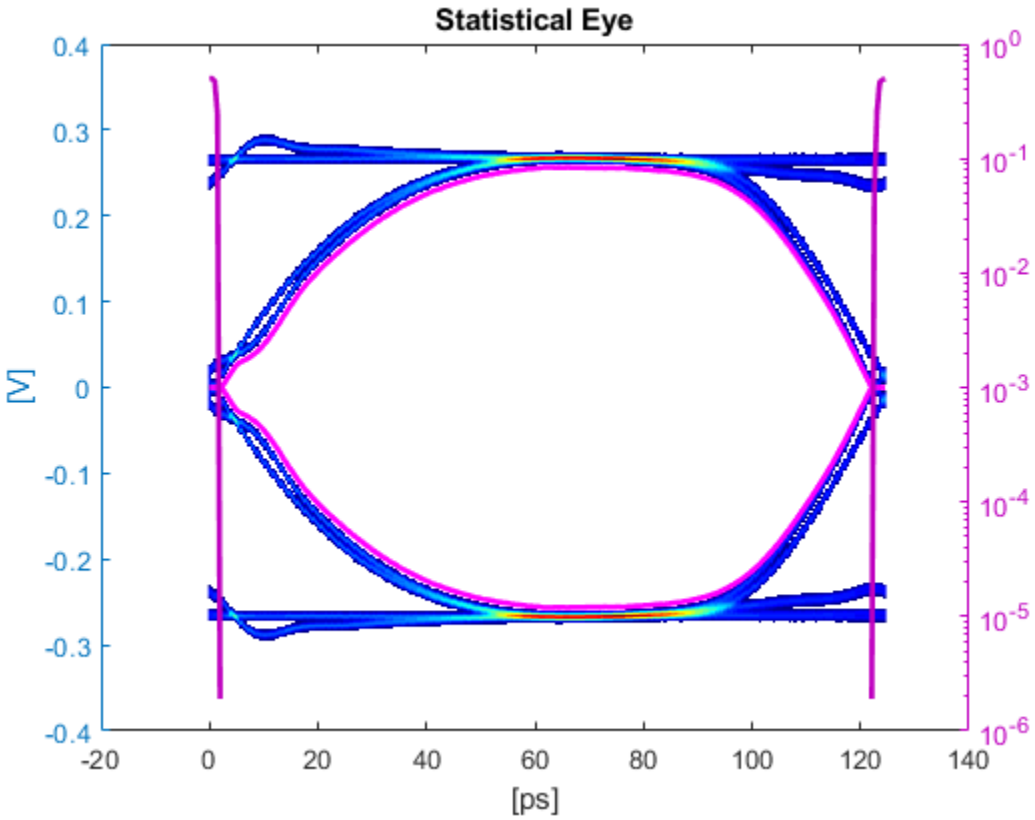
The MATLAB script contains the code to generate the transmitter and receiver building blocks and analog models. It also contains the channel information and SerDes system configuration. The script exposes every parameter that is part of the SerDes system. You can modify the parameters to further explore the SerDes system.

For example, to see the effect of **Channel loss** on the SerDes system, scroll down to the section of the MATLAB script that says `% Build ChannelData`. Replace the default code section with the following code:

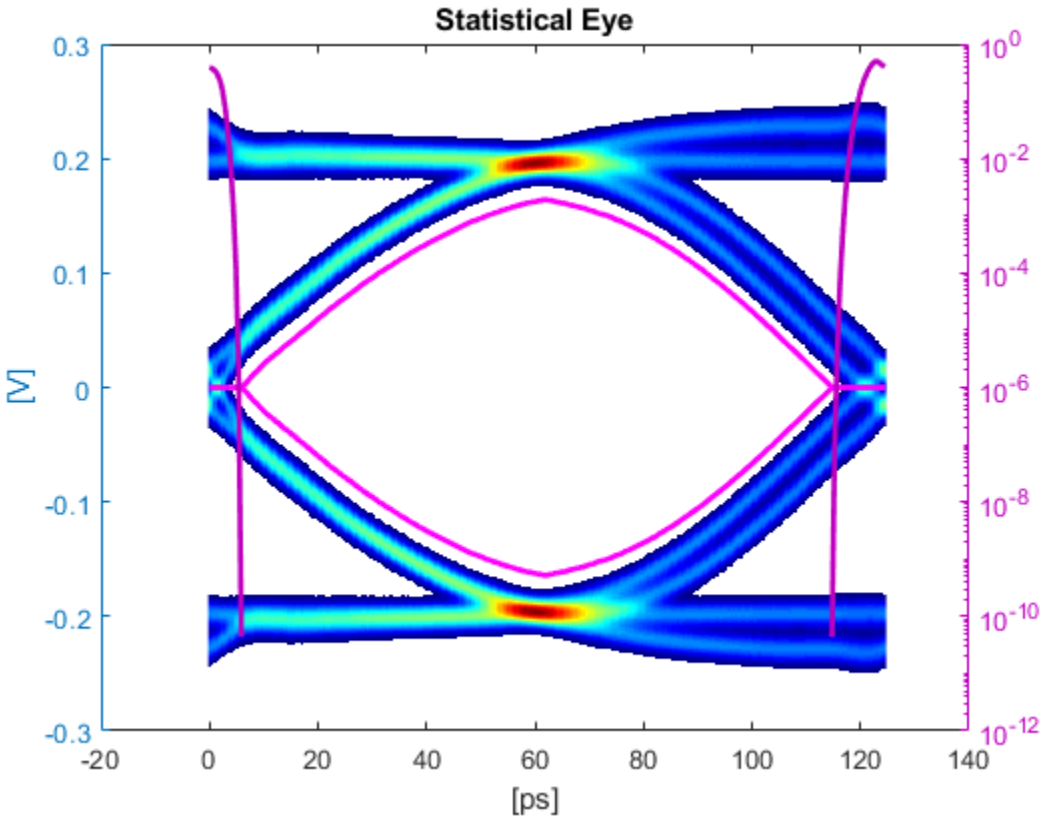
```
% Build ChannelData:
channelLoss = 5;
channel = ChannelData( ...
    'ChannelLossdB',channelLoss, ...
    'ChannelLossFreq',5000000000, ...
    'ChannelDifferentialImpedance',100);
```

Save the change and run the script. Keep changing the value of `channelLoss` to see the effect of changing channel loss.

The eye diagram when the **Channel loss** is set to 5 dB:



The eye diagram when the **Channel loss** is set to 16 dB:



After you finalize the SerDes system with your desired Channel Loss, you can export the MATLAB script of the SerDes system as a Simulink model. From the Simulink canvas, you can perform further time-domain analysis, or export the system to a AMI model.

See Also

AGC | CTLE | DFECDR | FFE | **SerDes Designer** | `serdes.ChannelLoss`

Create and Customize IBIS-AMI Models Topics

SiSoft Link

The SiSoft Link app is used to test the SerDes models developed in Simulink using SerDes Toolbox in SiSoft Quantum Channel Designer (QCD) and Quantum Signal Integrity (QSI) software. You can transfer the data required to reproduce a QCD or QSI test case back to Simulink® for debugging and refinement. You need SiSoft 2018.07-SP4 or later software.

Using the SiSoft Link app, you can:

- Create a QCD project.
- Create a QSI project.
- Import QCD or QSI simulation data into Simulink.
- Update QCD or QSI with new data from Simulink.

To test the SerDes model in QSI or QCD software, first download the SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software from the Add-On Explorer. For more information on downloading add-ons, see “Get and Manage Add-Ons” (MATLAB).

To access the SiSoft link app:

- From the Apps tab in the MATLAB toolstrip, click on SiSoft Link app icon.
- In the MATLAB command prompt, enter `sisoftLink`.

See Also

More About

- “SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software” on page 3-3

External Websites

- https://sisoft.com

SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software

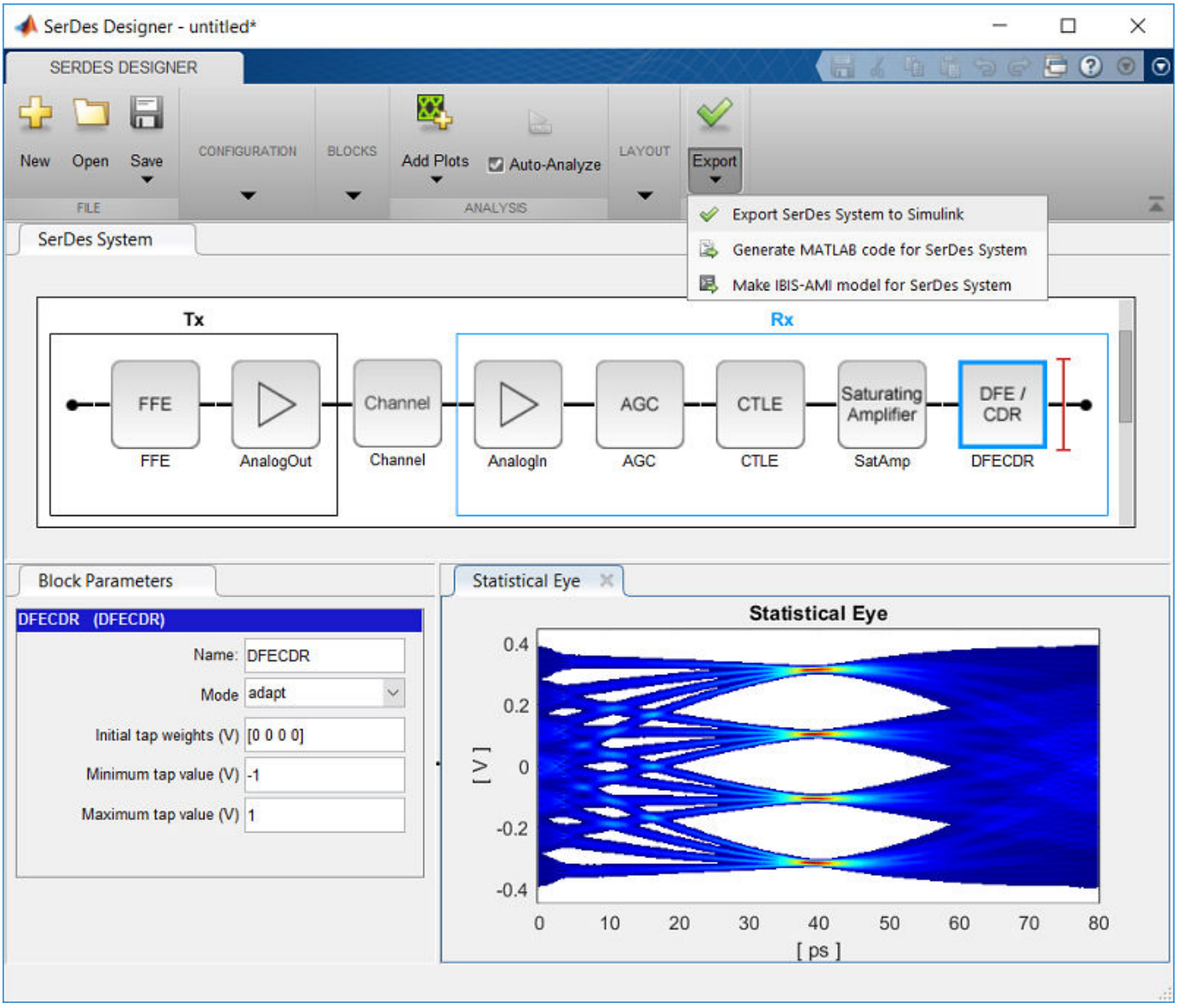
This example shows how to use SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software support package to test IBIS-AMI SerDes models developed in Simulink using SerDes Toolbox, in SiSoft Quantum Channel Designer (QCD) or Quantum Signal Integrity (QSI) software. You can transfer the data required to reproduce a QCD or QSI test case back to Simulink for debugging and refinement. You need SiSoft 2018.07-SP4 or later software to run this example. You must also have installed the SiSoft Link app provided with the support package.

SerDes Development Flow

SerDes model development begins with the SerDes Designer app. The app exports a Simulink model with transmitter (Tx) and receiver (Rx) SerDes models and a testbench to simulate and further develop the SerDes designs. Test the models in QCD or QSI to verify proper IBIS-AMI model operation in a target EDA tool. Due to the high performance of IBIS-AMI executable models, run many simulations to verify the full range of model capabilities, testing with all possible AMI parameters and a variety of stimuli and interconnect channels. Replicate the simulation cases warranting closer inspection in Simulink to reproduce and debug the test. Repeat this cycle as many times as needed, updating the QCD/QSI project and Simulink model.

Create SerDes Toolbox System Model

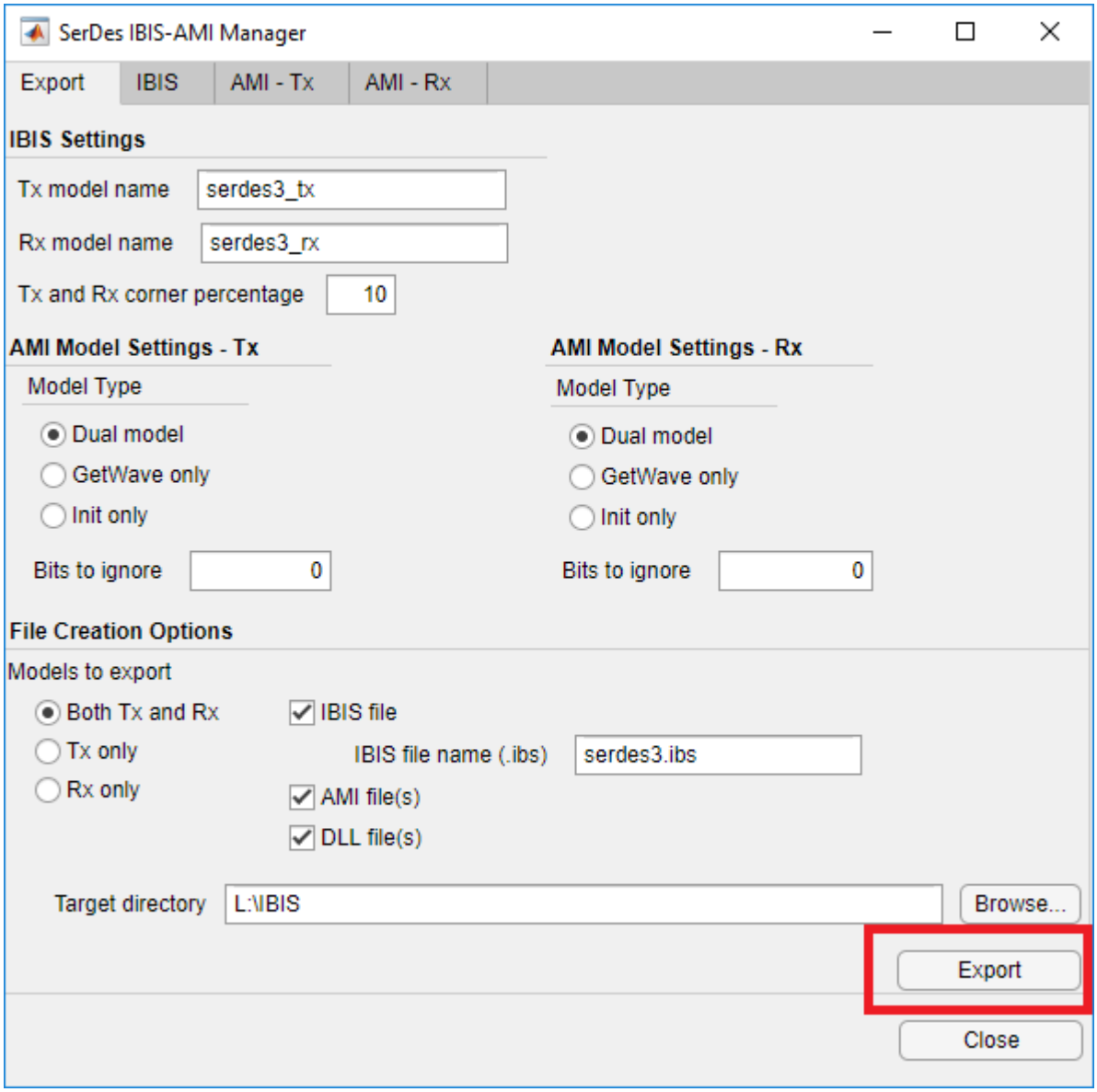
Open the **SerDes Designer** app from the Apps toolstrip. Use the app to quickly prototype and statistically analyze a SerDes system with a Tx and an Rx.



Add blocks from the Blocks gallery to the Tx and Rx sides. If you change the block parameters, the statistical eye display shows the performance changes. Click on **Export SerDes System to Simulink** from the Export dropdown menu to create a Simulink model for the system.

Prepare SerDes Simulink Model for QCD/QSI

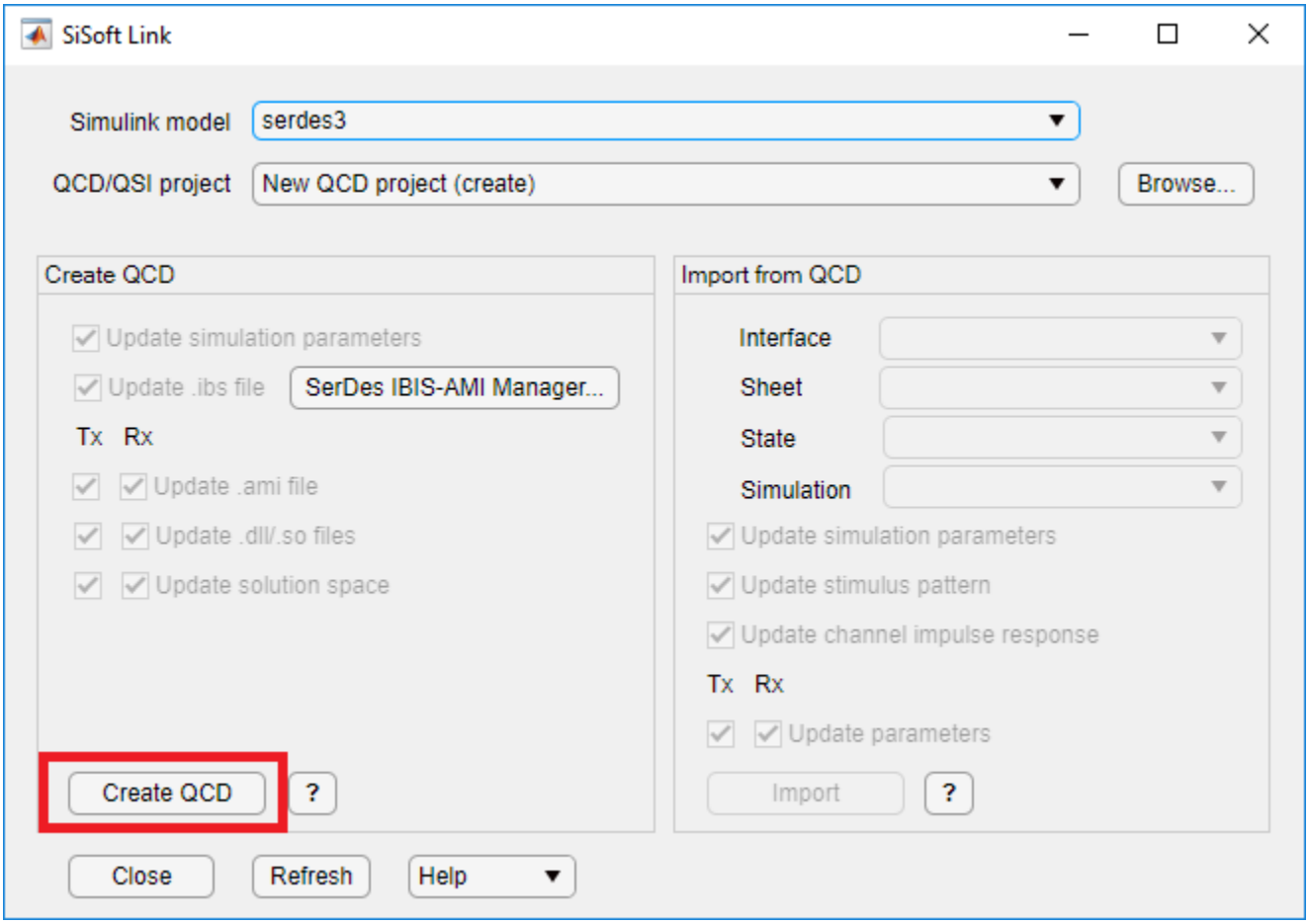
The SiSoft QCD and QSI software requires IBIS models to simulate the Tx and Rx of your system. Use the “Open SerDes IBIS-AMI Manager” button in the Configuration block to produce the IBIS files. In the **Export** tab of the SerDes IBIS-AMI Manager dialog box choose a target directory and click the **Export** button to create the set of IBIS files.



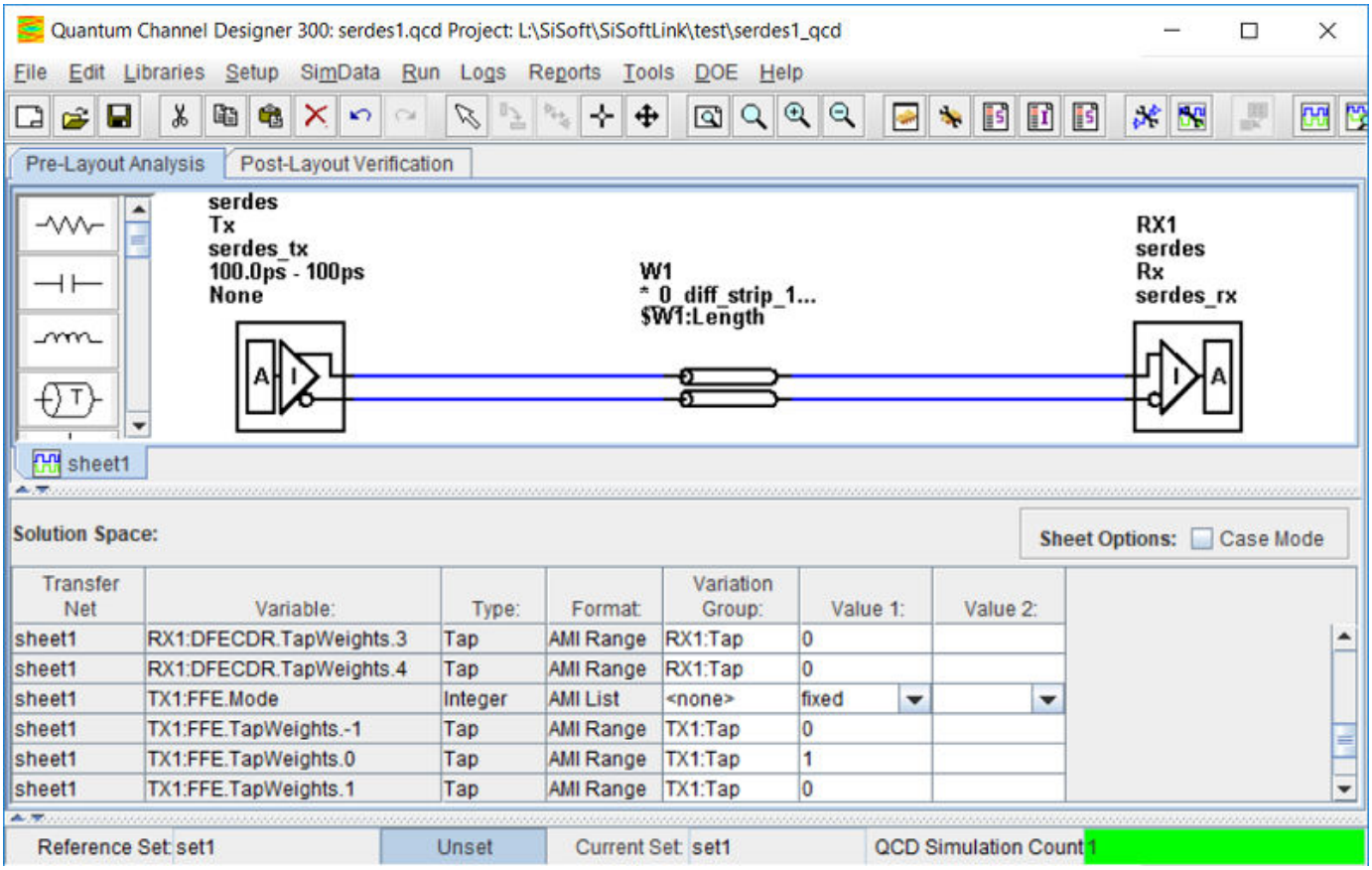
Create QCD Project

Click the SiSoft Link  icon from the Apps tab in the MATLAB toolstrip to open the SiSoft Link app.

If your SerDes system model is open in Simulink, it is listed in the **Simulink Model** dropdown menu in the SiSoft Link app. Click the **Refresh** button if your model is not listed. Set the **QCD/QSI project** dropdown menu to `New QCD project (create)` and click **Create QCD**. If there are unresolved issues regarding the selected Simulink model, **Create QCD** button remains disabled.



Choose a folder in which the QCD project resides and a name for the project folder. The folder path and project name must not have spaces. If you have not yet used SiSoft Link to create a project, the system asks you to locate the folder containing your SiSoft software. A report window appears and QCD opens executing a script produced by SiSoft Link. When script execution finishes, the QCD project interface is renamed after your SerDes system model, with a single sheet `sheet1`.

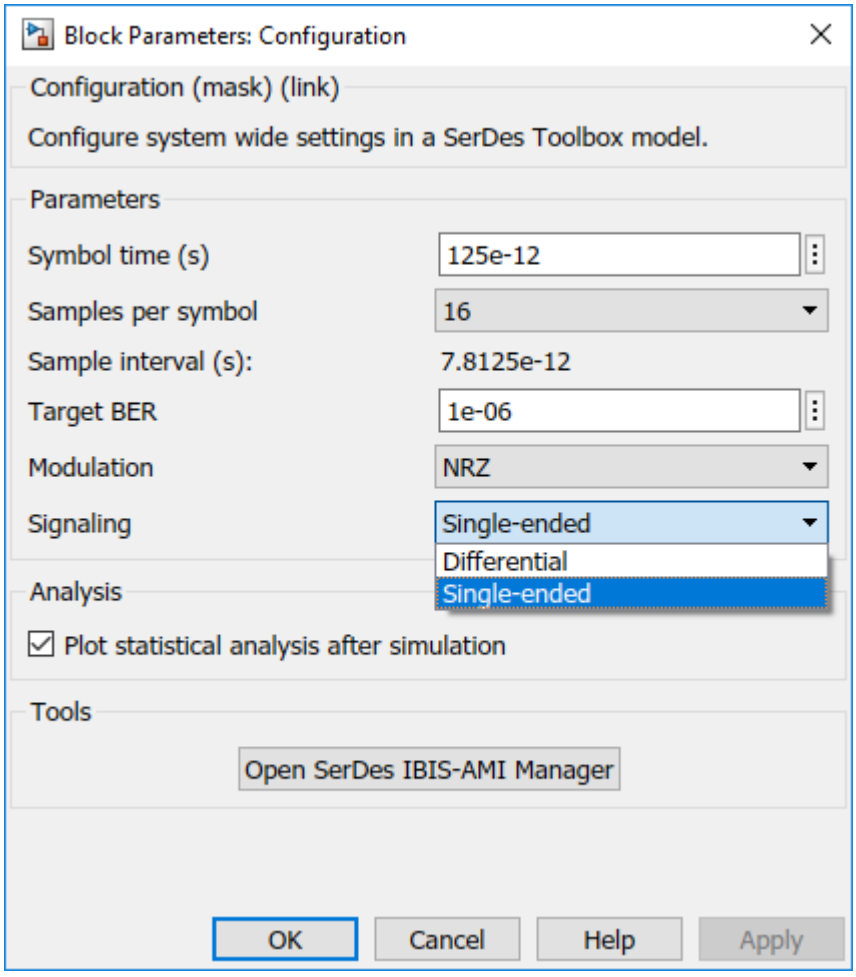


The following data are copied from Simulink to QCD:

- The QCD interface has the same name as the Simulink model.
- QCD has one sheet, `sheet1`.
- All IBIS files is copied into the QCD project `si_lib/ibis` folder.
- All Tx and Rx model parameter values from Simulink is set in the QCD solution space.
- Simulation parameters are set: **UI**, **Samples_Per_Bit**, and **TargetBER**.

Create QSI Project

To create a QSI project, set the **QCD/QSI project** dropdown menu to `New QSI project (create)` and click the **Create QSI** button. The process is otherwise similar to that for QCD. Typically, IBIS-AMI models are used in QSI for analysis of single-ended DDR4/5 DQS signals with equalization. If that is the case, double click the Configuration block in the Simulink model to open it, and set **Signaling** to `Single-ended` before creating the QSI project.

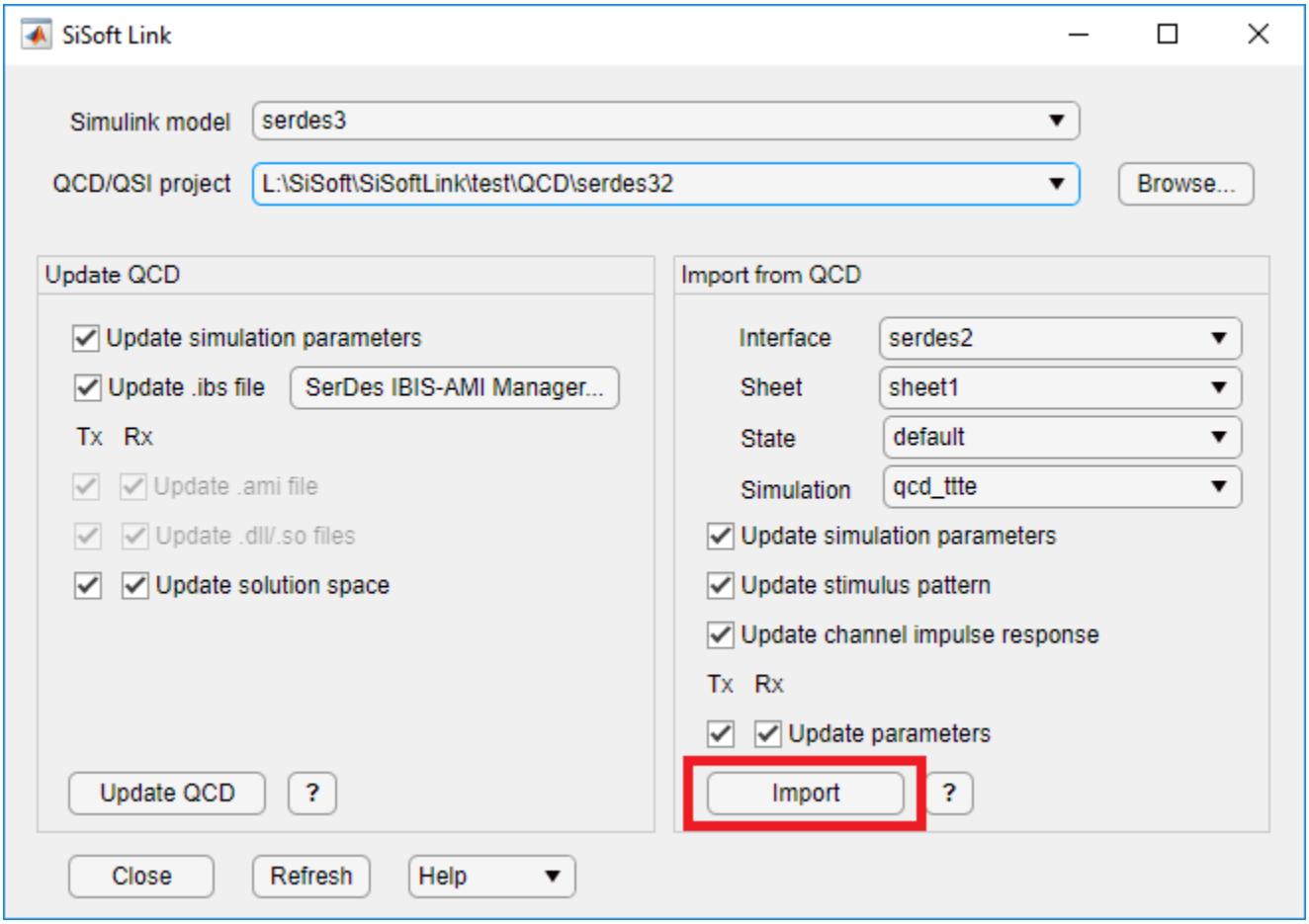


For QSI the following simulation parameters are set:

- The QSI interface has the same name as the Simulink model.
- QSI has one sheet, `sheet1`.
- All IBIS files is copied into the QSI project `si_lib/ibis` folder.
- All Tx and Rx model parameter values from Simulink is set in the QSI solution space.
- Simulation parameters are set: **UI**, **Samples_Per_Bit**, and **TargetBER**.
- The Tx **rise_time** is copied from the typical corner value in the IBIS file.
- **Time_Domain_Stop** is set to `Ignore_Bits + 20,000 UI`.
- **Record_Bits** is set to `100` and **Record_Start** is set accordingly.

Import QCD or QSI Simulation Data into Simulink

After simulating in QCD or QSI, you can import data to reproduce a simulation in Simulink. You must select the project in the **QCD/QSI project** dropdown menu. Click the **Browse...** button to choose a desired QCD or QSI project if it is not listed in the **QCD/QSI project** dropdown menu.



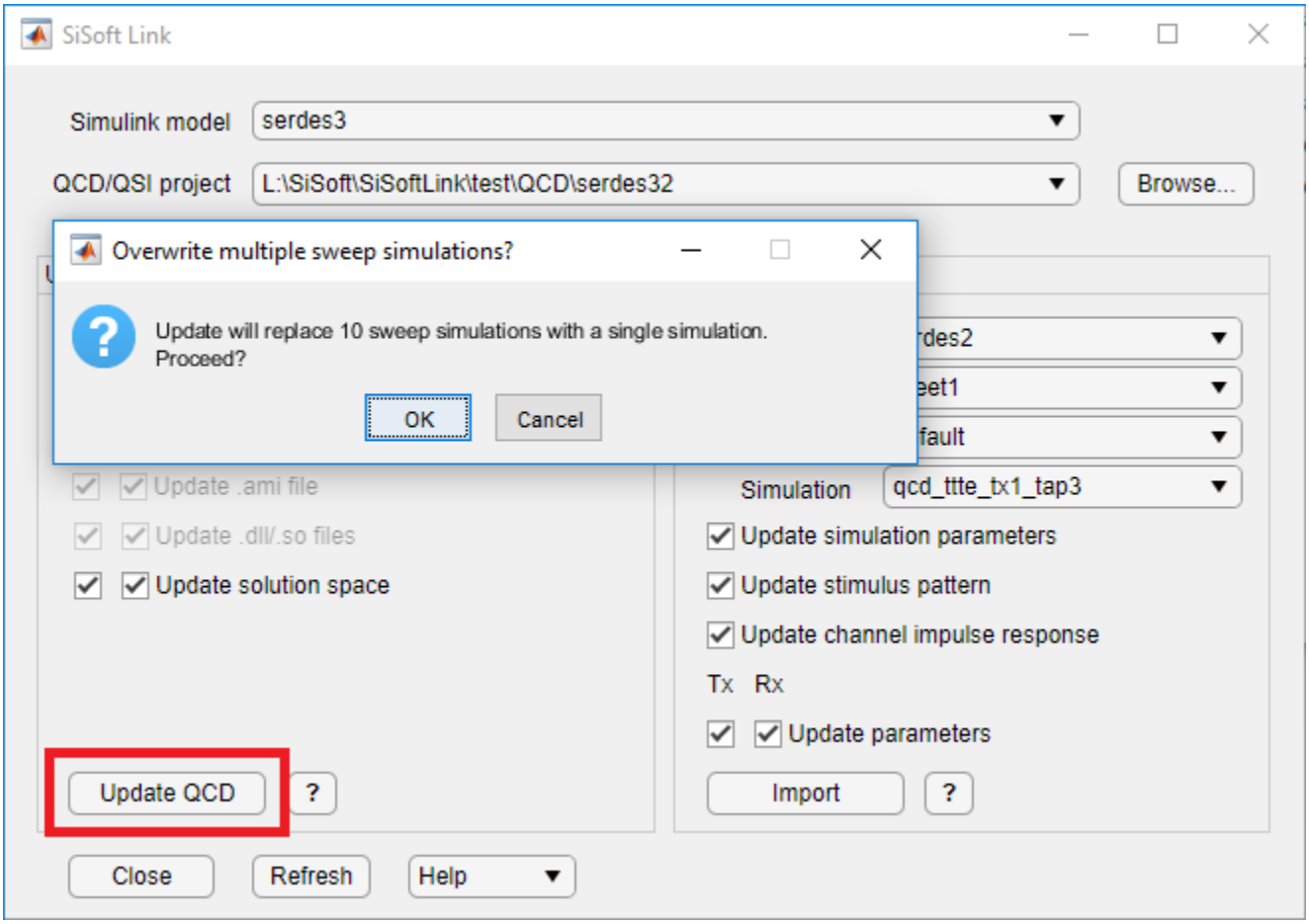
The following data are copied from QCD/QSI to Simulink, as enabled by the Import section checkboxes:

- All Tx and Rx model parameter values from the selected simulation are set in corresponding blocks in the Simulink model.
- **Modulation**, **SymbolTime**, and **SampleInterval** are set in the Configuration block.
- The time domain stimulus pattern is set in the Stimulus block, even if only statistical simulations are run in QCD/QSI.
- The channel impulse response from QCD/QSI is set in the Analog Channel block.

A report is generated giving the details of the import.

Update QCD or QSI with New Data from Simulink

To support iterative development, selectively update a QCD or QSI project with data from Simulink. When a QCD or QSI project path is selected in **QCD/QSI project** dropdown menu, the **Create QCD** (or **Create QSI**) button becomes **Update QCD** (or **Update QSI**). The checkboxes above the button are enabled to choose the data to be updated. If **Update .ibs** file is checked, the checkboxes for .ami files and .dll/.so files are forced on, since importing the .ibis file in QCD or QSI always imports the other files along with it.



Clicking **Update QCD** (or **Update QSI**) runs the QCD (or QSI) to open the project and makes the changes. To avoid conflicts, you must close the project before updating it.

See Also

Analog Channel | Configuration | **SerDes Designer** | Stimulus

More About

- “SiSoft Link” on page 3-2

External Websites

- https://sisoft.com

Design and Simulate SerDes Systems Examples

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

This example shows how to configure the **Specification** parameter `GPZ Matrix` of a CTLE in the SerDes Designer app to use zeros, poles, and gains output by the `zpk` function, given poles and residues output by the `rational` function. You can reformat the set of zeros, poles, and gains output by the `zpk` function to use as a `GPZ matrix` in a CTLE block.

Import Transfer Function

Import a .csv file containing a transfer function using the `readmatrix` function.

```
ctle_transfunc = readmatrix('transfer_function.csv','Range','A7:C775');
freq = ctle_transfunc(:,1);
ri = ctle_transfunc(:,2:end);
```

Convert Transfer Function to Complex Form

To prepare data for use by the `rational` function, convert the real numbers from the transfer function to complex numbers using the `complex` function.

```
data = complex(ri(:,1:2:end),ri(:,2:2:end));
```

Find Rational of Transfer Function

You can use the `rational` function to find the best fit to the transfer function. The `rational` function performs iterations to identify a fit with the lowest error. It is important to set the argument `TendsToZero` to `true` to add a pole so that the fit tends to zero as S approaches infinity. This meets the requirement to have one more pole than the number of zeros in the GPZ matrix.

```
bestfit = rational(freq,data,'Tolerance',-40,'TendsToZero',true,'MaxPoles',8,'Display','on');
```

```
nSurrogate=1; reduced to 100.0%
min achievable error=-Inf
Region 1 of 1 init: np=0 errdb=0
Region 1 of 1: np=0 errdb=1.90887
Region 1 of 1: np=2 errdb=-25.4479
Region 1 of 1: np=4 errdb=-98.7295
final: np=4 errdb=-102.116
```

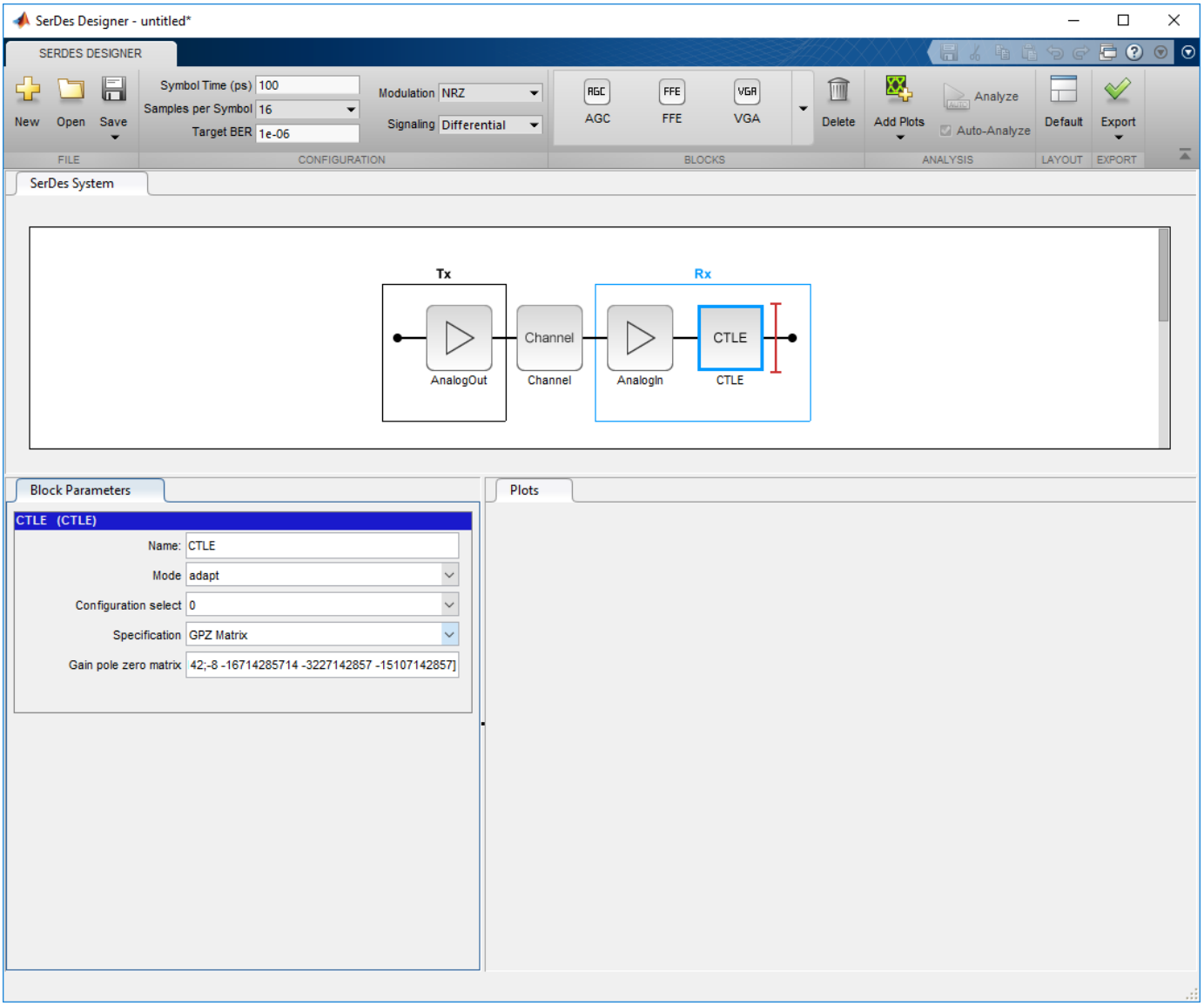
Convert to Zeros, Poles, Gains from Poles and Residues

The `rational` function returns poles and residues, but you need to convert these into zeros, poles and gains for a CTLE block. The CTLE can be configured to use **Specification** parameter `GPZ Matrix` where the units for gains, poles and zeros are dB, Hz, and Hz, respectively.

```
[z,p,~,dcgain]=zpk(bestfit);

gpz(1,1) = dcgain;
gpz(1,2:2:length(p)*2) = p;
gpz(1,3:2:length(z)*2+1) = z;
```

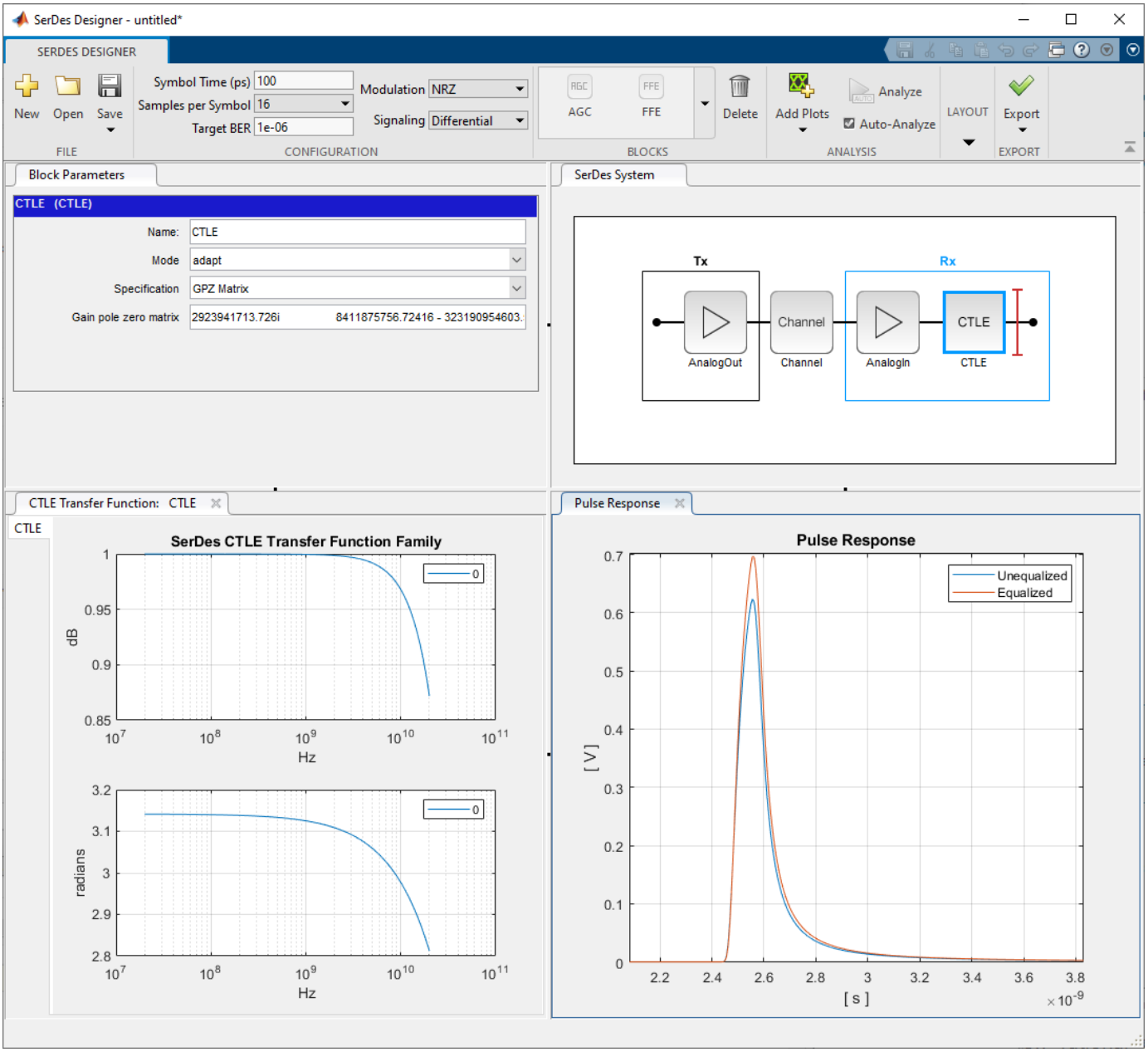
Configure CTLE Block in SerDes Designer

Launch the SerDes Designer app. Place a CTLE block after the analog model of the receiver. Select the CTLE and from the **Block Parameters** pane, set the **Specification** parameter to `GPZ Matrix`. Then copy the value of the `gpz` variable and paste it to the **Gain pole zero matrix** parameter.



Correlate Pulse Response in SerDes Designer to IBIS-AMI Simulation

In the SerDes Designer app, plot the CTLE Transfer Function and Pulse Response from the **Add Plots** button.



Then click the **Export > Make IBIS AMI Model for SerDes System** button. The IBIS-AMI model may be loaded into an appropriate EDA tool to plot the Pulse Response from the model. For correlation purposes, you can compare the plots for Pulse Response from the SerDes Designer app and the EDA tool.

See Also

CTLE | **SerDes Designer**

More About

- “Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance” on page 4-11

Convert Scattering Parameter to Impulse Response for SerDes System

This example covers two topics: the first focuses on the concatenation of three scattering parameters (S-Parameters) that represent a communication channel along with the analog components of a transmitter and of a receiver into a single 4-port S-Parameter, the second focuses on the conversion of this 4-port S-Parameter to an impulse response for use with SerDes Toolbox™. You will use the `rationalfit` function from the RF Toolbox™ to find the impulse response of the S-Parameter that represents a communication channel employing True and Complement differential signaling. For more information, see Modeling a High-Speed Backplane (4-Port S-Parameters to a Rational Function).

Configure Variables

The S-Parameter file must be a 4-port `.s4p` file. Normalize the transmitter (Tx) and receiver (Rx) impedances to `50` Ohms. The `Datarate` and `SamplesPerSymb` valuses must match the settings of the SerDes system and are set to `10` GHz and 32, respectively. The control parameters for the `rationalfit` function include `delayStart`, `delayStep`, `delayStop`, and `nPoles`. Adjust the resolution of the impulse response using `pointsInImpulse.`

```
filename = 'default.s4p';
TxR = 50;
TxC = 0.1e-12;
RxR = 50;
RxC = 0.2e-12;
datarate = 10*1e9;
samplespersymb = 32;

delayStart = 0.68;
delayStep = 0.01;
delayStop = 0.73;
nPoles = 120;
Rshort = 1.0e-6;
pointsInImpulse = 16384;
showPlots = true;
```

Create Differential S-Parameter Representation of Communication Channel

Calculate the symbol time and the sample interval. Import the S-Parameter data file and keep the original frequency content.

```
pulsewidth = 1/datarate;
ts = pulsewidth/samplespersymb;
origSparam = sparameters(filename);
freq = origSparam.Frequencies;
```

Combine S-Parameter of Communication Channel with Analog Sections of Transmitter and Receiver

Create the analog Tx and Rx circuits as S-Parameters. Then combine the S-Paramater models of the Tx and Rx circuits with S-Parameter model of Communication Channel.

Convert Analog Model of Transmitter to S-Parameter

```
cktTx = circuit('txAnalog');
add(cktTx,[1 3],resistor(TxR,'R1'))
```

```
add(cktTx,[3 0],capacitor(TxC,'C1'))
add(cktTx,[2 4],resistor(TxR,'R2'))
add(cktTx,[4 0],capacitor(TxC,'C2'))
setports(cktTx,[1 0],[2 0],[3 0],[4 0])
StxAnalog = sparameters(cktTx,freq,50);
```

Configure Port Order for S-Parameter of Communication Channel

```
cktCh = circuit('ChannelSparam');
channel = nport(filename); % Create s-parameter circuit element
add(cktCh,[1 3 2 4],channel);
setports(cktCh,[1 0],[2 0],[3 0],[4 0])
Schannel = sparameters(cktCh,freq,50);
```

Convert Analog Model of Receiver to S-Parameter

```
cktRx = circuit('rxAnalog');
add(cktRx,[1 0],resistor(RxR,'R3'))
add(cktRx,[1 0],capacitor(RxC,'C3'))
add(cktRx,[2 0],resistor(RxR,'R4'))
add(cktRx,[2 0],capacitor(RxC,'C4'))
add(cktRx,[1 3],resistor(Rshort,'R5'))
add(cktRx,[2 4],resistor(Rshort,'R6'))
setports(cktRx,[1 0],[2 0],[3 0],[4 0])
SrxAnalog = sparameters(cktRx,freq,50);
```

Create Combined S-Parameter Object

Concatenate the S-Parameters using `cascadesparams`. Set values of `data`, `freq` and `z0` for use by the `rationalfit` function. Use `s2sdd` to convert the data to a 4-port differential S-Parameter `diffdata` and set the port order to `1234`. **Note:** RF Toolbox applies ports first from top to bottom on an S-parameter, then from left to right.

```
sparamWithAnalog = cascadesparams(StxAnalog,Schannel,SrxAnalog);

data = sparamWithAnalog.Parameters;
freq = sparamWithAnalog.Frequencies;
z0 = sparamWithAnalog.Impedance;

diffdata = s2sdd(data,2);
diffz0 = 2*z0;
diffsparams = sparameters(diffdata,freq,diffz0);
```

Compute Analytical Form of Transfer Function with Rational Fit

Find the transfer function from the `diffdata`, reference impedance of the S-Parameters, source impedenance in the Tx analog model, and the load impedance in the Rx analog model.

```
zRef = diffz0;   % Reference impedance of S-Parameters
zSource = 1e-6;  % Short circuit the source impedance since it is included in the Tx Analog mode
zLoad = 1e6;     % Open circuit the load impedance since it is included in the Rx Analog model.
difftransfunc = s2tf(diffdata,zRef,zSource,zLoad);
```

Create the `rationalfit` of the diff S-Parameter. Sweep the delay factor and keep the best fit. Then plot the derived `rationalfit` result.

```
bestErr = 0;
bestDelay = 0;
```

```
bestRationalFit = rfmodel.rational();

for delaySweep = delayStart:delayStep:delayStop
    [rationalfunc,errdb] = rationalfit(freq,difftransfunc,-50,'DelayFactor',delaySweep,'Iteratio
    if errdb < bestErr
        bestErr = errdb;
        bestDelay = delaySweep;
        bestRationalFit = rationalfunc;
    end
    fprintf('.');
end

......


rationalfunc = bestRationalFit;
npoles = length(rationalfunc.A);
fprintf('\nThe derived rational function achieved %f dB fit with %f delay and %d poles.\n',bestE

The derived rational function achieved -70.026558 dB fit with 0.680000 delay and 120 poles.
```

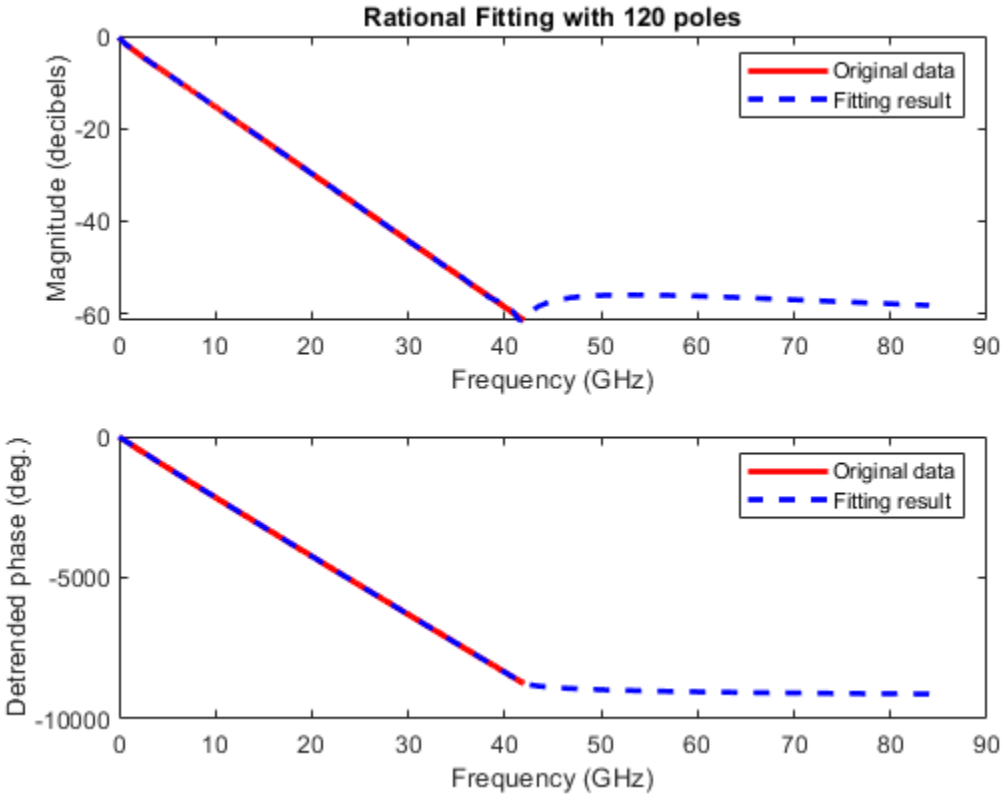
Create the impulse response.

```
[imp,impt]=impulse(rationalfunc,ts,pointsInImpulse);
```

Evaluate Differential-Mode Frequency Response

Plot the magnitude and phase of the original transfer function and the output of `rationalfit`.

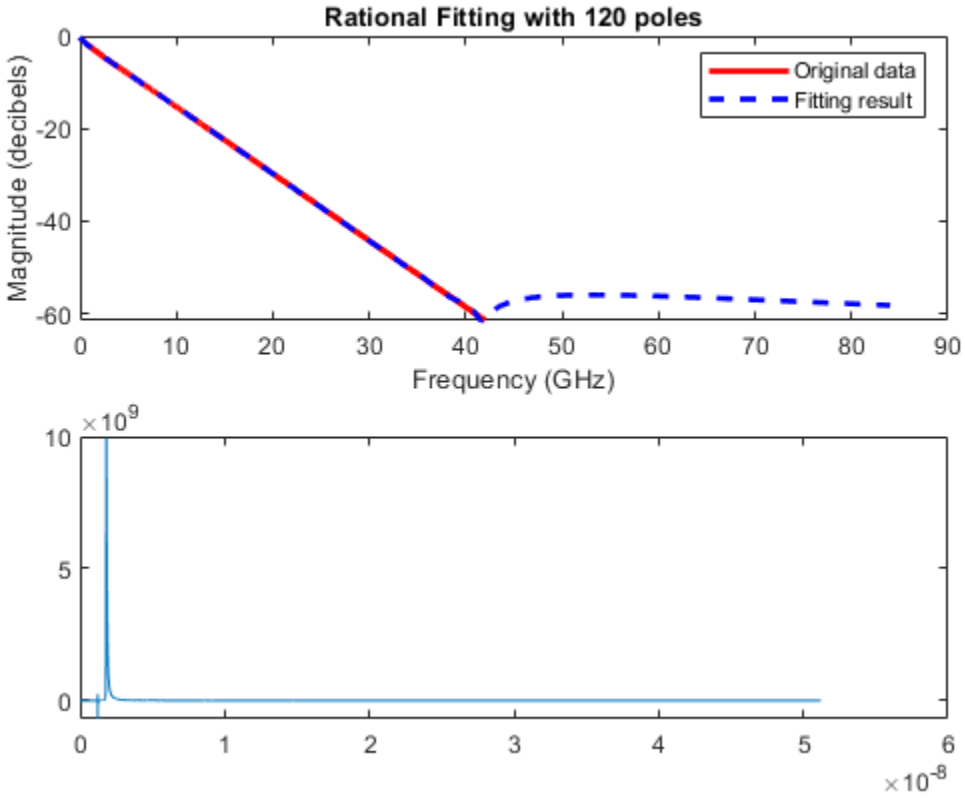
```
    freqsforresp = linspace(0,max(freq)*2,length(freq))';
    resp = freqresp(rationalfunc,freqsforresp);

    figure(11)
    subplot(2,1,1)
    plot(freq*1.e-9,20*log10(abs(difftransfunc)),'r',freqsforresp*1.e-9, ...
        20*log10(abs(resp)),'b--','LineWidth',2)
    title(sprintf('Rational Fitting with %d poles',npoles),'FontSize',12)
    ylabel('Magnitude (decibels)')
    xlabel('Frequency (GHz)')
    legend('Original data','Fitting result')

    subplot(2,1,2)
    origangle = unwrap(angle(difftransfunc))*180/pi+360*freq*rationalfunc.Delay;
    plotangle = unwrap(angle(resp))*180/pi+360*freqsforresp*rationalfunc.Delay;
    plot(freq*1.e-9,origangle,'r',freqsforresp*1.e-9,plotangle,'b--', ...
        'LineWidth',2)
    ylabel('Detrended phase (deg.)')
    xlabel('Frequency (GHz)')
    legend('Original data','Fitting result')
```



Convert Transfer Function to Impulse Response and Plot for Evaluation

Use the transfer function from `rationalfit` to find the impulse response using `impulse`.

```
[imp,impt]=impulse(rationalfunc,ts,pointsInImpulse);
plot(impt,imp);
```



See Also

SerDes Designer

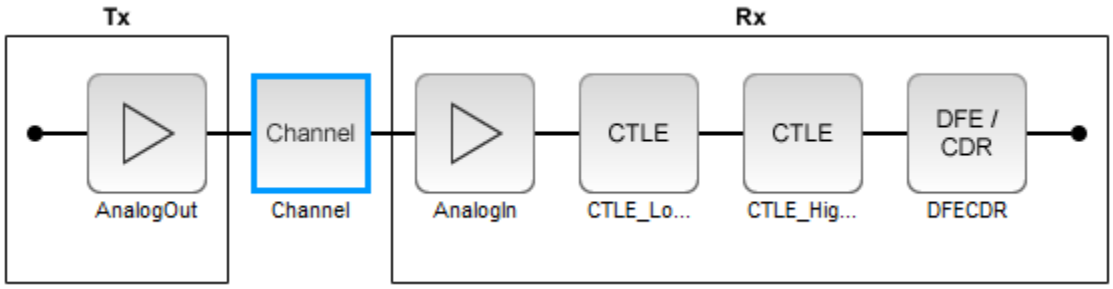
Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance

This example shows how to perform optimization of a set of receiver components as a system using function `optPulseMetric` to calculate metrics such as eye height, width and channel operating margin (COM) estimate from a pulse response at a target bit error rate (BER) to evaluate the optimal performance of a particular configuration. The adaptation is performed as statistical analysis (Init), then the optimized result is passed to time-domain (GetWave).

Initialize SerDes System with Multiple CTLEs and DFECDR

This example uses the SerDes Designer model `rx_ctle_adapt_dfe_train` as a starting point. Type the following command in the MATLAB® command window to open the model:

```
>> serdesDesigner('rx_ctle_adapt_dfe_train.mat')
```



This project contains a receiver section with two CTLE blocks followed by a DFECDR block. In their default configuration, these blocks optimize individually. The goal of this example is to optimize the blocks as a system.

For the `CTLE_LowFreq` block, the **Peaking frequency (GHz)** is set to `[10 11 12 13 14 15 16]`, the **DC gain (dB)** is set to `[0 0 0 0 0 0 0]`, and the **Peaking gain (dB)** is set to `0`. All other parameters are kept at their default values.

For the `CTLE_HighFreq` block, the **Specification** is set to `DC Gain and AC Gain`, the **Peaking frequency (GHz)** is set to `14`, the **DC gain (dB)** is set to `0`, and the **AC gain (dB)** is set to `[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]`. All other parameters are kept at their default values.

For the DFECDR block, the **Initial tap weights (V)** is set to `[0 0 0 0 0 0 0 0 0 0]`. All other parameters are kept at their default values.

Export the SerDes system to a Simulink® model.

Add Code to Optimize CTLEs and DFECDR as System

Double click the Init subsystem inside the Rx block and click on the **Show Init** button. Copy the complete code listed at the end of this example inside the **Custom user code area**. Save the model. The code is broken down below in several subsections for easy comprehension.

Note: To complete the example, you must place the following code sections together in the **Custom user code area** inside the Init subsystem. A complete set of code is provided at the end of this example. For more information about Init subsystem, see “Statistical Analysis in SerDes Systems” on page 1-18.

Initialize Receiver Parameters

The first section of the **Custom user code area** checks if both CTLEs are in adapt mode and instantiating variables to hold temporary values and the best configuration metrics.

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
% If both CTLEs are in Adapt mode, use global adaptation
if CTLE_LowFreqParameter.Mode == 2 && CTLE_HighFreqParameter.Mode == 2
    CTLE_LowFreqInitBestConfig = 0;
    CTLE_HighFreqInitBestConfig = 0;
    bestMetric = 0;
    SPB = SymbolTime/SampleInterval;
```

Sweep CTLE Parameters

The example code sets the CTLE.Mode parameter from `adapt` to `fixed` to allow algorithmic control of the values for each block. In this case the values are directly swept and the blocks are called to process the impulse response.

```
    CTLE_LowFreqInit.Mode = 1;
    CTLE_HighFreqInit.Mode = 1;
    for CTLE_LowFreqInitSweep = 0:1:6
        for  CTLE_HighFreqInitSweep = 0:1:15
            % Set current sweep configs on each CTLE
            CTLE_LowFreqInit.ConfigSelect = CTLE_LowFreqInitSweep;
            CTLE_HighFreqInit.ConfigSelect = CTLE_HighFreqInitSweep;
            % Call CTLEs and DFE
            [sweepImpulse, ~] = CTLE_LowFreqInit(LocalImpulse);
            [sweepImpulse, ~] = CTLE_HighFreqInit(sweepImpulse);
            [sweepImpulse, ~, ~, ~, ~] = DFECDRInit(sweepImpulse);
```

Convert Impulse Response to Pulse Response and Evaluate with optPulseMetric

Convert the impulse response to a pulse response for evaluation by function `optPulseMetric`. A pulse response lends itself to metrics-based evaluation more readily than an impulse response. The `optPulseMetric` function is used to optimize the SerDes system as a whole. Many metrics are reported by this function and you can use an algorithm to evaluate multiple receiver components together as a system.

```
            % Convert impulse after DFE to pulse then calculate eye metrics
            sweepPulse = impulse2pulse(sweepImpulse,SPB,SampleInterval);
            eyeMetric = optPulseMetric(sweepPulse,SPB,SampleInterval,1e-6);
            % Select eye metric to evaluate results
            sweepMetric = eyeMetric.maxMeanEyeHeight;
%             sweepMetric = eyeMetric.eyeHeightMax;
%             sweepMetric = eyeMetric.COMMax;
%             sweepMetric = eyeMetric.meanHeightCenter;
%             sweepMetric = eyeMetric.eyeHeightCenter;
%             sweepMetric = eyeMetric.COMCenter;
```

Evaluate optPulseMetric Results

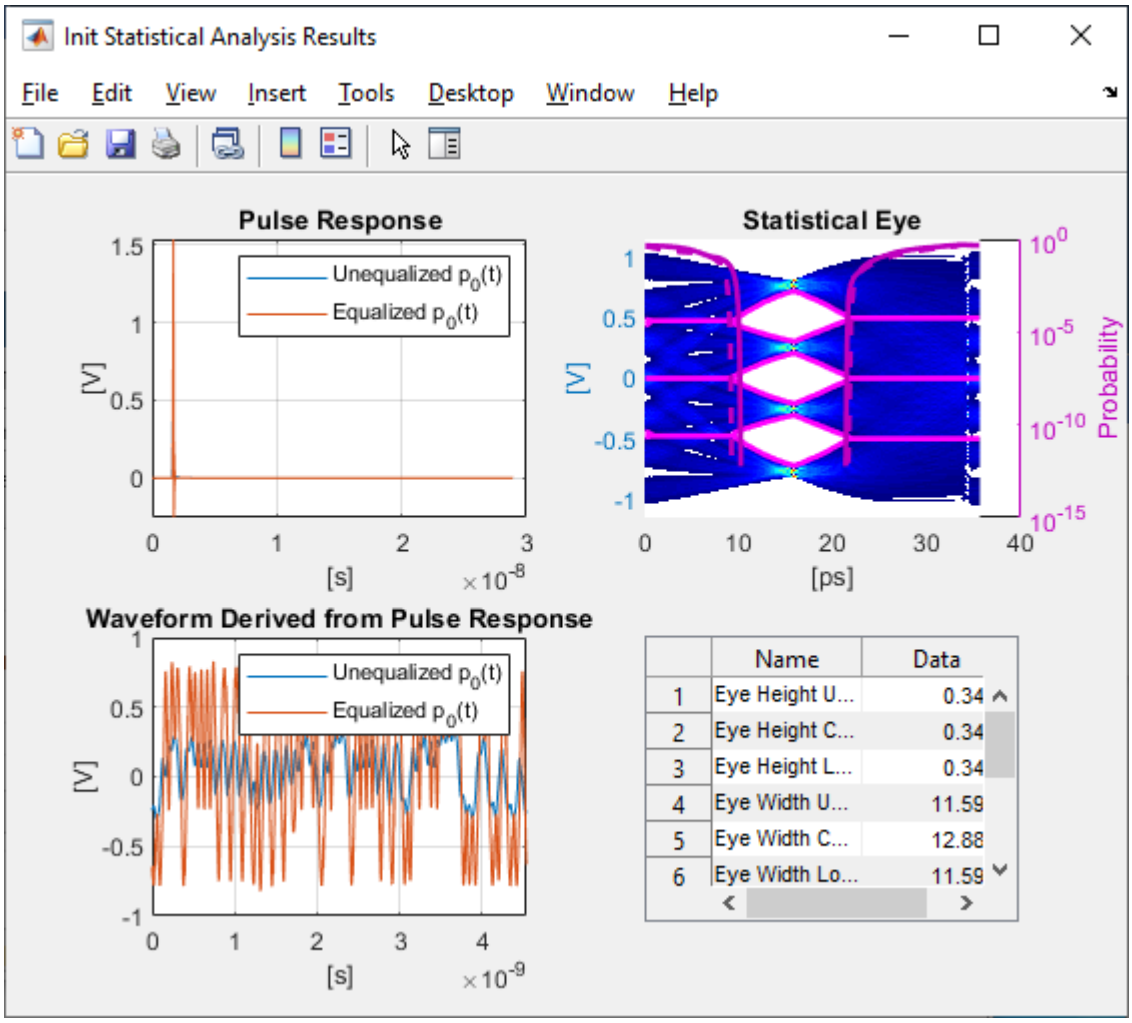
Save the CTLE configurations based on comparison to previous results. The final best configurations are saved on the blocks for a final statistical (Init) analysis and then passed to time-domain (GetWave) simulation.

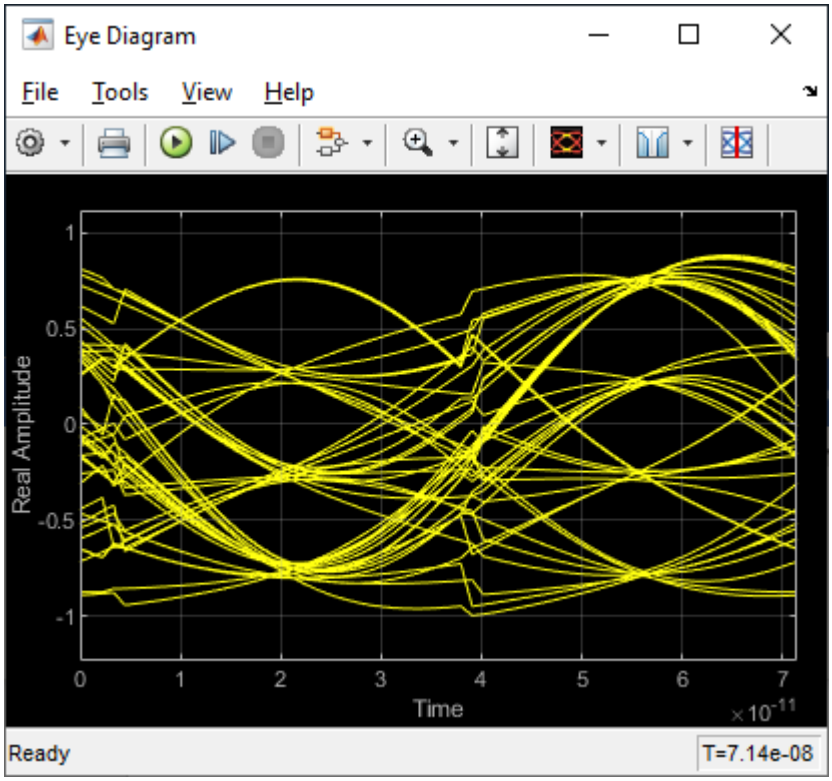
```
     % If current sweep metric is better than previous, save the CTLE configs
            if sweepMetric > bestMetric
```

```
                bestMetric = sweepMetric;
                CTLE_LowFreqInitBestConfig = CTLE_LowFreqInitSweep;
                CTLE_HighFreqInitBestConfig = CTLE_HighFreqInitSweep;
            end
        end
    end
    % Set CTLEs to best configs from sweep
    CTLE_LowFreqInit.ConfigSelect = CTLE_LowFreqInitBestConfig;
    CTLE_HighFreqInit.ConfigSelect = CTLE_HighFreqInitBestConfig;
end
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

Run SerDes System

Run the SerDes system and observe the optimizing behavior. You can try changing which metric is evaluated to perform different optimizations.





Complete Code

Copy-paste the following code within the **Custom user code area** inside the Init subsystem of the receiver.

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
% If both CTLEs are in Adapt mode, use global adaptation
if CTLE_LowFreqParameter.Mode == 2 && CTLE_HighFreqParameter.Mode == 2
    CTLE_LowFreqInitBestConfig = 0;
    CTLE_HighFreqInitBestConfig = 0;
    bestMetric = 0;
    SPB = SymbolTime/SampleInterval;
    CTLE_LowFreqInit.Mode = 1;
    CTLE_HighFreqInit.Mode = 1;
    for CTLE_LowFreqInitSweep = 0:1:6
        for  CTLE_HighFreqInitSweep = 0:1:15
            % Set current sweep configs on each CTLE
            CTLE_LowFreqInit.ConfigSelect = CTLE_LowFreqInitSweep;
            CTLE_HighFreqInit.ConfigSelect = CTLE_HighFreqInitSweep;
            % Call CTLEs and DFE
            [sweepImpulse, ~] = CTLE_LowFreqInit(LocalImpulse);
            [sweepImpulse, ~] = CTLE_HighFreqInit(sweepImpulse);
            [sweepImpulse, ~, ~, ~, ~] = DFECDRInit(sweepImpulse);
            % Convert impulse after DFE to pulse then calculate eye metrics
            sweepPulse = impulse2pulse(sweepImpulse,SPB,SampleInterval);
            eyeMetric = optPulseMetric(sweepPulse,SPB,SampleInterval,1e-6);
            % Select eye metric to evaluate results
            sweepMetric = eyeMetric.maxMeanEyeHeight;
%             sweepMetric = eyeMetric.eyeHeightMax;
%             sweepMetric = eyeMetric.COMMax;
```

```
%             sweepMetric = eyeMetric.meanHeightCenter;
%             sweepMetric = eyeMetric.eyeHeightCenter;
%             sweepMetric = eyeMetric.COMCenter;
% If current sweep metric is better than previous, save the CTLE configs
            if sweepMetric > bestMetric
                bestMetric = sweepMetric;
                CTLE_LowFreqInitBestConfig = CTLE_LowFreqInitSweep;
                CTLE_HighFreqInitBestConfig = CTLE_HighFreqInitSweep;
            end
        end
    end
    % Set CTLEs to best configs from sweep
    CTLE_LowFreqInit.ConfigSelect = CTLE_LowFreqInitBestConfig;
    CTLE_HighFreqInit.ConfigSelect = CTLE_HighFreqInitBestConfig;
end
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

See Also

CTLE | DFECDR | optPulseMetric

More About

- “Find Zeros, Poles, and Gains for CTLE from Transfer Function” on page 4-2
- “Implement Custom CTLE in SerDes Toolbox PassThrough Block” on page 5-20
- “Statistical Analysis in SerDes Systems” on page 1-18

Customize SerDes Systems

Customizing SerDes Toolbox Datapath Control Signals

This example shows how to customize the control signals in a SerDes system datapath by adding new custom AMI parameters and using MATLAB® function blocks. This allows you to customize existing control parameters without modifying the built-in blocks in the SerDes Toolbox™ library.

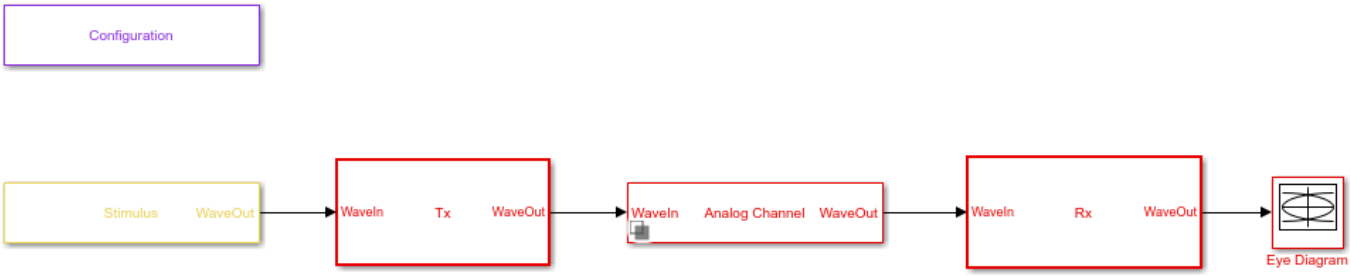
This example shows how to add a new AMI parameter to control the operation of the three transmitter taps used by the FFE block. The custom AMI parameter simultaneously sets all three taps to one of the ten values defined by the PCIe4 specification or allows you to enter three custom floating-point tap values. To know more about how to define a PCIe4 transmitter model, see “PCIe4 Transmitter/Receiver IBIS-AMI Model” on page 7-2.

PCIe4 Transfer Model

The transmitter model in this example complies with the PCIe4 specification. The receiver is a simple pass-through model. A PCIe4 compliant transmitter uses a 3-tap feed forward equalizer (FFE) with one pre-tap and one post-tap, and ten presets.

Open the model `adding_tx_ffe_params`. The SerDes system Simulink® model consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks.

```
open_system('adding_tx_ffe_params.slx')
```



- The Tx subsystem contains an FFE block to model the time-domain portion of the AMI model and an Init block to model the statistical portion.
- The Analog Channel block has the PCIe4 parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx analog model parameters.
- The Rx subsystem has a Pass-Through block and an Init block.

Add New AMI Parameter

Add a new AMI parameter to the transmitter which is available to both the Init and GetWave datapath blocks and functions. The parameter is also included in the Tx IBIS-AMI file.

Double-click the Configuration block to open the Block Parameters dialog box. Click the **Open SerDes IBIS-AMI Manager** button. Go to the **AMI-Tx** tab of the SerDeS IBIS-AMI Manager dialog box.

- Select the FFE parameter, then click **Add Parameter...** to add a new FFE sub-parameter.
- Set the Parameter name to `ConfigSelect`.
- Keep the **Current value** as `0`.

- In the Description, add `Pre/Main/Post tap configuration selector`.
- Keep the **Usage** as `In`.
- Set the **Type** to `Integer`.
- Set the **Format** to `List`.
- Under the **List Format details**, set **Default** to `0`.
- Set **List values** to `[-1 0 1 2 3 4 5 6 7 8 9]`
- Set **List_Tip values** to `["User Defined" "P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9"]`

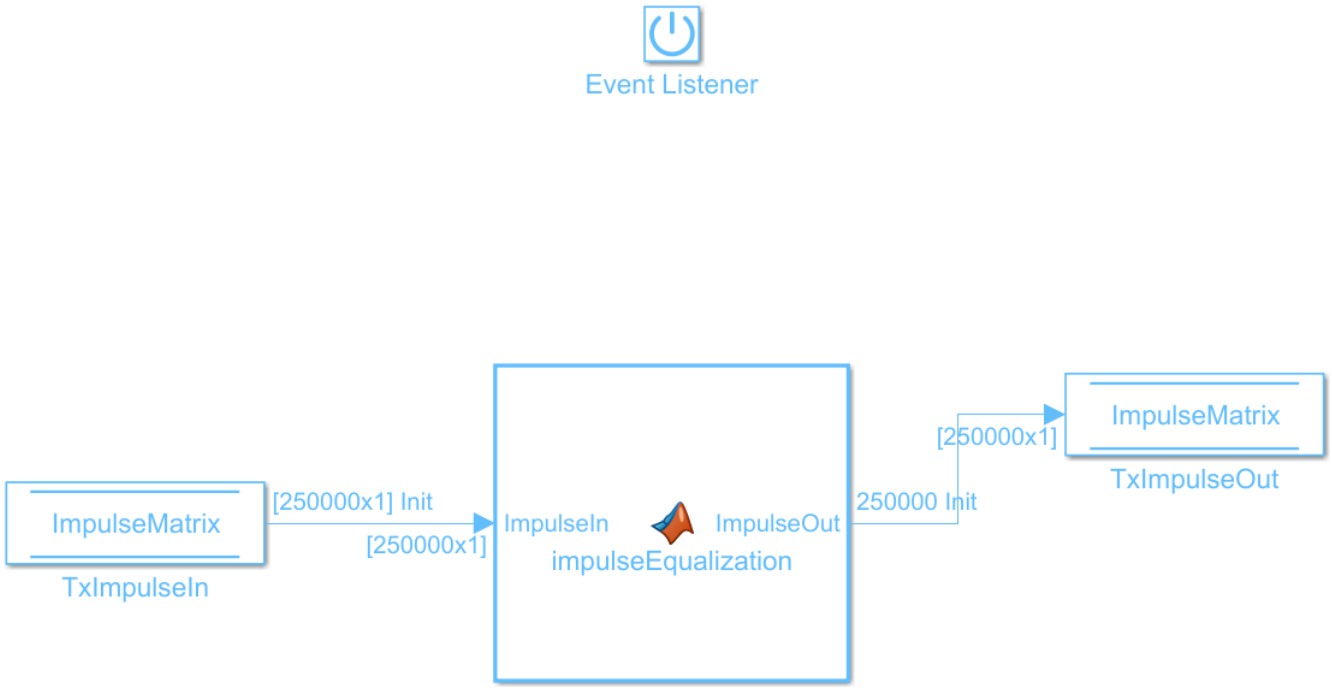
A new parameter **ConfigSelect*** is added to the **AMI-Tx** tab.

Modify Init

Modify the Initialize MATLAB function inside the Init block in the Tx subsystem to use the newly added **ConfigSelect***parameter. The **ConfigSelect*** parameter controls the existing three transmitter taps. To accomplish this, add a switch statement that takes in the values of **ConfigSelect*** and automatically sets the values for all three Tx taps, ignoring the user defined values for each tap. If a **ConfigSelect** value of `-1` is used, then the user-defined Tx tap values are passed through to the FFE datapath block unchanged.

Inside the Tx subsystem, double-click the Init block to open the Block Parameters dialog box and click the **Refresh Init** button to propagate the new AMI parameter to the Initialize sub-system.

Type **Ctrl-U** to look under the mask for the Init block, then double-click on the initialize block to open the Initialize Function.



Double-click on the impulseEqualization MATLAB function block to open the function in MATLAB. This is an automatically generated function which provides the impulse response processing of the

SerDes system block (IBIS AMI-Init). The `%% BEGIN:` and `% END:` lines denote the section where custom user code can be entered. Data in this section will not get over-written when Refresh Init is run:

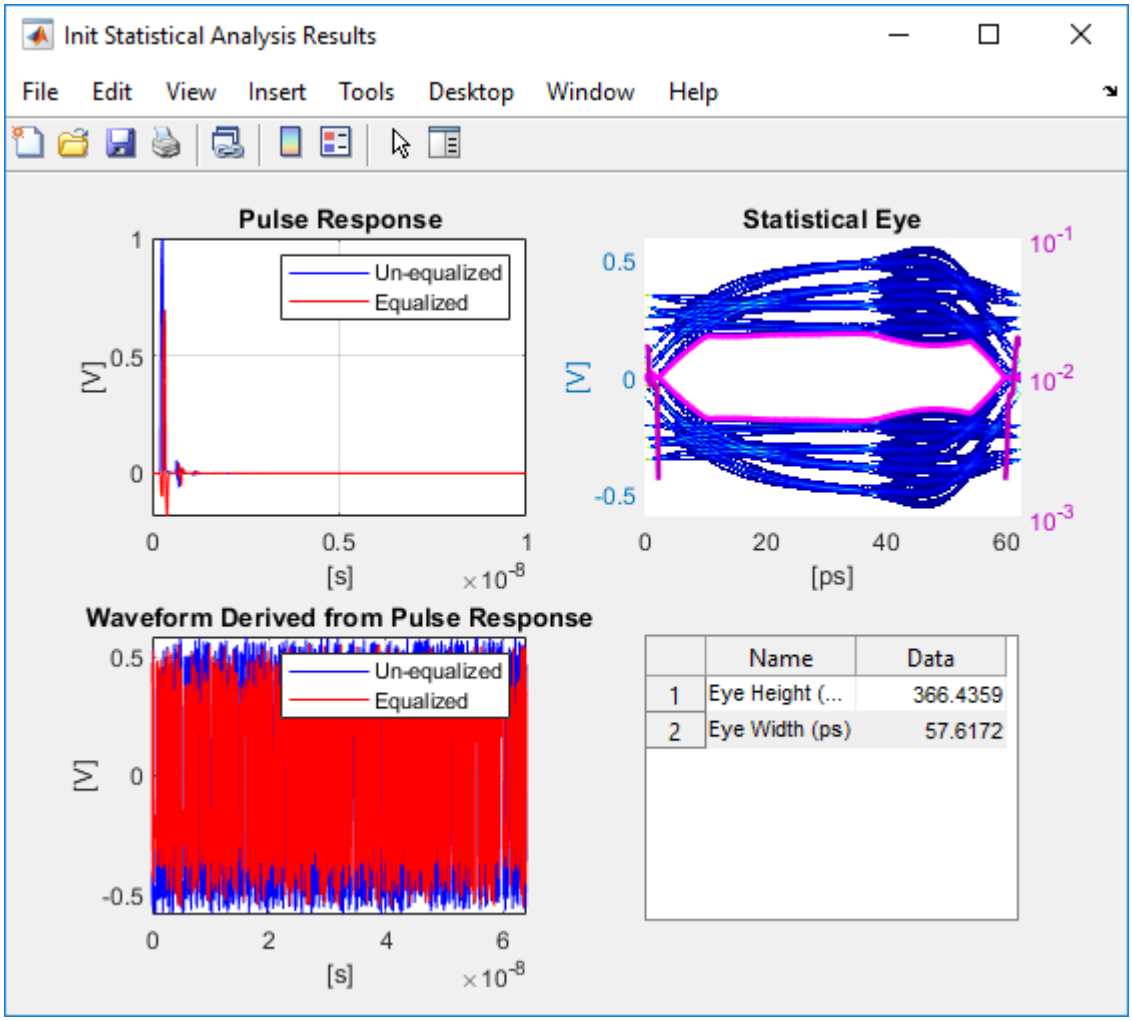
```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
FFEParameter.ConfigSelect; % User added AMI parameter
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

To add the custom ConfigSelect control code, scroll down the Customer user code area, comment out the `FFEParameter.ConfigSelect` line, then enter the following code:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
%FFEParameter.ConfigSelect; % User added AMI parameter
switch FFEParameter.ConfigSelect
case -1  % User defined tap weights
FFEInit.TapWeights = FFEParameter.TapWeights;
case 0  % PCIe Configuration: P0
FFEInit.TapWeights = [0.000 0.750 -0.250];
case 1  % PCIe Configuration: P1
FFEInit.TapWeights = [0.000 0.830 -0.167];
case 2  % PCIe Configuration: P2
FFEInit.TapWeights = [0.000 0.800 -0.200];
case 3  % PCIe Configuration: P3
FFEInit.TapWeights = [0.000 0.875 -0.125];
case 4  % PCIe Configuration: P4
FFEInit.TapWeights = [0.000 1.000 0.000];
case 5  % PCIe Configuration: P5
FFEInit.TapWeights = [-0.100 0.900 0.000];
case 6  % PCIe Configuration: P6
FFEInit.TapWeights = [-0.125 0.875 0.000];
case 7  % PCIe Configuration: P7
FFEInit.TapWeights = [-0.100 0.700 -0.200];
case 8  % PCIe Configuration: P8
FFEInit.TapWeights = [-0.125 0.750 -0.125];
case 9  % PCIe Configuration: P9
FFEInit.TapWeights = [-0.166 0.834 0.000];
otherwise
FFEInit.TapWeights = FFEParameter.TapWeights;
end
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

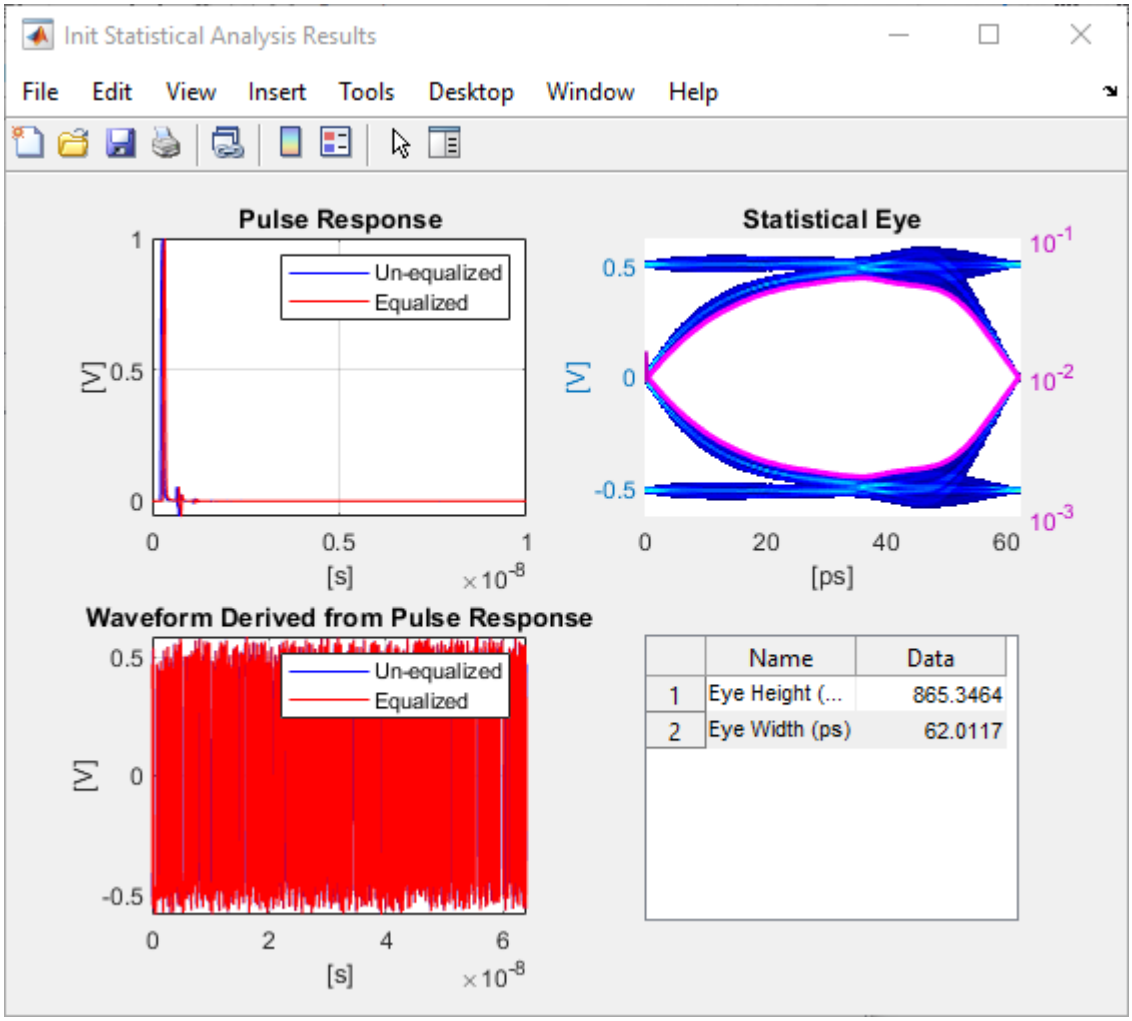
To test that the new FFE control parameter is working correctly, open the SerDes IBIS-AMI Manager dialog box from the Configuration block. In the **AMI-Tx** tab, edit the **ConfigSelect*** parameter to set **Current value** to `P7`. This corresponds to PCIe Configuration P7: Pre = -0.100, Main = 0.700 and Post = -0.200.

Run the simulation and observe the results of Init statistical analysis.



Next, set the **Current value** of the **ConfigSelect*** parameter to `User Defined`. This corresponds to user-defined tap weights: Pre = 0.000, Main = 1.000 and Post = 0.000.

Run the simulation and observe the results of Init statistical analysis.



Try different values of **ConfigSelect*** to verify proper operation. The statistical eye opens and closes based on the amount of equalization applied by the FFE. How much the eye changes, and the tap values that create the most open eye varies based on the loss defined in the Analog Channel block.

Modify GetWave

To modify GetWave, add a new MATLAB function that operates in the same manner as the Initialize function.

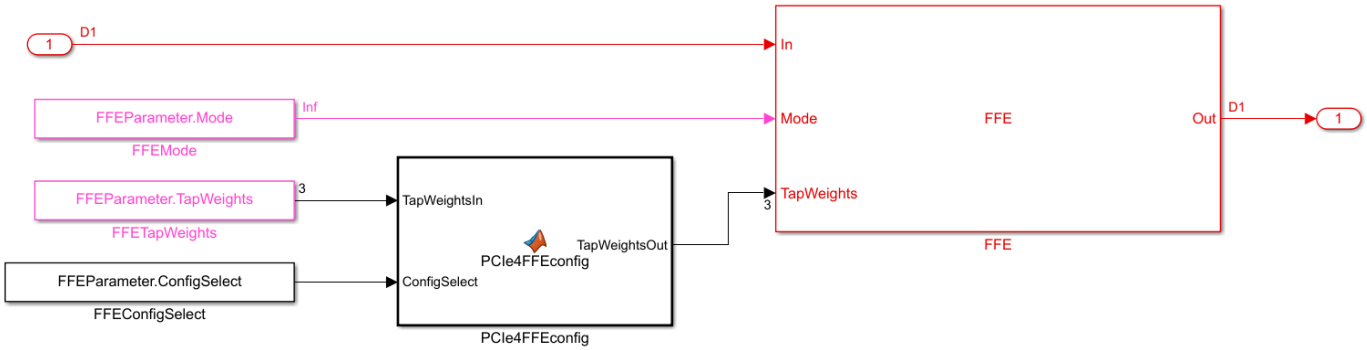
Inside the Tx subsystem, type **Ctrl-U** to look under the mask of the FFE block.

- Add a Constant block to the canvas from the Simulink/Sources library.
- Rename the Constant block as `FFEConfigSelect` and set the **Constant value** to `FFEParameter.ConfigSelect`.
- Add a MATLAB Function block to the canvas from the Simulink/User-Defined library.
- Rename the MATLAB Function block to `PCIe4FFEconfig`.
- Double-click the MATLAB Function block and replace the template code with the following:

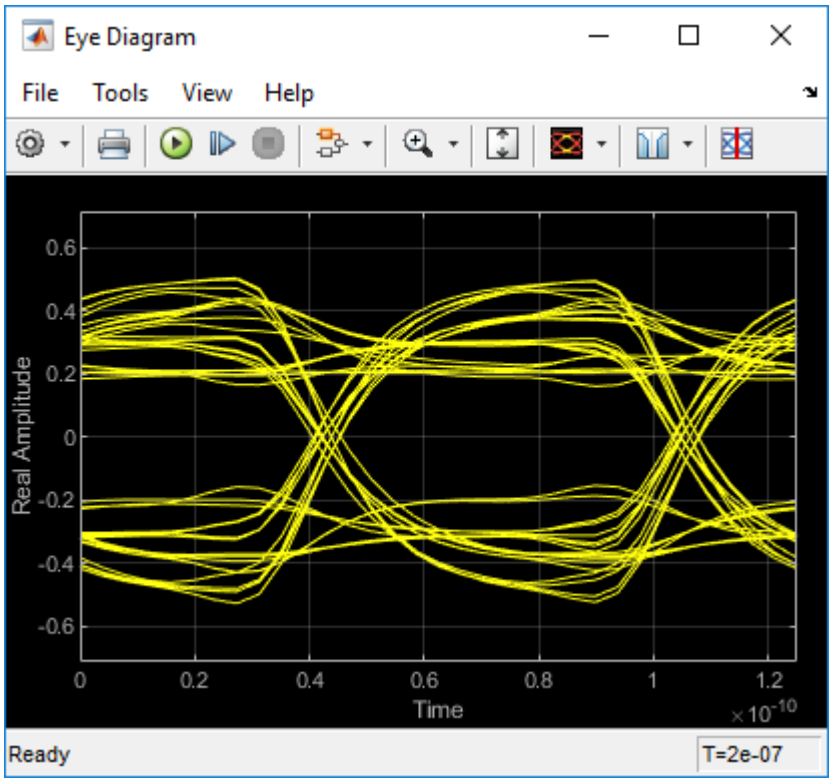
```
% PCIe4 tap configuration selector
%   Selects pre-defined Tx FFE tap weights based on PCIe4 specified
%   configurations.
%
% Inputs:
%   TapWeightsIn: User defined floating point tap weight values.
%   ConfigSelect: 0-9: PCIe4 defined configuration (P0-P9).
%                 -1:  User defined configuration (from TapWeightsIn).
% Outputs:
%   TapWeightsOut: Array of tap weights to be used.
%
function TapWeightsOut = PCIe4FFEconfig(TapWeightsIn, ConfigSelect)

switch ConfigSelect
    case -1  % User defined tap weights
         TapWeightsOut = TapWeightsIn;
    case 0   % PCIe Configuration: P0
        TapWeightsOut = [0.000 0.750 -0.250];
    case 1   % PCIe Configuration: P1
        TapWeightsOut = [0.000 0.833 -0.167];
    case 2   % PCIe Configuration: P2
         TapWeightsOut = [0.000 0.800 -0.200];
    case 3   % PCIe Configuration: P3
        TapWeightsOut = [0.000 0.875 -0.125];
    case 4   % PCIe Configuration: P4
        TapWeightsOut = [0.000 1.000 0.000];
    case 5   % PCIe Configuration: P5
        TapWeightsOut = [-0.100 0.900 0.000];
    case 6   % PCIe Configuration: P6
        TapWeightsOut = [-0.125 0.875 0.000];
    case 7   % PCIe Configuration: P7
        TapWeightsOut = [-0.100 0.700 -0.200];
    case 8   % PCIe Configuration: P8
        TapWeightsOut = [-0.125 0.750 -0.125];
```

```
    case 9   % PCIe Configuration: P9
        TapWeightsOut = [-0.166 0.834 0.000];
    otherwise
        TapWeightsOut = TapWeightsIn;
end
```

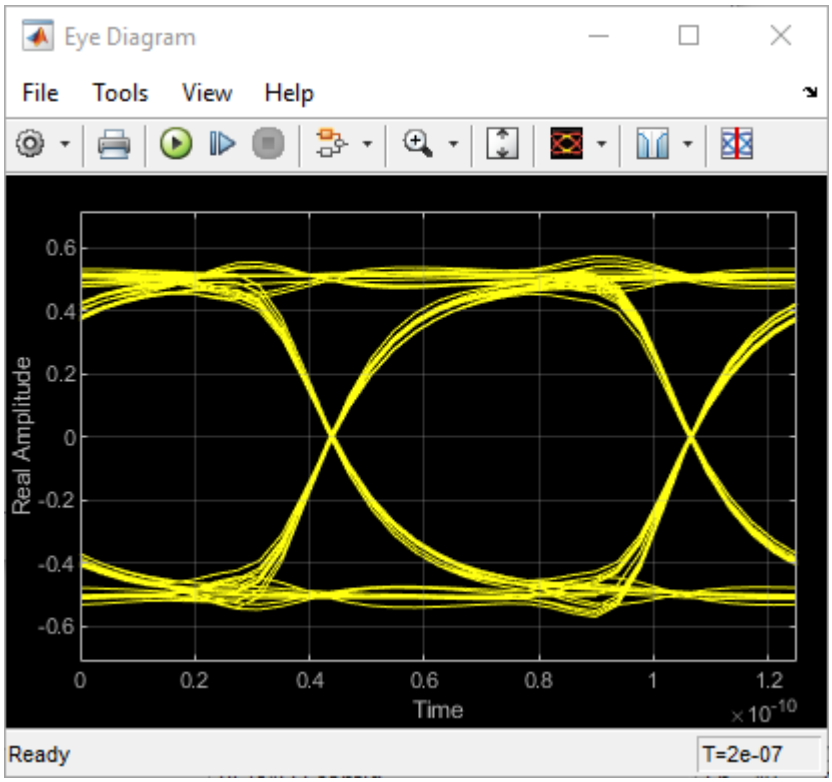
Re-wire the FFE sub-system so that the FFETapWeights and FFEConfigSelect constant blocks connect to the inputs of the newly defined PCIe4FFEconfig MATLAB function block. The TapWeightsOut signal from the PCIe4FFEconfig block connects to the **TapWeights** port of the FFE block.



To test that the new FFE control parameter is working correctly, open the SerDes IBIS-AMI Manager dialog box from the Configuration block. In the **AMI-Tx** tab, edit the **ConfigSelect*** parameter to set **Current value** to P7. This corresponds to PCIe Configuration P7: Pre = -0.100, Main = 0.700 and Post = -0.200. Observe the output waveform.



Next, set the **Current value** of the **ConfigSelect*** parameter to `User Defined`. This corresponds to user-defined tap weights: Pre = 0.000, Main = 1.000 and Post = 0.000. Observe how the output waveform changes.



Try different values of **ConfigSelect*** to verify proper operation. The time-domain eye opens and closes based on the amount of equalization applied by the FFE. How much the eye changes, and the tap values that create the most open eye varies based on the loss defined in the Analog Channel block.

Export the Tx IBIS-AMI Model

Verify that both Init and GetWave are behaving as expected, then generate the final IBIS-AMI compliant PCIe4 model executables, IBIS and AMI files.

Double-click the Configuration block to open the Block Parameters dialog box. Click the **Open SerDes IBIS-AMI Manager** button, then select the **Export** tab:

- Update the **Tx model name** to `pcie4_tx`.
- **Tx and Rx corner percentage** is set to `10`. This will scale the min/max analog model corner values by +/-10%.
- Verify that **Dual model** is selected as **Model Type** for the Tx. This will create model executables that support both statistical (Init) and time domain (GetWave) analysis.
- Set the Tx model **Bits to ignore** parameter to 3 since there are three taps in the Tx FFE.
- Set the **Models to export** to **Tx only**.
- Set the **IBIS file name (.ibs)** to `pcie4_tx_serdes.ibs`

- Click the **Export** button to generate models in the **Target directory**.

Test Generated IBIS-AMI Model

The PCIe4 transmitter IBIS-AMI model is now complete and ready to be tested in any industry standard AMI model simulator.

References

PCI-SIG.

See Also

FFE | PassThrough | **SerDes Designer**

More About

- “Managing AMI Parameters” on page 6-2
- “PCIe4 Transmitter/Receiver IBIS-AMI Model” on page 7-2

Customizing Datapath Building Blocks

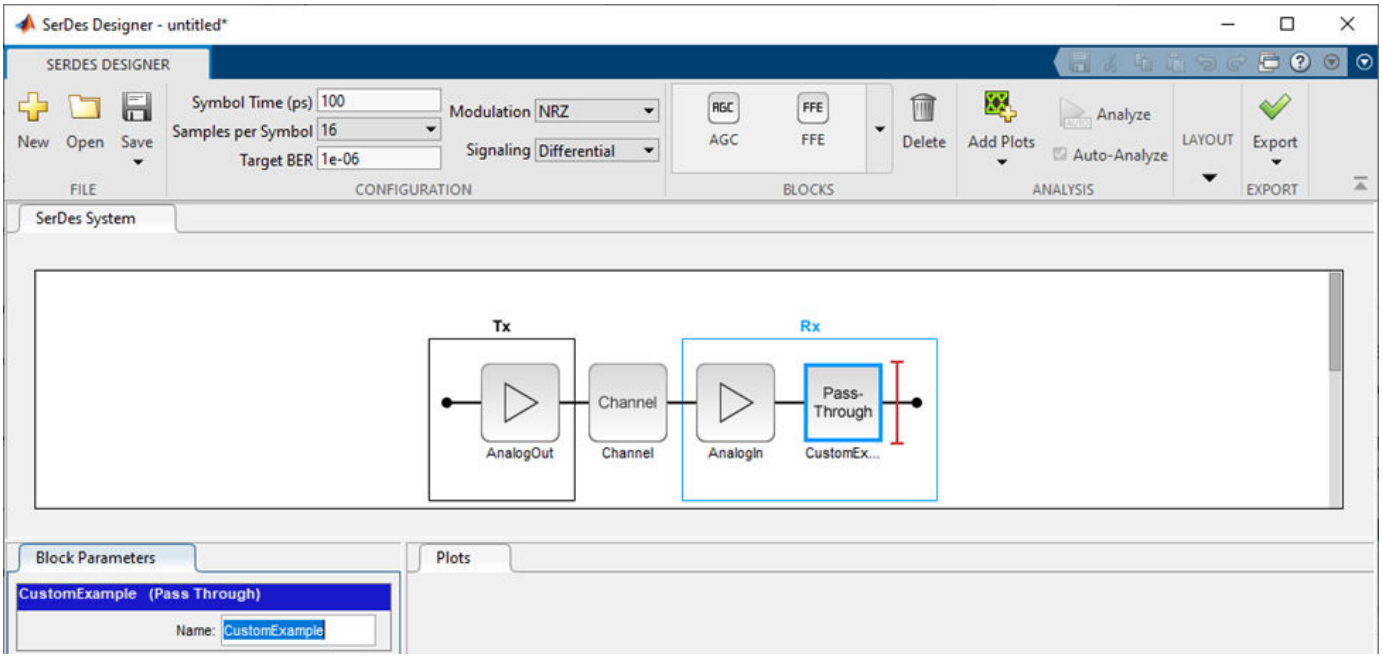
This example shows how to customize a PassThrough block in Simulink® using other Simulink library blocks. This example shows the implementation of a receiver gain or attenuation stage controlled by an IBIS-AMI parameter. You can also use this example as a guide to modify PassThrough blocks to implement custom functions for a SerDes system.

PassThrough Block Function and Use

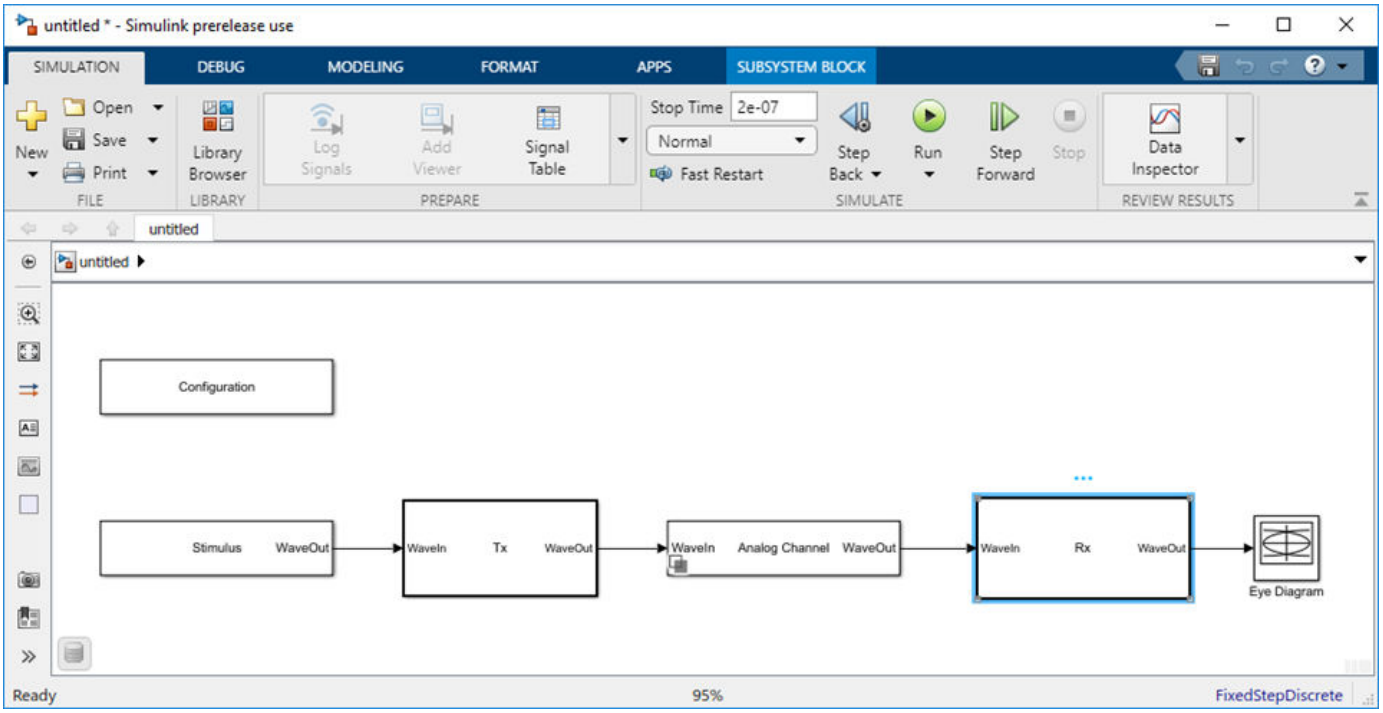
By default, PassThrough block is, as the name implies, a block that passes the input impulse or waveform to the output with no modifications. This block can be used as a floor planning tool in the SerDes Designer App and then customized after exporting to Simulink. Under the mask of a PassThrough block is a MATLAB® System block referencing the serdes.PassThrough System object™, which when called by Simulink forwards the input to the output. The MATLAB System block can be updated to reference other SerDes System objects or can be replaced with other Simulink blocks as this example outlines. For an example of customizing with System objects, see “Implement Custom CTLE in SerDes Toolbox PassThrough Block” on page 5-20.

Create SerDes System in SerDes Designer App

Launch the SerDes Designer app. Place a PassThrough block after the analog model of the receiver. Change the name of the PassThrough block from `PT` to `CustomExample`.

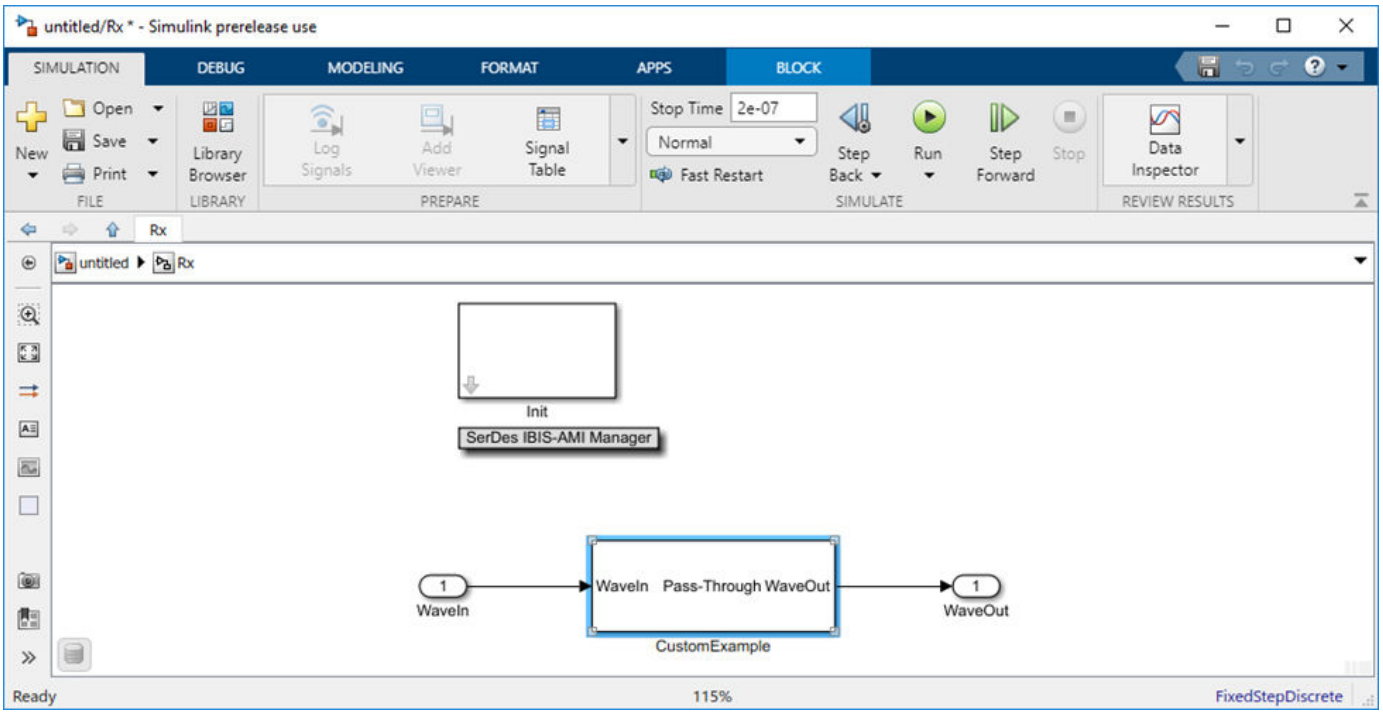


Export the SerDes system to Simulink.



Add AMI Parameter to Control Gain

Double click on the Rx block to look inside the Rx subsystem and open the SerDes IBIS-AMI Manager dialog box.



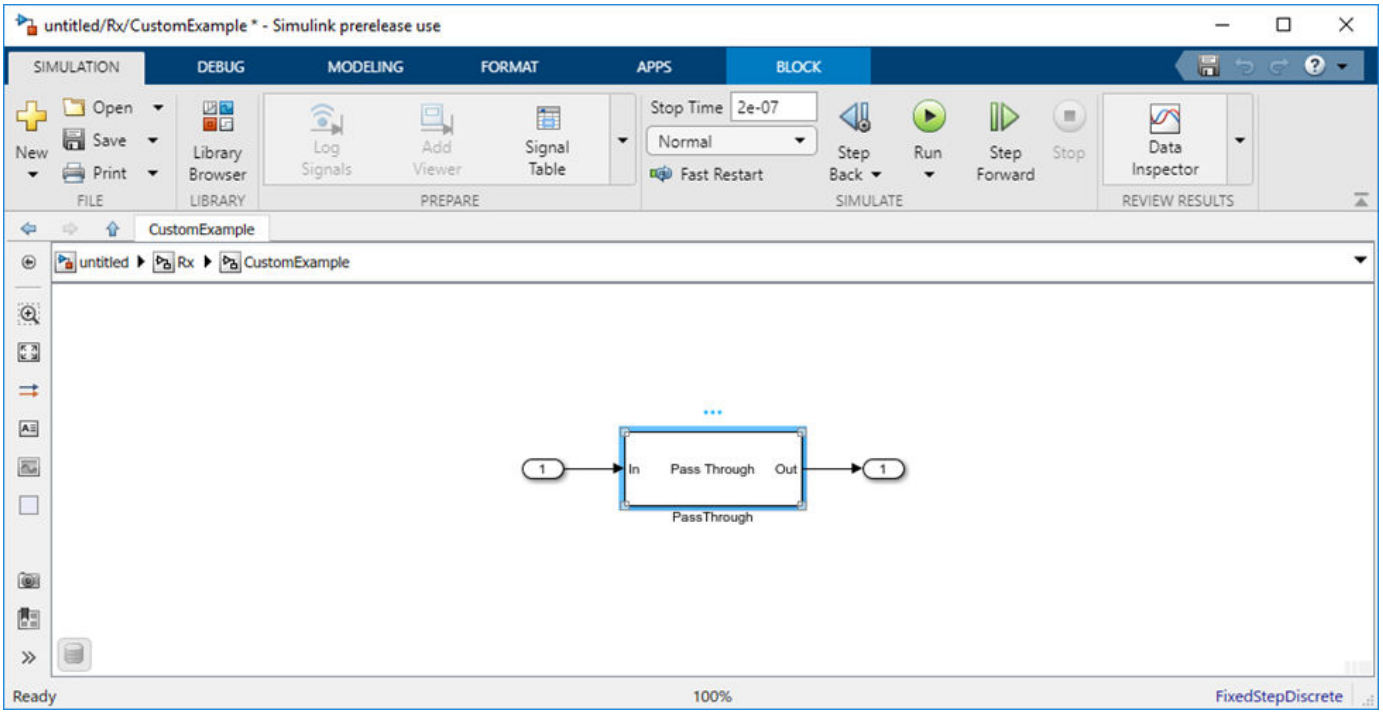
In the **AMI-Rx** tab, select the **CustomExample** node. Click on the **Add Parameter** button and set the variables:

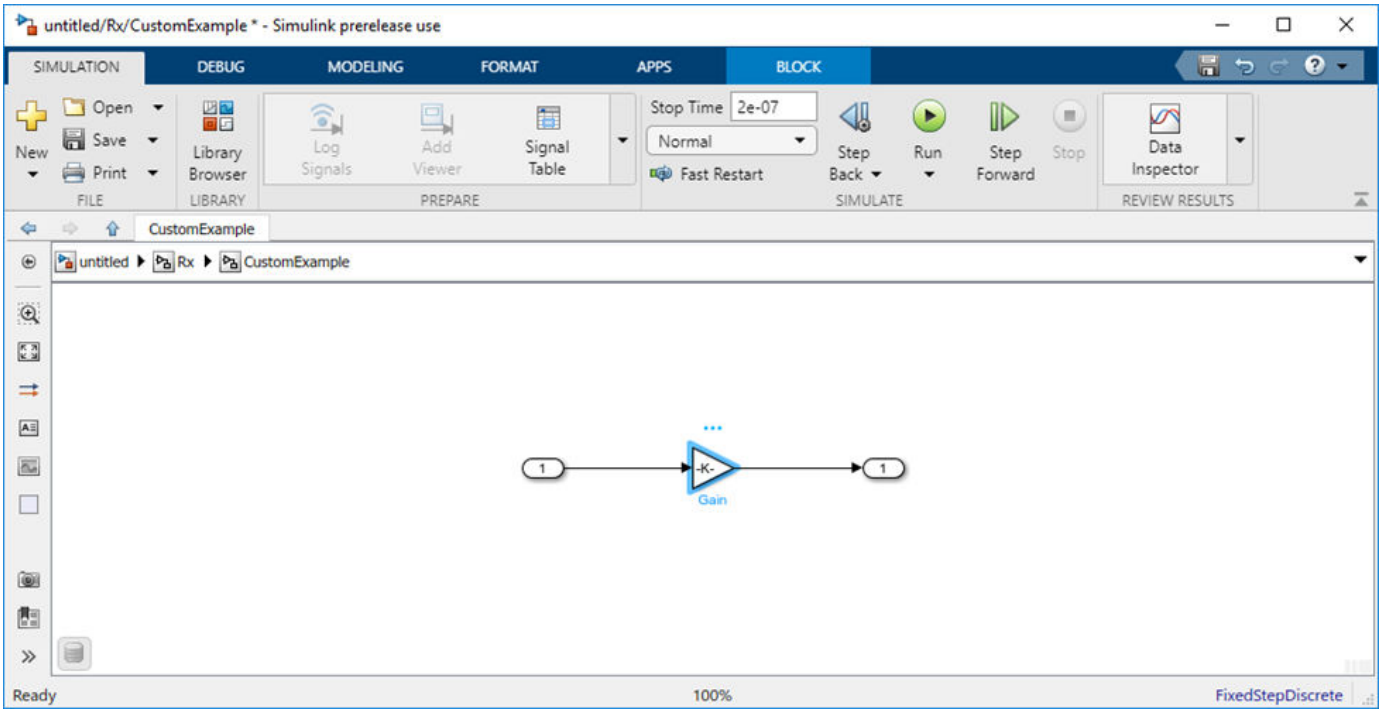
- **Parameter name** to `ExampleGain`
- **Description** to `Gain setting for Receiver`
- **Format** to `Range`
- **Typ** to `0.8`
- **Min** to `0`
- **Max** to `1`.

Current value, **Usage**, and **Type** are kept at their default values `0`, `In`, and `Float`, respectively.

Add Gain Block to PassThrough

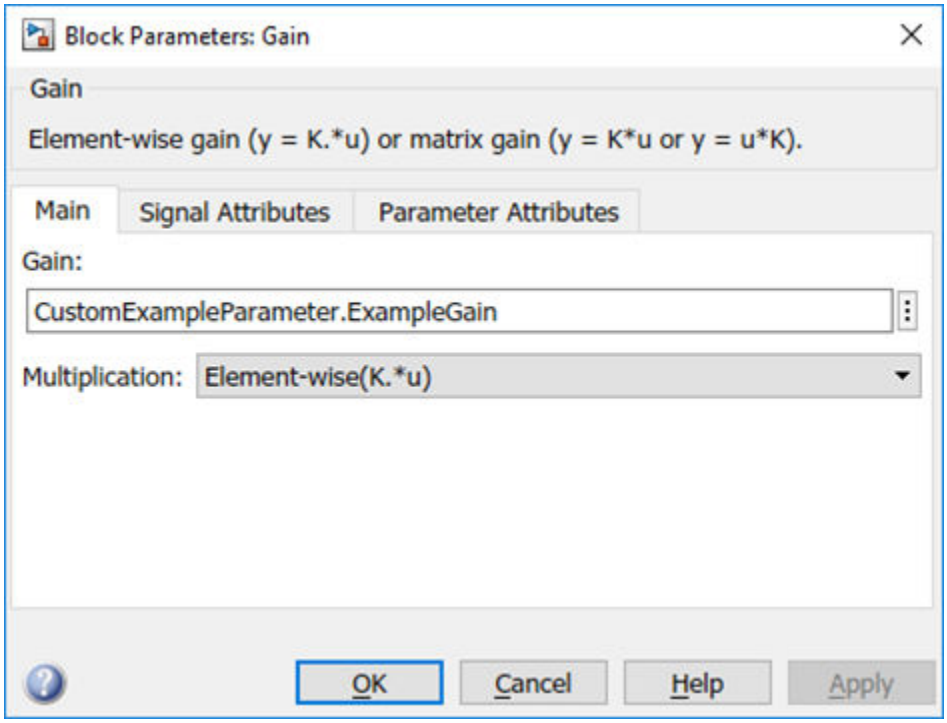
To configure the time-domain portion of the model, a Gain block is added within the PassThrough block. Look under the mask of the PassThrough block. Delete the MATLAB System block that points to the serdes.PassThrough System Object. Add a Gain block from the Simulink > MathOperators library and connect the Gain block between the input and output ports.





Connect Block Parameters of Gain Block to Added AMI Parameter

Constants are represented as Simulink parameters. Double click the Gain block to open the Block Parameters dialog box. Set **Gain** value to `CustomExampleParameter.ExampleGain`.



Update Code that Runs During Statistical Analysis

To enable the gain to be applied to the impulse response during statistical analysis, double click the Init block inside the Rx subsystem. Click the **Refresh Init** button to add the new AMI parameter to the Init code. Click the **Show Init** button to open the MATLAB editor window and look for the **Custom user code area** surrounded by `%%BEGIN` and `%END` comments. Your code associated with the customized PassThrough block is encapsulated in this section.

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
CustomExampleParameter.ExampleGain; % User added AMI parameter from SerDes IBIS-AMI Manager
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

Implement Gain

In the **Custom user code area**, edit your customized code to perform a Gain operation on the local variable containing the Impulse Response. To do this, replace the code `CustomExampleParameter.ExampleGain` with `LocalImpulse = LocalImpulse*CustomExampleParameter.ExampleGain`. Save the changes.

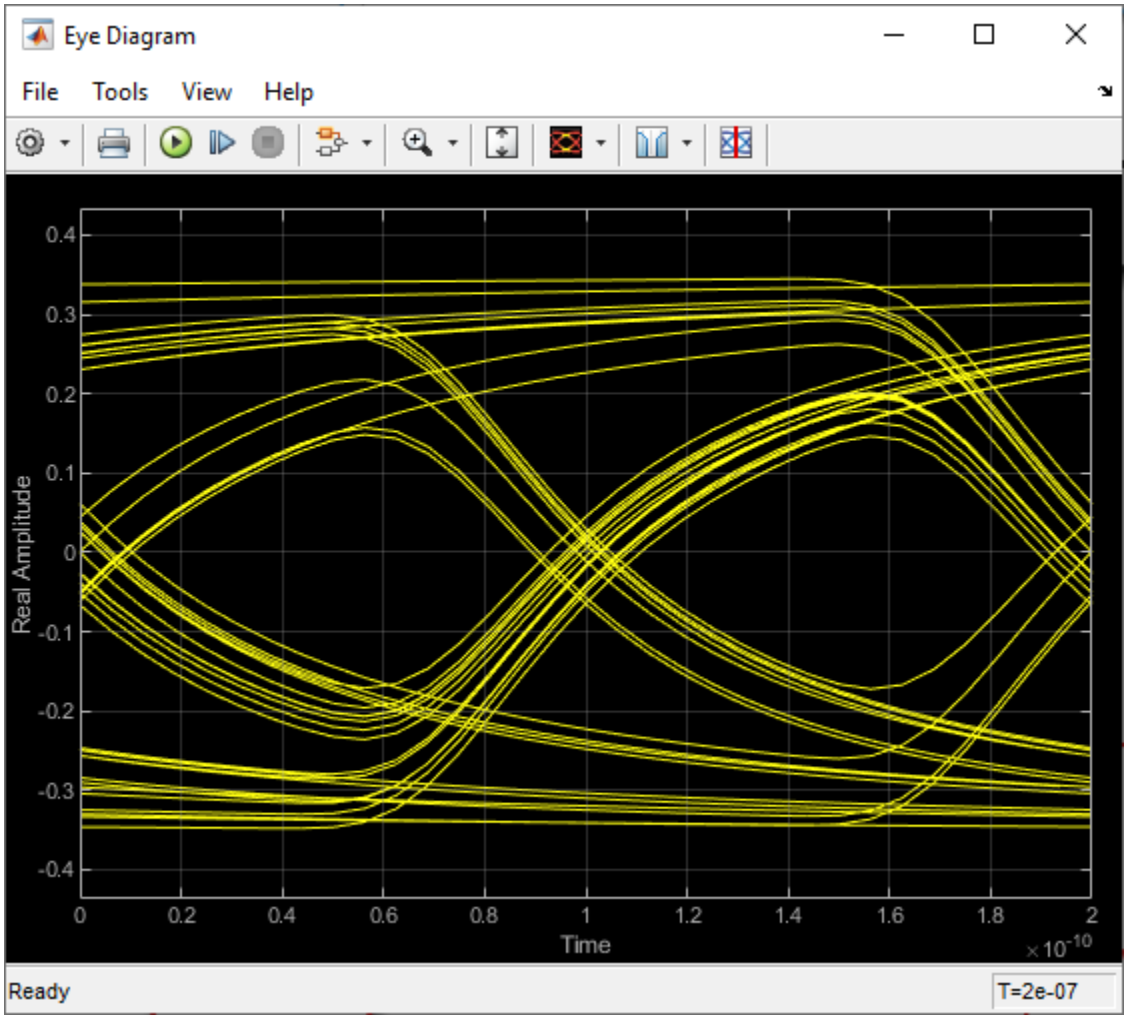
```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
LocalImpulse = LocalImpulse*CustomExampleParameter.ExampleGain; % User added AMI parameter from SerDes IBIS-AMI Manager
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

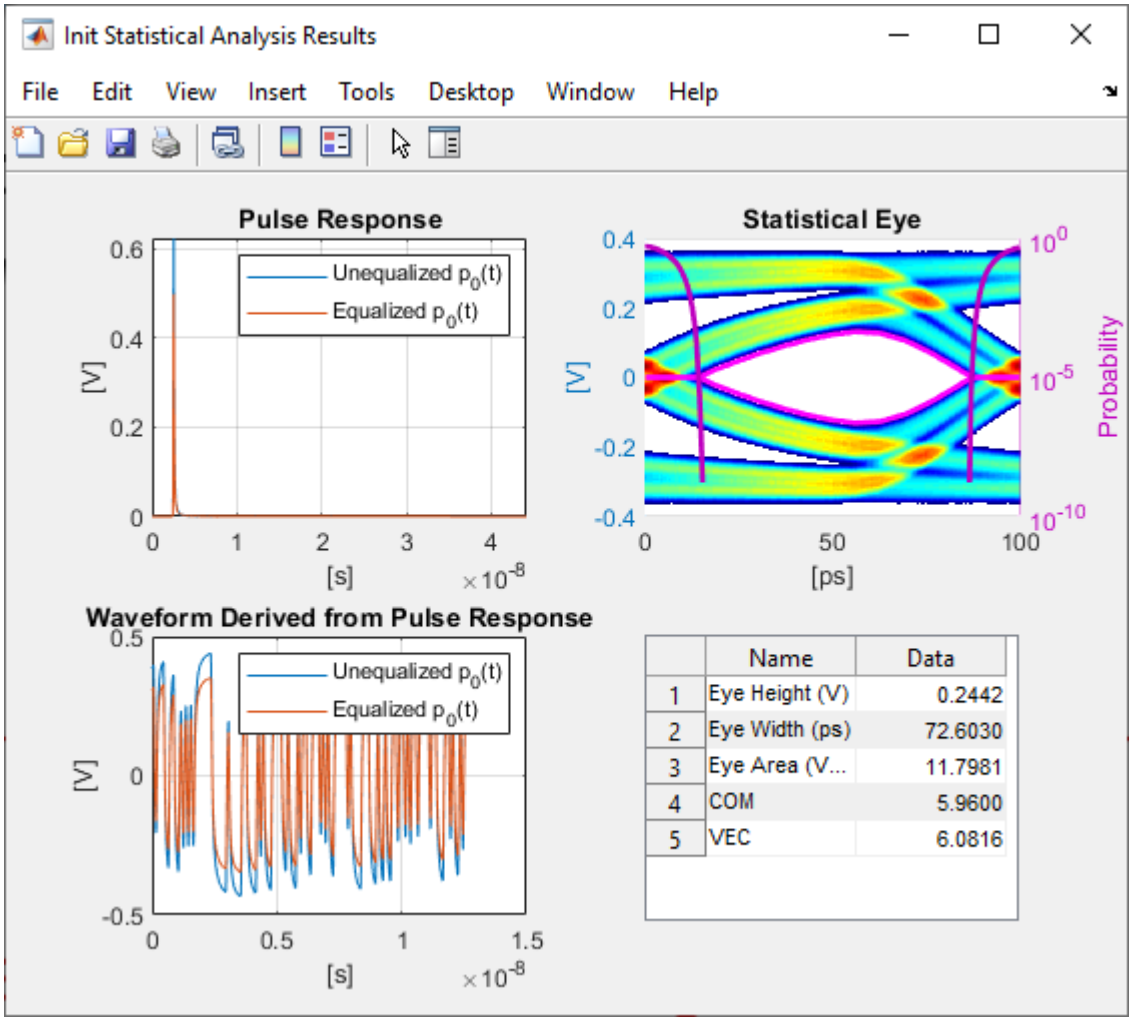
Note: If Init code is not modified, results from the Statistical simulation does not reflect the gain operation and is only shown in the results from the Time-Domain (GetWave) simulation.

Run Simulation with Gain Setting

Open the SerDes IBIS-AMI Manager dialog box and click on the **AMI-Rx** tab. Select the **ExampleGain*** node and set the **Current value** to `0.8`.

Run the simulation and observe amplitude of the waveform from Time-Domain (GetWave) and the waveform from Statistical (Init) results.

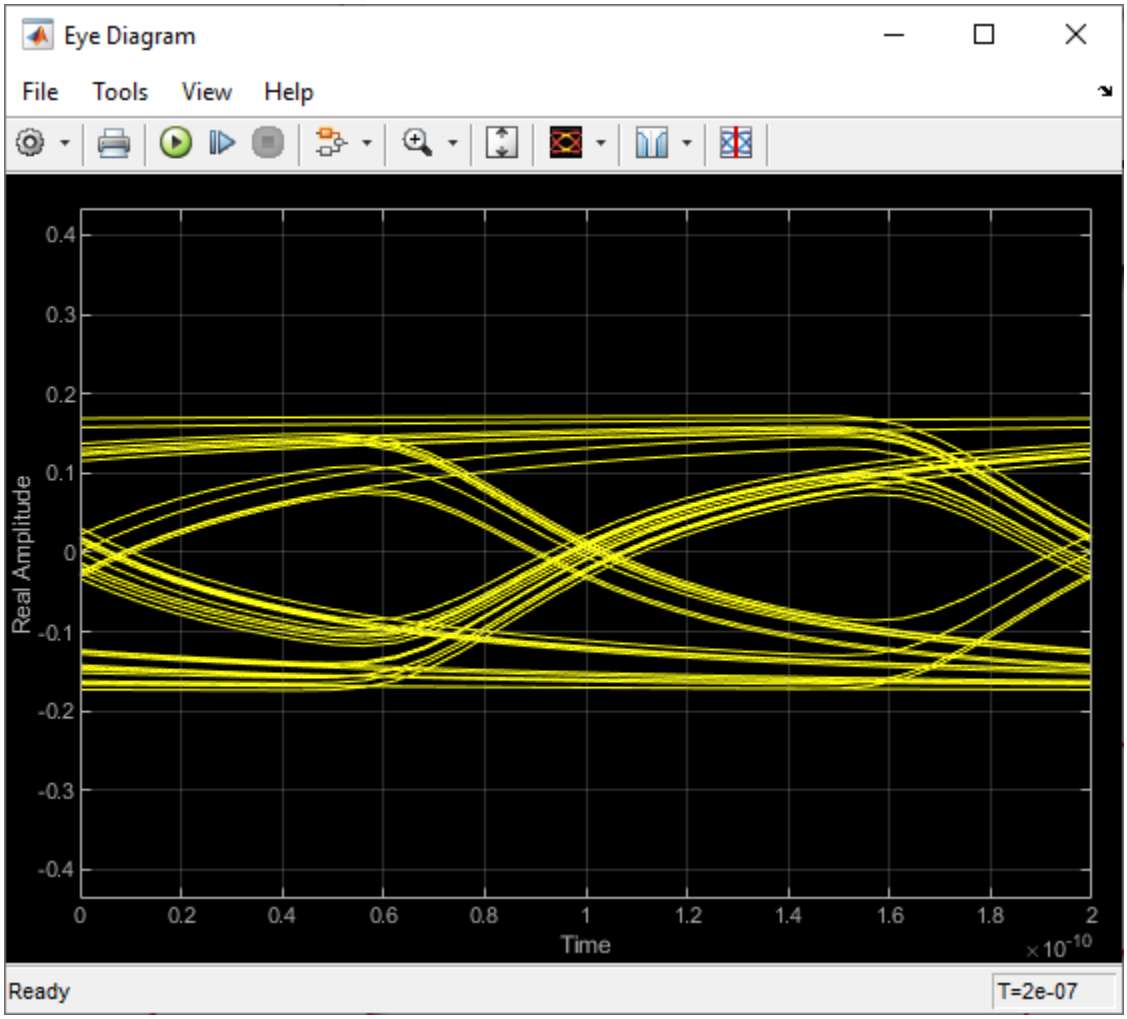

Eye Diagram
File
Tools
View
Help
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
Real Amplitude
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Time
×10^-10
Ready
T=2e-07

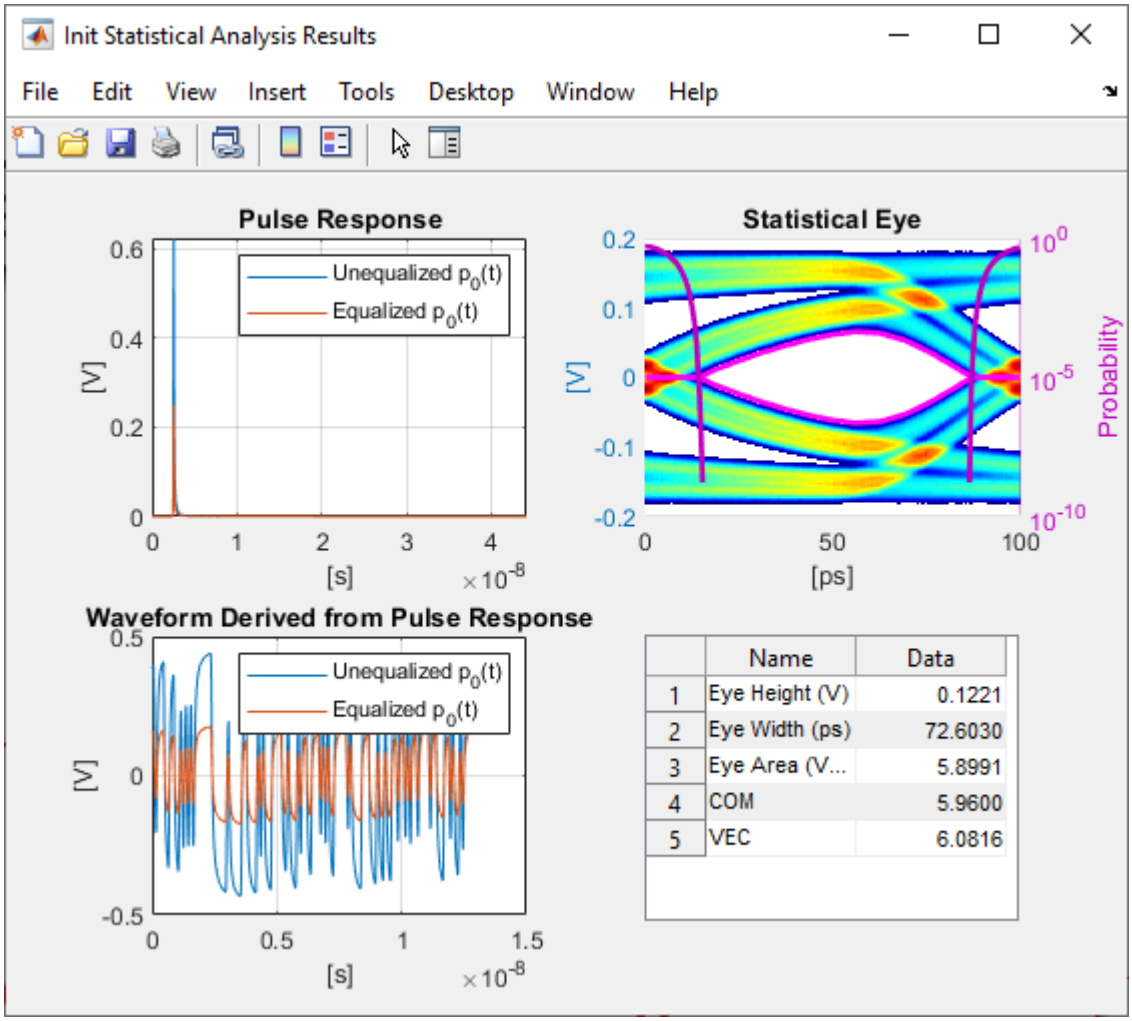


Change Gain Setting and Observe Change

Open the SerDes IBIS-AMI Manager dialog box and click on the **AMI-Rx** tab. Select the **ExampleGain*** node and set the **Current value** to `0.4`.

Run the simulation again and observe how the amplitude changes for both the waveform from Time-Domain (GetWave) and the waveform from Statistical (Init).


Eye Diagram
File
Tools
View
Help
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
Real Amplitude
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Time
×10^-10
Ready
T=2e-07



See Also

Configuration | PassThrough | **SerDes Designer**

More About

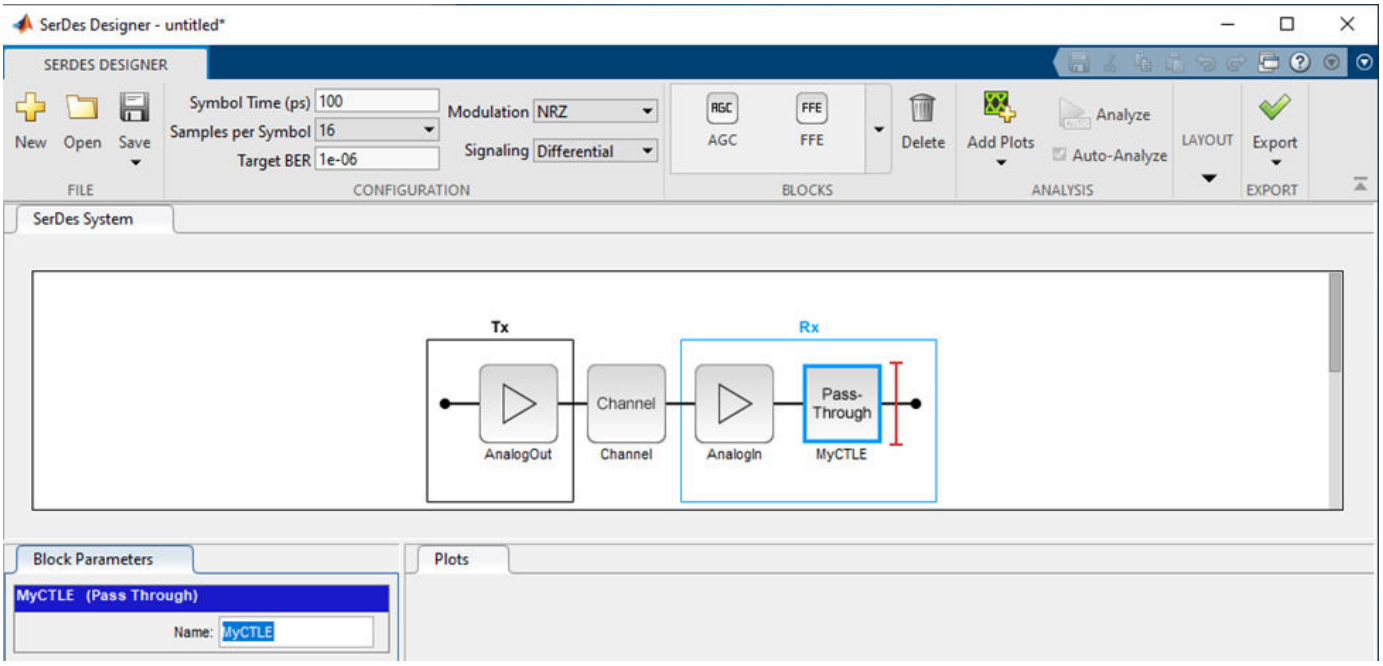
- “Implement Custom CTLE in SerDes Toolbox PassThrough Block” on page 5-20

Implement Custom CTLE in SerDes Toolbox PassThrough Block

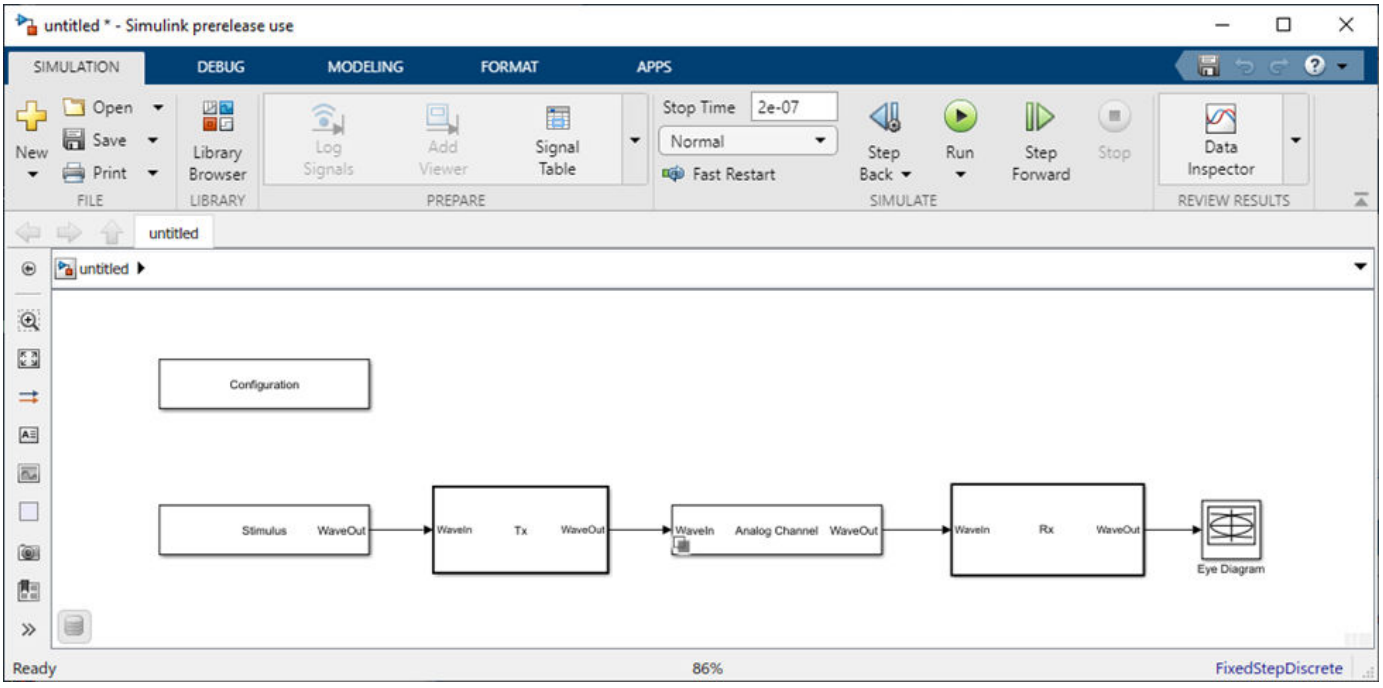
This example shows how to customize a PassThrough Block in Simulink® to implement a CTLE System Object™ with user defined AMI parameters. You can use this example as a guide for modifying PassThrough blocks that leverage system objects. For more information on the purpose of the PassThrough block and an example of using other Simulink library blocks within them, see “Customizing Datapath Building Blocks” on page 5-11.

Create SerDes System in SerDes Designer App

In MATLAB®, type `serdesDesigner` to launch the SerDes Designer app. Place a PassThrough block after the analog model in the receiver. Change the name of the PassThrough block from `PT` to `MyCTLE`.



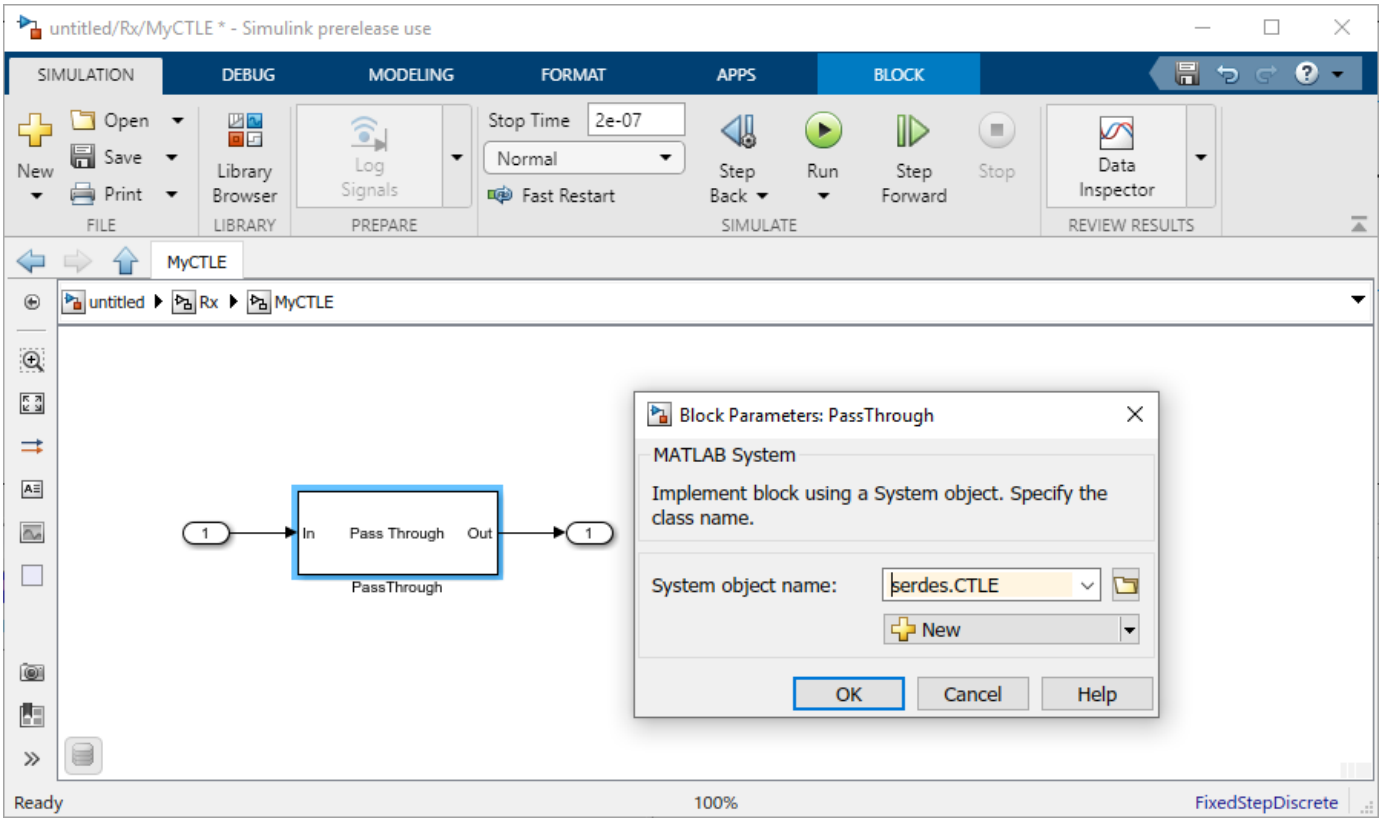
Export the SerDes system to Simulink.



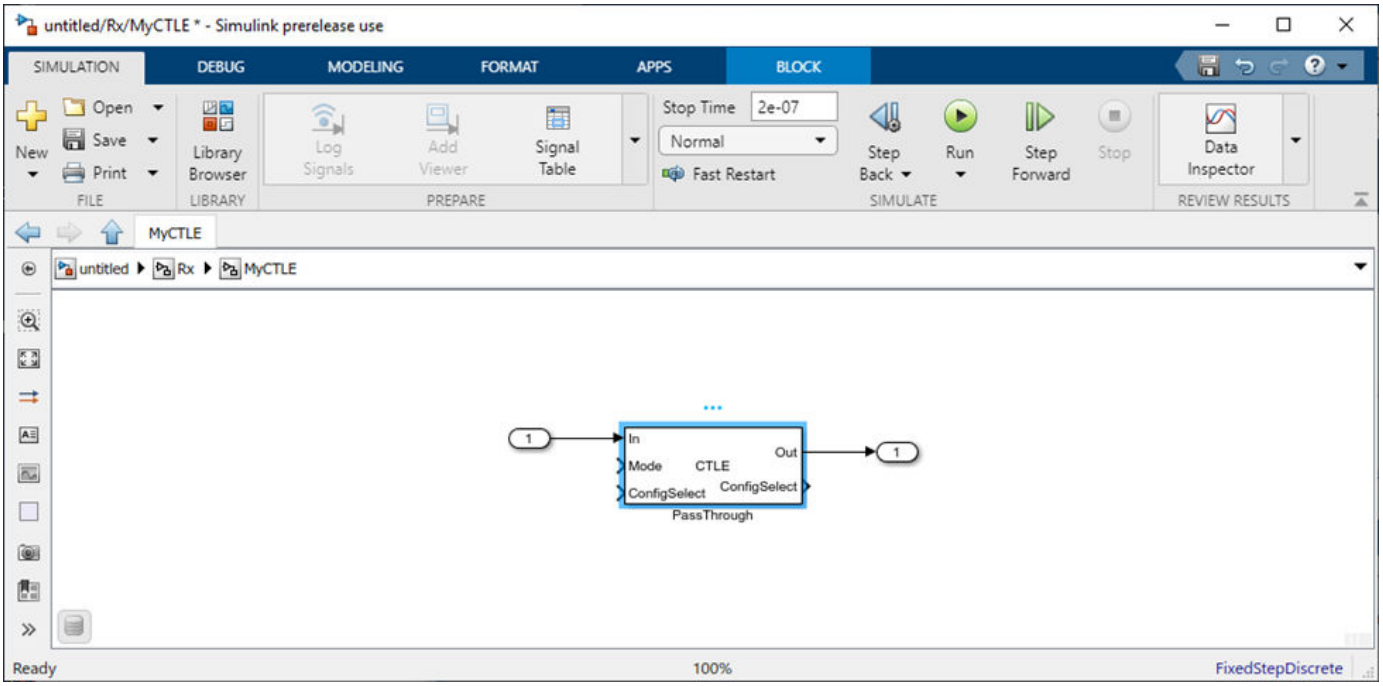
Modify PassThrough Block to Implement CTLE

This example builds a custom replica of the CTLE bloc from SerDes Toolbox™. First modify the contents of PassThrough block to reference a new system object and then implement and connect its parameters. This addresses the time-domain (GetWave) function of the model. The Init code is then updated to mirror the functionality of time-domain (GetWave) in the statistical analysis. This example walks you through the whole process using `serdes.CTLE` System object.

Inside the Rx subsystem, look under mask of PassThrough block `MyCTLE`. Select the PassThrough block, press **Ctrl+U** to open the Block Parameters dialog box of the MATLAB System, and change the **System object name** from `serdes.PassThrough` to `serdes.CTLE`.



Click **OK** to save the changes.



Note: You can use your own custom System object as well. For example, if you wanted to create a custom CTLE with a change in the adaptation algorithm:

1. Open the source code of `serdes.CTLE`.
2. Save a local copy of the source code in a directory.
3. Make the desired changes in the code.
4. Then reference the customized code with the MATLAB System.

To properly link the CTLE to the system-wide parameters **SymbolTime** and **SampleInterval**, you need to set the CTLE to use these parameters as variables rather than hard-coded values. Otherwise incorrect or unexpected values may be included in the simulation and result in invalid data. Double click the PassThrough block that now points to the CTLE system object to open the Block parameters dialog window. In the Advanced tab, set **Symbol time (s)** to `SymbolTime` and **Sample interval (s)** to `SampleInterval`. Click **OK** to save the changes.

Add AMI Parameters to PassThrough Block

Open the SerDes IBIS-AMI Manager dialog box. Under the `Model_Specific` parameters in the **AMI-Rx** tab, select the **MyCTLE** node and add two new parameters, **CTLEMode** and **CTLEConfigSelect**.

To add **CTLEMode** parameter, click on the **Add Parameter** button and set the variables:

- **Parameter name** to `CTLEMode`
- **Current value** to `0`
- **Description** to `CTLE Mode: 0 = off, 1 = fixed, 2 = adapt`
- **Type** to `Integer`
- **Format** to `Range`
- **Typ** to `1`
- **Min** to `0`
- **Max** to 2.

Save the changes.

To add **CTLEConfigSelect** parameter, select the **MyCTLE** node again, click on the **Add Parameter** button and set the variables:

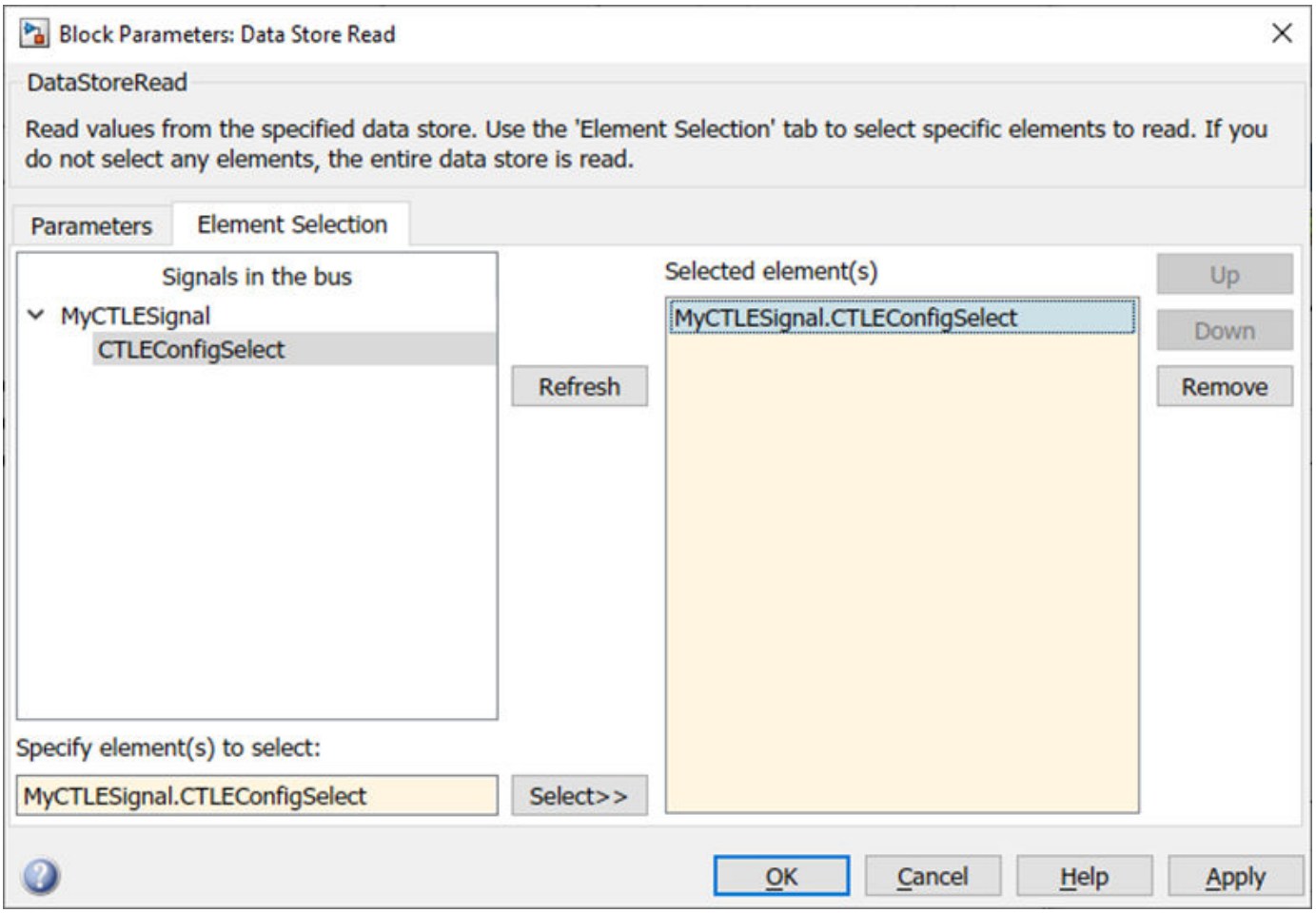
- **Parameter name** to `CTLEConfigSelect`
- **Current value** to `0`
- **Description** to `CTLE Config Select has a range from 0 to 8`
- **Usage** to `InOut`
- **Type** to `Integer`
- **Format** to `Range`
- **Typ** to `0`
- **Min** to `0`
- **Max** to 8.

Save the changes and close the SerDes IBIS-AMI Manager dialog box.

Implement AMI Parameters

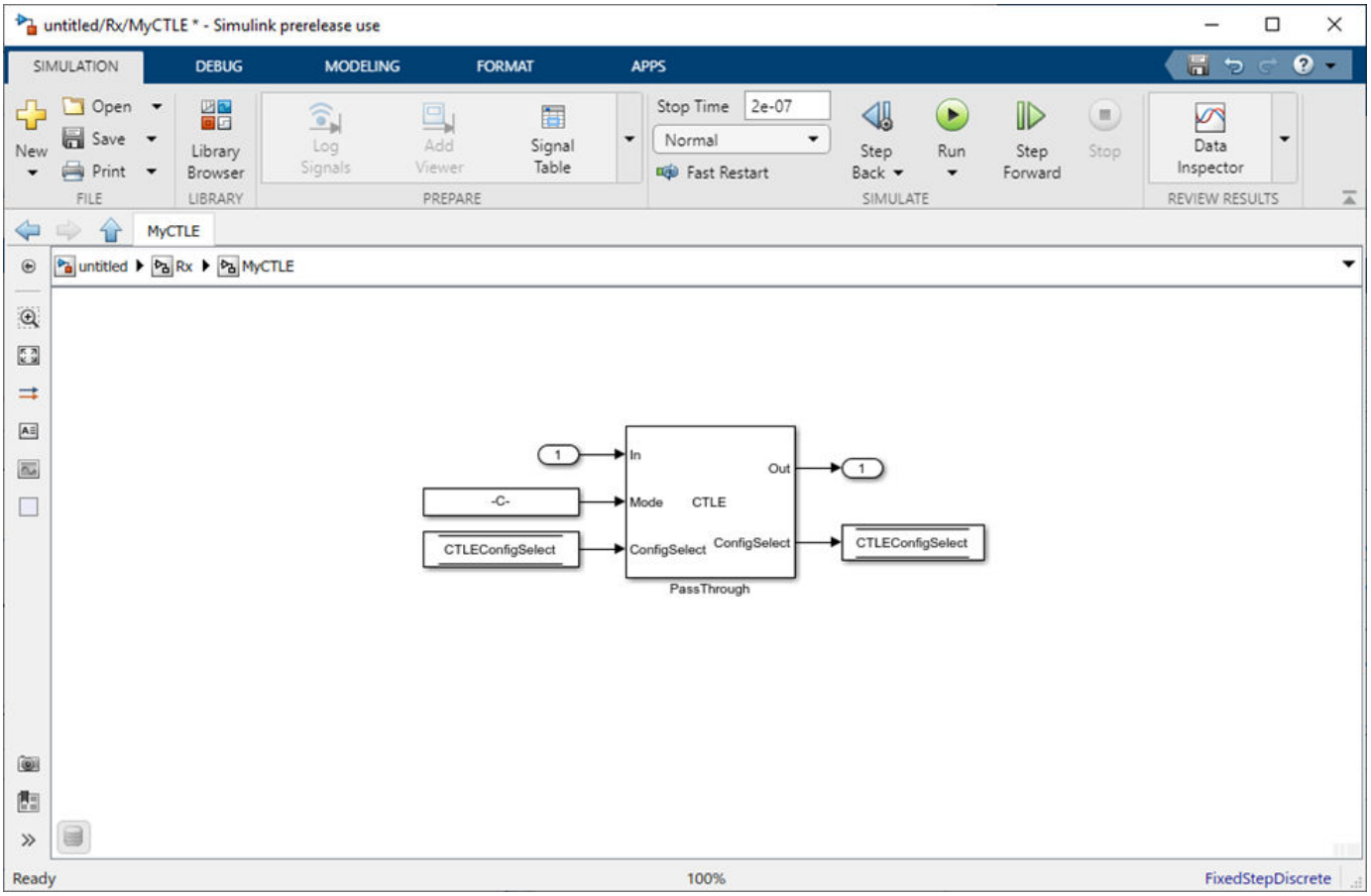
Look under the mask of the PassThrough Block. Add a Constant block from the **Simulink > Sources** library on the canvas. Double click the Constant block to open the Block Parameters dialog box and set the **Constant value** to `MyCTLEParameter.CTLEMode`. Connect the Constant block to the **Mode** input of the PassThrough block. For more information, see “Managing AMI Parameters” on page 6-2.

Add a Data Store Read block from the **Simulink > Signal Routing** library on the canvas. Double click the Data Store Read block to open the Block Parameters dialog box. In the **Parameters** tab, set **Data store name** to `MyCTLESignal`. In the **Element Selection** tab, expand **MyCTLESignal**, select **CTLEConfigSelect**, and press the **Select >>** button to add the selected element. Connect the Data Store Read block to the **ConfigSelect** input of the PassThrough block.



Add a Data Store Write block from the **Simulink > Signal Routing** library on the canvas. Double click the Data Store Write block to open the Block Parameters dialog box. In the **Parameters** tab, set **Data store name** to `MyCTLESignal`. In the **Element Selection** tab, expand **MyCTLESignal**, select **CTLEConfigSelect**, and press the **Select >>** button to add the selected element. Connect the Data Store Read block to the **ConfigSelect** input of the PassThrough block.

At this point, this is how the `MyCTLE` PassThrough block configuration should look. This completes setup for the time-domain (GetWave) simulation.



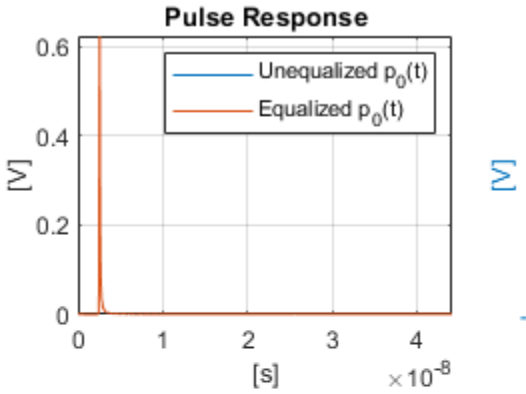
Verify Code for Statistical Analysis

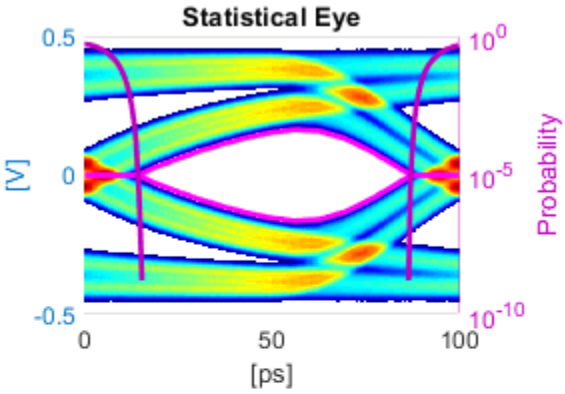
Double click the Init subsystem inside the Rx block to open the Block Parameter dialog box. To connect the AMI parameters as connected within the `MyCTLE` block, click the **Refresh Init** button. Since you used a system object, this connectivity is generated automatically. To verify this, click the **Show Init** button to open the MATLAB code for Init subsystem. You should find code related to the CTLE AMI parameter connections in the Custom user code area surrounded by the `%% Begin` and `% End` statements.

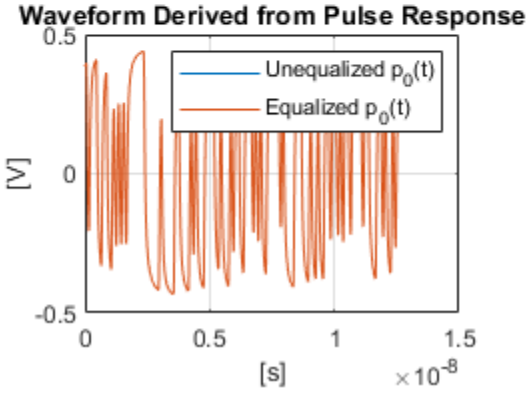
```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
MyCTLEInit.ConfigSelect = MyCTLEParameter.CTLEConfigSelect; % User added AMI parameter from SerDes IBIS-AMI Manager
MyCTLEInit.Mode = MyCTLEParameter.CTLEMode; % User added AMI parameter from SerDes IBIS-AMI Manager
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

Verify Operation of Custom CTLE

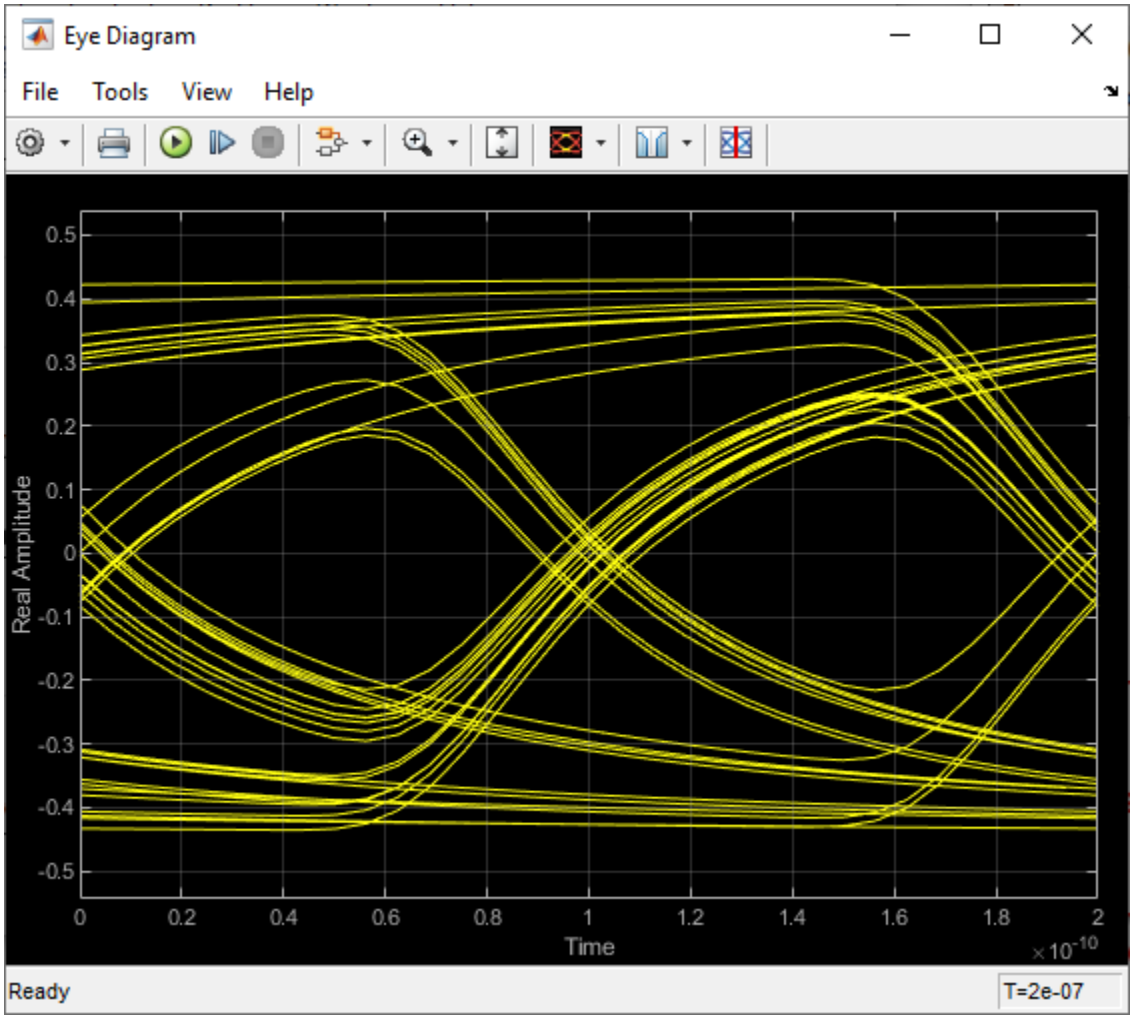
Run the simulation.



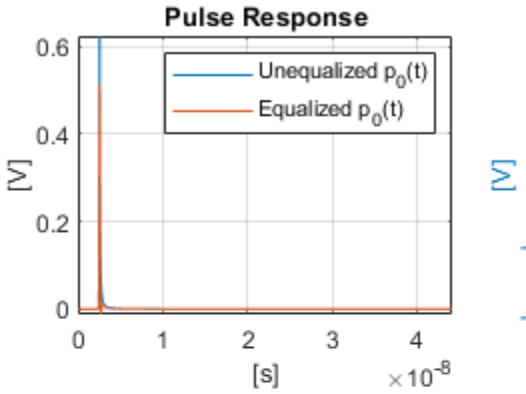


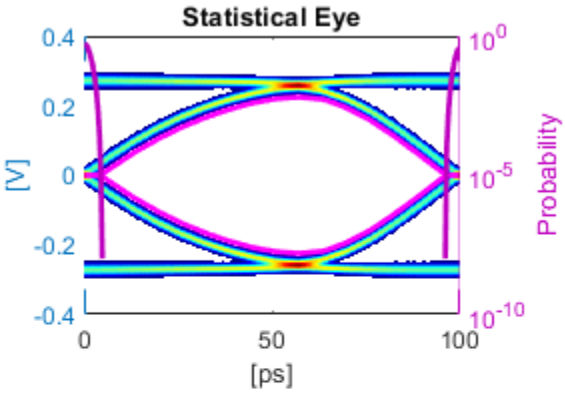


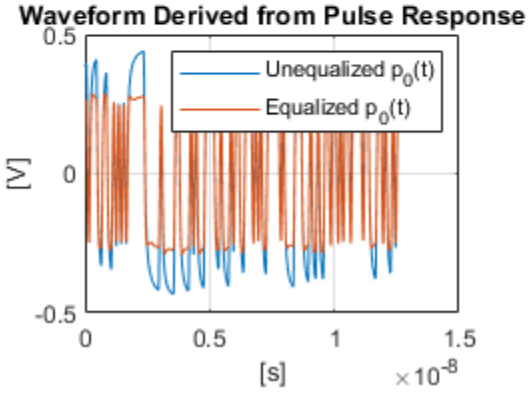
	Name	Data
1	Eye Height (V)	0.3053
2	Eye Width (ps)	72.6030
3	Eye Area (V...	14.7477
4	COM	5.9600
5	VEC	6.0816



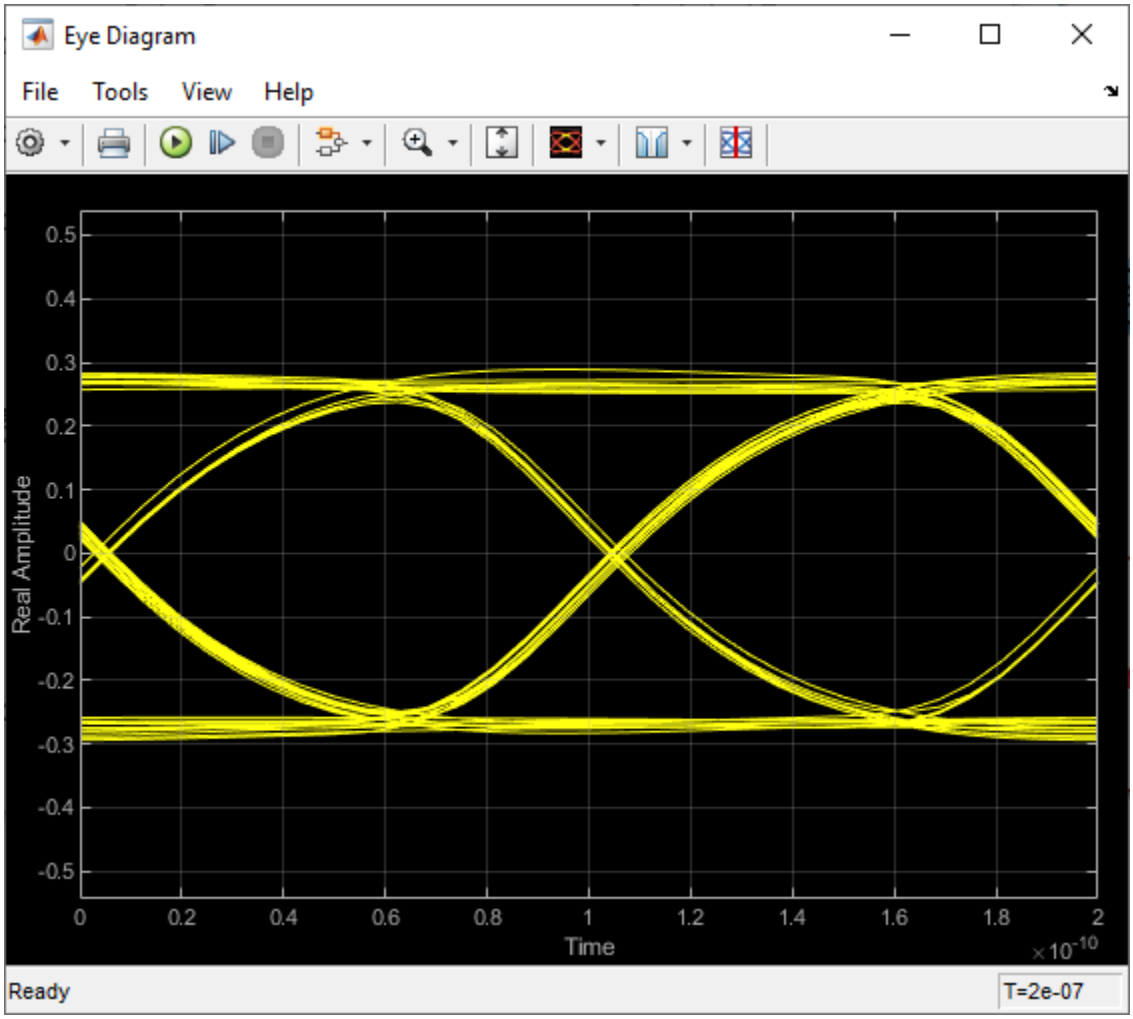
To evaluate the effect of the CTLE on output waveforms, open the SerDes IBIS-AMI manager dialog box. In the **AMI-Rx** tab, set **Current value** of **CTLEMode*** parameter to 1 to use fixed mode operation, and set **Current value** of **CTLEConfigSelect*** parameter to 4. Re-run the simulation.







	Name	Data
1	Eye Height (V)	0.4335
2	Eye Width (ps)	92.1840
3	Eye Area (V...	26.1905
4	COM	16.3293
5	VEC	1.4382



See Also

CTLE | Configuration | PassThrough | **SerDes Designer**

More About

- “Customizing Datapath Building Blocks” on page 5-11
- “Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance” on page 4-11

Customize IBIS-AMI Models

Managing AMI Parameters

This example shows how to add, delete, modify, rename and hide AMI parameters in SerDes Toolbox. These parameters are then available to be used with the existing datapath blocks, user created MATLAB function blocks or optimization control loop. These parameters can be passed to or returned from the AMI model executables (DLLs) created by SerDes Toolbox.

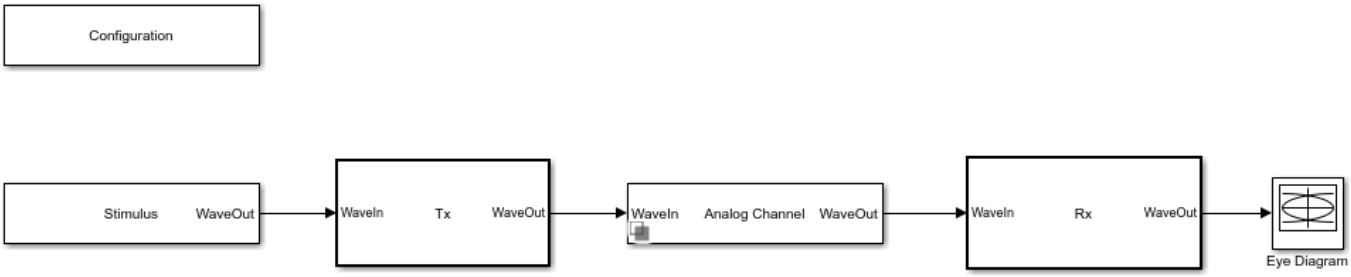
Example Setup

This example will be adding a new InOut Parameter 'Count' alongside the Pass-through datapath block. This parameter will count the number of passes through AMI_Init (which should be 1), then pass the result to AMI_GetWave where it will continue to count the total number of passes. While this may not be especially useful functionality for AMI model development, it will serve to demonstrate how new AMI parameters are added and used during model generation.

Inspect the Model

This example starts with a simple receiver model that only uses a pass-through block.

```
open_system('serdes_add_param.slx')
```

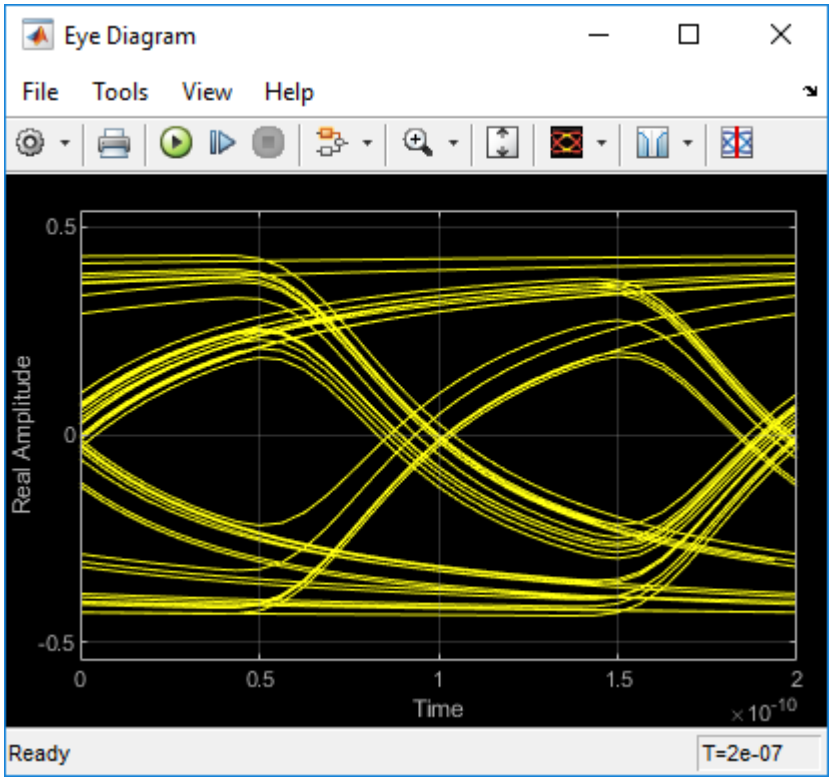


This Simulink SerDes System consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks.

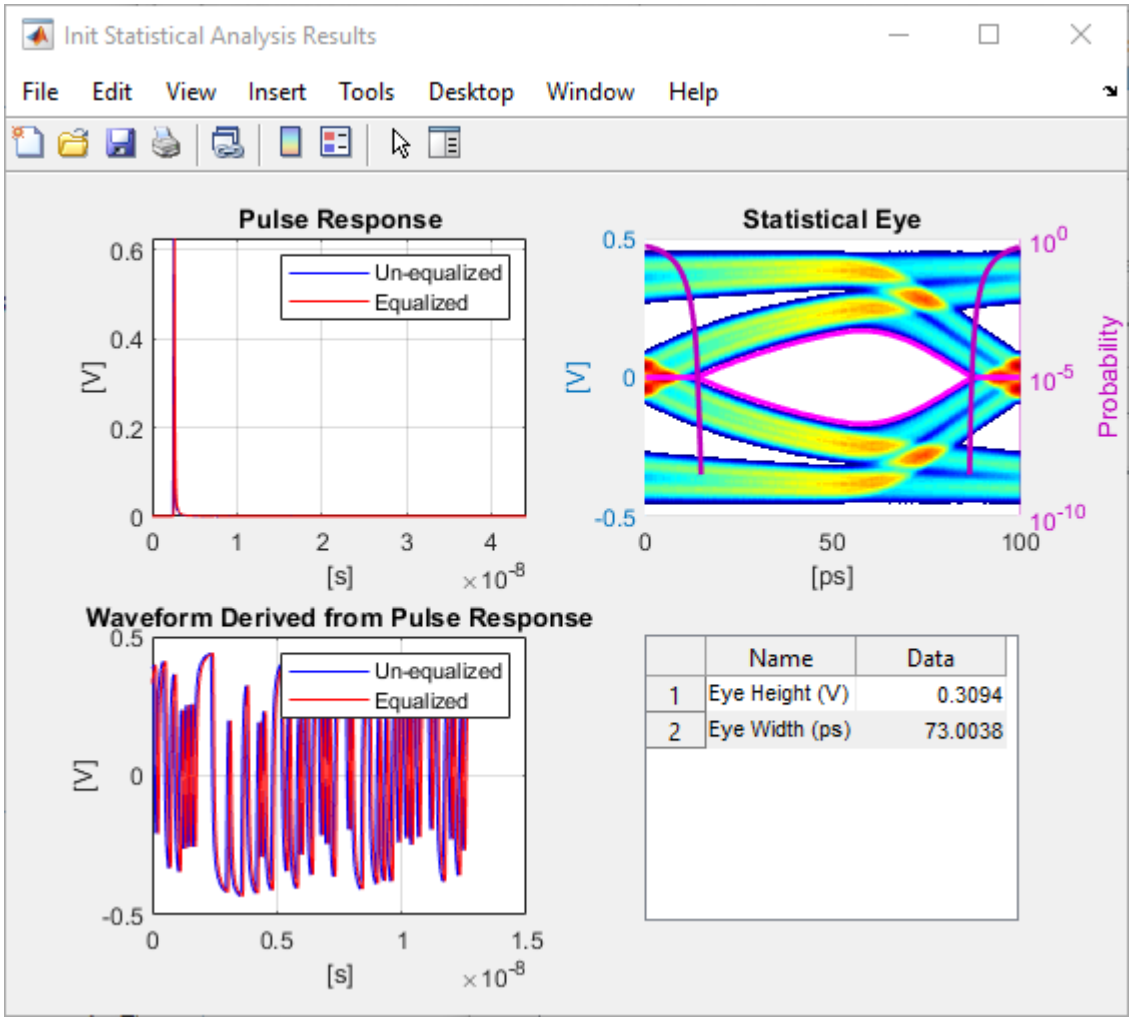
- The Tx subsystem has the FFE datapath block to model the time domain portion of the AMI model and an Init block to model the statistical portion. The Tx subsystem will not be used in this example.
- The Analog Channel block has the parameter values for Target frequency, Loss, Impedance and Tx/Rx analog model parameters.
- The Rx subsystem has the Pass-Through datapath block and an Init block to model the statistical portion of the AMI model.

Run the Model

Run the model to verify that the base configuration is working as expected before editing. Two plots are generated. The first is a live time domain (GetWave) eye diagram that is updated as the model is running.



The second plot contains four views of the statistical (Init) results.



How to Add a new Parameter

Open the Block Parameter dialog box for the Configuration block, then click on the **Open SerDes IBIS-AMI Manager** button and select the **AMI-Rx** tab.

1. Highlight the **PT** datapath block and press **Add Parameter...**

2. Change the **Parameter Name** to: `Count`

3. Verify that the **Current value** is set to `0` (this will be the starting point for our count).

4. In the **Description**, type: `Starting value of iteration count`.

There are four possible values for **Usage**:

- `In`: These parameters are required inputs to the AMI Executable.
- `Out`: These parameters are output from the AMI_Init and/or AMI_GetWave functions and returned to the EDA tool.
- `InOut`: These parameters are required inputs to the AMI Executable and can also return values from AMI_Init and/or AMI_GetWave to the EDA tool.

- `Info`: These parameters are information for the User and/or the simulation tool and are not used by the model.

5. Set the **Usage** to: `InOut`

There are six possible parameter **Types**:

- `Float`: A floating point number.
- `Integer`: Integer numbers without a fractional or decimal component.
- `UI`: Unit Interval (the inverse of the data rate frequency).
- `Tap`: A floating point number for use by Tx FFE and Rx DFE delay lines.
- `Boolean`: True and False values, without quotation marks.
- `String`: A sequence of ASCII characters enclosed in quotation marks.

6. Set the **Type** to: `Integer`

There are three possible parameter **Formats**:

- `Value`: A single data value.
- `List`: A discrete set of values from which the user may select one value.
- `Range`: A continuous range for which the user may select any value between Min and Max.

7. Set the **Format** to: `Value`

8. Hit **OK** to create the new parameter, then close the SerDes IBIS-AMI Manager.

Accessing a new Parameter from the Initialize Function

New parameters are accessed from the Initialize function (for statistical analysis) through the impulseEqualization MATLAB function block. This example has added an InOut parameter. To use the new InOut Parameter 'Count' in AMI_Init:

1. Inside the Rx subsystem, double click on the Init block to open the mask.

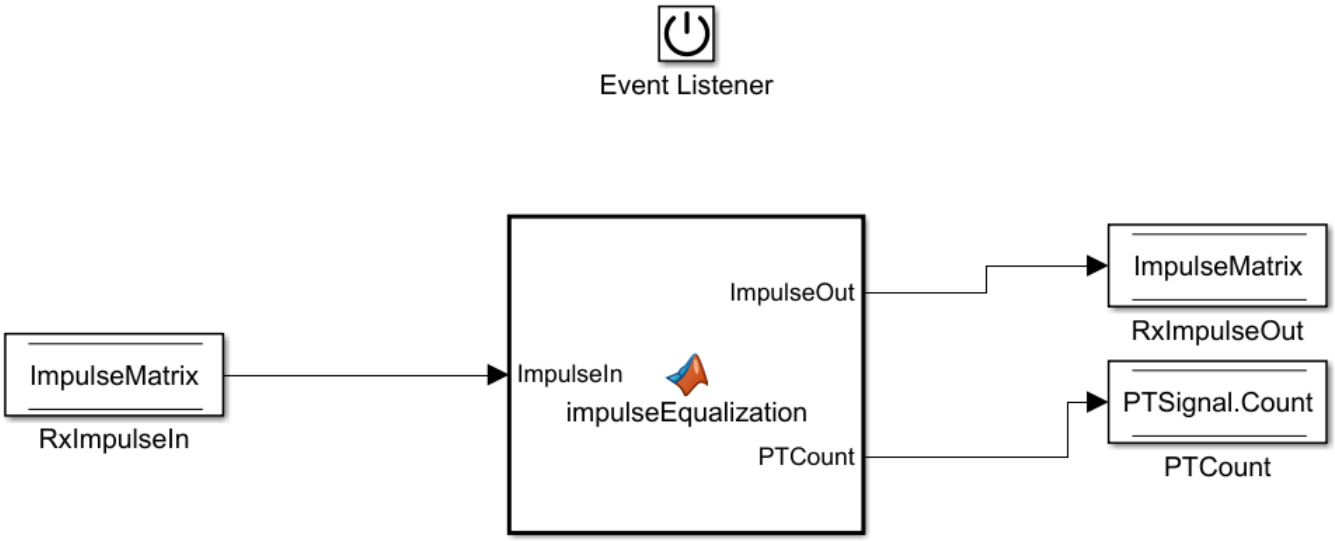
2. Press the **Refresh Init** button to propagate the new AMI parameter(s) to the initialize subsystem.

3. Click **OK** to close the mask.

4. Click on the Init block again and type **Ctrl-U** to look under the Init mask, then double-click on the initialize block to open the Initialize Function.

The impulseEqualization MATLAB function block is an automatically generated function which provides the impulse response processing of the SerDes system block (IBIS-AMI Init).

Note that the new Count parameter has been automatically added as an output of this MATLAB function as a Data Store Write block. No Data Store Read is required because the input parameters are passed in as a PTSignal Simulink.Parameter.



5. Double-click on the **impulseEqualization MATLAB function block** to open the function in MATLAB. The '%% BEGIN:' and '% END:' lines within this function block denote the section where custom user code can be entered. Data in this section will not get over-written when Refresh Init is run:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)

% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

When **Refresh Init** was run, it added our new parameter to the Custom user code area so that it can be used as needed:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
PTCount = PTParameter.Count; % User added AMI parameter from SerDes IBIS-AMI Manager
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

6. To add our custom code, scroll down to the Custom user code section, then enter `PTCount = PTCount + 1;` The Custom user code section should look like this:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
PTCount = PTParameter.Count; % User added AMI parameter from SerDes IBIS-AMI Manager
PTCount = PTCount + 1;  % Count each iteration through this function.
% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

7. Save the updated MATLAB function, then run the Simulink project to test the new code. Using the Simulation Data Inspector, verify that the value of Count after Init is now '1'.

Note that the final value for Count was written to the PTSignal data store so that it is now available in AMI_GetWave.

How Usage affects Parameters in Init

Depending on what Usage was selected, parameters show up in the Custom User code area of the impulseEqualization MATLAB function block in different ways:

Info Parameters

Info parameters are informational for the user or simulation tool and are not passed to, or used by the model, therefore they will not show up in the Initialize code.

In Parameters

In parameters are Simulink.Parameter objects that show up as a constant that can be used as needed. For example, an In parameter named 'InParam' that was added to the VGA block would show up as follows:

```
VGAParameter.InParam; % User added AMI parameter from SerDes IBIS-AMI Manager
```

Out Parameters

Out parameters are Simulink.Signal objects that show up as a parameter with the initial value defined in the IBIS-AMI Manager. For example, an Out parameter named 'OutParam' that was added to the VGA block with a current value of '2' would show up as follows:

```
VGAOutParam=2; % User added AMI parameter from SerDes IBIS-AMI Manager
```

Output parameters use a Data Store Write block to store values for passing out of Init to the EDA tool (via the AMI_Parameters_Out string) and for use in GetWave (if desired). In the above example, a Data Store Write block named 'OutParam' was automatically added to the Initialize Function:

InOut Parameters

InOut parameters use both a Simulink.Parameter object and a Simulink.Signal object. For example, an InOut parameter named 'InOutParam' that was added to the VGA block would show up as follows:

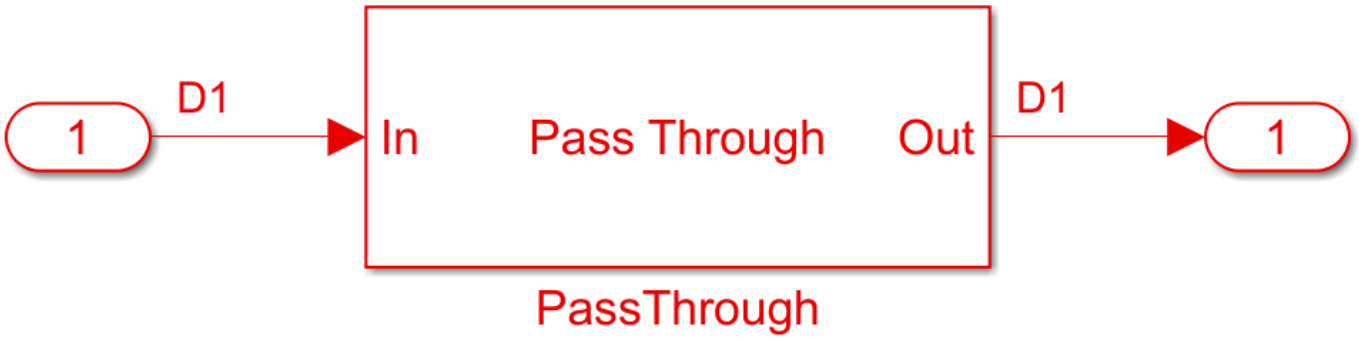
```
VGAInOutParam = VGAParameter.InOutParam; % User added AMI parameter from SerDes IBIS-AMI Manager
```

The Input value is accessed by using the Simulink.Parameter reference VGAParameter.InOutParam, while the output value uses a Data Store Write block to store values. In the above example, a Data Store Write block named 'InOutParam' was automatically added to the Initialize Function for passing values out of Init to the EDA tool (via the AMI_Parameters_Out string) and for use in GetWave (if desired):

Accessing a new Parameter from the GetWave Function

New parameters are accessed from the GetWave function (for time-domain analysis) by adding a Constant, Data Store Read or Data Store Write block to a datapath block. This example has added an InOut parameter. To use the new InOut Parameter 'Count' in GetWave:

1. Inside the Rx subsystem, click on the Pass-Through datapath block and type **Ctrl-U** to look under the Pass-Through mask.



2. Add a Simulink/Signal Routing **Data Store Read** block to the canvas

- Name the Data Store Read block: `PTCount_Read`
- Double-click on the Data Store Read block and change the Data store name to: `PTSignal`
- On the Element Selection tab, in the Specify element(s) to select box type: `PTSignal.Count` and press the **Select>>** button to select the Count element.
- Resize the block to make all names and element properties visible.
- Click **OK** to close the dialog.

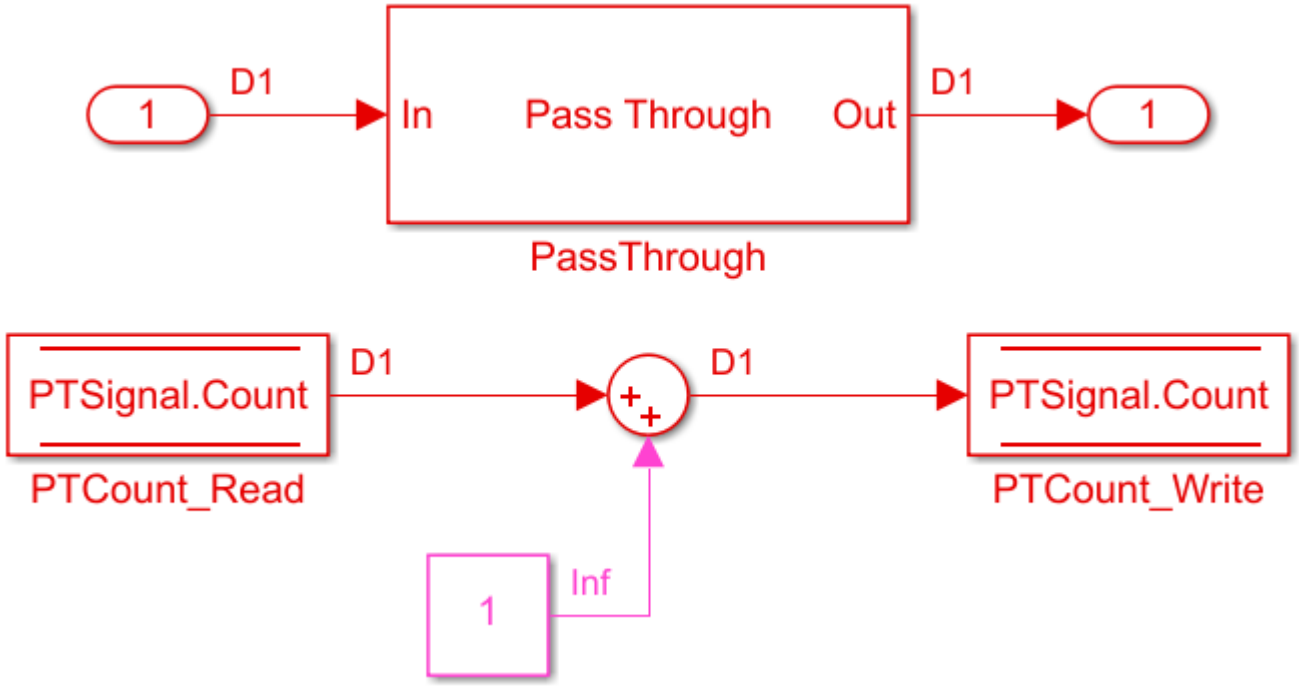
3. Add a Simulink/Math Operations **Sum** block to the canvas.

4. Add a Simulink/Sources **Constant** block to the canvas and set the value to `1`.

5. Add a Simulink/Signal Routing **Data Store Write** block to the canvas.

- Name the Data Store Write block: `PTCount_Write`
- Double-click on the Data Store Write block and change the Data store name to: `PTSignal`
- On the Element Assignment tab, in the Specify element(s) to select box type: `PTSignal.Count` and press the **Select>>** button to select the Count element.
- Resize the block to make all names and element properties visible.
- Click **OK** to close the dialog.

6. Wire up each of the elements so that the Pass Through block now looks like the following:



7. Save, then run the Simulink project to test the new code.

By adding Value Labels to the output port of the Sum block, see that the value of Count after GetWave is 3.2e+04 (Samples Per Symbol * Number of symbols). After generating AMI model executables, the value of Count will be available in the Parameters out string in an AMI simulator.

How Usage affects Parameters in GetWave

New parameters are accessed from the GetWave function in different ways, depending on what Usage was selected.

Info Parameters

Info parameters are informational for the user or simulation tool and cannot be used by the model.

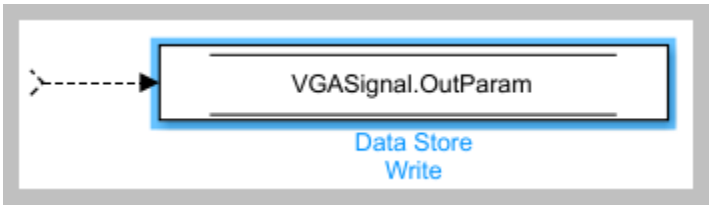
In Parameters

In parameters are Simulink.Parameter objects that are used by adding a Constant block. For example, an In parameter named 'InParam' that was added to the Rx VGA block can be accessed by any of the Rx blocks by adding a Constant block like this:

For more information, see “Customizing SerDes Toolbox Datapath Control Signals” on page 5-2.

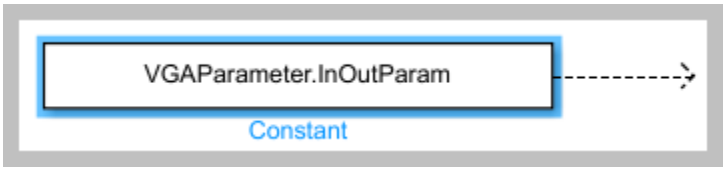
Out Parameters

Out parameters are Simulink.Signal objects that use a Data Store Write block to store values for passing out of GetWave to the EDA tool (via the AMI_Parameters_Out string) or to other Rx blocks. For example, an Out parameter named 'OutParam' that was added to the Rx VGA block can be written to with a Data Store Write block like this:

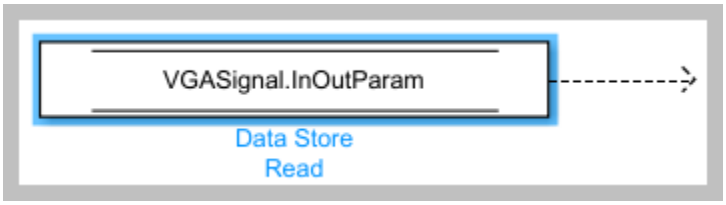


InOut Parameters

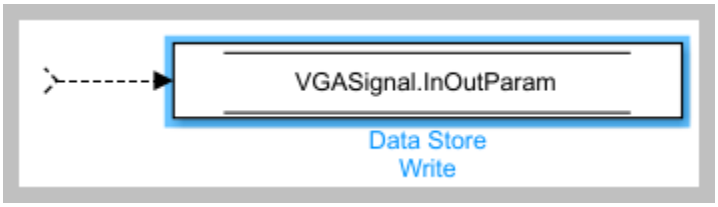
InOut parameters use both a Simulink.Parameter object and a Simulink.Signal object. The Input value can be accessed with either a constant block or with a Data Store Read block, while the output value uses a Data Store Write block to store values for passing out of GetWave to the EDA tool (via the AMI_Parameters_Out string) or to other Rx blocks. For example, if an InOut parameter named 'InOutParam' is added to the Rx VGA block, the initial Input value can be accessed by any Rx block by adding a Constant block like this:



Alternately, the updated Input value can be accessed with a Data Store Read block like this:



The output value can be written to with a Data Store Write block like this:



How to Rename a Parameter

The parameters used by the SerDes Toolbox built-in System Objects can be modified or hidden but cannot be renamed.

User generated AMI parameters are renamed as follows.

Update the AMI Parameters

1. Open the Block Parameter dialog box for the Configuration block, then click on the **Open SerDes IBIS-AMI Manager** button.

2. Go to either the **AMI-Tx** or **AMI-Rx** tab where the parameter resides.

3. Highlight the parameter to be renamed and press **Edit...**

4. In the Parameter name field, changed the name as desired.

5. Hit **OK**, then **Close** the SerDes IBIS-AMI Manager

Update Init

1. Push into either the Tx or Rx subsystem block where the parameter is used.

2. Double click on the Init block to open the mask.

3. Press the **Refresh Init** button to propagate the AMI parameter name change to the initialize subsystem.

4. Click **OK** to close the mask.

5. Click on the Init block again and type **Ctrl-U** to look under the Init mask, then double-click on the initialize block to open the Initialize Function.

6. Double-click on the **impulseEqualization MATLAB function block** to open the function in MATLAB.

7. Scroll down to the section titled:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
```

8. Rename all instances of the parameter.

9. Save and close the MATLAB function block.

Update GetWave

Push into each datapath block where the renamed parameter was used and rename each instance of the parameter.

Verify Results

Run a simulation to verify that the project still operates with no errors or warnings.

How to Delete a Parameter

The parameters used by the SerDes Toolbox built-in System Objects can be modified or hidden but cannot be deleted.

User generated AMI parameters are deleted as follows.

Update the AMI Parameters

1. Open the Block Parameter dialog box for the Configuration block, then click on the **Open SerDes IBIS-AMI Manager** button.

2. Go to either the **AMI-Tx** or **AMI-Rx** tab where the parameter resides.

3. Highlight the parameter to be deleted and press **Delete Parameter**.

4. Confirm the deletion, then **Close** the SerDes IBIS-AMI Manager.

Update Init

1. Push into either the Tx or Rx subsystem block where the parameter was used.

2. Double click on the Init block to open the mask.

3. Press the **Refresh Init** button to remove any deleted Out or InOut parameter Data Stores from the initialize subsystem.

4. Click **OK** to close the mask.

5. Click on the Init block again and type **Ctrl-U** to look under the Init mask

6. Double-click on the initialize block to open the Initialize Function.

7. Double-click on the **impulseEqualization MATLAB function block** to open the function in MATLAB.

8. Scroll down to the section titled:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
```

9. Delete or comment out all instances of the removed parameter.

10. Save and close the MATLAB function block.

Update GetWave

Push into each datapath block where the removed parameter was used and delete each instance of the parameter.

Verify Results

Run a simulation to verify that the project still operates with no errors or warnings.

How to Hide a Parameter

There may be times when a parameter is required for model functionality, but needs to be hidden from the user. For example, to keep a user from changing the FFE mode, remove this parameter from .ami file - effectively hardcoding the parameter to a single value. The mode parameter is still present in the code so that the FFE continues to work as expected, but the user can no longer change the value.

To hide a parameter from both Init and GetWave:

1. Open the mask by double-clicking on the datapath block of interest.

2. Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.

3. Deselect the parameter(s) to be hidden.

A few things to keep in mind about hiding parameters:

- When hiding parameters, verify that the current parameter value(s) are correct. The current value will now always be used as the default value for that parameter.
- Hiding a parameter has no effect on the model executable. It only removes the parameter from the generated .ami file.
- If the hidden parameter is of type Out or InOut, it will still show up in the AMI_Parameters_Out string of the model executable.

How to Modify a Parameter

All the parameters used in SerDes Toolbox are modified via the SerDes IBIS-AMI Manager dialog by using the **Edit...** button. However, the parameter values that can be modified vary depending on which type of parameters they are.

For the built-in System Objects, only the following fields can be modified:

- Current Value
- Description
- Format
- Default
- List values (for Format List)
- Typ/Min/Max values (for Format Range)

For the user defined parameters all fields can be modified.

How to add Jitter Parameters

Jitter and noise parameters such as Tx_Rj, Tx_Dj, Tx_DCD, Rx_Rj, Rx_Dj and Rx_DCD or other reserved parameters such Rx_Receiver_Sensitivity are post-processing parameters that are used by an IBIS-AMI compliant simulator to modify the simulation results accordingly. These parameters are added via the SerDes IBIS-AMI Manager dialog by using the **Reserved Parameters...** button.

For example, to add Rx_Receiver_Sensitivity and Rx_Dj to a receiver .ami file, click the **Reserved Parameters...** button to bring up the Rx Add/Remove Jitter&Noise dialog, select the **Rx_Receiver_Sensitivity** and **Rx_Dj** boxes, then click **OK** to add these parameters to the Reserved Parameters section of the Rx AMI file.

To set the values for these two new parameters:

- Select **Rx_Receiver_Sensitivity**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.04`
- Change the **Format** to `Value`.
- Click **OK** to save the changes.
- Select **Rx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.

- Set the **Typ** value to `0.05`
- Set the **Min** value to `0.0`
- Set the **Max** value to `0.1`
- Click **OK** to save the changes.

These two parameters will show up in the Reserved_Parameters section of the .ami file like this:

```
(Rx_Receiver_Sensitivity (Usage Info)(Type Float)(Value 0.04))
```

```
(Rx_Dj (Usage Info) (Type UI) (Range 0.05 0.0 0.01))
```

For more information on IBIS reserved parameters see the IBIS specification.

References

IBIS 6.1 Specification

See Also

FFE | PassThrough | **SerDes Designer**

More About

- “Customizing SerDes Toolbox Datapath Control Signals” on page 5-2

Industry Standard IBIS-AMI Models

PCIe4 Transmitter/Receiver IBIS-AMI Model

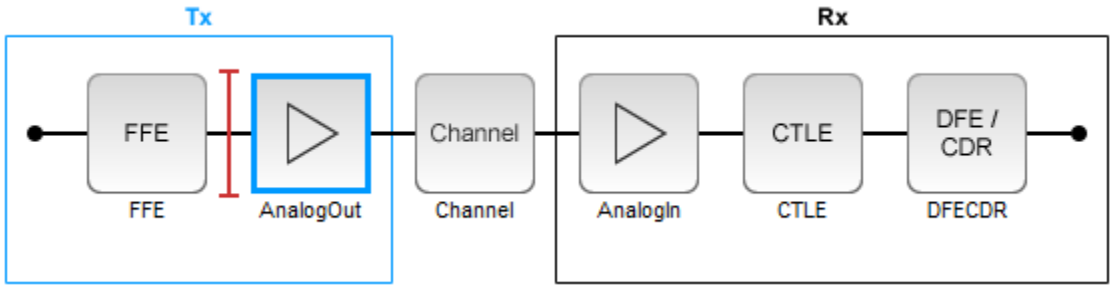
This example shows how to create generic PCIe Generation 4 (PCIe4) transmitter and receiver IBIS-AMI models using the library blocks in SerDes Toolbox™. The generated models conform to the IBIS-AMI and PCI-SIG PCIe4 specifications.

PCIe4 Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example sets up the target transmitter and receiver AMI model architecture using the blocks required for PCIe4 in the SerDes Designer app. The model is then exported to Simulink® for further customization.

This example uses the SerDes Designer model `pcie4_txrx_ami`. Type the following command in the MATLAB® command window to open the model:

```
>> serdesDesigner('pcie4_txrx_ami')
```



A PCIe4 compliant transmitter uses a 3-tap feed forward equalizer (FFE) with one pre-tap and one post-tap, and ten presets. The receiver model uses a continuous time linear equalizer (CTLE) with seven pre-defined settings, and a 2-tap decision feedback equalizer (DFE). To support this configuration the SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to `62.5` ps, since the maximum allowable PCIe4 operating frequency is `16` GHz
- **Target BER** is set to `1e-12`.
- **Samples per Symbol**, **Modulation**, and **Signaling** are kept at default values, which are respectively `16`, `NRZ` (non-return to zero), and `Differential`.

Transmitter Model Setup

- The Tx FFE block is set up for one pre-tap and one post-tap by including three tap weights. Specific tap presets will be added in later in the example when the model is exported to Simulink.
- The Tx AnalogOut model is set up so that **Voltage** is `1.05` V, **Rise time** is `12` ps, **R** (output resistance) is `50` Ohms, and **C** (capacitance) is `0.5` pF according to the PCIe4 specification.

Channel Model Setup

- **Channel loss** is set to 15 dB.
- **Target Frequency** is set to the Nyquist frequency, `8` GHz.
- **Differential impedance** is kept at default `100` Ohms.

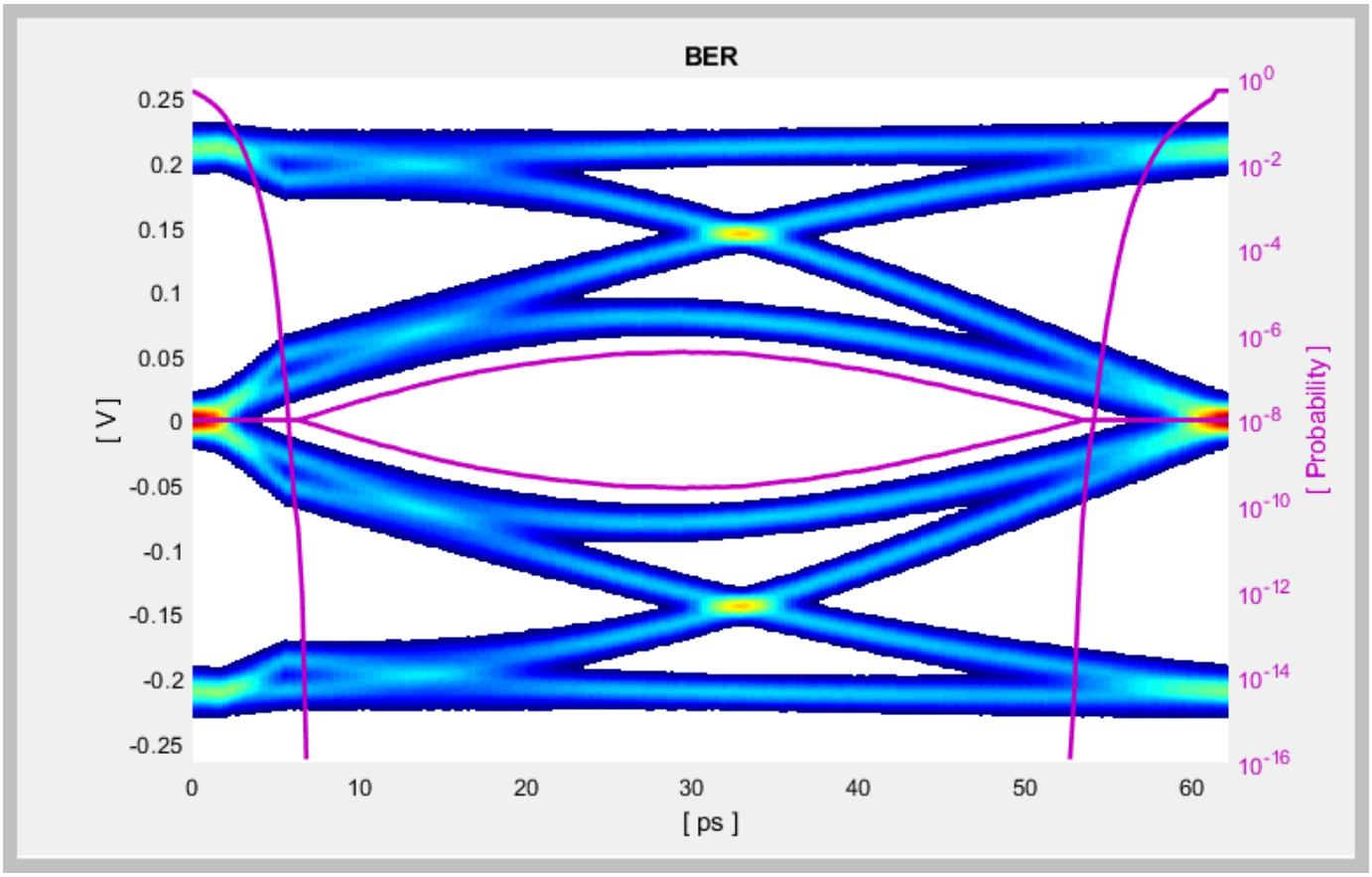
Receiver Model Setup

- The Rx Analogin model is set up so that **R** (input resistance) is `50` Ohms and **C** (capacitance) is `0.5` pF according to the PCIe4 specification.
- The Rx CTLE block is set up for 7 configurations. The **GPZ** (Gain Pole Zero) matrix data is derived from the transfer function given in the PCIe4 Behavioral CTLE specification.
- The Rx DFE/CDR block is set up for two DFE taps. The limits for each tap have been individually defined according to the PCIe4 specification to `+/-30` mV for tap1 and `+/-20` mV for tap2.

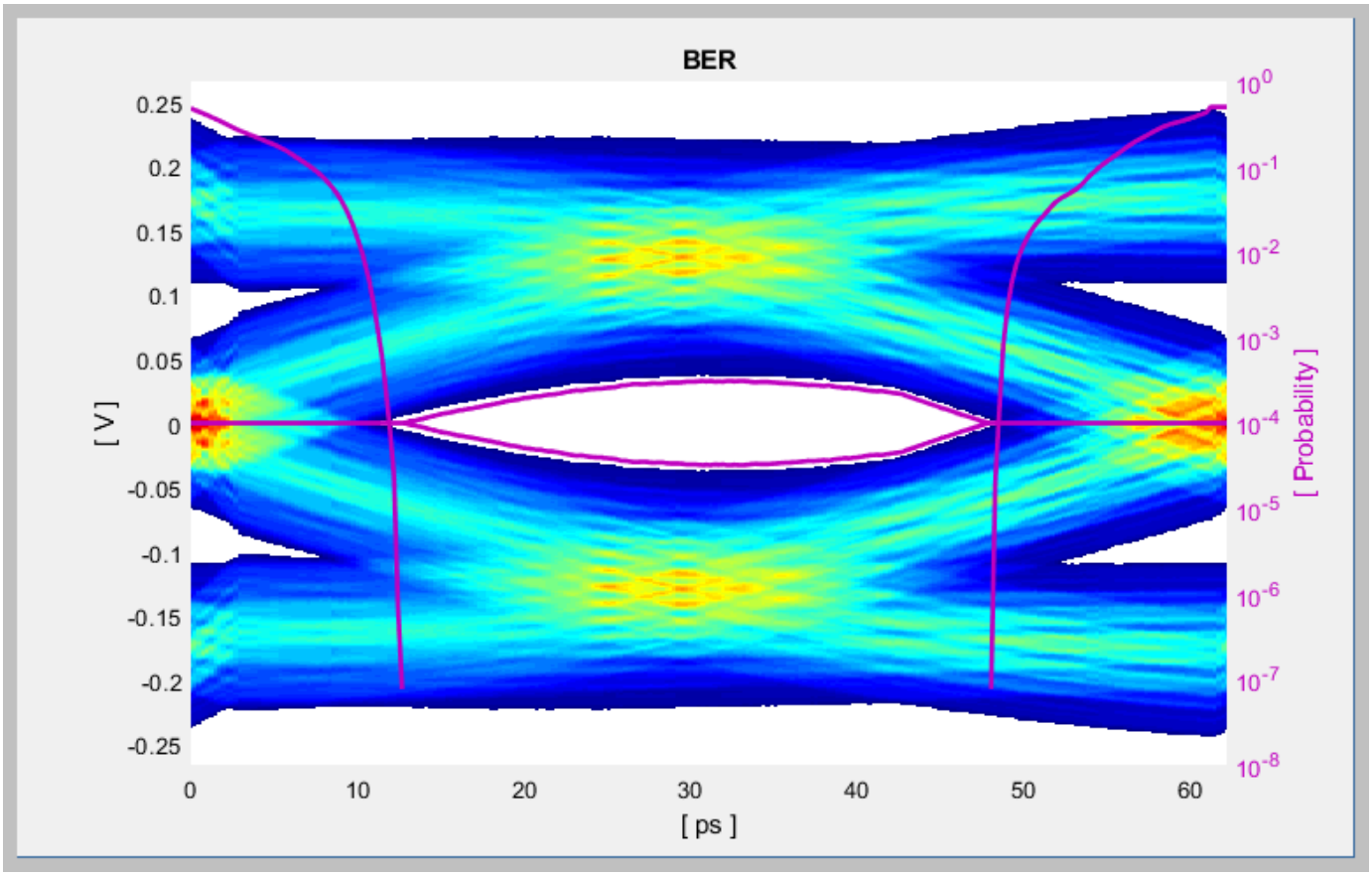
Plot Statistical Results

Use the SerDes Designer plots to visualize the results of the PCIe4 setup.

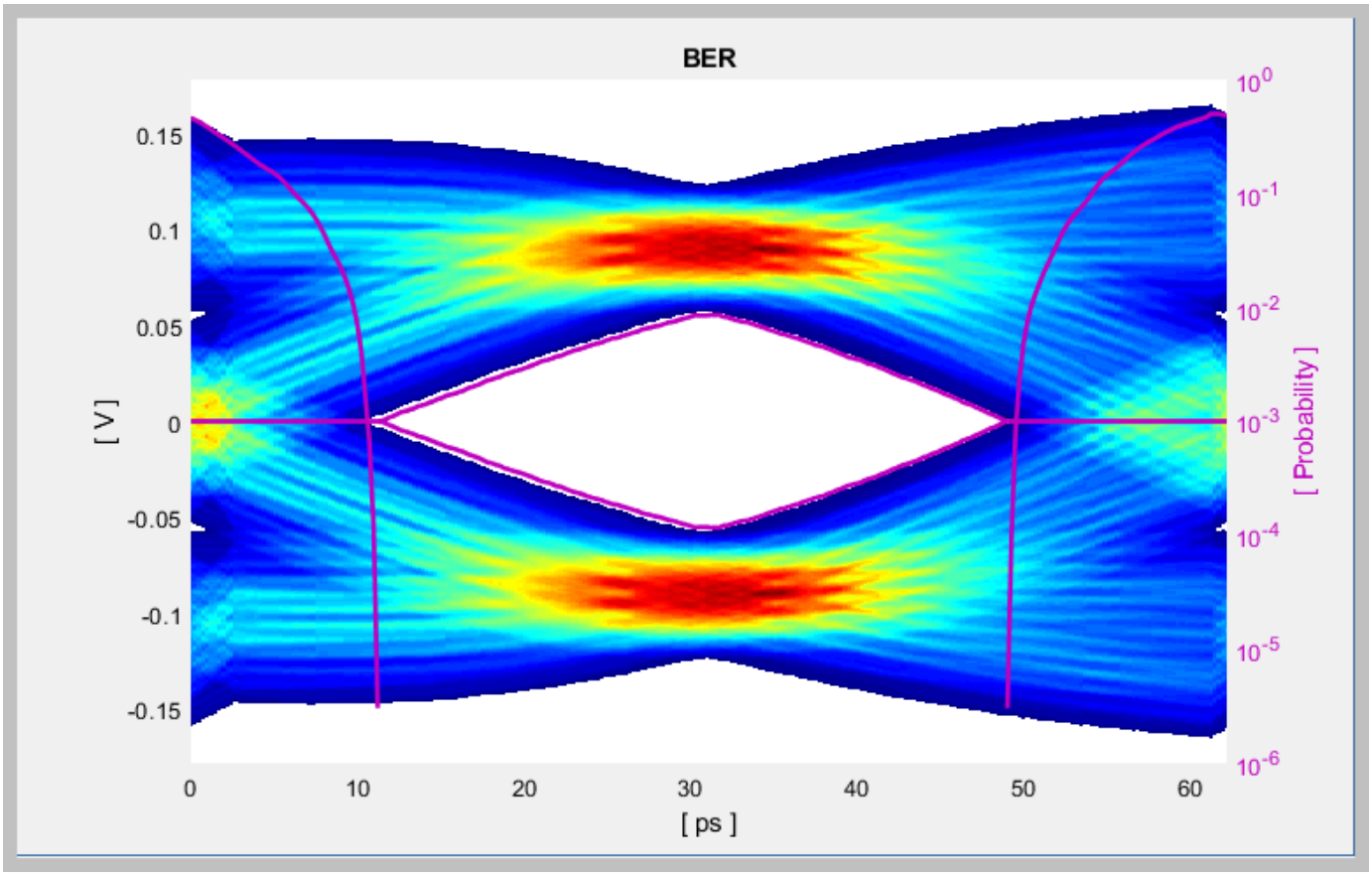
Add the **BER** plot from **ADD Plots** and observe the results.



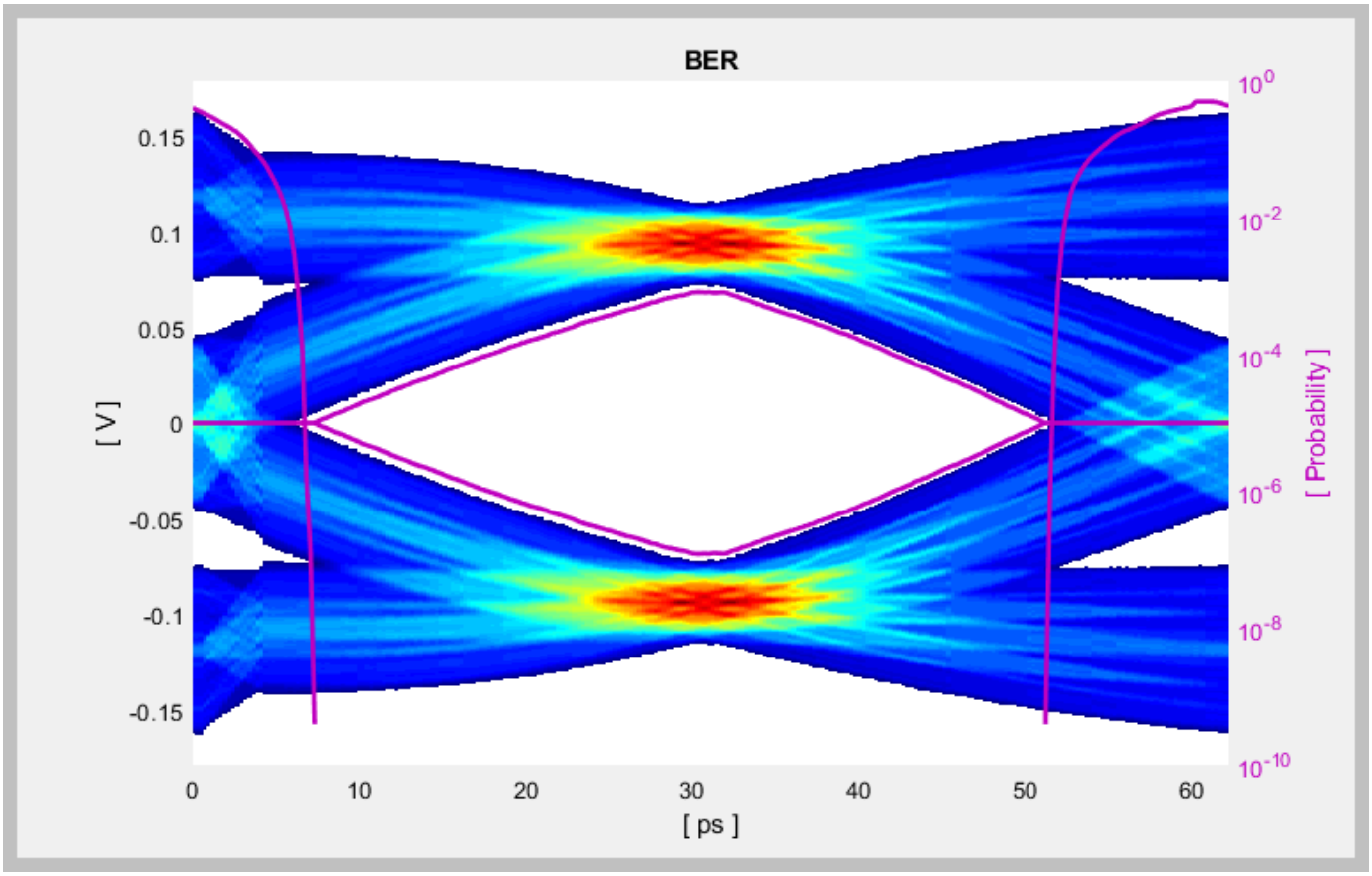
Change the Rx CTLE **Configuration select** parameter value from `0` to 6 and observe how this changes the data eye.



Change the value of the Tx FFE **Tap weights** from `[0 1 0]` to `[-0.125 0.750 -0.125]` and observe the results.



Change the Rx CTLE **Mode** to `Adapt` and observe the results. In this mode all CTLE values are swept to find the optimal setting.



Before continuing, reset the value of the Tx FFE **TapWeights** back to `[0 1 0]` and Rx CTLE **ConfigSelect** back to `0`. Leave the Rx **CTLE** Mode at `Adapt`. Resetting these values here will avoid the need to set them again after the model has been exported to Simulink. These values will become the defaults when the final AMI models are generated.

Export SerDes System to Simulink

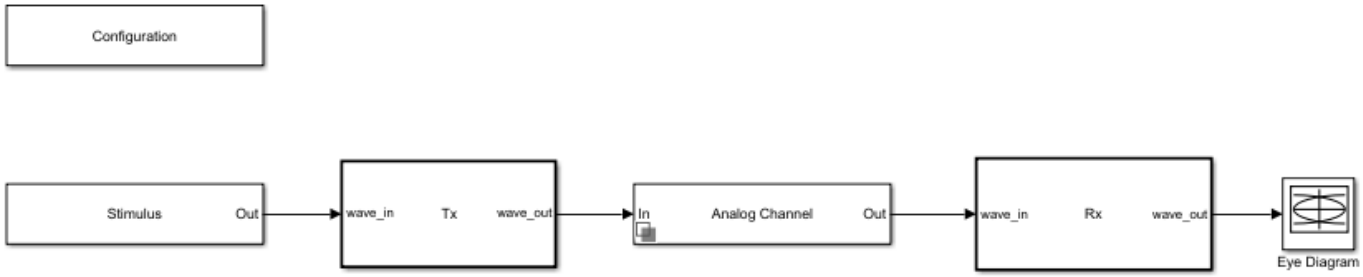
Click on the **Export** button to export the above configuration to Simulink for further customization and generation of the AMI model executables.

PCIe4 Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the SerDes Designer app and customize it as required for PCIe4 in Simulink.

Review Simulink Model Setup

The SerDes System imported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app have been transferred to the Simulink model. Save the model and review each block setup.

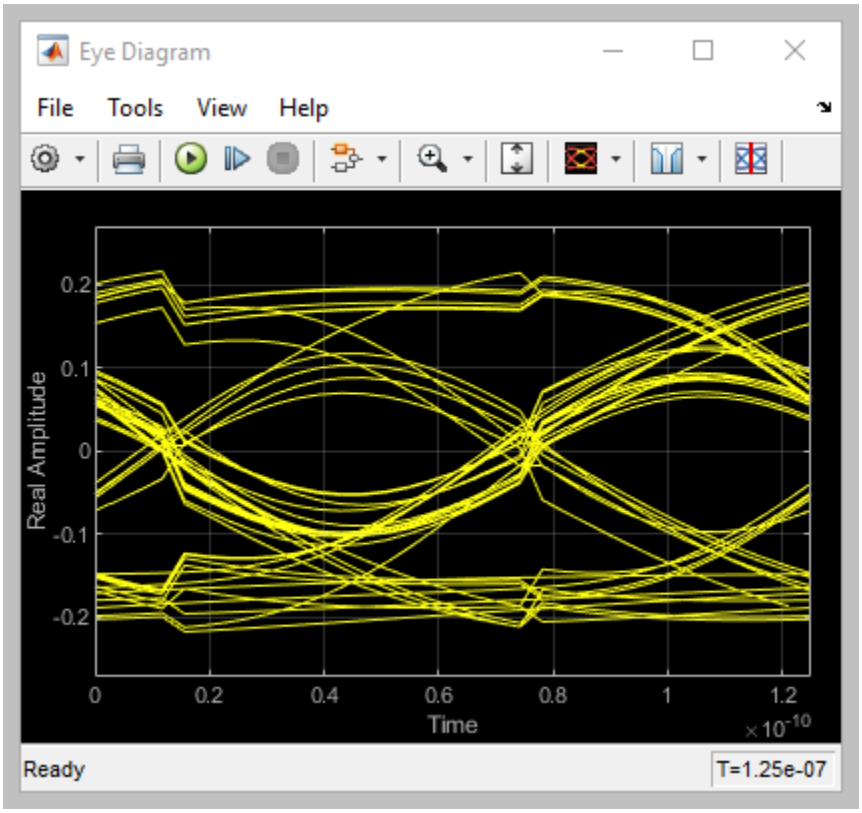


- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** is carried over from the SerDes Designer app.
- Double click the Stimulus block to open the Block Parameters dialog box. You can set the **PRBS** (pseudorandom binary sequence) order and the number of symbols to simulate. This block is not carried over from the SerDes Designer app.
- Double click the Tx block to look inside the Tx subsystem. The subsystem has the FFE block carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.
- Double click the Analog Channel block to open the Block Parameters dialog box. The parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx analog model parameters is carried over from the SerDes Designer app.
- Double click on the Rx block to look inside the Rx subsystem. The subsystem has the CTLE and DFECDR blocks carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.

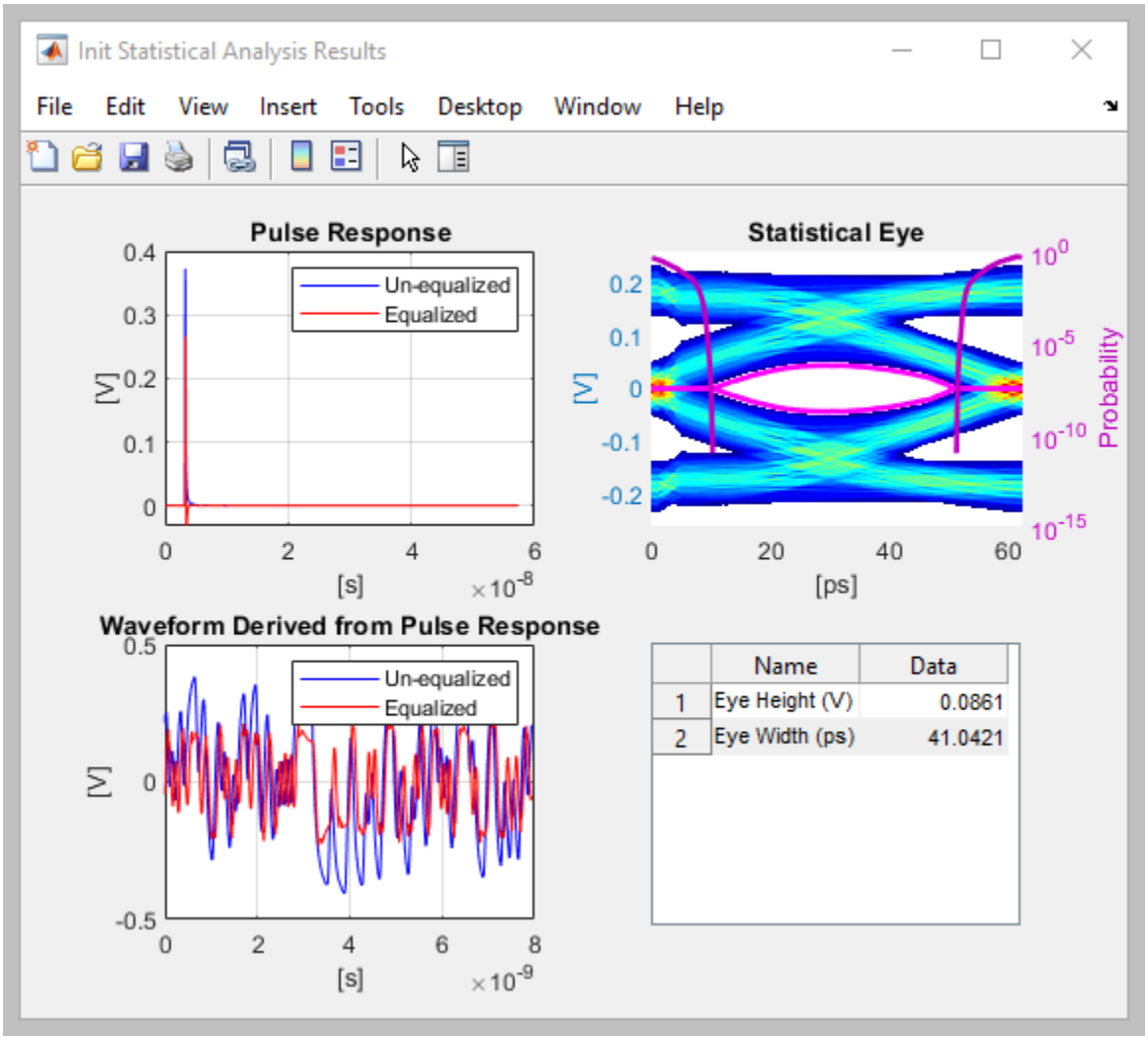
Run the Model

Run the model to simulate the SerDes System.

Two plots are generated. The first is a live time-domain (GetWave) eye diagram that is updated as the model is running.



The second plot contains four views of the statistical (Init) results, similar to what is available in the SerDes Designer App.



Update Tx FFE Block

- Inside the Tx subsystem, double click the FFE block to open the FFE Block Parameters dialog box.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect the **Mode** parameter to remove this parameter from the AMI file, effectively hard-coding the current value of **Mode** in final AMI model to `Fixed`.

Review Rx CTLE Block

- Inside the Rx subsystem, double click the CTLE block to open the CTLE Block Parameters dialog box.
- **Gain pole zero** data is carried over from the SerDes Designer app. This data is derived from the transfer function given in the PCIE4 Behavioral CTLE specification.
- CTLE **Mode** is set to `Fixed`, which means an optimization algorithm built into the CTLE system object selects the optimal CTLE configuration at run time.

Update Rx DFECDR Block

- Inside the Rx subsystem, double click the DFECDR block to open the DFECDR Block Parameters dialog box.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Clear the **Phase offset** and **Reference offset** parameters to remove these parameters from the AMI file, effectively hard-coding these parameters to their current values.

Generate PCIe4 Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model, modifies the AMI parameters for PCIe4, then generates IBIS-AMI compliant PCIe4 model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **Open SerDes IBIS/AMI Manager** button. In the **IBIS** tab inside the SerDes IBIS/AMI manager dialog box, the analog model values are converted to standard IBIS parameters that can be used by any industry standard simulator. In the **AMI-Rx** tab in the SerDes IBIS/AMI manager dialog box, the reserved parameters are listed first followed by the model specific parameters following the format of a typical AMI file.

Update Transmitter AMI Parameters

Open the **AMI-Tx** tab in the SerDes IBIS/AMI manager dialog box. Following the format of a typical AMI file, the reserved parameters are listed first followed by the model specific parameters.

Inside the **Model_Specific** parameters, you can set the TX FFE tap values in three different ways:

- Leave the Tx FFE tap values at their default configuration and you can enter any floating point value for the pre/main/post taps values.
- Create a new AMI parameter to automatically select preset values - see “Managing AMI Parameters” on page 6-2.
- Directly specify the ten preset coefficients as defined in the PCIe4 specification - shown below in this example.

When you directly specify the preset coefficients, you change the format of the three **TapWeights** and specify the exact value to use for each preset. Only these ten defined presets will be allowed, and all three taps must be set to the same preset to get the correct values.

Set Preshoot Tap

- Select **TapWeight -1**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.000`.
- Change the **Description** to `Preshoot tap value`.
- Change the **Format** from `Range` to `List`.
- Change the **Default value** to `0.000`.
- In the **List values** box enter: `[0.000 0.000 0.000 0.000 0.000 -0.100 -0.125 -0.100 -0.125 -0.166]`.
- In the **List_Tip values** box enter: `["P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9"]`.

- Click **OK** to save the changes.

Set Main Tap

- Select **TapWeight 0**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.750`.
- Change the **Description** to `Main tap value`.
- Change the **Format** from `Range` to `List`.
- Change the **Default value** to `0.750`.
- In the **List values** box enter: `[0.750 0.833 0.800 0.875 1.000 0.900 0.875 0.700 0.750 0.834]`.
- In the **List_Tip values** box enter: `["P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9"]`.
- Click **OK** to save the changes.

Set De-emphasis Tap

- Select **TapWeight 1**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `-0.250`.
- Change the **Description** to: `De-Emphasis tap value`.
- Change the **Format** from `Range` to `List`.
- Change the **Default value** to `-0.250`.
- In the **List values** box enter: `[-0.250 -0.167 -0.200 -0.125 0.000 0.000 0.000 -0.200 -0.125 0.000]`.
- In the **List_Tip values** box enter: `["P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9"]`.
- Click **OK** to save the changes.

Add Tx Jitter Parameters

To add Jitter parameters for the Tx model click the **Reserved Parameters...** button to bring up the Tx Add/Remove Jitter&Noise dialog, select the **Tx_DCD**, **Tx_Dj** and **Tx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Tx AMI file. The following ranges allow you to fine-tune the jitter values to meet PCIe4 jitter mask requirements.

Set Tx DCD Jitter Value

- Select **Tx_DCD**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `3.0e-11`
- Click **OK** to save the changes.

Set Tx Dj Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `3.0e-11`
- Click **OK** to save the changes.

Set Tx Rj Jitter Value

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `2.0e-12`
- Click **OK** to save the changes.

Update Receiver AMI Parameters

Open the **AMI-Rx** tab in the SerDes IBIS/AMI manager dialog box. Following the format of a typical AMI file, the reserved parameters are listed first followed by the model specific parameters.

Add Rx Jitter Parameters

To add Jitter parameters for the Rx model click the **Reserved Parameters...** button to bring up the Rx Add/Remove Jitter&Noise dialog, select the **Rx_DCD**, **Rx_Dj** and **Rx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Rx AMI file. The following ranges allow you to fine-tune the jitter values to meet PCIe4 jitter mask requirements.

Set Rx DCD Jitter Value

- Select **Rx_DCD**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `3.0e-11`
- Click **OK** to save the changes.

Set Rx Dj Jitter Value

- Select **Rx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.

- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to 3`.0e-11`
- Click **OK** to save the changes.

Set Rx Rj Jitter Value

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to 1`.0e-12`
- Click **OK** to save the changes.

Export Models

Open the **Export** tab in the SerDes IBIS/AMI manager dialog box.

- Update the **Tx model name** to `pcie4_tx`.
- Update the **Rx model name** to `pcie4_rx`.
- Note that the **Tx and Rx corner percentage** is set to `10`. This will scale the min/max analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx AMI Model Settings. This will create model executables that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Tx model Bits to ignore value** to `3` since there are three taps in the Tx FFE.
- Set the **Rx model Bits to ignore value** to `20,000` to allow sufficient time for the Rx DFE taps to settle during time domain simulations.
- Set **Models to export** as **Both Tx and Rx** so that all the files are selected to be generated (IBIS file, AMI files and DLL files).
- Set the **IBIS file name** to be `pcie4_serdes`.
- Press the **Export** button to generate models in the Target directory.

Test Generated IBIS-AMI Models

The PCIe4 transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry standard AMI model simulator.

References

1 PCI-SIG, https://pcisig.com.

2 SiSoft Support Knowledge Base Article: PCIe-Gen4 Compliance Kit, https://sisoft.na1.teamsupport.com/knowledgeBase/15488464.

See Also

CTLE | DFECDR | FFE | **SerDes Designer**

More About

- “Managing AMI Parameters” on page 6-2
- “Customizing SerDes Toolbox Datapath Control Signals” on page 5-2

External Websites

- https://www.sisoft.com/support/

DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model

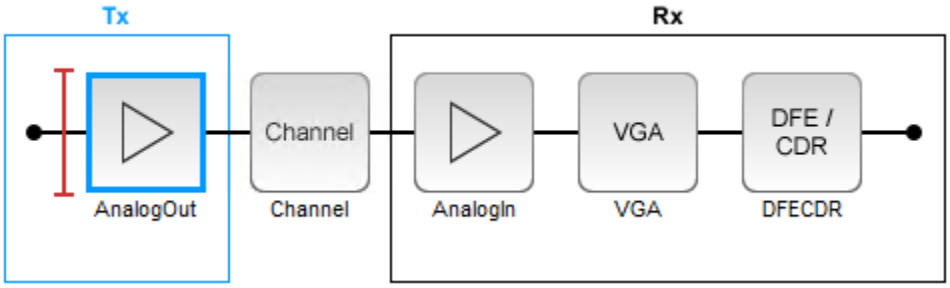
This example shows how to create generic DDR5 transmitter and receiver IBIS-AMI models using the library blocks in SerDes Toolbox™. Since DDR5 DQ signals are bidirectional, this example creates Tx and Rx models for the SDRAM. The generated models conform to the IBIS-AMI specification.

DDR5 SDRAM Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example sets up and explores the target transmitter and receiver architectures using the blocks required for DDR5 in the SerDes Designer app. The SerDes system is then exported to Simulink® for further customization and IBIS-AMI model generation.

Type the following command in the MATLAB® command window to open the `ddr5_sdram` model:

```
>> serdesDesigner('ddr5_sdram')
```



The SDRAM has a DDR5 transmitter (Tx) using no equalization. The SDRAM also has a DDR5 receiver (Rx) using a variable gain amplifier (VGA) with 7 pre-defined settings and a 4-tap decision feedback equalizer (DFE) with built-in clock data recovery.

Configuration Setup

- **Symbol Time** is set to `208.3` ps, since the target operating rate is `4.8`Gbps for DDR5-4800.
- **Target BER** is set to `100e-18`.
- **Signaling** is set to `Single-ended`.
- **Samples per Symbol** and **Modulation** are kept at default values, which are `16` and `NRZ` (nonreturn to zero), respectively.

Transmitter Model Setup

- The DDR5 SDRAM has no transmit equalization, so only an analog model is required.
- The Tx AnalogOut model is set up so that **Voltage** is `1.1` V, **Rise time** is `100` ps, **R** (output resistance) is `48` ohms, and **C** (capacitance) is `0.65` pF. The actual analog models used in the final model will be generated later in this example.

Channel Model Setup

- **Channel loss** is set to `5` dB, which is typical of DDR channels.
- **Single-ended impedance** is set to `40` ohms.
- **Target Frequency** is set to `2.4` GHz, which is the Nyquist frequency for 4.8 GHz

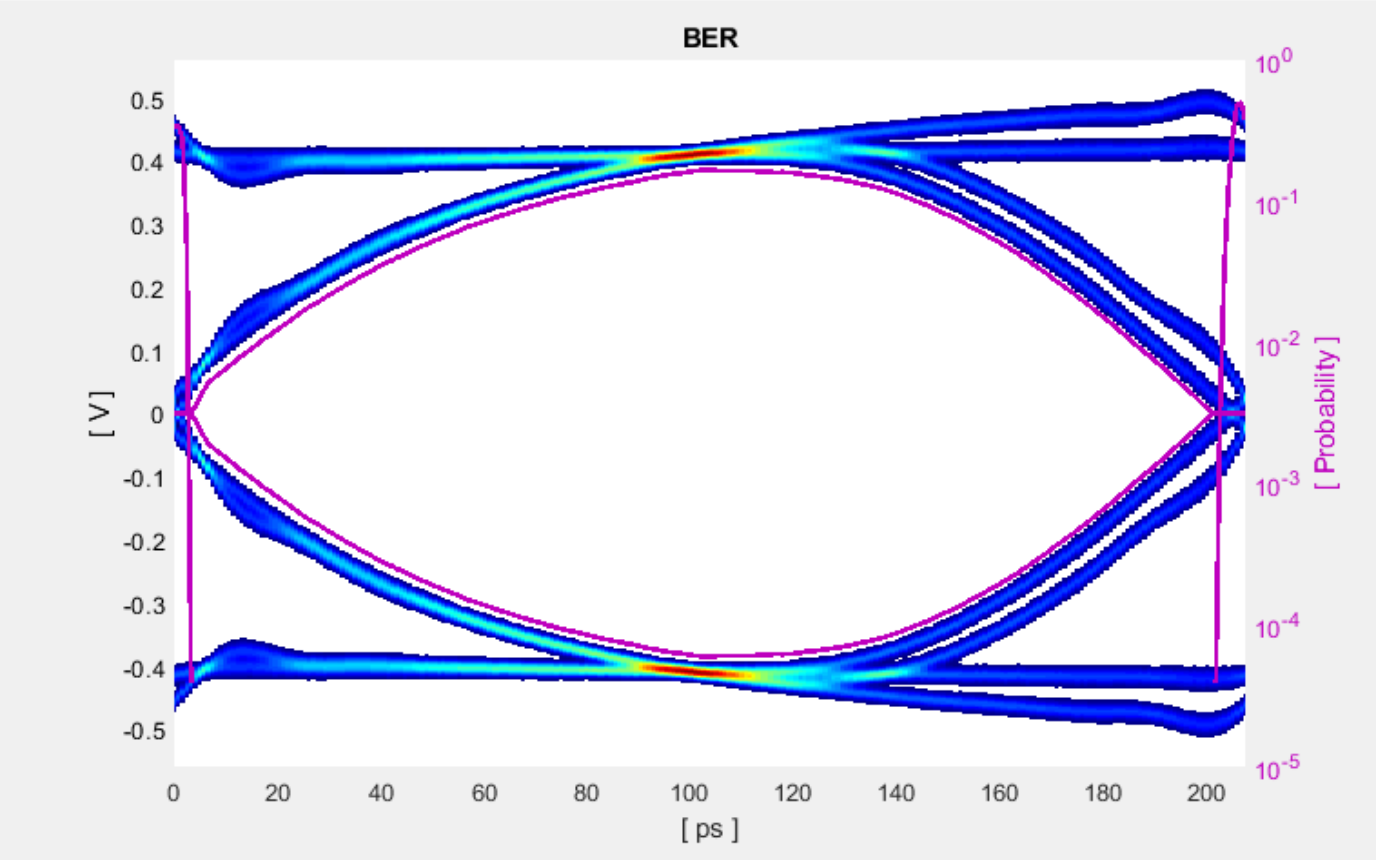
Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (input resistance) is `40` ohms and **C** (capacitance) is `0.65`pF. The actual analog models used in the final model will be generated later in this example.
- The VGA block is set up with a **Gain** of 1 and the **Mode** set to on. Specific VGA presets will be added later in this example after the model is exported to Simulink.
- The DFECDR block is set up for four DFE taps by including four **Initial tap weights** set to 0. The **Minimum tap value** is set to `[-0.2 -0.075 -0.06 -0.045]` V, and the **Maximum tap value** is set to `[0.05 0.075 0.06 0.045]` V.

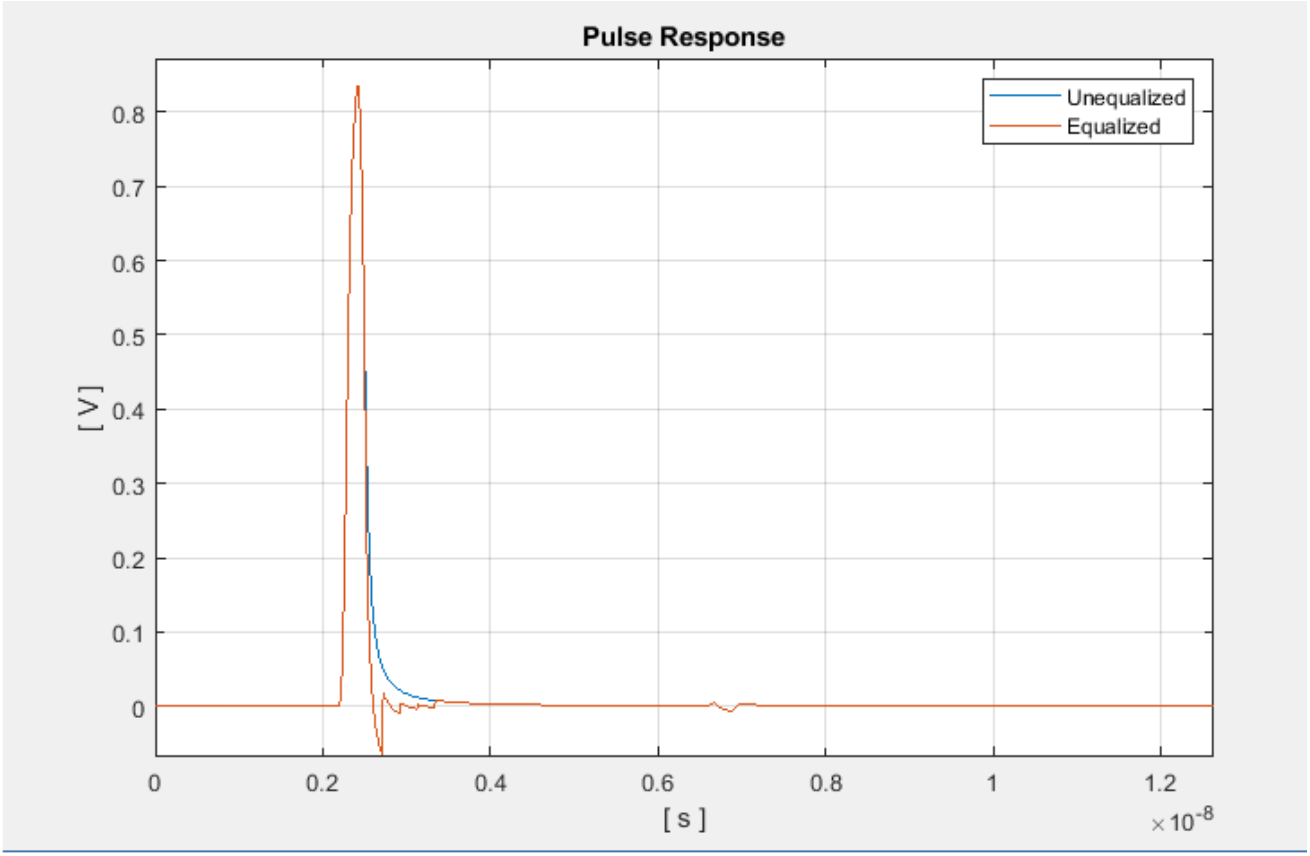
Plot Statistical Results

Use the SerDes Designer **Add Plots** button to visualize the results of the DDR5 SDRAM setup.

- Add the BER plot from **Add Plots** and observe the results.



- Add the Pulse Response plot from **Add Plots** and zoom into the pulse area to observe the results.



Export SerDes System to Simulink

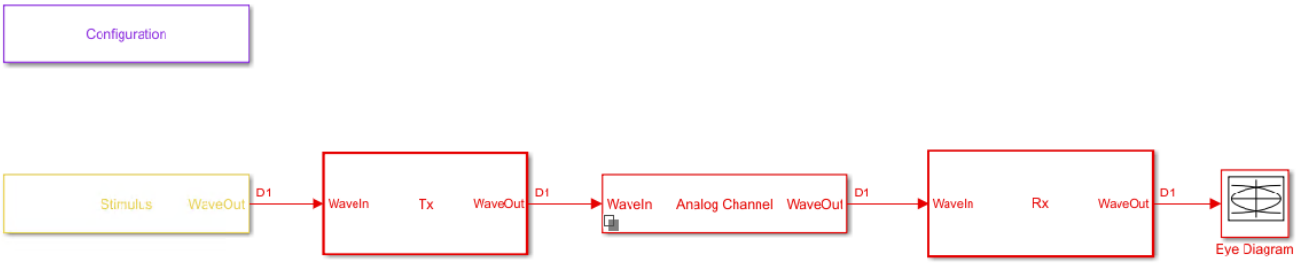
Click **Save** and then click on the **Export** button to export the configuration to Simulink for further customization and generation of the AMI model executables.

DDR5 SDRAM Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the SerDes Designer app and customizes it as required for DDR5 in Simulink.

Review the Simulink Model Setup

The SerDes System imported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app have been transferred to the Simulink model. Save the model and review each block setup.

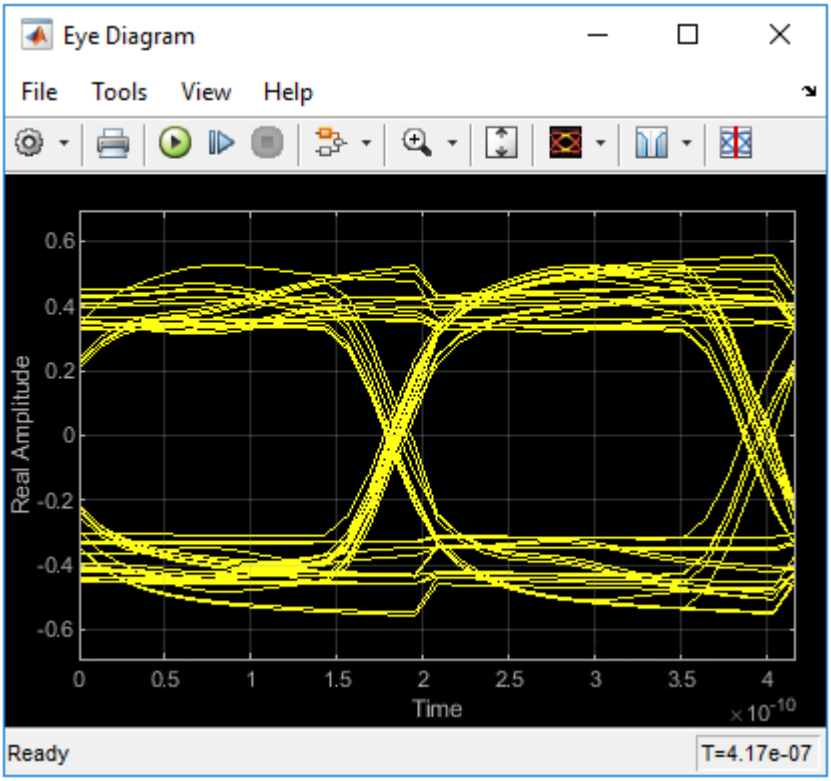


- Double-click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** are carried over from the SerDes Designer app.
- Double-click the Stimulus block to open the Block Parameters dialog box. You can set the **PRBS** (pseudorandom binary sequence) order and the number of symbols to simulate. This block is not carried over from the SerDes Designer app.
- Double-click the Tx block to look inside the Tx subsystem. Since there is no algorithmic model for the transmitter, the Tx subsystem is simply a pass through from the WaveIn to WaveOut ports.
- Double-click the Analog Channel block to open the Block Parameters dialog box. The parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx **Analog Model** parameters are carried over from the SerDes Designer app.
- Double-click on the Rx block to look inside the Rx subsystem. The subsystem has the VGA and DFECDR blocks carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.

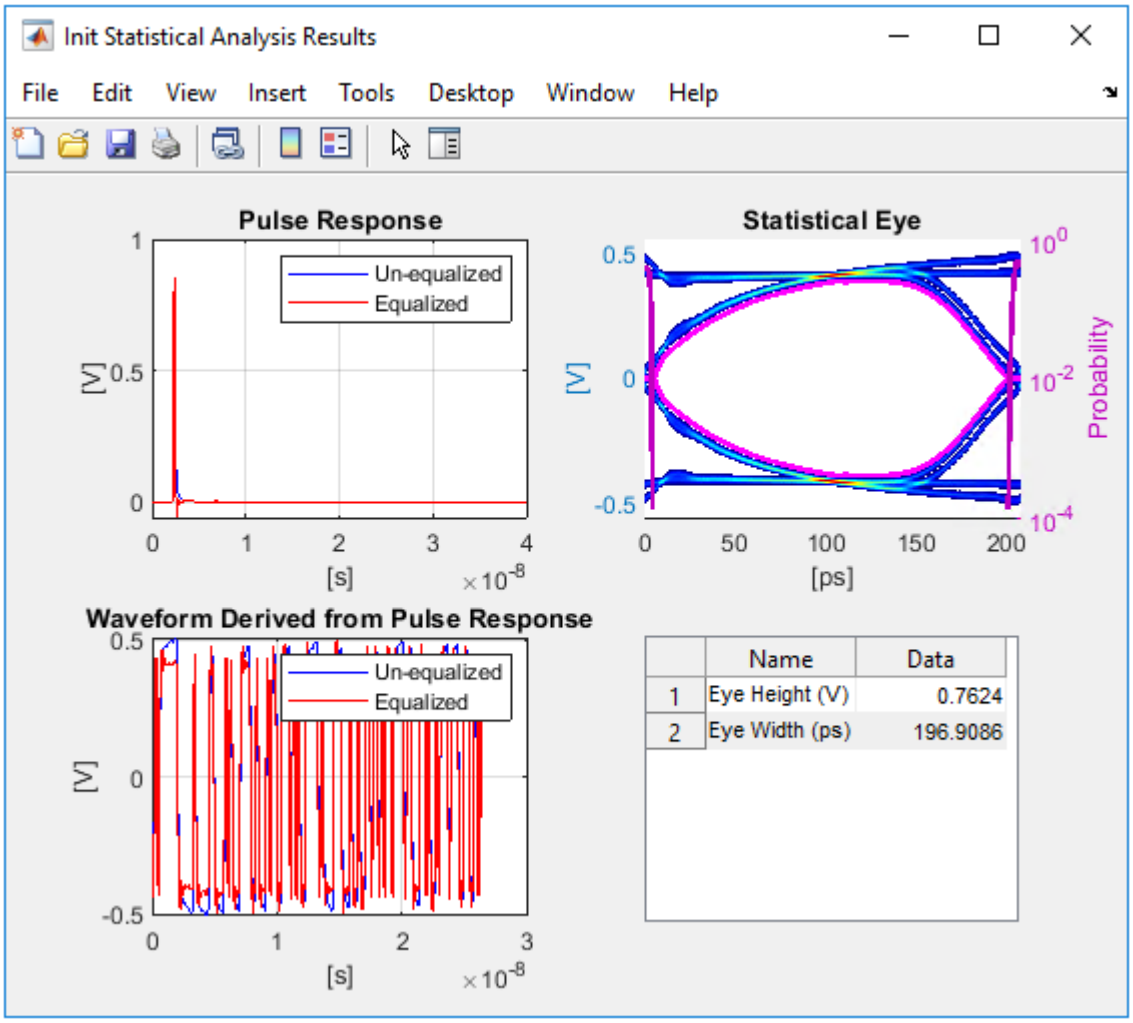
Run the Model

Run the model to simulate the SerDes system.

Two plots are generated. The first is a live time domain (GetWave) eye diagram that is updated as the model is running.



After the simulation has completed the second plot contains four views of the statistical (Init) results, similar to what is available in the SerDes Designer app.



Review Rx VGA Block

- Inside the Rx subsystem, double-click the VGA block to open the VGA Block Parameters dialog box.
- The **Mode** and **Gain** settings are carried over from the SerDes Designer app.

Update Rx DFECDR Block

- Inside the Rx subsystem, double-click the DFECDR block to open the DFECDR Block Parameters dialog box.
- The **Initial tap weights**, **Minimum DFE tap value**, and **Maximum tap value** RMS settings are carried over from the SerDes Designer app. The **Adaptive gain** and **Adaptive step size** are set to `3e-06` and `1e-06`, respectively, which are reasonable values based on DDR5 SDRAM expectations.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect **Phase offset** and **Reference offset** to remove these parameters from the AMI file, effectively hard-coding these parameters to their current values.

Generate DDR5 SDRAM IBIS-AMI Models

The final part of this example takes the customized Simulink model, modifies the AMI parameters for a DDR5 SDRAM, and then generates IBIS-AMI compliant DDR5 SDRAM model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **Open SerDes IBIS-AMI Manager** button. In the **IBIS** tab inside the SerDes IBIS-AMI manager dialog box, the analog model values are converted to standard IBIS parameters that can be used by any industry-standard simulator.

Review Transmitter (Tx) AMI Parameters

Open the **AMI-Tx** tab in the SerDes IBIS-AMI manager dialog box. Notice that there are no model-specific parameters since the DDR5 SDRAM Tx does not have any equalization.

Add Tx Jitter Parameters

To add Jitter parameters for the Tx model click the **Reserved Parameters...** button to bring up the Tx Add/Remove Jitter&Noise dialog, select the **Tx_Dj** and **Tx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Tx AMI file. The following values allow you to fine-tune the jitter values to meet DDR5 jitter mask requirements.

Note: All JEDEC DDR5 SDRAM values are currently TBD.

Set Tx Deterministic Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.1000`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Tx Random Jitter Value

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0050`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Update Receiver (Rx) AMI Parameters

Open the **AMI-Rx** tab in the SerDes IBIS-AMI manager dialog box. The reserved parameters are listed first followed by the model-specific parameters adhering to the format of a typical AMI file.

Set the VGA Gain:

- Highlight **Gain.**
- Click the **Edit...** button to launch the Add/Edit AMI Parameter dialog box.
- In the **Description** box, type `Rx Amplifier Gain`.

- Make sure **Format** is set to **List** and set **Default to** 1.
- In the **List values** box, enter `[0.5 0.631 0.794 1 1.259 1.585 2]`
- In the **List_Tip values** box, enter `["-6 dB" "-4 dB" "-2 dB" "0 dB" "2 dB" "4 dB" "6 dB"]`
- Click **OK** to save the changes.

Set First DFE Tap Weight

- Highlight **TapWeight 1**.
- Click the **Edit...** button to launch the Add/Edit AMI Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.2`, and **Max** = `0.05`.
- Click **OK.**

Set Second DFE Tap Weight

- Highlight **TapWeight 2**.
- Click the **Edit...** button to launch the Add/Edit AMI Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.075`, and **Max** = `0.075`
- Click **OK.**

Set Third DFE Tap Weight

- Highlight **TapWeight 3**.
- Click the **Edit...** button to launch the Add/Edit AMI Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.06`, and **Max** = `0.06`
- Click **OK.**

Set Fourth DFE Tap Weight

- Highlight **TapWeight 4**.
- Click the **Edit...** button to launch the Add/Edit AMI Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.045`, and **Max** = `0.045`
- Click **OK.**

Add Rx Jitter Parameters

To add Jitter parameters for the Rx model click the **Reserved Parameters...** button to bring up the Rx Add/Remove Jitter&Noise dialog, select the **Rx_Receiver_Sensitivity**, **Rx_Dj** and **Rx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Rx AMI file. The following values allow you to fine-tune the jitter values to meet DDR5 jitter mask requirements.

Note: All JEDEC DDR5 SDRAM values are currently TBD.

Set Rx Receiver Sensitivity Value

- Select **Rx_Receiver_Sensitivity**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.040`
- Change the **Format** to `Value`.

- Click **OK** to save the changes.

Set Rx Deterministic Jitter Value

- Select **Rx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.01750`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Rx Random Jitter Value

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.00375`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Export Models

Open the **Export** tab in the SerDes IBIS-AMI manager dialog box.

- Update the **Tx model name** to `ddr5_sdram_tx`.
- Update the **Rx model name** to `ddr5_sdram_rx`.
- Note that **Tx and Rx corner percentage** is set to `10`. This scales the minimum/maximum analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx AMI model settings. This creates model executables that support both statistical (Init) analysis and time-domain (GetWave) simulation.
- Set the Rx model **Bits to ignore** value to `250000` to allow sufficient time for the Rx DFE taps to settle during time domain simulations.
- Set the **Models to export** to **Both Tx and Rx** and ensure that all files have been selected to be generated (**IBIS file**, **AMI file(s)** and **DLL file(s)**). Note that while the Tx does not implement any equalization, we are still generating a pass-through model that will allow Tx jitter to be added to the simulation if desired.
- Set the **IBIS file name** to `temp_ddr5_sdram`.
- Click the **Export** button to generate models in the Target directory.

Update DDR5 Analog Models

To accommodate different topologies, loading configurations, data rates and transfers, DDR5 requires variable output drive strength and input on-die termination (ODT). While the same algorithmic AMI model is used, multiple analog models are required to cover all these use cases. The generation of these analog models is out of scope for this example, so a completed IBS file with the following analog models in it is available in the current example directory:

- POD11_IO_ZO34_ODTOFF: 34 ohm output impedance with no input ODT.
- POD11_IO_ZO48_ODTOFF: 48 ohm output impedance with no input ODT.

- POD11_IN_ODT34_C: Input with 34 ohm ODT.
- POD11_IN_ODT40_C: Input with 40 ohm ODT.
- POD11_IN_ODT48_C: Input with 48 ohm ODT.
- POD11_IN_ODT60_C: Input with 60 ohm ODT.
- POD11_IN_ODT80_C: Input with 80 ohm ODT.
- POD11_IN_ODT120_C: Input with 120 ohm ODT.
- POD11_IN_ODT240_C: Input with 240 ohm ODT.

To generate this complete IBIS file, the following changes were made to `temp_ddr5_sdram.ibs` using a text editor:

- Created one pin with a **signal_name** of DQ1_sdram and **model_name** of dq.
- Added two drivers with **Model_type** of I/O and named them POD11_IO_Z034_ODTOFF and POD11_IO_Z048_ODTOFF, respectively.
- Added seven receiver models and named them:

a) POD11_IN_ODT34_C

b) POD11_IN_ODT40_C

c) POD11_IN_ODT48_C

d) POD11_IN_ODT60_C

e) POD11_IN_ODT80_C

f) POD11_IN_ODT120_C

g) POD11_IN_ODT240_C

- Added VI curves and **Algorithmic Model** sections to all above mentioned models.
- Added a **Model Selector** section that references all above mentioned models.

Test Generated IBIS-AMI Models

The DDR5 transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry-standard AMI model simulator.

References

1 IBIS 7.0 Specification, https://ibis.org/ver7.0/ver7_0.pdf.

2 SiSoft Support Knowledge Base Article: DDR4 Registered - Rawcard B for 3 slot system, https://sisoft.na1.teamsupport.com/knowledgeBase/8976521.

See Also

DFECDR | **SerDes Designer** | VGA

More About

- “DDR5 Controller Transmitter/Receiver IBIS-AMI Model” on page 7-26

- “Design DDR5 IBIS-AMI Models to Support Back-Channel Link Training” on page 7-56

External Websites

- https://www.sisoft.com/support/

DDR5 Controller Transmitter/Receiver IBIS-AMI Model

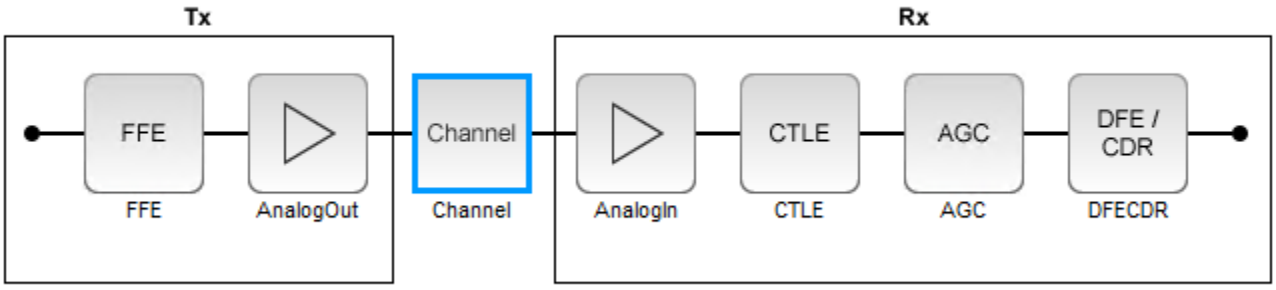
This example shows how to create generic DDR5 transmitter and receiver IBIS-AMI models using the library blocks in SerDes Toolbox™. Since DDR5 DQ signals are bidirectional, this example creates Tx and Rx models for the controller. The generated models conform to the IBIS-AMI specification.

DDR5 Controller Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example sets up and explores the target transmitter and receiver architectures using the blocks required for DDR5 in the SerDes Designer app. The SerDes system is then exported to Simulink® for further customization and IBIS-AMI Model generation.

Type the following command in the MATLAB® command window to open the `ddr5_controller` model:

```
>> serdesDesigner('ddr5_controller')
```



The controller has a DDR5 transmitter (Tx) using 5-tap feed forward equalization (FFE). The controller also has a DDR5 receiver (Rx) using a continuous time linear equalizer (CTLE) with 8 pre-defined settings, an automatic gain control (AGC), and a 4-tap decision feedback equalizer (DFE) with built-in clock data recovery.

Configuration Setup

- **Symbol Time** is set to `208.3` ps, since the target operating rate is `4.8` Gbps for DDR5-4800.
- **Target BER** is set to `100e-18`.
- **Signaling** is set to `Single-ended`.
- **Samples per Symbol** and **Modulation** are kept at default values, which are respectively `16` and `NRZ` (nonreturn to zero), respectively.

Transmitter Model Setup

- The Tx FFE block is set up for one pre-tap, one main-tap, and three post-taps by including five tap weights. This is done with the array [0 1 0 0 0], where the main tap is specified by the largest value in the array. Tap ranges will be added later in the example when the model is exported to Simulink.
- The Tx AnalogOut model is set up so that **Voltage** is `1.1` V, **Rise time** is `100` ps, **R** (output resistance) is `50` ohms, and **C** (capacitance) is `0.65` pF. The actual analog models used in the final model will be generated later in this example.

Channel Model Setup

- **Channel loss** is set to `5` dB, which is typical of DDR channels.
- **Single-ended impedance** is set to `40` ohms.
- **Target Frequency** is set to `2.4` GHz, which is the Nyquist frequency for 4.8 GHz

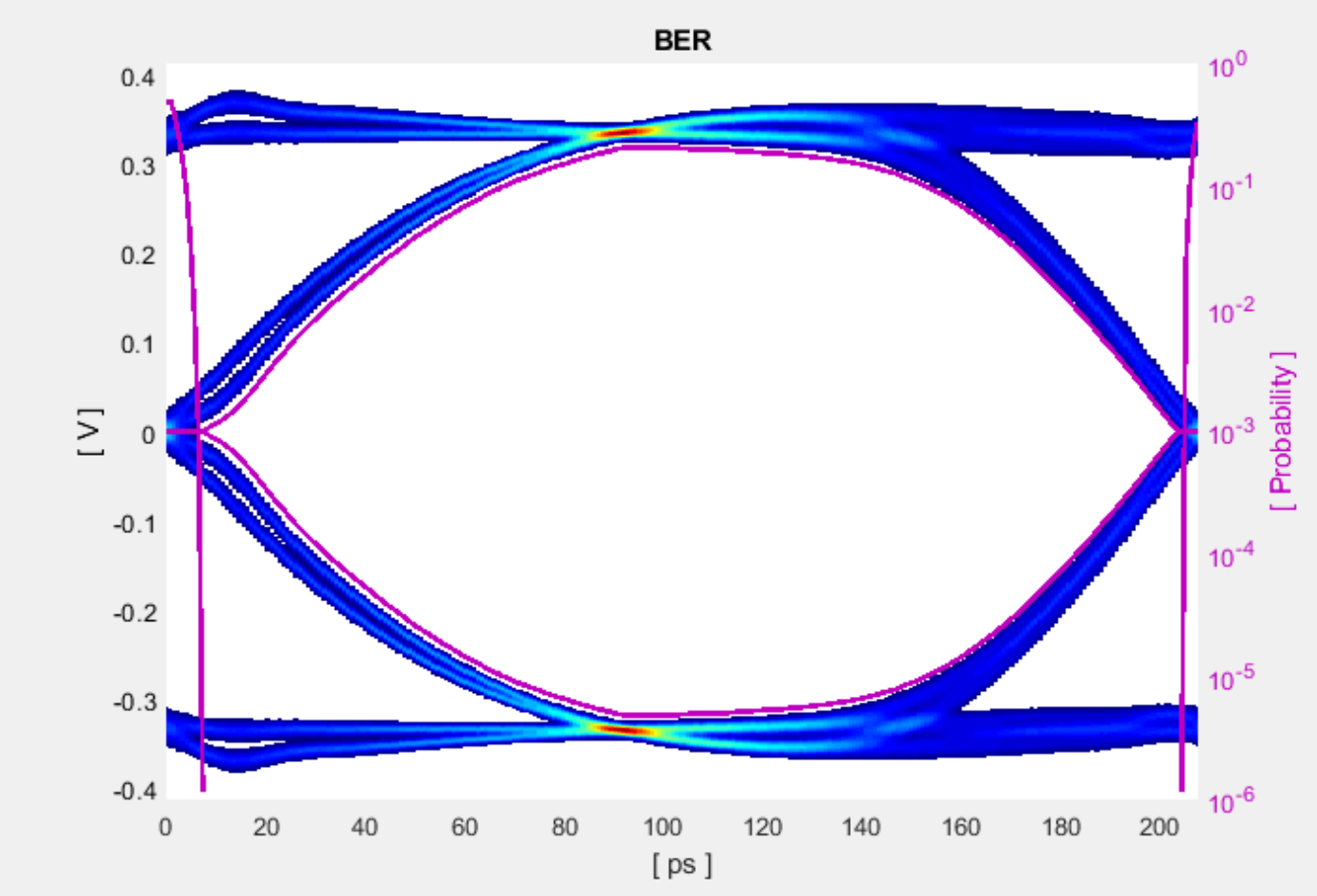
Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (input resistance) is `40` Ohms and **C** (capacitance) is `0.65`pF. The actual analog models used in the final model will be generated later in this example.
- The CTLE block is set up for `8` configurations. The **Specification** is set to `DC Gain and AC Gain`. **DC Gain** is set to `[0 -1 -2 -3 -4 -5 -6 -7]` dB. Peaking frequency is set to `2.4` GHz. All other parameters are kept at their default values.
- The AGC block has the default Target RMS voltage of `0.3` Volts.
- The DFECDR block is set up for four DFE taps by including four **Initial tap weights** set to 0. The **Minimum tap value** is set to `[-0.2 -0.075 -0.06 -0.045]` V and the **Maximum tap value** is set to `[0.05 0.075 0.06 0.045]` V.

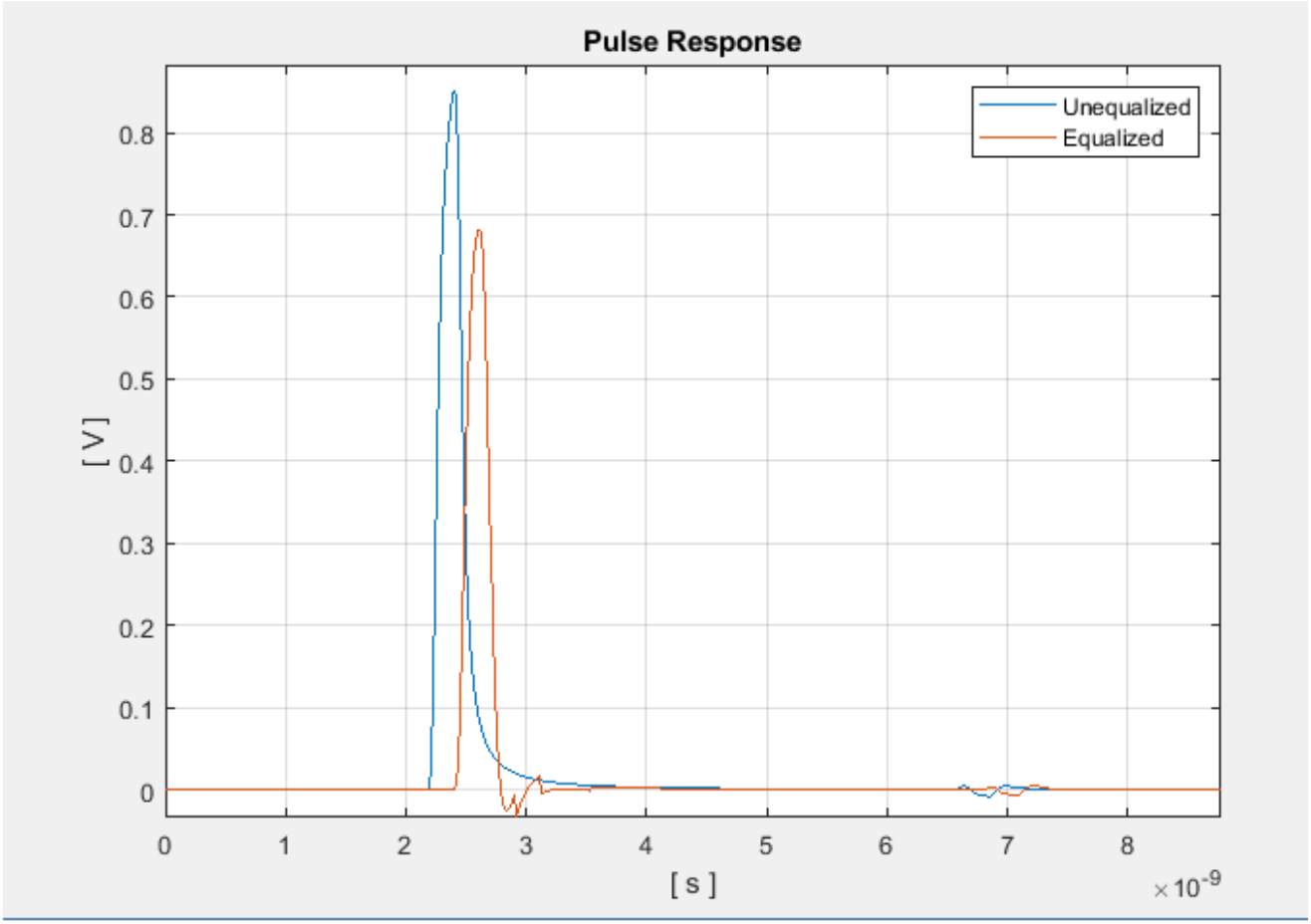
Plot Statistical Results

Use the SerDes Designer **Add Plots** button to visualize the results of the DDR5 Controller setup.

Add the BER plot from **Add Plots** and observe the results.



Add the Pulse Response plot from **Add Plots** and zoom into the pulse area to observe the results.



Export SerDes System to Simulink

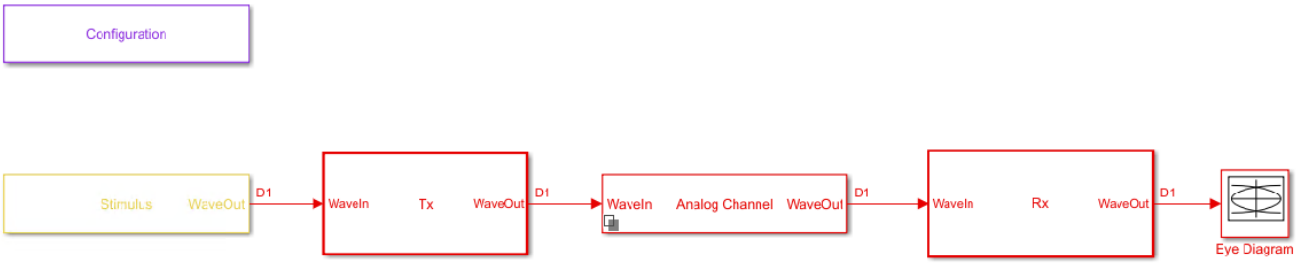
Click **Save** and then click on the **Export** button to export the configuration to Simulink for further customization and generation of the AMI model executables.

DDR5 Controller Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the SerDes Designer app and customizes it as required for DDR5 in Simulink.

Review the Simulink Model Setup

The SerDes System imported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app have been transferred to the Simulink model. Save the model and review each block setup.

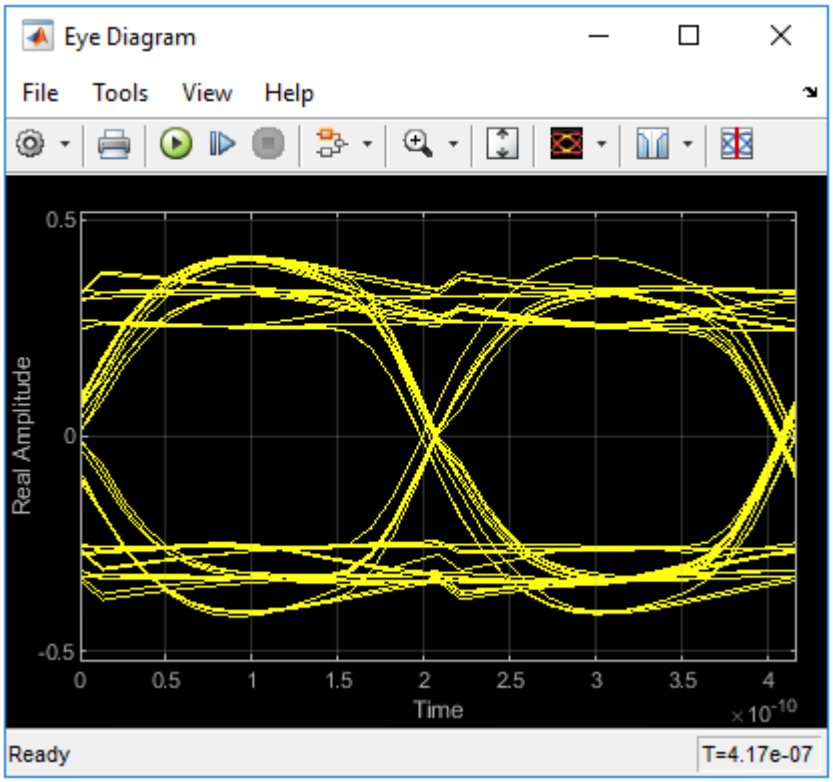


- Double-click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation,** and **Signaling** are carried over from the SerDes Designer app.
- Double-click the Stimulus block to open the Block Parameters dialog box. You can set the **PRBS** (pseudorandom binary sequence) order and the number of symbols to simulate. This block is not carried over from the SerDes Designer app.
- Double-click the Tx block to look inside the Tx subsystem. The subsystem has the FFE block carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.
- Double-click the Analog Channel block to open the Block Parameters dialog box. The parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx **Analog Model** parameters are carried over from the SerDes Designer app.
- Double-click on the Rx block to look inside the Rx subsystem. The subsystem has the CTLE, AGC and DFECDR blocks carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.

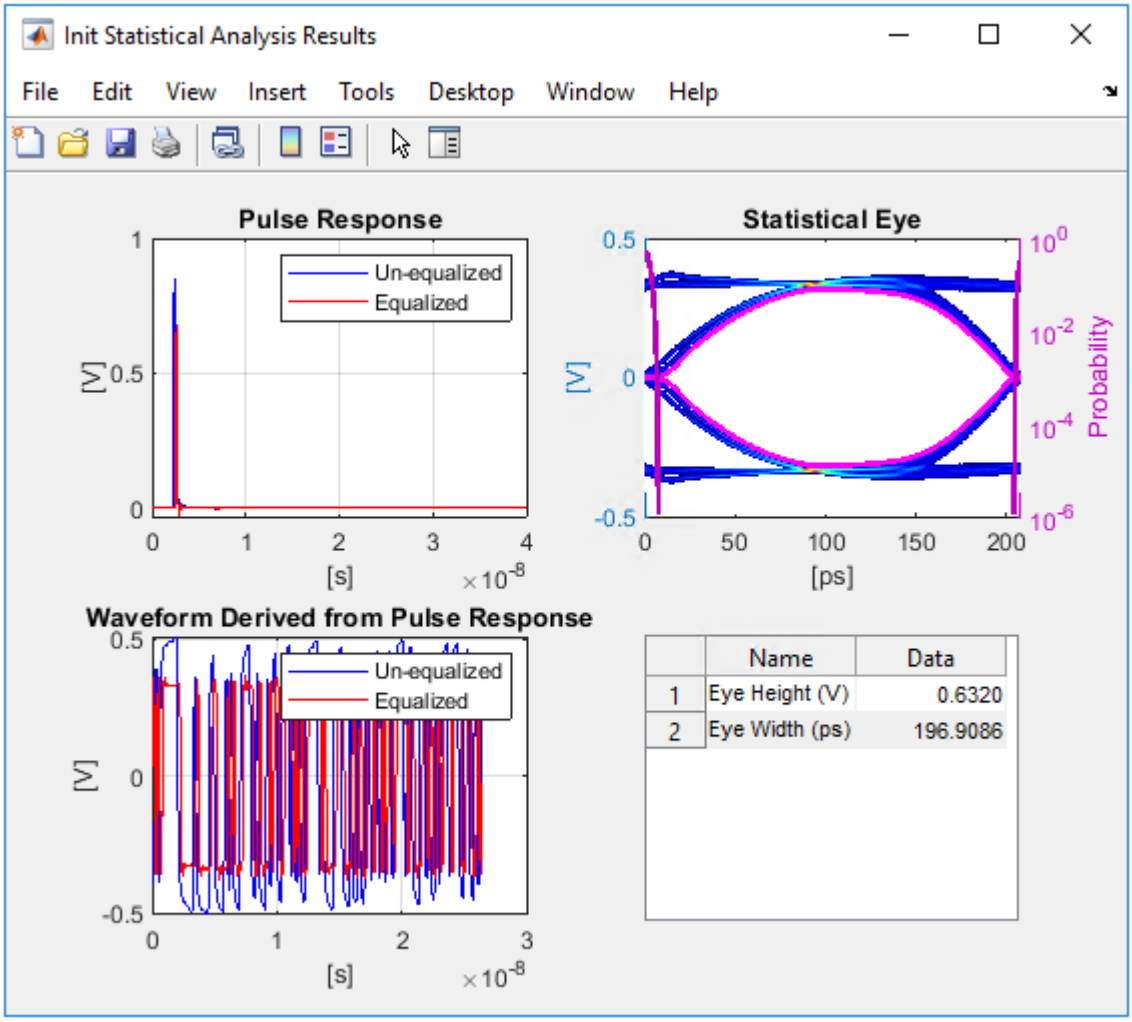
Run the Model

Run the model to simulate the SerDes system.

Two plots are generated. The first is a live time domain (GetWave) eye diagram that is updated as the model is running.



After the simulation has completed the second plot contains four views of the statistical (Init) results, similar to what is available in the SerDes Designer app.



Review Tx FFE Block

- Inside the Tx subsystem, double-click the FFE block to open the FFE Block Parameters dialog box.
- The **Tap Weights** are carried over from the SerDes Designer app.

Review Rx CTLE Block

- Inside the Rx subsystem, double-click the CTLE block to open the CTLE Block Parameters dialog box.
- **DC gain**, **AC gain**, and **Peaking frequency** are carried over from the SerDes Designer app.
- CTLE **Mode** is set to `Adapt`, which means an optimization algorithm built into the CTLE system object selects the optimal CTLE configuration at run time.

Review Rx AGC Block

- Inside the Rx subsystem, double-click the AGC block to open the AGC Block Parameters dialog box.
- The **Target RMS voltage** is carried over from the SerDes Designer app.

- The **Maximum gain** is set to `10` and **Averaging length** (the number of bits over which the average is calculated) is set to `100`. These values are reasonable for a generic controller model.

Update Rx DFECDR Block

- Inside the Rx subsystem, double-click the DFECDR block to open the DFECDR Block Parameters dialog box.
- The **Initial tap weights**, **Minimum DFE tap value,** and **Maximum tap value** RMS settings are carried over from the SerDes Designer app. The **Adaptive gain** and **Adaptive step size** are set to `3e-06` and `1e-06`, respectively, which are reasonable values based on DDR5 Controller expectations.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect **Phase offset** and **Reference offset** to remove these parameters from the AMI file, effectively hard-coding these parameters to their current values.

Generate DDR5 Controller IBIS-AMI Models

The final part of this example takes the customized Simulink model, modifies the AMI parameters for a DDR5 Controller, and then generates IBIS-AMI-compliant DDR5 Controller model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **Open SerDes IBIS-AMI Manager** button. In the **IBIS** tab inside the SerDes IBIS-AMI manager dialog box, the analog model values are converted to standard IBIS parameters that can be used by any industry-standard simulator.

Update Transmitter (Tx) AMI Parameters

Open the **AMI-Tx** tab in the SerDes IBIS-AMI manager dialog box. The reserved parameters are listed first followed by the model-specific parameters adhering to the format of a typical AMI file.

Set Pre-Emphasis Tap

- Highlight **TapWeight -1**
- Click the **Edit...** to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.2`, and **Max** = `0.2`.
- Click **OK** to save the changes**.**

Set Main Tap

- Highlight **TapWeight 0**.
- Click the **Edit...** button to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `1`, **Min** = `0.6`, and **Max** = `1`.
- Click **OK**.

Set First Post-Emphasis Tap

- Highlight TapWeight **1**.
- Select the **Edit...** button to launch the **Add/Edit Parameter** dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.2`, and **Max** = `0.2`.

- Click **OK**.

Set Second Post-Emphasis Tap

- Highlight TapWeight **2**.
- Select the **Edit...** button to launch the **Add/Edit Parameter** dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min**= `-0.1`, and **Max** = `0.1`.
- Click **OK**.

Set Third Post-Emphasis Tap

- Highlight TapWeight **3**.
- Select the **Edit...** button to launch the **Add/Edit Parameter** dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.1`, and **Max** = `0.1`.
- Click **OK**.

Add Tx Jitter Parameters

To add Jitter parameters for the Tx model click the **Reserved Parameters...** button to bring up the Tx Add/Remove Jitter&Noise dialog, select the **Tx_Dj** and **Tx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Tx AMI file. The following jitter values can be adjusted to meet the DDR5 mask requirements for a specific controller.

Set Tx Deterministic Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0500`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Tx Random Jitter Value

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0025`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Update Receiver (Rx) AMI Parameters

Open the **AMI-Rx** tab in the SerDes IBIS-AMI manager dialog box. The reserved parameters are listed first followed by the model-specific parameters adhering to the format of a typical AMI file.

Set First DFE Tap Weight

- Highlight **TapWeight 1.**
- Click the **Edit...** button to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.2`, and **Max** = `0.05`.

- Click **OK**.

Set Second DFE Tap Weight

- Highlight **TapWeight 2**.
- Click the **Edit...** button to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.075`, and **Max** = `0.075`.
- Click **OK**.

Set Third DFE Tap Weight

- Highlight **TapWeight 3**.
- Click the **Edit...** button to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.06`, and **Max** = `0.06`.
- Click **OK**.

Set Fourth DFE Tap Weight

- Highlight **TapWeight 4**.
- Click the **Edit...** button to launch the Add/Edit Parameter dialog box.
- Make sure **Format** is set to **Range** and set **Typ** = `0`, **Min** = `-0.045`, and **Max** = `0.045`.
- Click **OK**.

Add Rx Jitter and Noise Parameters

To add Jitter parameters for the Rx model click the **Reserved Parameters...** button to bring up the Rx Add/Remove Jitter&Noise dialog, select the **Rx_Receiver_Sensitivity**, **Rx_Dj, Rx_Noise, Rx_UniformNoise** and **Rx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Rx AMI file. The following jitter and noise values can be adjusted to meet the DDR5 mask requirements for a specific controller.

Set Rx Receiver Sensitivity Value

- Select **Rx_Receiver_Sensitivity**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.040`
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Rx Deterministic Jitter Value

- Select **Rx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0125`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Rx Gaussian Noise Value

- Select **Rx_Noise**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.

- Set the **Current Value** to `0.0015`
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Rx Uniform Noise Value

- Select **Rx_UniformNoise**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0025`
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Set Rx Random Jitter Value

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.00375`
- Change the **Type** to `UI`.
- Change the **Format** to `Value`.
- Click **OK** to save the changes.

Export Models

Open the **Export** tab in the SerDes IBIS-AMI manager dialog box.

- Update the **Tx model name** to `ddr5_controller_tx`
- Update the **Rx model name** to `ddr5_controller_rx`
- Note that **Tx and Rx corner percentage** is set to `10`. This scales the minimum/maximum analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx AMI model settings. This creates model executables that support both statistical (Init) analysis and time-domain (GetWave) simulation.
- Set the Tx model **Bits to ignore** to `5` since there are five taps in the Tx FFE.
- Set the Rx model **Bits to ignore** to `250000` to allow sufficient time for the Rx DFE taps to settle during time domain simulations.
- Verify that both Tx and Rx are set to export and that all files have been selected to be generated (**IBIS file**, **AMI file(s)** and **DLL file(s)**).
- Set the **IBIS file name** to `temp_ddr5_controller.ibs`
- Click the **Export** button to generate models in the Target directory.

Update DDR5 Analog Models

To accommodate different topologies, loading configurations, data rates and transfers, DDR5 requires variable output drive strength and input on-die termination (ODT). While the same algorithmic AMI model is used, multiple analog models are required to cover all these use cases. The generation of these analog models is out of scope for this example, so a completed IBS file with the following analog models in it is available in the current example directory:

- POD11_IO_ZO50_ODTOFF: 50 ohm output impedance with no input ODT.

- POD11_IN_ODT40_C: Input with 40 ohm ODT.
- POD11_IN_ODT60_C: Input with 60 ohm ODT.

To generate this complete IBIS file, the following changes were made to temp_ddr5_controller.ibs using a text editor:

- Created one pin with a signal_name of DQ1_controller and model_name of dq.
- Changed the driver Model_type to I/O and named it POD11_IO_Z050_ODTOFF.
- Added two receiver models and named them POD11_IN_ODT40_C and POD11_IN_ODT60_C, respectively.
- Added VI curves and Algorithmic Model sections to all above mentioned models.
- Added a Model Selector section that references the above mentioned models.

Test Generated IBIS-AMI Models

The DDR5 transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry-standard AMI model simulator.

References

1 IBIS 7.0 Specification, https://ibis.org/ver7.0/ver7_0.pdf.

2 SiSoft Support Knowledge Base Article: DDR4 Registered - Rawcard B for 3 slot system, https://sisoft.na1.teamsupport.com/knowledgeBase/8976521.

See Also

AGC | CTLE | DFECDR | FFE | **SerDes Designer**

More About

- “DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model” on page 7-15
- “Design DDR5 IBIS-AMI Models to Support Back-Channel Link Training” on page 7-56

External Websites

- https://www.sisoft.com/support/

CEI-56G-LR Transmitter/Receiver IBIS-AMI Model

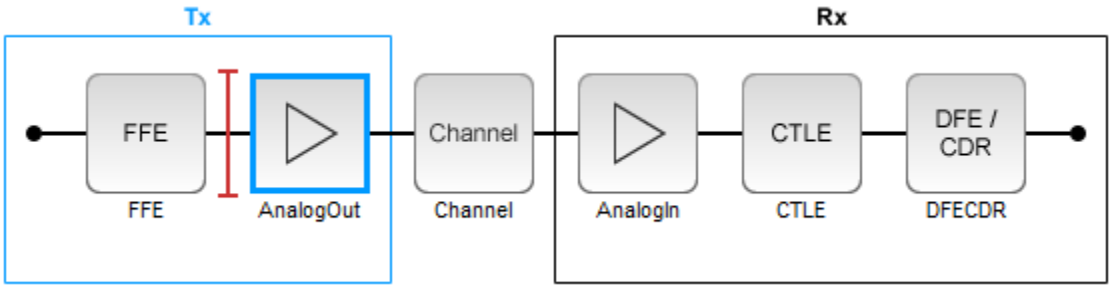
This example shows how to create generic CEI-56G-LR transmitter and receiver IBIS-AMI models using the library blocks in SerDes Toolbox™. The generated models conform to the IBIS-AMI and OIF-CEI-04.0 specifications.

CEI-56G-LR Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example sets up the target transmitter and receiver AMI model architecture using the datapath blocks required for CEI-56G in the SerDes Designer app. The model is then exported to Simulink® for further customization.

This example uses the SerDes Designer model cei_56G_lr_txrx. Type the following command in the MATLAB® command window to open the model:

```
>> serdesDesigner('cei_56g_lr_txrx')
```



A CEI-56G-LR compliant transmitter uses a 4-tap feed forward equalizer (FFE) with two pre-taps and one post-tap. The receiver model uses a continuous time linear equalizer (CTLE) with 17 pre-defined settings, and a 12 to 18 tap decision feedback equalizer (DFE). To support this configuration the SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to 35.71 ps, for a symbol rate of 28 GBaud and a PAM4 rate of 56 Gbps.
- **Target BER** is set to 100e-6, which assumes a compliant receiver with FEC.
- **Modulation** is set to PAM4.
- **Samples per Symbol** and **Signaling** are kept at default values, which are respectively 16 and Differential.

Transmitter Model Setup

- The Tx FFE block is set up for two pre-taps and one post-tap by including four tap weights, as specified in the OIF-CEI-04.0 specification. This is done with the array [0 0 1 0], where the main tap is specified by the largest value in the array.
- The Tx AnalogOut model is set up so that **Voltage** is 1.0 V, **Rise time** is 2.905 ps, **R** (single-ended output resistance) is 50 Ohms, and **C** (capacitance) is 0.16 pF.

Channel Model Setup

- **Channel loss** is set to 20 dB.

- **Differential impedance** is kept at default 100 Ohms.
- **Target Frequency** is set to the Nyquist frequency, 14 GHz.

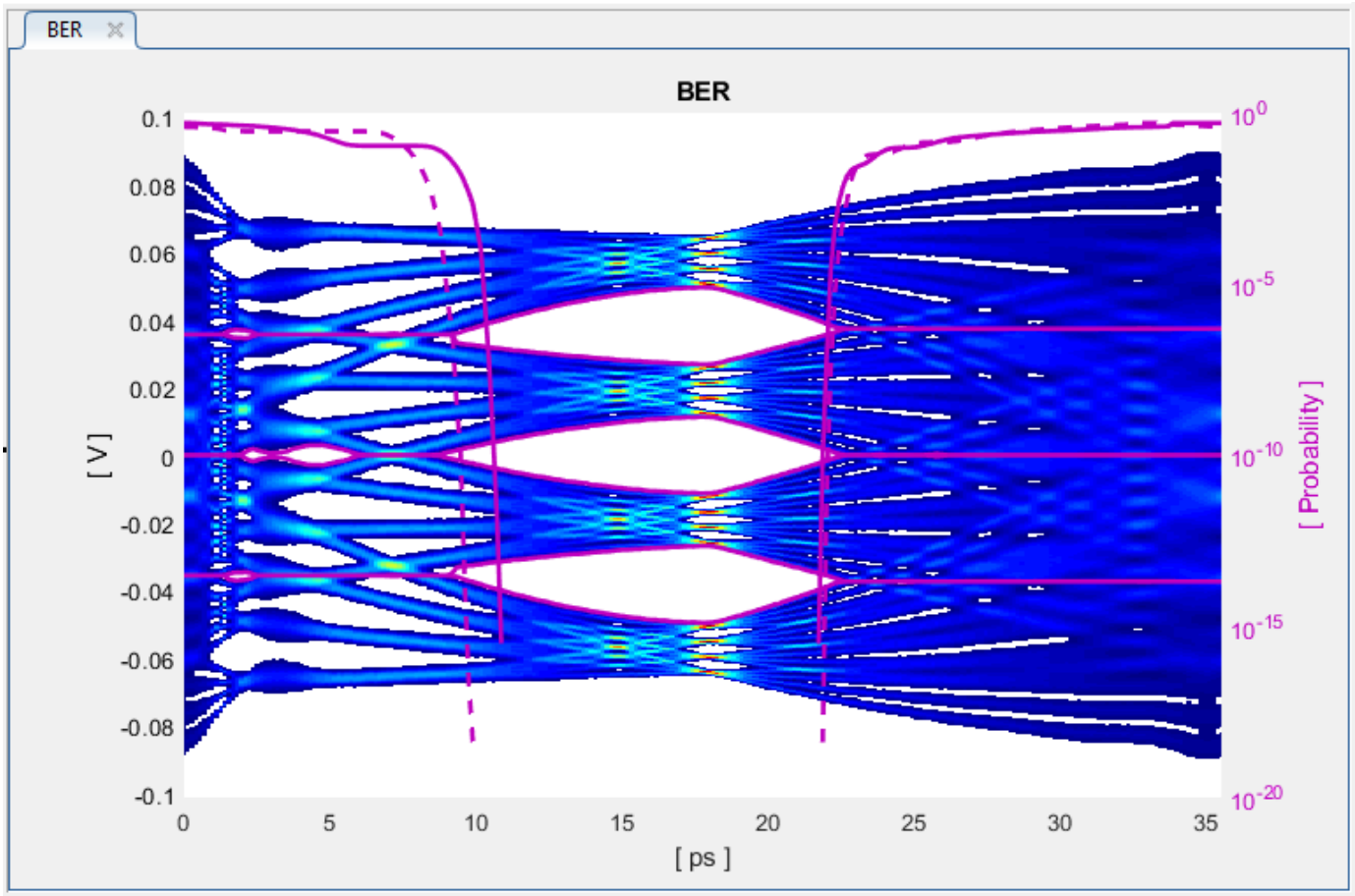
Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (single-ended input resistance) is 50 Ohms and **C** (capacitance) is 0.16 pF.
- The Rx CTLE block is set up for 147 configurations using the **GPZ** (Gain Pole Zero) matrix.
- The Rx DFE/CDR block is set up for 18 DFE taps. The limits for the taps are set to `-0.7` to `0.7`.

Plot Statistical Results

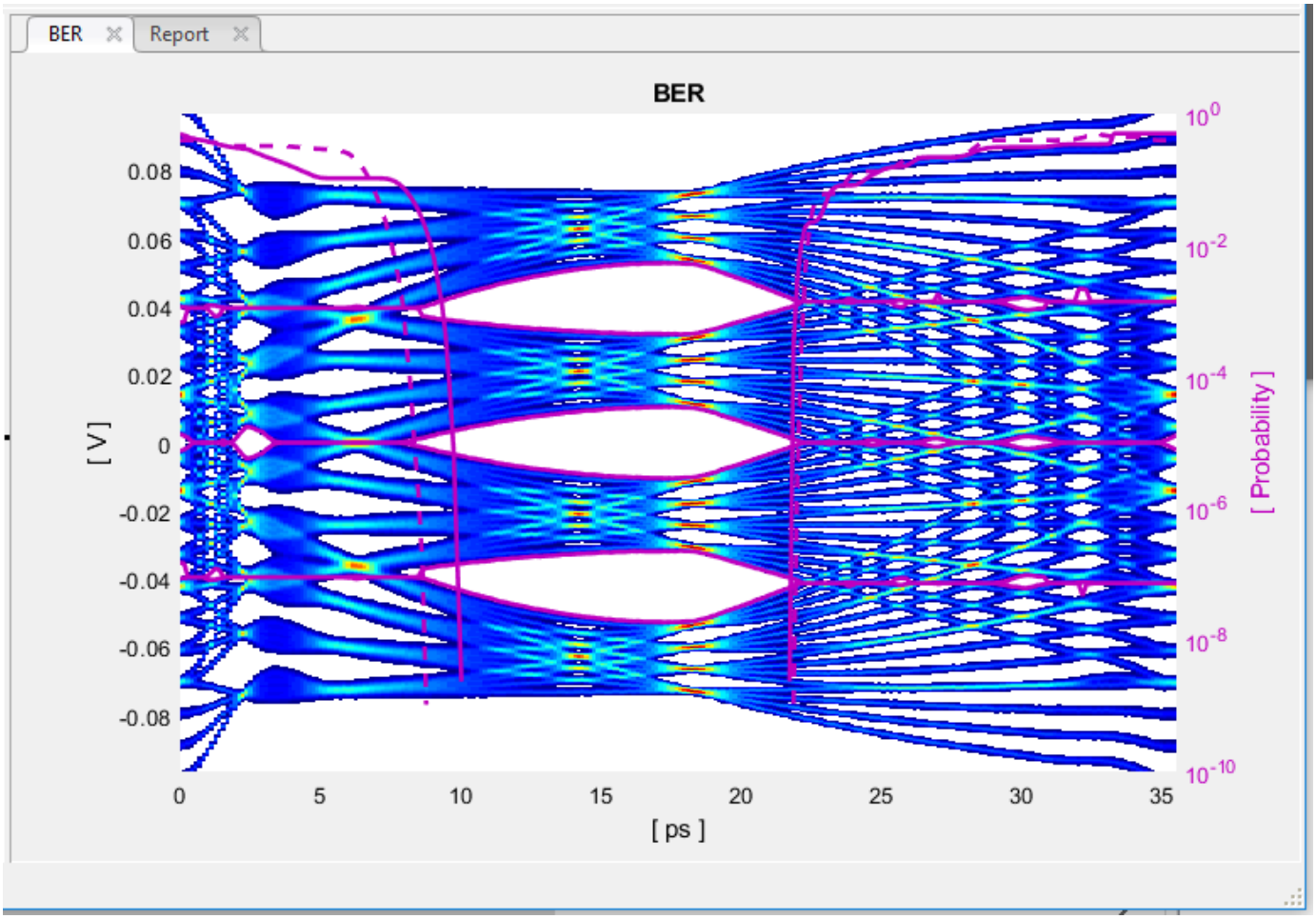
Use the SerDes Designer plots to visualize the results of the CEI-56G-LR setup.

Add the BER plot from **Add Plots** and observe the results.



Add the report from Add Plots and observe that the CTLE Config is 129.

Change the Rx CTLE **Mode** parameter to **`fixed`** and the **ConfigSelect** parameter value from 129 to 8 and observe how this changes the data eye.



Before continuing, reset the value of Rx CTLE **Mode** back to `adapt`. Resetting here will avoid the need to set it again after the model has been exported to Simulink.

Export SerDes System to Simulink

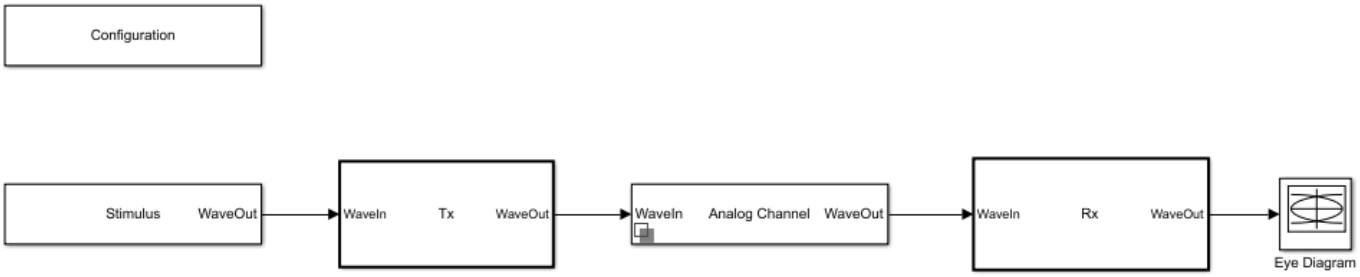
Click on the **Export** button to export the above configuration to Simulink for further customization and generation of the AMI model executables.

CEI-56G-LR Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the SerDes Designer app and customizes it as required for CEI-56G-LR in Simulink.

Review Simulink Model Setup

The SerDes System exported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app have been transferred to the Simulink model. Save the model and review each block setup.

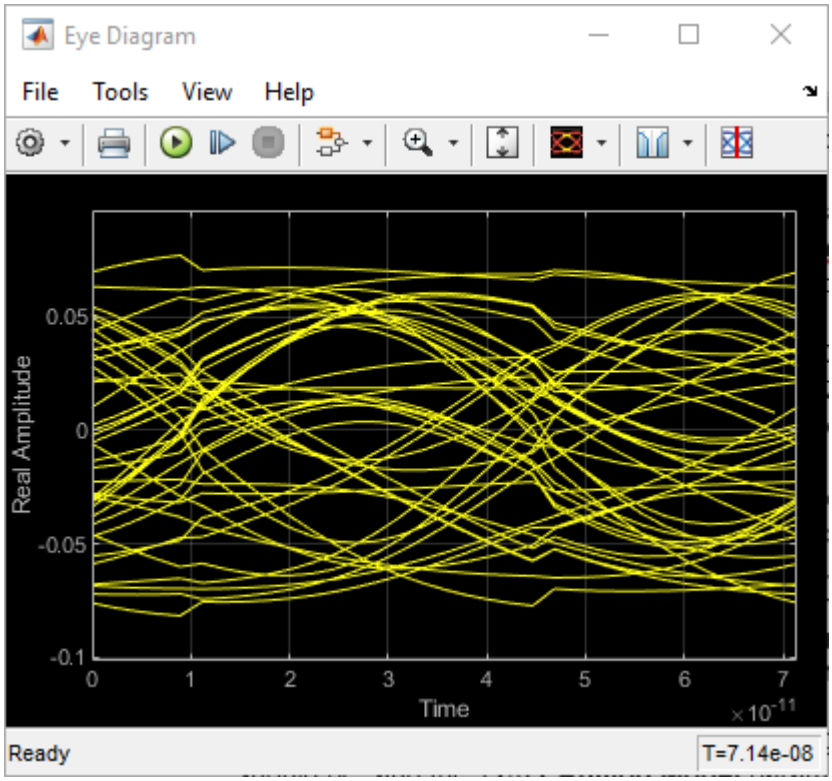


- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbo**l, **Target BER**, **Modulation** and **Signaling** are carried over from the SerDes Designer app.
- Double click the Stimulus block to open the Block Parameters dialog box. You can set the **PRBS** (pseudorandom binary sequence) order and the number of symbols to simulate. The settings for this block are not carried over from the SerDes Designer app.
- Double click the Tx block to look inside the Tx subsystem. The subsystem has the FFE block carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.
- Double click the Analog Channel block to open the Block Parameters dialog box. The parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx analog model parameters are carried over from the SerDes Designer app.
- Double click on the Rx block to look inside the Rx subsystem. The subsystem has the CTLE and DFECDR blocks carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.

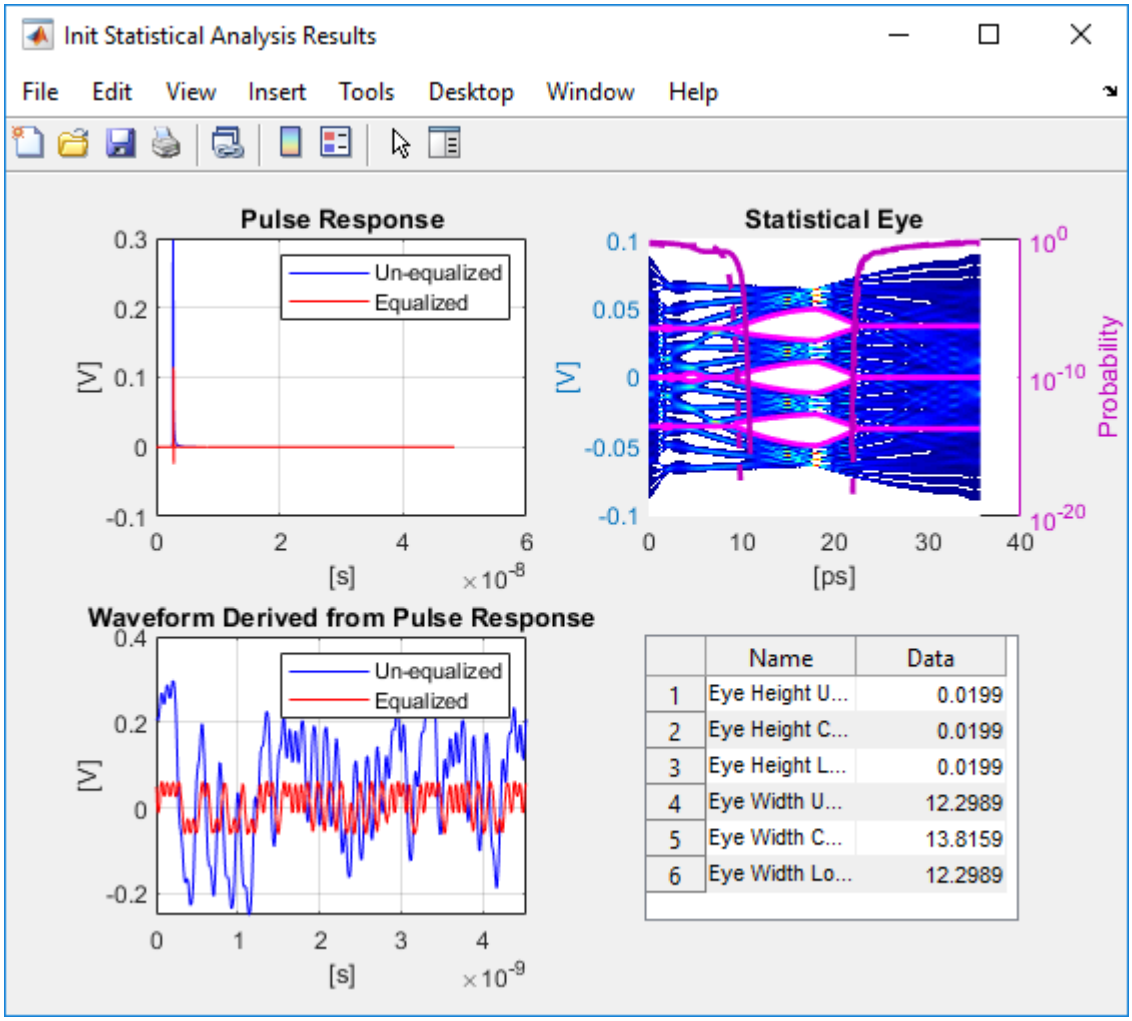
Run the Model

Run the model to simulate the SerDes System.

Two plots are generated. The first is a live time domain (GetWave) eye diagram that is updated as the model is running.



After the simulation has completed the second plot contains four views of the statistical (Init) results, similar to what is available in the SerDes Designer App.



Update Tx FFE Block

- Inside the Tx subsystem, double click the FFE block to open the FFE Block Parameters dialog box.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect the **Mode** parameter to remove this parameter from the AMI file, effectively hard-coding the current value of **Mode** in the final AMI model to Fixed.

Review Rx CTLE Block

- Inside the Rx subsystem, double click the CTLE block to open the CTLE Block Parameters dialog box.
- **Gain pole zero** data is carried over from the SerDes Designer app.
- CTLE **Mode** is set to Adapt, which means an optimization algorithm built into the CTLE system object selects the optimal CTLE configuration at run time.

Update Rx DFECDR Block

- Inside the Rx subsystem, double click the DFECDR block to open the DFECDR Block Parameters dialog box.

- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect the **Phase offset** and **Reference offset** parameters to remove these parameters from the AMI file, effectively hard-coding these parameters to their current values.

Generate CEI-56G-LR Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model, modifies the AMI parameters for CEI-56G-LR, then generates IBIS-AMI compliant CEI-56G-LR model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **SerDes IBIS-AMI Manager** button. In the **IBIS** tab inside the SerDes IBIS-AMI manager dialog box, the analog model values are converted to standard IBIS parameters that can be used by any industry standard simulator. In the **AMI-Tx** and **AMI-Rx** tabs in the SerDes IBIS-AMI manager dialog box, the reserved parameters are listed first followed by the model specific parameters following the format of a typical AMI file.

Add Tx Jitter Parameters

To add Jitter parameters for the Tx model, in the **AMI-Tx** tab click the **Reserved Parameters...** button to bring up the Tx Add/Remove Jitter&Noise dialog, select the **Tx_DCD**, **Tx_Dj** and **Tx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Tx AMI file. The following ranges allow you to fine-tune the jitter values to meet CEI-56G-LR jitter mask requirements.

Set Tx DCD Jitter Value

- Select **Tx_DCD**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.1`
- Click **OK** to save the changes.

Set Tx Dj Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.1`
- Click **OK** to save the changes.

Set Tx Rj Jitter Value

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.05`
- Click **OK** to save the changes.

Export Models

Select the **Export** tab in the **SerDes IBIS-AMI manager** dialog box.

- Update the **Tx model name** to `cei_56g_lr_tx`
- Update the **Rx model name** to `cei_56g_lr_rx`
- Note that the Tx and Rx **corner percentage** is set to 10%. This will scale the min/max analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx. This will create model executables that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Tx model Bits to ignore value** to 4 since there are four taps in the Tx FFE.
- Set the **Rx model Bits to ignore value** to 200000 to allow sufficient time for the Rx DFE taps to settle during time domain simulations.
- Verify that **Both Tx and Rx** are set to Export and that all files have been selected to be generated (IBIS file, AMI files and DLL files).
- Set the **IBIS file name** to be `cei_56g_lr_serdes.ibs`
- Press the **Export** button to generate models in the **Target directory**.

Test Generated IBIS-AMI Models

The CEI-56G-LR transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry standard AMI model simulator.

References

1 IBIS 6.1 Specification, https://ibis.org/ver6.1/ver6_1.pdf.

2 SiSoft Support Knowledge Base Article: CEI-56G-LR, https://sisoft.na1.teamsupport.com/knowledgeBase/11501730.

See Also

CTLE | DFECDR | FFE | **SerDes Designer**

More About

- “Managing AMI Parameters” on page 6-2

External Websites

- https://www.sisoft.com/support/

USB3.1 Transmitter/Receiver IBIS-AMI Model

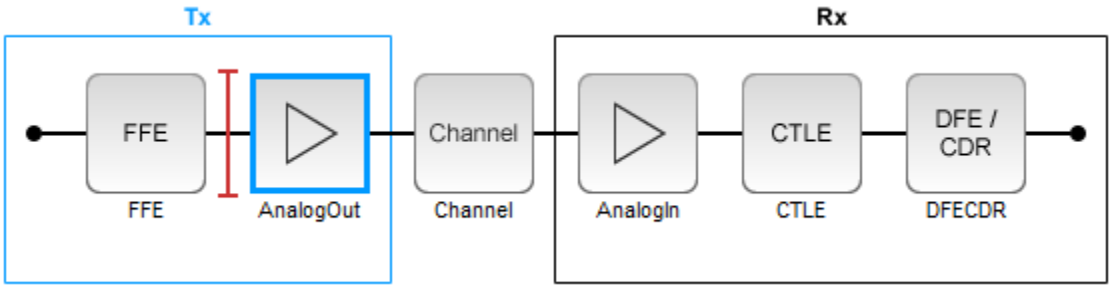
This example shows how to create generic Universal Serial Bus version 3.1 (USB3.1) transmitter and receiver IBIS-AMI models using the library blocks in SerDes Toolbox™. The generated models conform to the IBIS-AMI and USB3.1 specifications.

USB3.1 Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example sets up the target transmitter and receiver AMI model architecture using the datapath blocks required for USB3.1 in the SerDes Designer app. The model is then exported to Simulink® for further customization.

This example uses the SerDes Designer model usb3_1_txrx_ami. Type the following command in the MATLAB® command window to open the model:

```
>> serdesDesigner('usb3_1_txrx_ami')
```



A USB3.1 compliant transmitter uses a 3-tap feed forward equalizer (FFE) with one pre-tap and one post-tap. The receiver model uses a continuous time linear equalizer (CTLE) with seven pre-defined settings, and a 1-tap decision feedback equalizer (DFE). To support this configuration the SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to 100 ps, since the maximum allowable USB3.1 operating frequency is 10 GHz.
- **Target BER** is set to 1e-12 as specified in the USB3.1 specification.
- **Samples per Symbol**, **Modulation**, and **Signaling** are kept at default values, which are respectively 16, NRZ (non-return to zero), and Differential.

Transmitter Model Setup

- The Tx FFE block is set up for one pre- and one post-tap by including three tap weights, as specified in the USB3.1 specification. This is done with the array [0 1 0], where the main tap is specified by the largest value in the array.
- The Tx AnalogOut model is set up so that **Voltage** is 1.00 V, **Rise time** is 60 ps, **R** (single-ended output resistance) is 50 Ohms, and **C** (capacitance) is 0.5 pF.

Channel Model Setup

- **Channel loss** is set to 15dB.

- **Differential impedance** is kept at default 100 Ohms.
- **Target Frequency** is set to the Nyquist frequency, 5 GHz.

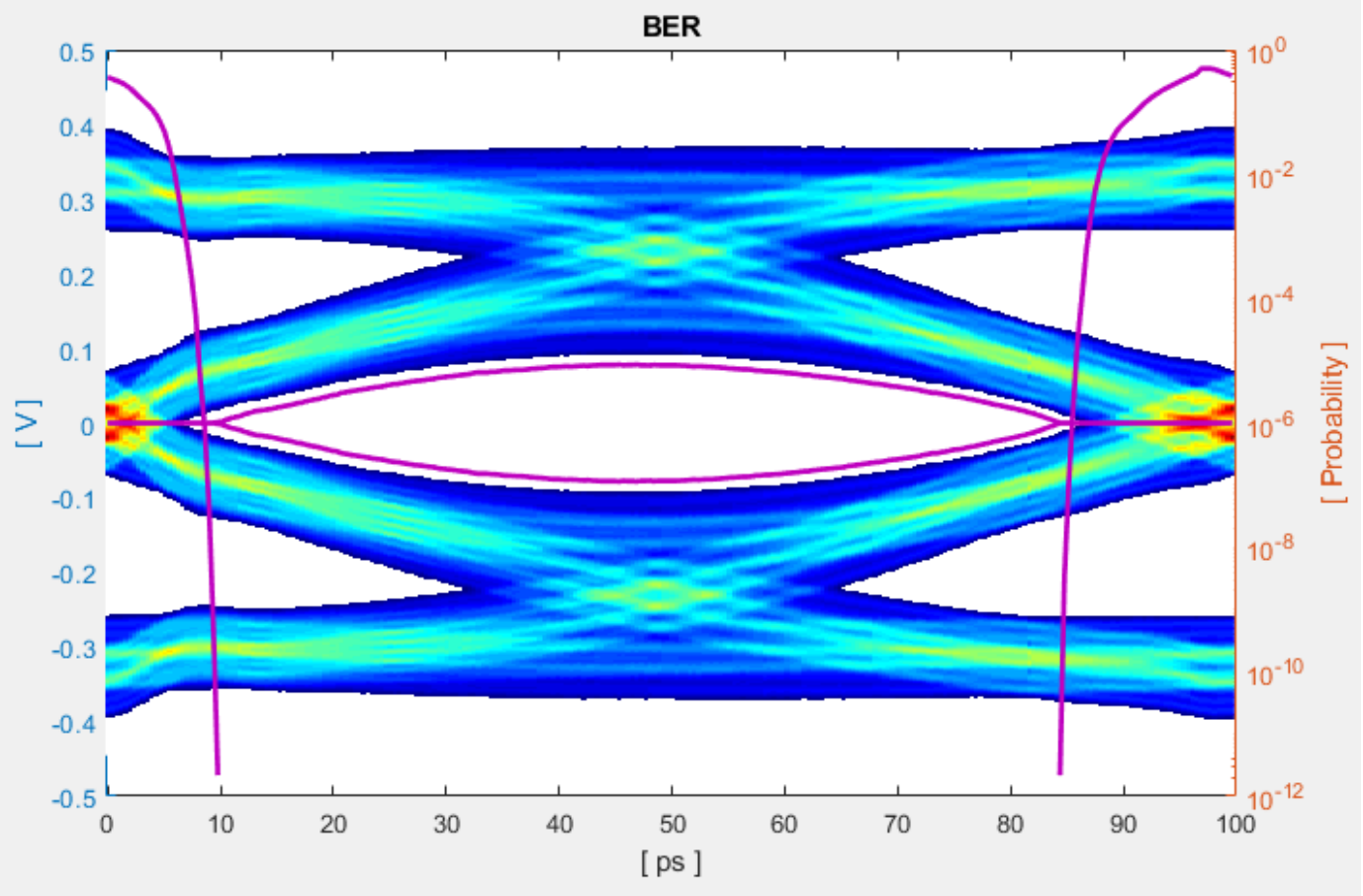
Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (single-ended input resistance) is 50 Ohms and **C** (capacitance) is 0.5 pF.
- The Rx CTLE block is set up for 7 configurations. The **GPZ** (Gain Pole Zero) matrix data is derived from the transfer function given in the USB3.1 Behavioral CTLE specification.
- The Rx DFE/CDR block is set up for one DFE tap. The limits for the tap are as defined by the USB3.1 specification: `+/-50 mV`.

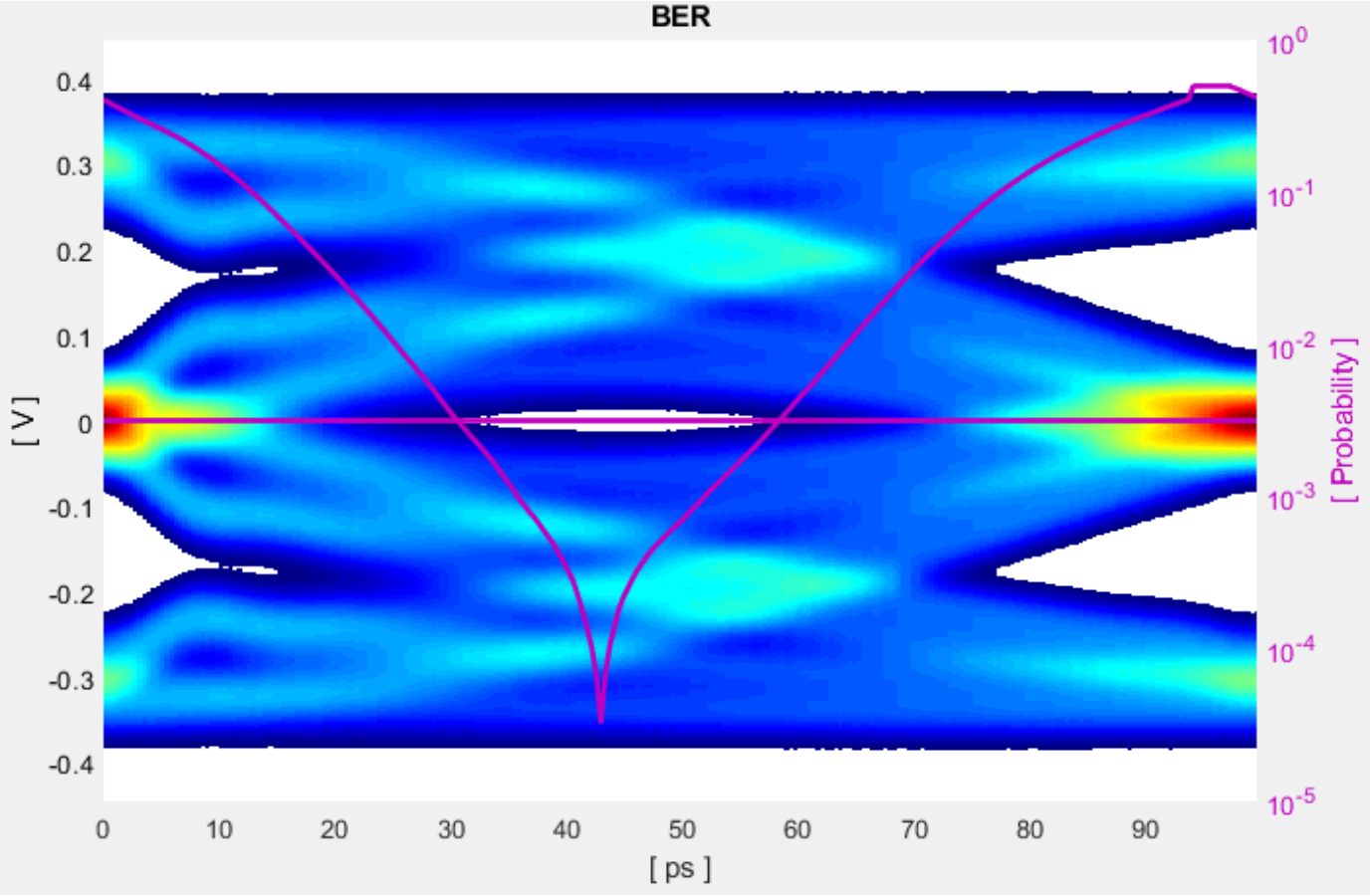
Plot Statistical Results

Use the SerDes Designer plots to visualize the results of the USB3.1 setup.

Add the BER plot from **ADD Plots** and observe the results.



Change the Rx CTLE **Mode** parameter from `adapt` to `fixed` and change the **ConfigSelect** parameter value from 6 to 0 and observe how this changes the data eye.



Before continuing, reset the value of Rx CTLE **Mode** back to `adapt`. Resetting the value here will avoid the need to set it again after the model has been exported to Simulink.

Export SerDes System to Simulink

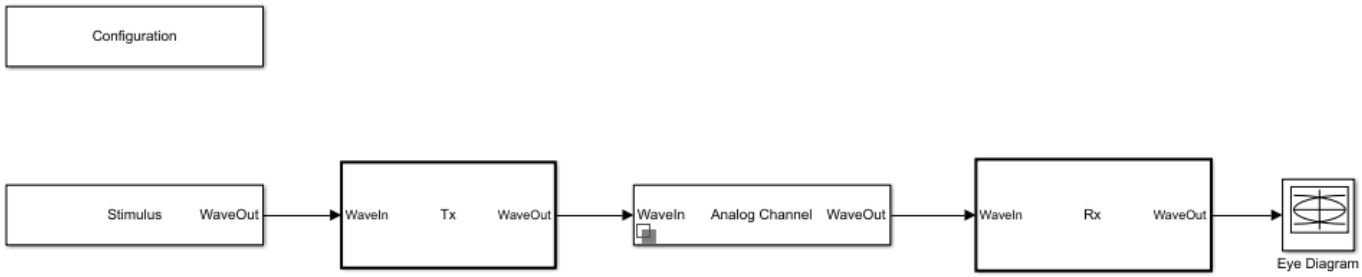
Click on the **Export** button to export the above configuration to Simulink for further customization and generation of the AMI model executables.

USB3.1 Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the SerDes Designer app and customizes it as required for USB3.1 in Simulink.

Review Simulink Model Setup

The SerDes System imported into Simulink consists of the Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app have been transferred to the Simulink model. Save the model and review each block setup.

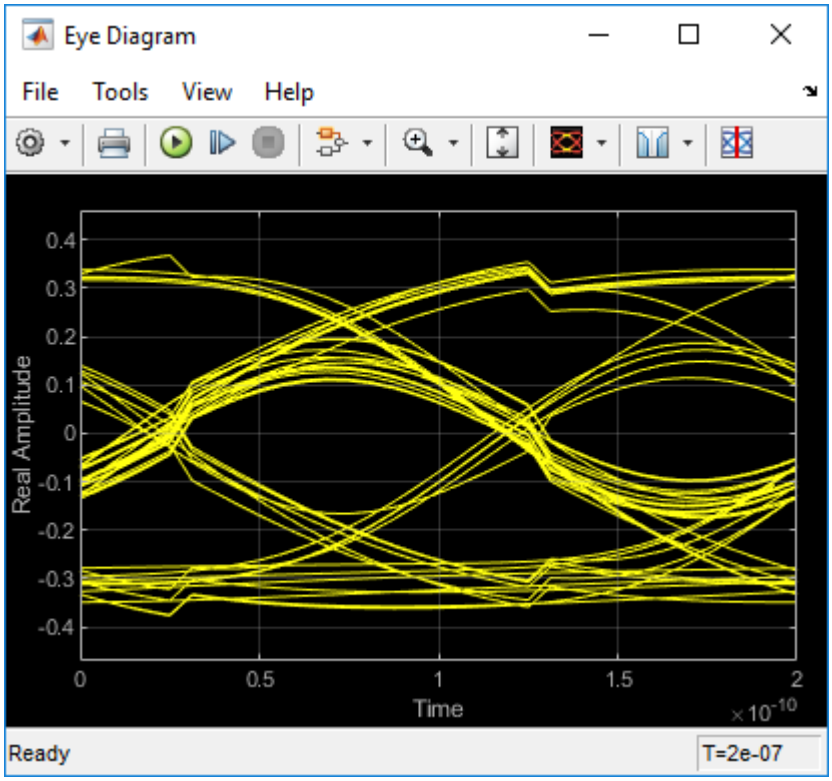


- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** are carried over from the SerDes Designer app.
- Double click the Stimulus block to open the Block Parameters dialog box. You can set the **PRBS** (pseudorandom binary sequence) order and the number of symbols to simulate. This block is not carried over from the SerDes Designer app.
- Double click the Tx block to look inside the Tx subsystem. The subsystem has the FFE block carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.
- Double click the Analog Channel block to open the Block Parameters dialog box. The parameter values for **Target frequency**, **Loss**, **Impedance** and Tx/Rx **Analog Model** parameters are carried over from the SerDes Designer app.
- Double click on the Rx block to look inside the Rx subsystem. The subsystem has the CTLE and DFECDR blocks carried over from the SerDes Designer app. An Init block is also introduced to model the statistical portion of the AMI model.

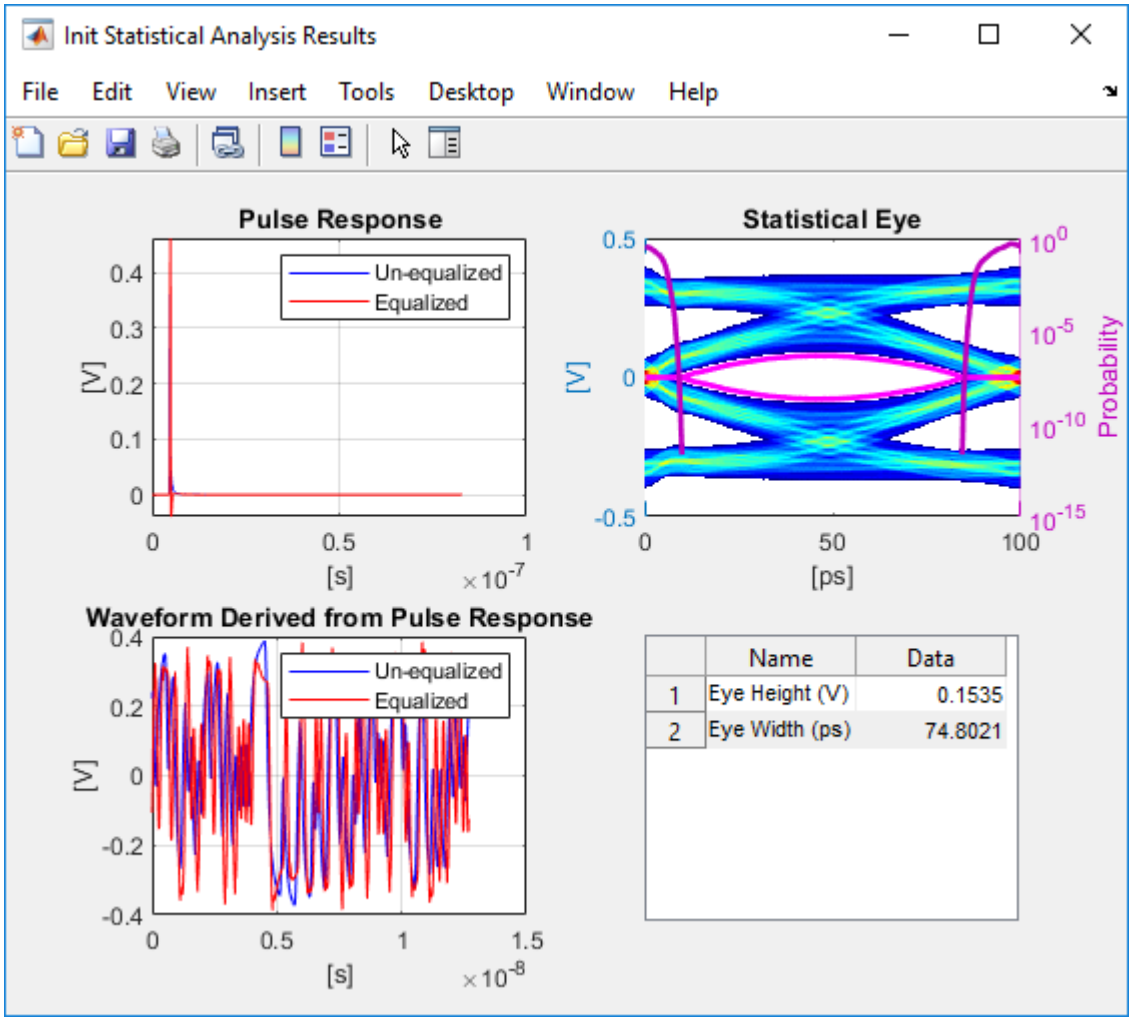
Run the Model

Run the model to simulate the SerDes System.

Two plots are generated. The first is a live time domain (GetWave) eye diagram that is updated as the model is running.



After the simulation has completed the second plot contains four views of the statistical (Init) results, similar to what is available in the SerDes Designer App.



Update Tx FFE Block

- Inside the Tx subsystem, double click the FFE block to open the FFE Block Parameters dialog box.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect the **Mode** parameter to remove this parameter from the AMI file, effectively hard-coding the current value of **Mode** in the final AMI model to Fixed.

Review Rx CTLE Block

- Inside the Rx subsystem, double click the CTLE block to open the CTLE Block Parameters dialog box.
- **Gain pole zero** data is carried over from the SerDes Designer app. This data is derived from the transfer function given in the USB3.1 Behavioral CTLE specification.
- CTLE **Mode** is set to Adapt, which means an optimization algorithm built into the CTLE system object selects the optimal CTLE configuration at run time.

Update Rx DFECDR Block

- Inside the Rx subsystem, double click the DFECDR block to open the DFECDR Block Parameters dialog box.
- Expand the **IBIS-AMI parameters** to show the list of parameters to be included in the IBIS-AMI model.
- Deselect the **Phase offset** and **Reference offset** parameters to remove these parameters from the AMI file, effectively hard-coding these parameters to their current values.

Generate USB3.1 Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model, modifies the AMI parameters for USB3.1, then generates IBIS-AMI compliant USB3.1 model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **SerDes IBIS-AMI Manager** button. In the **IBIS** tab inside the SerDes IBIS-AMI manager dialog box, the analog model values are converted to standard IBIS parameters that can be used by any industry standard simulator. In the **AMI-Tx** and **AMI-Rx** tabs in the SerDes IBIS-AMI manager dialog box, the reserved parameters are listed first followed by the model specific parameters following the format of a typical AMI file.

Add Tx Jitter Parameters

To add Jitter parameters for the Tx model, in the **AMI-Tx** tab click the **Reserved Parameters...** button to bring up the Tx Add/Remove Jitter&Noise dialog, select the **Tx_Dj** and **Tx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Tx AMI file. The following ranges allow you to fine-tune the jitter values to meet USB3.1 jitter mask requirements.

Set Tx Dj Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.17`
- Click **OK** to save the changes.

Set Tx Rj Jitter Value

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.012`
- Click **OK** to save the changes.

Add Rx Jitter and Noise Parameters

To add Jitter parameters for the Rx model, in the **AMI-Rx** tab click the **Reserved Parameters...** button to bring up the Rx Add/Remove Jitter&Noise dialog, select the **Rx_Receiver_Sensitivity**, **Rx_Dj** and **Rx_Rj** boxes and click **OK** to add these parameters to the Reserved Parameters section of the Rx AMI file. The following ranges allow you to fine-tune the jitter values to meet USB3.1 jitter mask requirements.

Set Rx Receiver_Sensitivity Value

- Select **Rx_Receiver_Sensitivity**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.025`
- Change the **Format** to `Range`.
- Set the **Typ** value to `0.025`
- Set the **Min** value to `0.015`
- Set the **Max** value to `0.100`
- Click **OK** to save the changes.

Set Rx Dj Jitter Value

- Select **Rx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.3`
- Click **OK** to save the changes.

Set Rx Rj Jitter Value

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Set the **Current Value** to `0.0`.
- Change the **Type** to `UI`.
- Change the **Format** to `Range`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `0.015`
- Click **OK** to save the changes.

Export Models

Select the **Export** tab in the SerDes IBIS-AMI manager dialog box.

- Update the **Tx model name** to `usb3_1_tx`
- Update the **Rx model name** to `usb3_1_rx`

- Note that the Tx and Rx **corner percentage** is set to 10%. This will scale the min/max analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx. This will create model executables that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Tx model Bits to ignore value** to 3 since there are three taps in the Tx FFE.
- Set the **Rx model Bits to ignore value** to 20000 to allow sufficient time for the Rx DFE taps to settle during time domain simulations.
- Verify that **Both Tx and Rx** are set to Export and that all files have been selected to be generated (IBIS file, AMI files and DLL files).
- Set the **IBIS file name** to be `usb3_1_serdes.ibs`
- Press the **Export** button to generate models in the **Target directory**.

Test Generated IBIS-AMI Models

The USB3.1 transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry standard AMI model simulator.

References

1. USB, https://www.usb.org.
2. IBIS 6.1 Specification, https://ibis.org/ver6.1/ver6_1.pdf.
3. SiSoft Support Knowledge Base Article: USB-3.1, https://sisoft.na1.teamsupport.com/knowledgeBase/8977326.

See Also

CTLE | DFECDR | FFE | **SerDes Designer**

More About

- “Managing AMI Parameters” on page 6-2

External Websites

- https://www.sisoft.com/support/

Design DDR5 IBIS-AMI Models to Support Back-Channel Link Training

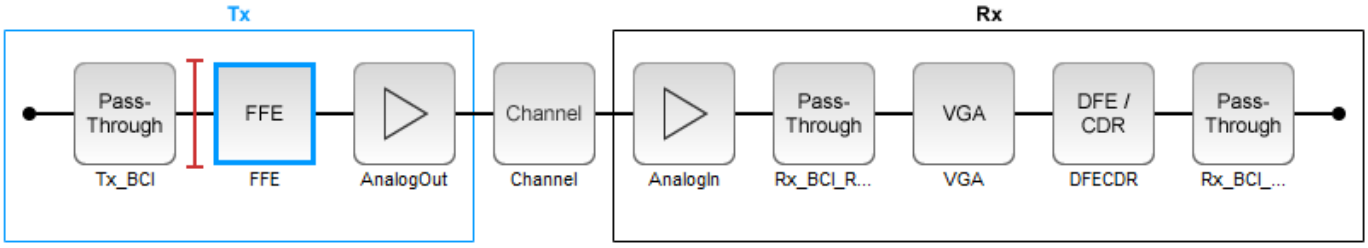
This example shows how to create transmitter and receiver AMI models that support link training communication (back-channel) as defined in the IBIS 7.0 specification by adding to the library blocks in SerDes Toolbox™. This example uses a DDR5 write transfer (Controller to SDRAM) to demonstrate the setup.

DDR5 Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

The first part of this example starts with the DDR5 controller transmitter model from “DDR5 Controller Transmitter/Receiver IBIS-AMI Model” on page 7-26 and the SDRAM receiver AMI model from “DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model” on page 7-15. Add a few additional pass-through blocks to support the back-channel communication and export the model to Simulink® for further customization.

Open the model `DDR5_Write_txrx_ami` by typing the following command in the MATLAB® command window:

```
>> serdesDesigner('DDR5_Write_txrx_ami')
```



For a write transaction, the transmitter (Tx) is a DDR5 controller using 3-tap feed forward equalization (FFE), while the receiver (Rx) is using a variable gain amplifier (VGA) with 7 pre-defined settings and a 4-tap decision feedback equalizer (DFE) with built-in clock data recovery. To support this configuration the SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to `208.3` ps, since the target operating rate is `4.8`Gbps for DDR5-4800.
- **Target BER** is set to `100e-18`.
- **Signaling** is set to `Single-ended`.
- **Samples per Symbol** and **Modulation** are kept at default values, which are `16` and `NRZ` (nonreturn to zero), respectively.

Transmitter Model Setup

- The Pass-Through block Tx_BCI is a block used to support this back-channel implementation. The operation of this block will be described later in this example.
- The Tx FFE block is set up for one pre-tap, one main-tap, and one post-tap by including three tap weights. This is done with the array [0 1 0], where the main tap is specified by the largest value in the array. Tap ranges will be added later in the example when the model is exported to Simulink.

- The Tx AnalogOut model is set up so that **Voltage** is `1.1` V, **Rise time** is `100` ps, **R** (output resistance) is `50 ohms`, and **C** (capacitance) is `0.65` pF. The actual analog models used in the final model will be generated later in this example.

Channel Model Setup

- **Channel loss** is set to `5` dB, which is typical of DDR channels.
- **Single-ended impedance** is set to `40` ohms.
- **Target Frequency** is set to `2.4` GHz, which is the Nyquist frequency for 4.8 GHz

Receiver Model Setup

- The Pass-Through block Rx_BCI_Read is a block used to support this back-channel implementation. The operation of this block will be described later in this example.
- The Rx AnalogIn model is set up so that **R** (input resistance) is `40` ohms and **C** (capacitance) is `0.65`pF. The actual analog models used in the final model will be generated later in this example.
- The VGA block is set up with a **Gain** of 1 and the **Mode** set to on. Specific VGA presets will be added later in this example after the model is exported to Simulink.
- The DFECDR block is set up for four DFE taps by including four **Initial tap weights** set to 0. The **Minimum tap value** is set to `[-0.2 -0.075 -0.06 -0.045]` V, and the **Maximum tap value** is set to `[0.05 0.075 0.06 0.045]` V.
- The Pass-Through block Rx_BCI_Write is a block used to support this back-channel implementation. The operation of this block will be described later in this example.

Export SerDes System to Simulink

Click on the **Export** button to export the configuration to Simulink for further customization and generation of the AMI model executables.

DDR5 Tx/Rx IBIS-AMI Model Setup in Simulink

This part of the example takes the SerDes system exported by the SerDes Designer app and customizes it as required for DDR5 back-channel operation in Simulink.

Review Simulink Model Setup

The SerDes System imported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks. All the settings from the SerDes Designer app are transferred to the Simulink model. Save the model and review each block setup.

- Inside the Tx subsystem, double click the FFE block to open the FFE Block Parameters dialog box. Expand the **IBIS-AMI parameters** and deselect the **Mode** parameter, effectively hard-coding the current value of **Mode** in the final AMI model to `Fixed`.
- Inside the Rx subsystem, double click the VGA block to open the VGA Block Parameters dialog box. The **Mode** and **Gain** settings are carried over from the SerDes Designer app.
- Inside the Rx subsystem, double click the DFECDR block to open the DFECDR Block Parameters dialog box. The **Initial tap weights**, **Minimum DFE tap value**, and **Maximum tap value** RMS settings are carried over from the SerDes Designer app. The **Adaptive gain** and **Adaptive step size** are set to `3e-06` and `1e-06`, respectively, which are reasonable values based on DDR5 SDRAM expectations. Expand the **IBIS-AMI parameters** and deselect **Phase offset** and **Reference offset** parameters, effectively hard-coding these parameters to their current values.

Update Transmitter (Tx) AMI Parameters

Open the **AMI-Tx** tab in the SerDes IBIS-AMI manager dialog box. The reserved parameters are listed first followed by the model-specific parameters adhering to the format of a typical AMI file.

- Set the pre-emphasis tap: Edit **TapWeights -1** and set **Format** to `Range`, **Typ** to `0`, **Min** to `-0.2`, and **Max** to `0.2`.
- Set the main tap: Edit **TapWeights 0** and set **Format** to `Range`, **Typ** to `1`, **Min** to `0.6`, and **Max** to `1`.
- Set the post-emphasis tap: Edit **TapWeights 1** and set **Format** to `Range`, **Typ** to `0`, **Min** to `-0.2`, and **Max** to `0.2`.

Create new Tx back-channel AMI parameters

To support back-channel operation, additional control parameters are needed. In the **AMI-Tx** tab in the SerDes IBIS-AMI manager dialog, highlight **Tx_BCI** and add the following 6 new parameters:

- **FFE_Tapm1:** This parameter creates a Data Store that is used to pass the FFE pre tap value between Tx blocks during training. Click the **Add Parameter...** button. Set **Parameter Name** to `FFE_Tapm1`, **Current Value** to `0`, **Usage** to `InOut`, **Type** to `Float`, and **Format** to `Value`. Set the **Description** as: `Tx FFE Tap -1 for back-channel training`. Save the changes.
- **FFE_Tap0:** This parameter creates a Data Store that is used to pass the FFE main tap value between Tx blocks during training. Click the **Add Parameter...** button. Set **Parameter Name** to `FFE_Tap0`, **Current Value** to `0`, **Usage** to `InOut`, **Type** to `Float`, and **Format** to `Value`. Set the **Description** as: `Tx FFE Tap 0 for back-channel training`. Save the changes.
- **FFE_Tap1:** This parameter creates a Data Store that is used to pass the FFE post tap value between Tx blocks during training. Click the **Add Parameter...** button. Set **Parameter Name** to `FFE_Tap1`, **Current Value** to `0`, **Usage** to `InOut`, **Type** to `Float`, and **Format** to `Value`. Set the **Description** as: `Tx FFE Tap 1 for back-channel training`. Save the changes.
- **BCI_Protocol:** This parameter is only used to generate a parameter named "BCI_Protocol" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_Protocol`, **Current Value** to `"DDRx_Write"`, **Usage** to `Info`, **Type** to `String`, and **Format** to `Value`. Set the **Description** as: `This model supports the DDRx Write Example back-channel protocol. NOTE: This model does not currently support BCI_Protocol as an input to the model.` Save the changes.
- **BCI_ID:** This parameter is only used to generate a parameter named "BCI_ID" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_ID`, **Current Value** to `"bci_comm"`, **Usage** to `Info`, **Type** to `String`, and **Format** to `Value`. Set the **Description** as: `This model creates files with names beginning with 'bci_comm' for back-channel communication. NOTE: This model does not currently support BCI_ID as an input to the model.` Save the changes.
- **BCI_State:** This parameter creates a Data Store that is used to communicate the status of back-channel training: 1=Off, 2=Training, 3=Converged, 4=Failed, 5=Error. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_State`, **Usage** to `InOut`, **Type** to `Integer`, and **Format** to `List`. Set the **Description** as: `Back channel training status.` Set the **Default** to 2, **List values** to `[1 2 3 4 5]`, and **List_Tip values** to `["Off" "Training" "Converged" "Failed" "Error"],` then set the **Current Value** to `"Training".` Save the changes.

Update Receiver (Rx) AMI Parameters

On the **AMI-Rx** tab in the SerDes IBIS-AMI manager dialog box, the reserved parameters are listed first followed by the model-specific parameters adhering to the format of a typical AMI file.

- Set the VGA gain: Edit **Gain**. Set **Description** as: `Rx Amplifier Gain`. Make sure **Format** is set to `List` and set **Default to** `1`. Set **List values** as `[0.5 0.631 0.794 1 1.259 1.585 2]` and **List_Tip values** as `["-6 dB" "-4 dB" "-2 dB" "0 dB" "2 dB" "4 dB" "6 dB"]`, then set the **Current Value** to `0dB`. Save the changes.
- Set the first DFE tap weight: Edit **TapWeights 1**. Make sure **Format** is set to `Range` and set **Typ** = `0`, **Min** = `-0.2`, and **Max** = `0.05`. Save the changes.
- Set the second DFE tap weight: Edit **TapWeights 2**. Make sure **Format** is set to `Range` and set **Typ** = `0`, **Min** = `-0.075`, and **Max** = `0.075`. Save the changes.
- Set the third DFE tap weight: Edit **TapWeights 3**. Make sure **Format** is set to `Range` and set **Typ** = `0`, **Min** = `-0.06`, and **Max** = `0.06`. Save the changes.
- Set the fourth DFE tap weight: Edit **TapWeights 4**. Make sure **Format** is set to `Range` and set **Typ** = `0`, **Min** = `-0.045`, and **Max** = `0.045`. Save the changes.

Create new Rx back-channel AMI parameters

To support back-channel operation, additional control parameters are needed. In the **AMI-Rx** tab in the SerDes IBIS-AMI manager dialog, highlight **Rx_BCI_Write** and add the following new parameters (Note: **Rx_BCI_Read** does not require any additional parameters):

- **sampleVoltage:** This parameter creates a Data Store that will be used to pass the CDR sample voltage to the other Rx blocks during training. Click the **Add Parameter...** button. Set **Parameter Name** to `sampleVoltage`, **Current Value** to `0`, **Usage** to `InOut`, **Type** to `Float`, and **Format** to `Value`. Set the **Description** as: `Sample Voltage for back-channel training`. Save the changes.
- **BCI_Protocol:** This parameter only generates a parameter named "BCI_Protocol" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not be used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_Protocol`, **Current Value** to `"DDRx_Write"`, **Usage** to `Info`, **Type** to `String`, and **Format** to `Value`. Set the **Description** as: `This model supports the DDRx Write Example back-channel protocol. NOTE: This model does not currently support BCI_Protocol as an input to the model`. Save the changes.
- **BCI_ID:** This parameter only generates a parameter named "BCI_ID" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not be used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_ID`, **Current Value** to "bci_comm", **Usage** to `Info`, **Type** to `String`, and **Format** to `Value`. Set the **Description** as: `This model creates files with names beginning with 'bci_comm' for back-channel communication. NOTE: This model does not currently support BCI_ID as an input to the model`. Save the changes.
- **BCI_State:** This parameter creates a Data Store that is used to communicate the status of back-channel training: 1=Off, 2=Training, 3=Converged, 4=Failed, 5=Error. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_State`, **Usage** to `InOut`, **Type** to `Integer`, and **Format** to `List`. Set the **Description** as: `Back channel training status.` Set the **Default** to 2, **List values** to `[1 2 3 4 5]`, and **List_Tip values** to `["Off" "Training" "Converged" "Failed" "Error"],` then set the **Current Value** to `"Training".` Save the changes.

- **BCI_Message_Interval_UI:** This parameter only generates a parameter named "BCI_Message_Interval_UI" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not be used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_Message_Interval_UI`, **Current Value** to `64`, **Usage** to `Info`, **Type** to `Integer`, and **Format** to `Value`. Set the **Description** as: `BCI requires 1024 Samples Per Bit for proper operation`. Save the changes.
- **BCI_Training_UI:** This parameter only generates a parameter named "BCI_Training_UI" in the .ami file for compliance to the IBIS-AMI specification. This parameter is not be used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to `BCI_Training_UI`, **Current Value** to `100000`, **Usage** to `Info`, **Type** to `Integer`, and **Format** to `Value`. Set the **Description** as: `BCI training may require 100,000 UI to complete`. Save the changes.

Run Refresh Init

To propagate the new AMI parameters, run Refresh Init on both the Tx and Rx blocks.

- Double click the Init subsystem inside the Tx block and click the **Refresh Init** button.
- Double click the Init subsystem inside the Rx block and click the **Refresh Init** button.

Run the Model

Run the model to simulate the SerDes system and verify that the current setup compiles and runs with no errors or warnings. Two plots are generated. The first is a live time-domain (GetWave) eye diagram that is updated as the model is running. The second plot contains four views of the statistical (Init) results, like the plots available in the SerDes Designer App.

Supplied files

Three sets of external files are required to support back-channel training. The generation of these files is beyond the scope of this example, so they are included in this example. Download these files to the model directory (location of the Simulink .slx file) before running the complete SerDes system or generating AMI model executables.

Write to back-channel communication files

These three files are used to write the current state of the back-channel training parameters and eye metric(s) to an external file for communication between the Tx and Rx AMI models.

- MATLAB function file: *writeBCIfile.m*
- C++ files required for codegen: *writeamidata.cpp* and *writeamidata.h*

Read from back-channel communication files

These three files are used to read the current state of the back-channel training parameters and eye metric(s) from an external file for communication between the Tx and Rx AMI models.

- MATLAB function file: *readBCIfile.m*
- C++ files required for codegen: *readamidata.cpp* and *readamidata.h*

Write to back-channel log files

These three files are used to write current state of the back-channel training parameters and eye metric(s) after each training step to a log file for debug.

- MATLAB function file: *writeBCIhistory.m*
- C++ files required for codegen: *writebcihist.cpp* and *writebcihist.h*

Modify Tx FFE to enable external control of Tap values

To control the Tx FFE tap weights from the Tx_BCI block when back-channel training is enabled, replace the FFEParameter.TapWeights Constant block with a DataStoreRead block. This datastore allows the FFE tap values to change during the simulation and to be passed in and out of each of the datapath blocks.

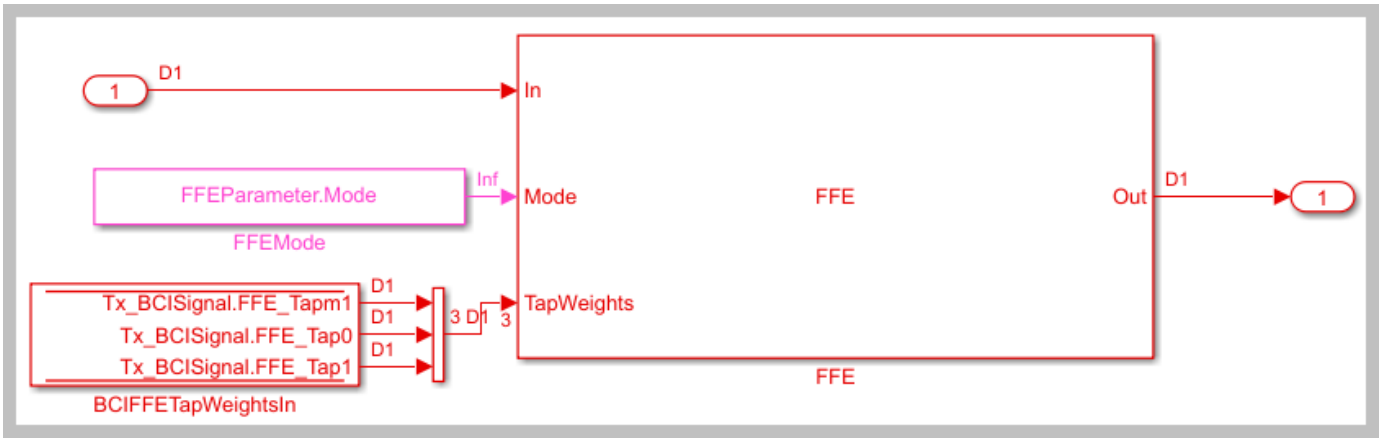
Inside the Tx subsystem, click on the FFE block and type **Ctrl-U** to look under the mask of the FFE block.

1. Delete the FFETapWeights Constant block.
2. Add a **DataStoreRead** block labeled `BCIFFETapWeightsIn`.
3. Double-click on the DataStoreRead block and set the Data store name to: `Tx_BCISignal`.
4. On the Element Selection tab, expand the signal Tx_BCISignal and highlight `FFE_Tapm1`, `FFE_Tap0` and `FFE_Tap1`.
5. Press the **Select>>** button to select these 3 elements.
6. Save the changes.

Add a Mux block and set the number of inputs to 3 to multiplex these three parameters into a vector for the FFE block.

Connect the output of the Mux block to the TapWeights input on the FFE.

The final FFE block should look like the following:



Type **Ctrl-D** to compile the model and check for errors.

Modify the DFECDR to output eye Sample Voltage

To determine the quality of a given set of equalization values during back-channel training, the voltage that is sampled by the CDR at the center of the eye for each symbol will be used. This value is captured by a DataStoreWrite block so that its value is available to the other BCI control blocks.

Inside the Rx subsystem, click on the DFECDR block and type **Ctrl-U** to look under the mask of the Rx DFECDR block.

Open the **BusSelector** object

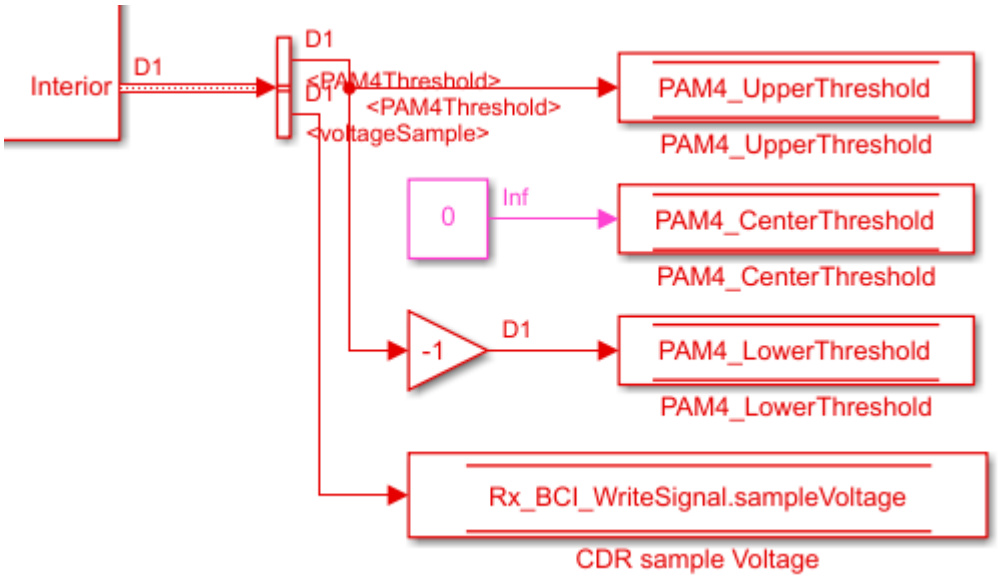
1. Highlight **voltageSample** from the list of Signals in the bus.
2. Hit **Select>>** to move it to the list of Selected signals.
3. Save the changes.

Add a **DataStoreWrite** block labeled `CDR sample Voltage`

1. Double click the DataStoreWrite block and set the Data store name to: `Rx_BCI_WriteSignal`.
2. On the Element Assignment tab, expand the signal Rx_BCI_WriteSignal and highlight **sampleVoltage**.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Connect the voltageSample output of the BusSelector to the input of the new DataStoreWrite block.

This portion of the DFECDR block should look like the following:



Type **Ctrl-D** to compile the model and check for errors.

Modify the DFECDR to override Mode when training is enabled

During back-channel training, both the FFE and DFE Modes need to be set to "Fixed". The FFE Mode has been hard-coded to "Fixed". A simple MATLAB function is used to allow you to set the DFE Mode when training is not enabled.

Inside the Rx subsystem, click on the DFECDR block and type **Ctrl-U** to look under the mask of the Rx DFECDR block.

Delete the connection between the DFECDRMode block and the DFECDR.

Add a new MATLAB function block and set the label to `DFEModeSelect`. This function block reads the values of BCI_State and DFE.Mode and forces the DFE Mode to 1 (Fixed) when training is

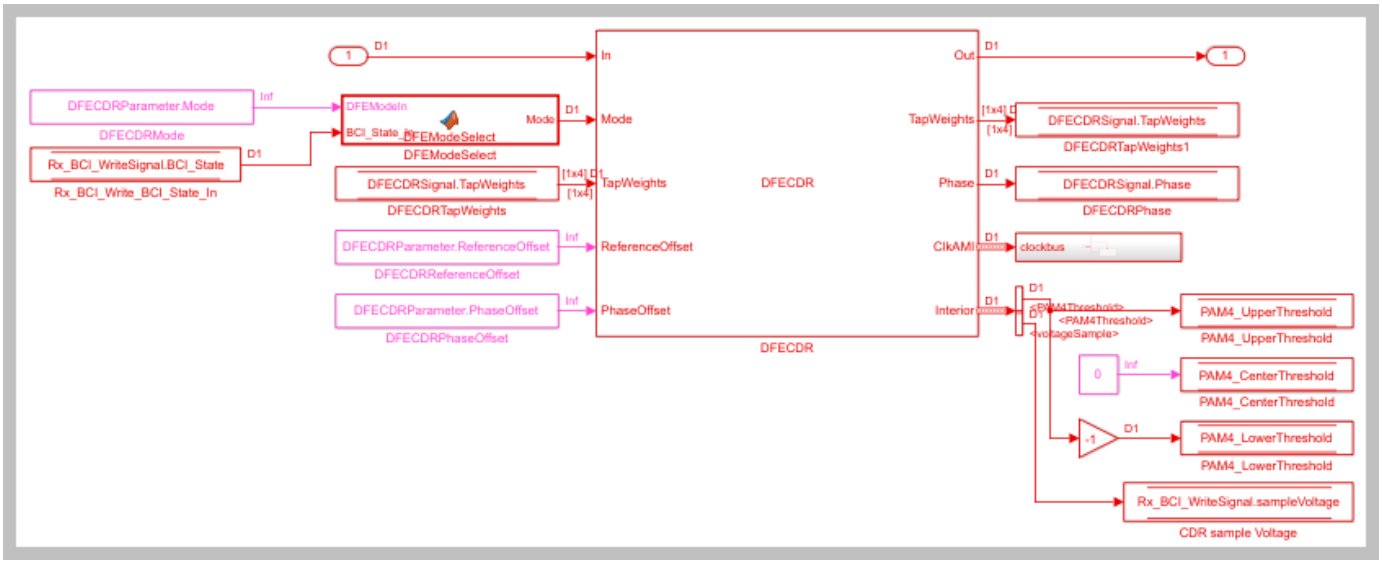
enabled or completed. Copy/Paste the following code into the DFEModeSelect MATLAB function block, replacing the default contents.

```
function Mode = DFEModeSelect(DFEModeIn, BCI_State_In)

if BCI_State_In == 1  % Training is Off
    Mode = DFEModeIn;
else
    Mode = 1;         % Force DFE Mode to Fixed for all other Training states
end
```

Add a **DataStoreRead** block labeled `Rx_BCI_Write_BCI_State_In`, so the value of BCI_State can be fed into the MATLAB function block.

1. Double click the DataStoreRead block and set the Data store name to: `Rx_BCI_WriteSignal`.
2. On the Element Selection tab, expand the signal Rx_BCI_WriteSignal and highlight **BCI_State**.
3. Press the **Select>> button** to select this element.
4. Save the changes.

Wire up these new blocks as shown. The final DFECDR block should look like the following:

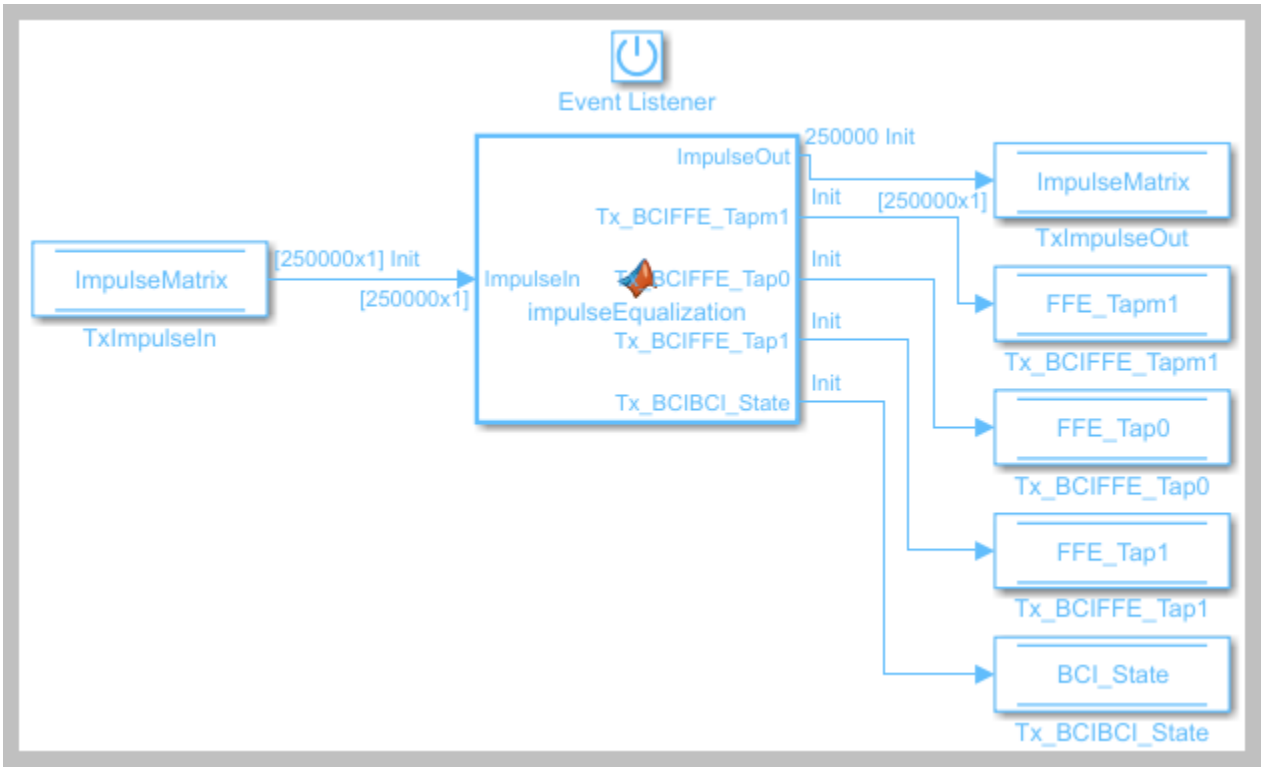


Type **Ctrl-D** to compile the model and check for errors.

Set up the Tx Init Custom Code

The Tx Initialize function is used to set up the Tx AMI model for running back-channel training during GetWave analysis. This creates the back-channel communication and log files, sets up the various parameters and overrides any user defined FFE tap values.

Inside the Tx subsystem, type **Ctrl-U** to look under the mask for the Init block, then double click on the initialize block to open the Initialize Function.



Double click on the impulseEquation MATLAB function block to open the function in MATLAB. This is an automatically generated function which provides the impulse response processing of the SerDes system block (IBIS AMI-Init). The `%% BEGIN:` and `% END:` lines denote the section where custom user code can be entered. Data in this section is not over-written when Refresh Init is run:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
Tx_BCIBCI_State = Tx_BCIParameter.BCI_State; % User added AMI parameter from SerDes IBIS-AMI Mana
Tx_BCIFFE_Tap0 = Tx_BCIParameter.FFE_Tap0; % User added AMI parameter from SerDes IBIS-AMI Manage
Tx_BCIFFE_Tap1 = Tx_BCIParameter.FFE_Tap1; % User added AMI parameter from SerDes IBIS-AMI Manage
Tx_BCIFFE_Tapm1 = Tx_BCIParameter.FFE_Tapm1; % User added AMI parameter from SerDes IBIS-AMI Mana

% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

Use this custom user code area to initialize the back-channel parameters, write the first entry in the back-channel communication file "BCI_comm.csv" and create the back-channel log file "BCI_comm_log.csv". To add the custom back-channel control code, scroll down to the custom user code area and Copy/Paste the following code:

```
Tx_BCIBCI_State = Tx_BCIParameter.BCI_State; % User added AMI parameter from SerDes IBIS-AMI Mana
Tx_BCIFFE_Tap0 = Tx_BCIParameter.FFE_Tap0; % User added AMI parameter from SerDes IBIS-AMI Manage
Tx_BCIFFE_Tap1 = Tx_BCIParameter.FFE_Tap1; % User added AMI parameter from SerDes IBIS-AMI Manage
Tx_BCIFFE_Tapm1 = Tx_BCIParameter.FFE_Tapm1; % User added AMI parameter from SerDes IBIS-AMI Mana

%% Set up for GetWave back-channel operation
if Tx_BCIBCI_State == 2 % Training enabled
    bciWrFile = 'BCI_comm.csv'; %% Tx/Rx back-channel communication file
    Protocol = ['DDR5' 0];      %% Null terminate string to keep fprintf happy in C++
    State = ['Training' 0];     %% Null terminate string to keep fprintf happy in C++
    Sequence = 1;               %% Initialize sequence counter
    EyeHeight = 0.0;            %% Initialize training metric
```

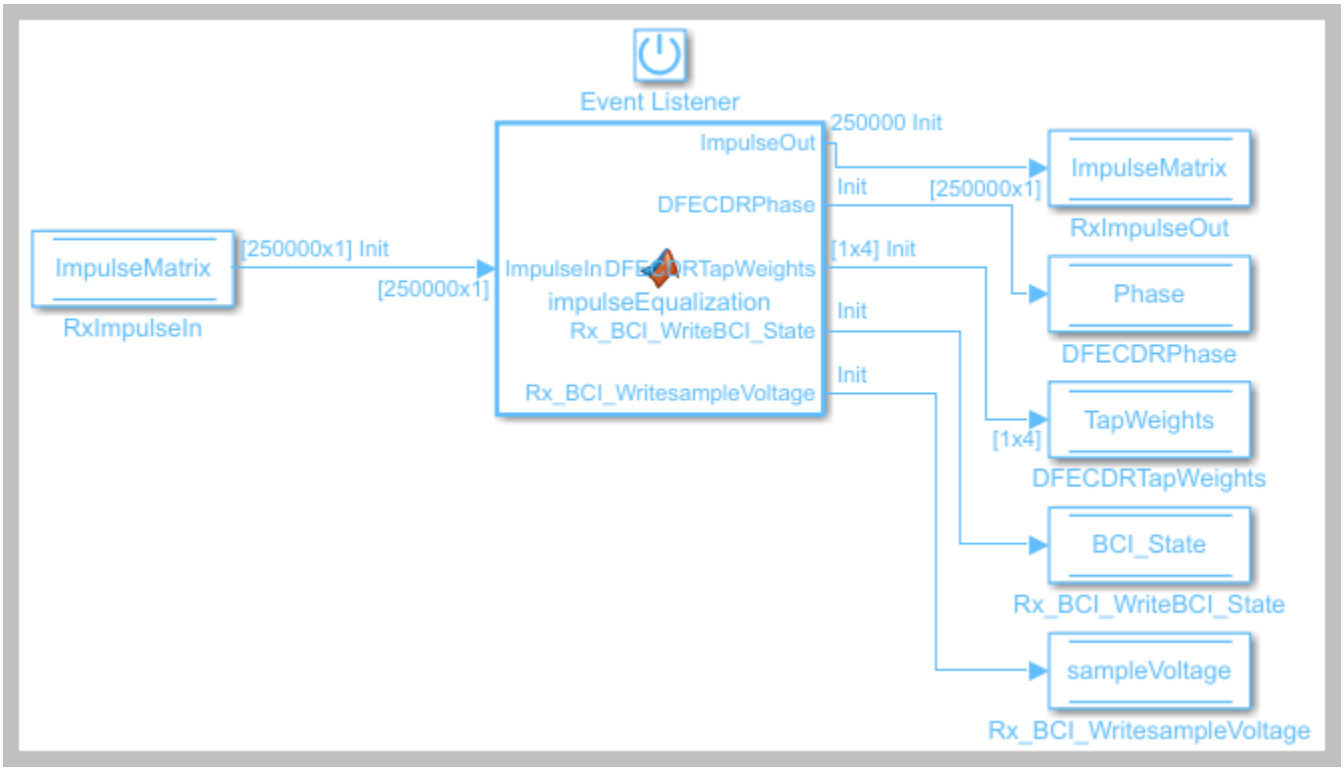
```
    % Publish Tx capabilities
    numFFEtaps = 3;
    FFEtaps = [0.0, 1.0, 0.0];
    FFEInit.TapWeights = [0.0, 1.0, 0.0];
    % Initialize Rx capabilities (actual values set by Rx)
    numDFEtaps = 1;
    DFEtaps = 0.0000;

    % Create new file for back-channel communication
    writeBCIfile(bciWrFile, 'w', Protocol, numDFEtaps, numFFEtaps, DFEtaps, FFEtaps, Sequence, S

    % Create new BCI_ID_log.csv file (for back-channel history)
    logFileName = 'BCI_comm_log.csv';
    writeBCIhistory(logFileName, 'Tx', 'Init', 0, Tx_BCIBCI_State, numDFEtaps, numFFEtaps, DFEtap
end
```

To test that the new user code is working correctly, run the model, verify that the new back-channel communication (BCI_comm.csv) and log (BCI_comm_log.csv) files have been created and that the values in the files match the values set above.

Set up the Rx Init Custom Code

The Rx Initialize function is used to set up the Rx AMI model for running back-channel training during GetWave analysis. This reads in the back-channel communication file and then updates the file with the Rx configuration information (number of DFE taps and DFE tap values). It also updates the log file.

Inside the Rx subsystem type **Ctrl-U** to look under the mask for the Init block, then double click on the initialize block to open the Initialize Function.



Double-click on the impulseEqualization MATLAB function block to open the function in MATLAB. This is an automatically generated function which provides the impulse response processing of the SerDes system block (IBIS AMI-Init). The `%% BEGIN:` and `% END:` lines denote the section where custom user code can be entered. Data in this section is not over-written when Refresh Init is run:

```
%% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
Rx_BCI_WritesampleVoltage = Rx_BCI_WriteParameter.sampleVoltage; % User added AMI parameter from
Rx_BCI_WriteBCI_State = Rx_BCI_WriteParameter.BCI_State; % User added AMI parameter from SerDes I

% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

Use this custom user code area to read the configuration from the Tx, initialize the additional back-channel parameters required by the Rx, write the next entry in the back-channel communication file "BCI_comm.csv", and append to the back-channel log file "BCI_comm_log.csv". To add the custom back-channel control code, scroll down the custom user code area and Copy/Paste the following code:

```
Rx_BCI_WritesampleVoltage = Rx_BCI_WriteParameter.sampleVoltage; % User added AMI parameter from
Rx_BCI_WriteBCI_State = Rx_BCI_WriteParameter.BCI_State; % User added AMI parameter from SerDes I

%% Set up for GetWave back-channel operation
if Rx_BCI_WriteBCI_State == 2 % Training enabled
    %% Read from back-channel communication file to get setup from Tx
    bciRdFile = 'BCI_comm.csv';
    [Protocol, ~, numFFEtaps, ~, FFEtaps, Sequence, State, EyeHeight] = readBCIfile(bciRdFile);

    %% Write Rx setup to back-channel communication file.
    bciWrFile = 'BCI_comm.csv';
    Sequence = Sequence + 1;   %% Initialize sequence counter
    % Publish Rx capabilities
    numDFEtaps = 4;
    DFEtaps = [0.0000, 0.0000, 0.0000, 0.0000];

    writeBCIfile(bciWrFile, 'w', Protocol, numDFEtaps, numFFEtaps, DFEtaps, FFEtaps, Sequence, St

    % Write to log file
    logFileName = 'BCI_comm_log.csv';
    writeBCIhistory(logFileName, 'Rx', 'Init', 0, Rx_BCI_WriteBCI_State, numDFEtaps, numFFEtaps,

    % Force DFE Mode to Fixed when training is enabled.
    DFECDRInit.Mode = 1;

end
```

To test that the new user code is working correctly, run the model, verify that the back-channel communication (BCI_comm.csv) and log (BCI_comm_log.csv) files have been created and that the values in the files match the values set above.

Set up the Tx Tx_BCI pass-through block

The Tx_BCI block is used to control the entire back-channel training process. The first time through it initializes all the Tx and Rx parameters that will be optimized during training. After every back-channel training cycle this block will read the current eye metric supplied by the Rx, store this value, then update the Tx and Rx parameters for the next pass. When training is complete this block will signal completion of training, set all Tx and Rx parameters to their optimal values and then return the models to regular operation.

The first step is to set up the Tx_BCI block for back-channel operation. The MATLAB function block that controls the operation of the Tx_BCI block is written later in this example.

Look under the mask in the Tx_BCI block.

Delete the Pass-Through system object since it is not used. Be sure to connect the Inport to the Outport.

Add a **Constant** block labeled `FFETapWeights` and set the constant value to `FFEParameter.TapWeights`.

- Double click the Constant block to open the mask.
- Uncheck the **Interpret vector parameters** as 1-D check-box to prevent the incoming Tap Weights row vector from being converted to a column vector.

Add a **DataStoreRead** block labeled `TxBCIStateIn`

1. Double click the DataStoreRead block and set the Data store name to `Tx_BCISignal`.
2. On the Element Selection tab, expand the signal Tx_BCISignal and highlight `BCI_State`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Add a **DataStoreWrite** block labeled `BCIFFETapWeightsOut`

1. Double click on the DataStoreWrite block and set the Data store name to `Tx_BCISignal.`
2. On the Element Assignment tab, expand the signal Tx_BCISignal and highlight `FFE_Tapm1`, `FFE_Tap0` and `FFE_Tap1`
3. Press the **Select>>** button to select these elements.
4. Save the changes.

Add a **DataStoreWrite** block labeled `TxBCIStateOut`

1. Double click the DataStoreWrite block and set the Data store name to: `Tx_BCISignal`.
2. On the Element Assignment tab, expand the signal Tx_BCISignal and highlight `BCI_State`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Add a **Demux** block and set the number of outputs to 3 to demultiplex the tapWeightsOut vector into three separate signals.

Add a new **MATLAB function** block and set the label to `Counter`. This MATLAB function returns a count of the total number of samples processed by the model and the resulting number of UI. Open this new MATLAB function block then Copy/Paste the following code, replacing the default contents.

```
function [sampCount, uiCount] = counter(SymbolTime, SampleInterval)

% Calculate Samples Per Bit
sampBit = round(SymbolTime/SampleInterval);

% Set up persistant variables
persistent x y
if isempty(x)
```

```
    x = int32(1);
    y = int32(1);
else
    x = x + 1;
end

% Start counting by UI
if mod(x,sampBit) == 0
    y = y + 1;
end

% Output results
sampCount = x;
uiCount   = y;
```

The values for two of the inputs to this function, **SymbolTime** and **SampleInterval**, are inherited from the Model Workspace and therefore do not need to show up as nodes on the MATLAB function block. To remove these nodes from the MATLAB function block:

1. Save the MATLAB function.
2. Open the Model Explorer and navigate to Tx->Tx_BCI->Counter.
3. Highlight the parameter **SymbolTime**.
4. Update the Scope from Input to `Parameter` and click **Apply**.
5. Repeat this process for **SampleInterval**.

The Data Type for the outputs of this function, **sampCount** and **uiCount**, are set to Inherit by default. Since this function block is creating the values for these two parameters their Data Type needs to be explicitly defined instead of determined based on heuristics. To explicitly define the Data Types for these two parameters:

1. Open the Model Explorer and navigate to Tx->Tx_BCI->Counter.
2. Highlight the parameter **sampCount**.
3. Update the Type from Inherit to `int32` and click **Apply**.
4. Repeat this process for **uiCount**.

Add another new **MATLAB function** block and set the label to `txBackChannel`. This MATLAB function block is used to control the back-channel training process. The contents of this function is covered later in this example. However, to complete the Tx_BCI block connections you must display all the correct nodes. To enable this:

1. Double click the txBackChannel MATLAB function block to open in the MATLAB editor.
2. Delete all the default contents.
3. Insert the following function signature:

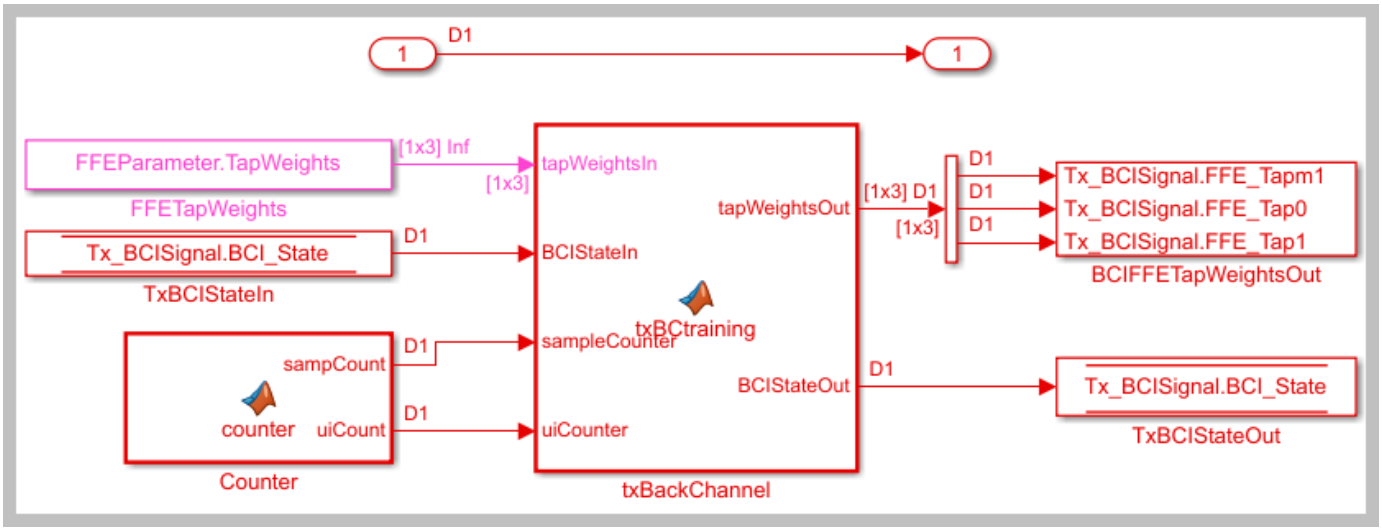
```
function [tapWeightsOut, BCIStateOut] = txBCtraining(tapWeightsIn, BCIStateIn, sampleCounter, ui
```

The values for two of the inputs to this function, **SymbolTime** and **SampleInterval**, are inherited from the Model Workspace and therefore do not need to show up as nodes on the MATLAB function block. To remove these nodes from the MATLAB function block:

1. Save the MATLAB function.
2. Highlight the parameter **SymbolTime**.

3 Right-click on the parameter and select Data Scope for "SymbolTime".
4 Set the value to Parameter.
5 Repeat this process for **SampleInterval**.

Connect everything together as shown below:



Set up the Rx Rx_BCI_Read block

The Rx_BCI_Read block is used to read the Rx parameters values requested by the Tx_BCI block and set those values for the next back-channel training cycle. If the Tx_BCI block signals that training is complete, this block sets the final values to be used for the remainder of the simulation.

The first step is to set up the Rx_BCI_Read block for back-channel operation. The MATLAB function block that controls the operation of the Rx_BCI_Read block is written later in the example.

Look under the mask in the Rx_BCI_Read block.

Delete the Pass-Through system object since it will not be used. Be sure to connect the Inport to the Outport.

Add a **DataStoreRead** block labeled `DFECDRTapWeightsIn`

1 Double click the DataStoreRead block and set the Data store name to: `DFECDRSignal`.
2 On the Element Selection tab, expand the signal DFECDRSignal and highlight `TapWeights [1,4]`.
3 Press the **Select>>** button to select this element.
4 Save the changes.

Add a **DataStoreRead** block labeled `RxBCIStateIn`

1 Double click the DataStoreRead block and set the Data store name to: `Rx_BCI_WriteSignal`.
2 On the Element Selection tab, expand the signal Rx_BCI_WriteSignal and highlight `BCI_State`.
3 Press the **Select>>** button to select this element.
4 Save the changes.

Add a **DataStoreWrite** block labeled `RxBCIStateOut`

1. Double click the DataStoreWrite block and set the Data store name to: `Rx_BCI_WriteSignal`.
2. On the Element Assignment tab, expand the signal Rx_BCI_WriteSignal and highlight `BCI_State`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Add a **DataStoreWrite** block labeled `DFECDRTapWeightsOut`

1. Double-click on the DataStoreWrite block and set the Data store name to: `DFECDRSignal`.
2. On the Element Assignment tab, expand the signal DFECDRSignal and highlight `TapWeights [1,4]`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Copy the **Counter** MATLAB function block from the Tx Tx_BCI block into this block.

Add a new **MATLAB function** block and set the label to `rxBackChannelRead`. This MATLAB function block is used to control the back-channel training process. The contents of this function is covered later in this example. However, to complete the Rx_BCI_Read block connections you must display all the correct nodes. To enable this:

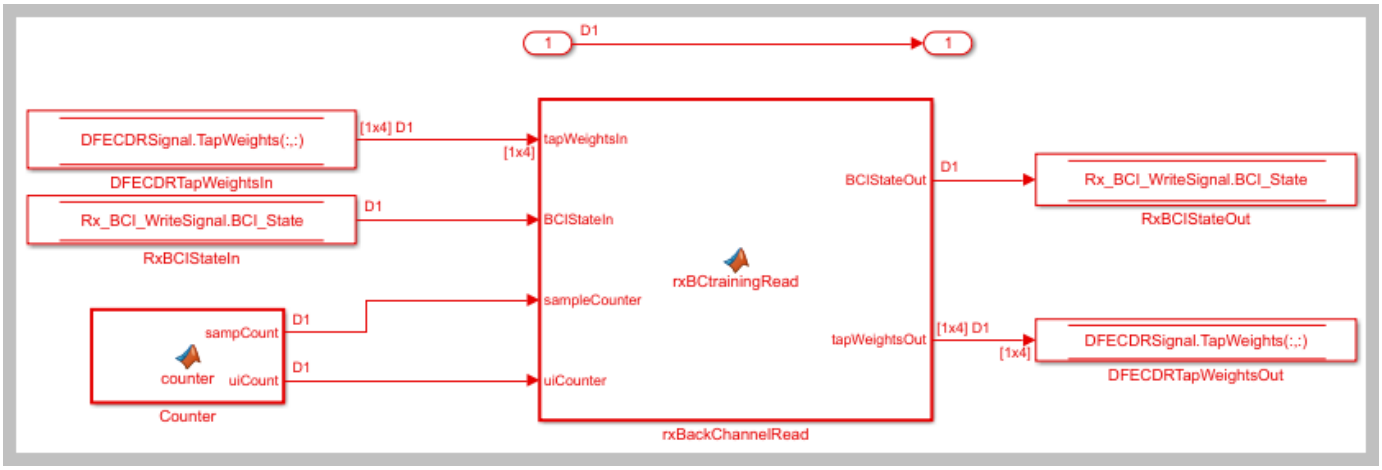
1. Double click the rxBackChannelRead MATLAB function block to open in the MATLAB editor.
2. Delete all the default contents.
3. Insert the following function signature:

```
function [BCIStateOut, tapWeightsOut] = rxBCtrainingRead(tapWeightsIn, BCIStateIn, sampleCounter
```

The values for two of the inputs to this function, **SymbolTime** and **SampleInterval**, are inherited from the Model Workspace and therefore do not need to show up as nodes on the MATLAB function block. To remove these nodes from the MATLAB function block:

1. Save the MATLAB function block.
2. Highlight the parameter **SymbolTime.**
3. Right-click on the parameter and select Data Scope for "SymbolTime".
4. Set the value to Parameter.
5. Repeat this process for **SampleInterval.**

Connect everything together as shown below:



Set up the Rx Rx_BCI_Write block

The Rx_BCI_Write block is used at the end of each back-channel training cycle to calculate the current eye metrics and report those metrics back to the Tx_BCI block for analysis.

The first step is to set up the Rx_BCI_Write block for back-channel operation. The MATLAB function block that controls the operation of the Rx_BCI_Write block is written later in the example.

Look under the mask in the Rx_BCI_Write block.

Delete the Pass-Through system object since it is not used. Be sure to connect the Inport to the Outport.

Add a **DataStoreRead** block labeled `CDRSampleVoltageIn`.

1. Double click the DataStoreRead block and set the Data store name to: `Rx_BCI_WriteSignal`.
2. On the Element Selection tab, expand the signal Rx_BCI_WriteSignal and highlight `sampleVoltage`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Add a **DataStoreRead** block labeled `DFECDRTapWeightsIn`.

1. Double-click on the DataStoreRead block and set the Data store name to `DFECDRSignal`.
2. On the Element Selection tab, expand the signal DFECDRSignal and highlight `TapWeights [1,4]`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Add a **DataStoreRead** block labeled `RxBCIStateIn`

1. Double click the DataStoreRead block and set the Data store name to: `Rx_BCI_WriteSignal`
2. On the Element Selection tab, expand the signal Rx_BCI_WriteSignal and highlight `BCI_State`
3. Press the **Select>>** button to select this element
4. Press **OK** to close the DataStoreRead dialog.

Add a **DataStoreWrite** block labeled `RxBCIStateOut`

1. Double click the DataStoreWrite block and set the Data store name to: `Rx_BCI_WriteSignal`.
2. On the Element Assignment tab, expand the signal Rx_BCI_WriteSignal and highlight `BCI_State`.
3. Press the **Select>>** button to select this element.
4. Save the changes.

Copy the **Counter** MATLAB function block from the Tx Tx_BCI block into this block.

Add a new **MATLAB function** block and set the label to `rxBackChannelWrite`. This MATLAB function block is used to control the back-channel training process. The contents of this function is covered later in this example. However, to complete the Rx_BCI_Write block connections you must display all the correct nodes. To enable this:

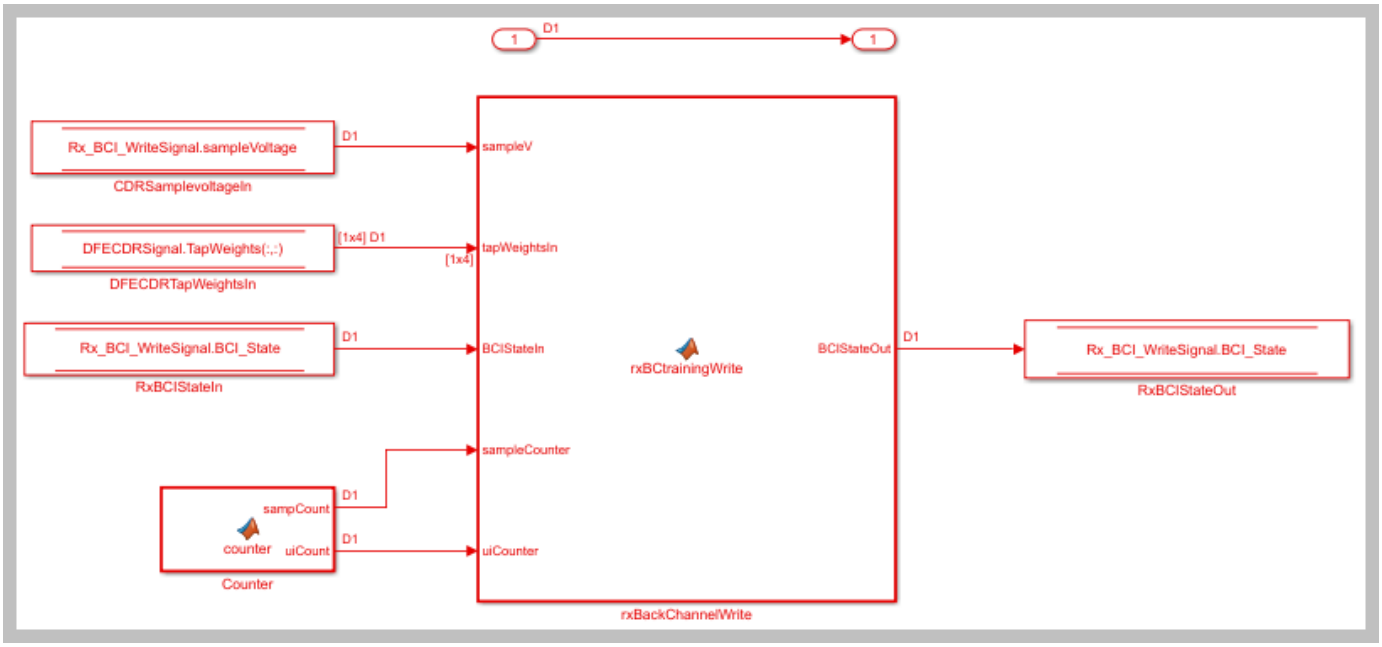
1. Double click the rxBackChannelWrite MATLAB function block to open in the MATLAB editor.
2. Delete all the default contents.
3. Insert the following function signature:

```
function BCIStateOut = rxBCtrainingWrite(sampleV, tapWeightsIn, BCIStateIn, sampleCounter, uiCou
```

The values for two of the inputs to this function, **SymbolTime** and **SampleInterval**, are inherited from the Model Workspace and therefore do not need to show up as nodes on the MATLAB function block. To remove these nodes from the MATLAB function block:

1. Save the MATLAB function block.
2. Highlight the parameter **SymbolTime**.
3. Right click on the parameter and select Data Scope for "SymbolTime".
4. Set the value to Parameter.
5. Repeat this process for **SampleInterval**.

Connect everything together as shown below:



Edit the txBCtraining MATLAB function block

The Tx_BCI block is used to control the entire back-channel training process. The first time through it initializes all the Tx and Rx parameters that will be optimized during training. After every back-channel training cycle, this block reads the current eye metric supplied by the Rx, stores this value, then updates the Tx and Rx parameters for the next pass. When training is complete this block signals completion of training, sets all Tx and Rx parameters to their optimal values and then returns the models to regular operation.

The Tx_BCI block was set up for back-channel operation earlier in this example. Now create the MATLAB function block at the heart of the Tx_BCI block. This MATLAB function block, which was labeled `txBackChannel`, controls the entire back-channel training process. The steps involved in this process are as follows:

1. Define the function signature
2. Initialize parameters and set persistent variables
3. Define the parameters to be swept and their ranges
4. On the first GetWave call, set up the initial starting parameter values for the Tx and the Rx
5. Every back-channel training cycle read the eye metrics calculated by the Rx and decide what to do next. When training is complete signal the completion of training, output the optimal Tx and Rx parameter values to be used during simulation and write these final values to the log file.
6. Set to proper training state and output the FFE parameters to be used

The following sections walks you through the code used in the txBackChannel MATLAB function block. In the Tx block, click on the Tx_BCI pass-through block and type **Ctrl-U** to push into the Tx_BCI pass-through block set up earlier. Double-click on the **txBackChannel** MATLAB function block, then Copy/Paste the code described in the following sections.

Define the function signature

The function signature for the txBCtraining block has 6 inputs and 2 outputs. The inputs are:

- **tapWeightsIn**: The FFE tap weights array as defined in the FFE mask.
- **BCIStateIn**: The back-channel state value from the TxBCIStateIn Data Store.
- **sampleCounter**: Count of total number of samples.
- **uiCounter**: Count of total number of UI.
- **SymbolTime**: The UI (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".
- **SampleInterval**: Simulation step size (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".

There are two outputs:

- **tapWeightsOut**: The FFE tap weights array output to the BCIFFETapWeightsOut Data Store.
- **BCIStateOut**: The back-channel state value output to the TxBCIStateOut Data Store.

The function signature was entered when initially creating the MATLAB function block and so is already present.

Initialize parameters and variables

This section sets up the three constants needed for calculating the size of the back-channel training cycle:

- **sampBit**: The number of samples in each UI.
- **messageInterval**: The length (in UI) of each back-channel training cycle. This value is currently set to ~2 PRBS7 iterations.
- **BCIwait**: The delay time (in UI) before starting back-channel training. This value is currently set to ~4 PRBS7 iterations.

In addition to the constant values, this section sets up the 11 persistent variables used by this function. Persistent variables retain their values between each call to this MATLAB function. The 11 persistent variables are:

- **Protocol**: The protocol being used by this back-channel model.
- **numDFEtaps**: The number of DFE taps being included in this back-channel training algorithm.
- **numFFEtaps**: The number FFE taps being included in this back-channel training algorithm.
- **DFEtaps**: The current DFE tap values.
- **FFEtaps**: The current FFE tap values.
- **Sequence**: A integer counter used to log the sequence of training events.
- **State**: The current back-channel training state.
- **EyeHeight**: The current eye height (in Volts) being reported by the Rx.
- **step**: The current training sequence step being run.
- **indx**: An index variable for control loops.
- **metric**: An array used to store the incoming eye heights from each training step.

To initialize these parameters and variables, Copy/Paste the following code into the txBackChannel MATLAB function block:

```
%% Setup
sampBit = round(SymbolTime/SampleInterval);  %% Calculate Samples Per Bit
messageInterval = 256;                       %% Length (in UI) of back-channel training cycle ite
BCIwait = 512;                               %% Delay time (in UI) before starting training(~4 PR

%% Read BCI file to determine training values
% Make variables available between time steps
persistent Protocol numDFEtaps numFFEtaps DFEtaps FFEtaps Sequence State EyeHeight step indx metr

% Initialize variable initial conditions
if isempty(Protocol)
    Protocol = 'Defaults';
end
if isempty(numDFEtaps)
    numDFEtaps = 4;
end
if isempty(numFFEtaps)
    numFFEtaps = 3;
end
if isempty(DFEtaps)
    DFEtaps = [0.000,0.000,0.000,0.000];
end
if isempty(FFEtaps)
    FFEtaps = [0.000,1.000,0.000];
end
if isempty(Sequence)
    Sequence = 1;
end
if isempty(State)
    State = 'Testing';
end
if isempty(EyeHeight)
    EyeHeight = 0.000;
end
if isempty(step)
    step = 1;
end
if isempty(indx)
    indx = 1;
end
if isempty(metric)
    metric = zeros(50,1);
end
```

Define swept parameters

The training algorithm implemented in this example sweeps the pre and post FFE tap values and all 4 of the DFE taps individually, then selects the optimal value for each tap. Eight parameters are used to define the ranges for each of the taps and the step size to be used during training:

- **ffeTapStep**: The step size to be used when sweeping the FFE taps. This value is negative since the FFE tap values are always <= 0.
- **dfeTapStep**: The step size to be used when sweeping the DFE taps.
- **regFFEtapm1**: The min/max range of values to be used when sweeping the FFE pre-tap.
- **regFFEtap1**: The min/max range of values to be used when sweeping the FFE post-tap.

- **regDFEtap1**: The min/max range of values to be used when sweeping the first DFE tap.
- **regDFEtap2**: The min/max range of values to be used when sweeping the second DFE tap.
- **regDFEtap3**: The min/max range of values to be used when sweeping the third DFE tap.
- **regDFEtap4**: The min/max range of values to be used when sweeping the fourth DFE tap.

To define all the parameters to be swept during training, Copy/Paste the following code into the txBackChannel MATLAB function block:

```
% Define parameter step sizes
ffeTapStep = -0.050;
dfeTapStep = 0.010;

% Map ranges to register values
regFFEtapm1 = ( 0.000:ffeTapStep:-0.300);
regFFEtap1  = ( 0.000:ffeTapStep:-0.300);
regDFEtap1  = (-0.200:dfeTapStep: 0.050);
regDFEtap2  = (-0.075:dfeTapStep: 0.075);
regDFEtap3  = (-0.060:dfeTapStep: 0.060);
regDFEtap4  = (-0.045:dfeTapStep: 0.045);
```

First GetWave call

When training is enabled, the very first call to this MATLAB function needs to read the back-channel communication file written during Init to determine the full capabilities of the Tx and Rx models. This section also sets up the initial values to be used for the first back-channel training cycle. Finally, all these values are written to the back-channel communication log file.

To implement the first GetWave call, Copy/Paste the following code into the txBackChannel MATLAB function block:

```
%% First Tx GetWave Call (Sequence=3)
if sampleCounter == 1 && BCIStateIn == 2 % Training enabled
    % Read back-channel communication file to get current settings
    bciRdFile = 'BCI_comm.csv';
    [~, numDFEtaps, numFFEtaps, ~, ~, Sequence, ~, EyeHeight] = readBCIfile(bciRdFile);

    % Decide what to do first
    % Tx Params
    FFEtaps = [0.000,1.000,0.000];
    % Rx Params
    DFEtaps = [0.0000, 0.0000, 0.0000, 0.0000];

    % Write back-channel communication file with first pass settings for Rx
    bciWrFile = 'BCI_comm.csv';
    Protocol = ['DDR5' 0];   %% Null terminate string to keep fprintf happy in C++
    State = ['Training' 0];  %% Null terminate string to keep fprintf happy in C++
    Sequence = Sequence + 1;
    writeBCIfile(bciWrFile, 'w', Protocol, numel(DFEtaps), numel(FFEtaps), DFEtaps, FFEtaps, Sequ

    % Write to log file
    logFileName = 'BCI_comm_log.csv';
    writeBCIhistory(logFileName, 'Tx', 'GetW', sampleCounter, BCIStateIn, numel(DFEtaps), numel(
end
```

Back-channel training algorithm

When training is enabled, after waiting the number of UI as defined by the constant **BCIwait**, the back-channel training algorithm is called every training block as defined by the **messageInterval** constant. First the current metrics reported by the Rx are read, then those results are written to the back-channel communication log file. The training algorithm uses the following steps:

1. Sweep all values of the FFE pre-tap and determine which value results in the largest eye opening.
2. Sweep all values of the FFE post-tap and determine which value results in the largest eye opening.
3. Sweep all values of DFE tap 1 and determine which value results in the largest eye opening.
4. Sweep all values of DFE tap 2 and determine which value results in the largest eye opening.
5. Sweep all values of DFE tap 3 and determine which value results in the largest eye opening.
6. Sweep all values of DFE tap 4 and determine which value results in the largest eye opening.
7. When training is complete, change the State to "Converged" and write the final values to the back-channel communication log file.

To implement the back-channel training algorithm, Copy/Paste the following code into the txBackChannel MATLAB function block:

```
%% Each subsequent BCI Block (Sequence=5,7,9,11...)
if  uiCounter > BCIwait + 2 && mod(sampleCounter - 1, (messageInterval * sampBit)) == 0 && BCISta
    % Read setup used for previous 16 GetWaveblocks from back-channel communication file
    bciRdFile = 'BCI_comm.csv';
    [~, ~, ~, ~, ~, Sequence, ~, EyeHeight] = readBCIfile(bciRdFile);

    % Write current results to log file
    Sequence = Sequence + 1;
    logFileName = 'BCI_comm_log.csv';
    writeBCIhistory(logFileName, 'Tx', 'GetW', sampleCounter, BCIStateIn, numel(DFEtaps), numel(
    if indx ~= 1
        % Store current metrics
        metric(indx - 1) = EyeHeight;
    end

    % Decide what to do next
    switch step
        case 1  % Step 1: Determine best value for FFE tap -1
            State = ['Training' 0];  %% Null terminate string to keep fprintf happy in C++
            if indx <= length(regFFEtapm1)
                % Set values for next iteration
                FFEtaps(1) = regFFEtapm1(indx);
                FFEtaps(3) = 0.0;
                FFEtaps(2) = 1 - abs(FFEtaps(1)) - abs(FFEtaps(3));
                indx = indx + 1;
            elseif indx == length(regFFEtapm1) + 1
                % Set best metric
                [~, jj] = max(metric);
                FFEtaps(1) = regFFEtapm1(jj);
                FFEtaps(3) = 0.0;
                FFEtaps(2) = 1 - abs(FFEtaps(1)) - abs(FFEtaps(3));

                % Done. Set up for next step
```

```
        metric = zeros(50,1);
        step = step + 1;
        indx = 1;
    end
case 2  % Step 2: Determine best value for FFE tap 1
    State = ['Training' 0];
    if indx <= length(regFFEtap1)
        % Set values for next iteration
        %FFEtaps(1) = 0.0;   %% Use value from step 1
        FFEtaps(3) = regFFEtap1(indx);
        FFEtaps(2) = 1 - abs(FFEtaps(1)) - abs(FFEtaps(3));
        indx = indx + 1;
    elseif indx == length(regFFEtap1) + 1
        % Set best metric
        [~, jj] = max(metric);
        FFEtaps(3) = regFFEtap1(jj);
        FFEtaps(2) = 1 - abs(FFEtaps(1)) - abs(FFEtaps(3));

        % Done. Set up for next step
        metric = zeros(50,1);
        step = step + 1;
        indx = 1;
    end
case 3  % Step 3: Determine best value for DFE tap 1
    State = ['Training' 0];
    if indx <= length(regDFEtap1)
        % Set values for next iteration
        DFEtaps = [regDFEtap1(indx), 0.0000, 0.0000, 0.0000];
        indx = indx + 1;
    elseif indx == length(regDFEtap1) + 1
        % Set best metric
        [~, jj] = max(metric);
        DFEtaps = [regDFEtap1(jj), 0.0000, 0.0000, 0.0000];

        % Done. Set up for next step
        metric = zeros(50,1);
        step = step + 1;
        indx = 1;
    end
case 4  % Step 4: Determine best value for DFE tap 2
    State = ['Training' 0];
    if indx <= length(regDFEtap2)
        % Set values for next iteration
        DFEtaps(2:4) = [regDFEtap2(indx), 0.0000, 0.0000];
        indx = indx + 1;
    elseif indx == length(regDFEtap2) + 1
        % Set best metric
        [~, jj] = max(metric);
        DFEtaps(2:4) = [regDFEtap2(jj), 0.0000, 0.0000];

        % Done. Set up for next step
        metric = zeros(50,1);
        step = step + 1;
        indx = 1;
    end
case 5  % Step 5: Determine best value for DFE tap 3
    State = ['Training' 0];
    if indx <= length(regDFEtap3)
```

```
                % Set values for next iteration
                DFEtaps(3:4) = [regDFEtap3(indx), 0.0000];
                indx = indx + 1;
            elseif indx == length(regDFEtap3) + 1
                % Set best metric
                [~, jj] = max(metric);
                DFEtaps(3:4) = [regDFEtap3(jj), 0.0000];

                % Done. Set up for next step
                metric = zeros(50,1);
                step = step + 1;
                indx = 1;
            end
        case 6  % Step 6: Determine best value for DFE tap 4
            State = ['Training' 0];
            if indx <= length(regDFEtap4)
                % Set values for next iteration
                DFEtaps(4) = regDFEtap4(indx);
                indx = indx + 1;
            elseif indx == length(regDFEtap4) + 1
                % Set best metric
                [~, jj] = max(metric);
                DFEtaps(4) = regDFEtap4(jj);

                % Done. Set up for next step
                metric = zeros(50,1);
                step = step + 1;
                indx = 1;
            end
        case 7  % Step 7: Training is complete
            State = ['Converged' 0];
            % Write final entry in log file
            logFileName = 'BCI_comm_log.csv';
            Sequence = Sequence + 1;
            writeBCIhistory(logFileName, 'Tx', 'GetW', sampleCounter, 3, numel(DFEtaps), numel(FF
        otherwise
            State = ['Error' 0];
    end

    % Write to back-channel communication file with next pass settings for Rx
    bciWrFile = 'BCI_comm.csv';
    Protocol = ['DDR5' 0];   %% Null terminate string to keep fprintf happy in C++
    writeBCIfile(bciWrFile, 'w', Protocol, numel(DFEtaps), numel(FFEtaps), DFEtaps, FFEtaps, Sequ

end
```

Set training State and output parameter values

The last thing that needs to be done in by this MATLAB function is to update the State for the BCI_State Data Store and to update the FFE tap array values.

To set the training state and output values, Copy/Paste the following code into the txBackChannel MATLAB function block:

```
%% Set back-channel state
if strcmpi(State,'Off') || strcmpi(State,['Off' 0])
    BCIStateOut = 1;
elseif strcmpi(State,'Training') || strcmpi(State,['Training' 0])
```

```
    BCIStateOut = 2;
elseif strcmpi(State,'Converged') || strcmpi(State,['Converged' 0])
    BCIStateOut = 3;
elseif strcmpi(State,'Failed') || strcmpi(State,['Failed' 0])
    BCIStateOut = 4;
else %Error
    BCIStateOut = 5;
end

%% Set output FFE values based on Training
if BCIStateOut == 2 || BCIStateOut == 3 % Training enabled/Converged
    tapWeightsOut = FFEtaps(1,1:3);
else  % Training Off/Failed/Error
    tapWeightsOut = tapWeightsIn;
end
```

Save and close this MATLAB function block.

Edit the rxBCtrainingRead MATLAB function block

The Rx_BCI_Read block is used to read the Rx parameters values requested by the Tx_BCI block and set them for the next back-channel training cycle. If the Tx_BCI block signals that the training is complete, this block sets the final values to be used by the Rx for the remainder of the simulation.

The Rx_BCI_Read block was set up for back-channel operation earlier in this example. Now create the MATLAB function block at the center of the Rx_BCI_Read block. This MATLAB function block, which was labeled `rxBCtrainingRead`, sets the Rx DFE values to be used. The steps involved in this process are as follows:

1. Define the function signature.
2. Initialize parameters and set persistent variables.
3. On the first GetWave call, and at the beginning of every back-channel training cycle, read the Rx DFE tap values to be used as specified by the Tx back-channel training algorithm.
4. Set the proper training state and output the DFE parameters to be used.

The following sections walk you through the code used in the rxBCtrainingRead MATLAB function block. In the Rx block, click on the Rx_BCI_Read pass-through block and type **Ctrl-U** to push into the Rx_BCI_Read pass-through block set up earlier. Double click the rxBCtrainingRead MATLAB function block, then Copy/Paste the code described in the following sections.

Define the function signature

The function signature for the rxBCtrainingRead block has 6 inputs and 2 outputs. The inputs are:

- **tapWeightsIn**: The DFE tap weights array as defined in the DFECDRTapWeightsIn Data Store.
- **BCIStateIn**: The back-channel state value from the RxBCIStateIn Data Store.
- **sampleCounter**: Count of total number of samples.
- **uiCounter**: Count of total number of UI.
- **SymbolTime**: The UI (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".

- **SampleInterval**: Simulation step size (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".

There are two outputs:

- **tapWeightsOut**: The DFE tap weights array output to the DFECDRTapWeightsOut Data Store.
- **BCIStateOut**: The back-channel state value output to the RxBCIStateOut Data Store.

The function signature was entered when initially creating the MATLAB function block and so is already present.

Initialize parameters and variables

This section sets up the three constants needed for calculating the size of the back-channel training cycle:

- **sampBit**: The number of samples in each UI.
- **messageInterval**: The length (in UI) of each back-channel training cycle. This value is currently set to ~2 PRBS7 iterations.
- **BCIwait**: The delay time (in UI) before starting back-channel training. This value is currently set to ~4 PRBS7 iterations.

In addition to the constant values, this section sets up the 7 persistent variables used by this function. Persistent variables retain their values between each call to this MATLAB function. The 7 persistent variables are:

- **Protocol**: The protocol being used by this back-channel model.
- **numDFEtaps**: The number of DFE taps being included in this back-channel training algorithm.
- **numFFEtaps**: The number FFE taps being included in this back-channel training algorithm.
- **DFEtaps**: The current DFE tap values.
- **FFEtaps**: The current FFE tap values.
- **Sequence**: A integer counter used to log the sequence of training events.
- **State**: The current back-channel training state.

To initialize the parameters and variables, Copy/Paste the following code into the rxBCtrainingRead MATLAB function block:

```
%% Setup
sampBit = round(SymbolTime/SampleInterval);  %% Calculate Samples Per Bit
messageInterval = 256;                       %% Length (in UI) of back-channel training cycle ite
BCIwait = 512;                               %% Delay time (in UI) before starting training(~4 PR

% Make variables available between time steps
persistent Protocol numDFEtaps numFFEtaps DFEtaps FFEtaps Sequence State;

% Initialize variable initial conditions
if isempty(Protocol)
    Protocol = 'Defaults';
end
if isempty(numDFEtaps)
    numDFEtaps = 4;
end
```

```
if isempty(numFFEtaps)
    numFFEtaps = 3;
end
if isempty(DFEtaps)
    DFEtaps = tapWeightsIn;
end
if isempty(FFEtaps)
    FFEtaps = [0,0,0];
end
if isempty(Sequence)
    Sequence = 1;
end
if isempty(State)
    if BCIStateIn == 1     % Off
        State = ['Off' 0];
    elseif BCIStateIn == 2 % Training
        State = ['Training' 0];
    elseif BCIStateIn == 3 % Converged
        State = ['Converged' 0];
    elseif BCIStateIn == 4 % Failed
        State = ['Failed' 0];
    else  % Error
        State = ['Error' 0];
    end
end
```

Read DFE tap values to be used

When training is enabled, on the very first call to this MATLAB function and at the beginning of every training block as defined by the **messageInterval** constant, the back-channel communication file is read to determine the updated DFE tap values to be used for the next training cycle.

To set up the DFE tap values to be used, Copy/Paste the following code into the rxBCtrainingRead MATLAB function block:

```
%% First GetWave block of each BCI Block (Sequence=3,5,7,9,11,...)
% Read back-channel communication file to get current settings
if  (sampleCounter == 1 && BCIStateIn == 2) || ((uiCounter > BCIwait + 2 && mod(sampleCounter - 1
    bciRdFile = 'BCI_comm.csv';
    [Protocol, numDFEtaps, numFFEtaps, DFEtaps(1,1:4), FFEtaps, Sequence, State, ~] = readBCIfile

end
```

Set training State and output parameter values

The last thing that needs to be done in by this MATLAB function block is to update the State for the BCI_State Data Store and to update the DFE tap array values.

To set the State and output values, Copy/Paste the following code into the rxBCtrainingRead MATLAB function block:

```
%% Set back-channel state
if strcmpi(State,'Off') || strcmpi(State,['Off' 0])
    BCIStateOut = 1;
elseif strcmpi(State,'Training') || strcmpi(State,['Training' 0])
    BCIStateOut = 2;
elseif strcmpi(State,'Converged') || strcmpi(State,['Converged' 0])
    BCIStateOut = 3;
```

```
elseif strcmpi(State,'Failed') || strcmpi(State,['Failed' 0])
    BCIStateOut = 4;
else %Error
    BCIStateOut = 5;
end

%% Set output DFE values based on Training
if BCIStateOut == 2 % Training enabled
    tapWeightsOut = DFEtaps(1,1:4);
else
    tapWeightsOut = tapWeightsIn;
end
```

Save and close this MATLAB function block.

Edit the rxBCtrainingWrite MATLAB function block

The Rx_BCI_Write block is used at the end of each back-channel training cycle to calculate the current eye metrics and report those metrics back to the Tx_BCI block for analysis.

The Rx_BCI_Write block was set up for back-channel operation earlier in this example. Now the MATLAB function block at the center of the Rx_BCI_Write block will be created. This MATLAB function block, which we labeled `rxBCtrainingWrite`, will calculate the minimum eye height of the last 127 bits and write those values to the back-channel communication file and log file. The steps involved in this process are as follows:

1. Define the function signature.
2. Initialize parameters and set persistent variables.
3. Store a vector of voltages to be used when calculating the minimum eye height.
4. At the end of each back-channel training cycle calculate the minimum eye height and write it to the back-channel communication file.
5. Update the training state.

The following sections will walk through the code used in the rxBCtrainingWrite MATLAB function block. In the Rx block, click on the Rx_BCI_Write pass-through block and type **Ctrl-U** to push into the Rx_BCI_Write pass-through block set up earlier. Double-click on the rxBCtrainingWrite MATLAB function block, then Copy/Paste the code described in the following sections.

Define the function signature

The function signature for the rxBCtrainingWrite block has 7 inputs and 1 output. The inputs are:

- **sampleV**: The voltage at the CDR sample time.
- **tapWeightsIn**: The DFE tap weights array as defined in the DFECDRTapWeightsIn Data Store.
- **BCIStateIn**: The back-channel state value from the RxBCIStateIn Data Store.
- **sampleCounter**: Count of total number of samples.
- **uiCounter**: Count of total number of UI.
- **SymbolTime**: The UI (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".

- **SampleInterval**: Simulation step size (in seconds). This value is inherited from the Model Workspace and therefore does not need to show up as a node on the MATLAB function block. To remove this node from the MATLAB function block, the Data Scope has been set to "Parameter".

There is one output:

- **BCIStateOut**: The back-channel state value output to the RxBCIStateOut Data Store.

The function signature was entered when initially creating the MATLAB function block and so is already present.

Initialize parameters and variables

This section sets up the four constants needed for calculating the size of the back-channel training cycle:

- **sampBit**: The number of samples in each UI.
- **messageInterval**: The length (in UI) of each back-channel training cycle. This value is currently set to ~2 PRBS7 iterations.
- **BCIwait**: The delay time (in UI) before starting back-channel training. This value is currently set to ~4 PRBS7 iterations.
- **windowLength**: The length of the window (in UI) used to calculate the minimum eye height. This value is currently set to 1 PRBS7 iteration.

In addition to the constant values, this section sets up the 5 persistent variables used by this function. Persistent variables retain their values between each call to this MATLAB function. The 5 persistent variables are:

- **Protocol**: The protocol being used by this back-channel model.
- **Sequence**: A integer counter used to log the sequence of training events.
- **State**: The current back-channel training state.
- **EyeHeight**: The calculated inner eye height value (in Volts).
- **vSamp**: The sample voltage being reported by the CDR block.

To initialize all parameters and variables for this block, Copy/Paste the following code into the rxBCtrainingWrite MATLAB function block:

```
%% Setup
sampBit = round(SymbolTime/SampleInterval);  %% Calculate Samples Per Bit
messageInterval = 256;                       %% Length (in UI) of back-channel training cycle ite
BCIwait = 512;                               %% Delay time (in UI) before starting training(~4 PR
windowLength = 127;                          %% Length of window (in UI) used to calculate minimu

% Make variables available between time steps
persistent Protocol Sequence State EyeHeight vSamp

if isempty(State)
    if BCIStateIn == 1     % Off
        State = ['Off' 0];
    elseif BCIStateIn == 2 % Training
        State = ['Training' 0];
    elseif BCIStateIn == 3 % Converged
        State = ['Converged' 0];
    elseif BCIStateIn == 4 % Failed
```

```
        State = ['Failed' 0];
    else  % Error
        State = ['Error' 0];
    end
end
```

Store vector of reported voltages

This section accumulates a rolling vector of voltages to be used in the minimum eye height calculation. Assume that these voltages are symmetric around 0V, so the absolute value is used.

To store the report eye voltage values, Copy/Paste the following code into the rxBCtrainingWrite MATLAB function block:

```
% Accumulate rolling vector of voltages for minimum eye height calculations
if isempty(vSamp)
    vSamp = zeros(1, windowLength * sampBit);
end
vSamp = circshift(vSamp, 1);
vSamp(1) = abs(sampleV);      % Assume symmetry and only use positive values
```

Calculate minimum eye height and write to file

When training is enabled, after waiting the number of UI as defined by the constant **BCIwait** the back-channel metrics are calculated at the end of each training iteration as defined by the **messageInterval** constant. First the back-channel configuration is read from the back-channel communication file, then the inner eye height value is calculated and the results output to the back-channel communication file and the log file.

To calculate the eye metrics and write to the communication file every back-channel cycle, Copy/Paste the following code into the rxBCtrainingWrite MATLAB function block:

```
%% Write current state and eye metrics at the end of each BCI block
if  uiCounter > BCIwait + 2 && mod(sampleCounter, (messageInterval * sampBit)) == 0 && BCIStateIn

    % Read setup used for last 16 GetWaveblocks from back-channel communication file
    bciRdFile = 'BCI_comm.csv';
    [Protocol, ~, ~, ~, FFEtaps, Sequence, State, ~] = readBCIfile(bciRdFile);

    % Calculate inner eye height from sampled voltage:
    EyeHeight = min(vSamp) * 2;    % 2x since using absolute value.

    % Write new back-channel communication file with end of BCI-Block metrics
    bciWrFile = 'BCI_comm.csv';
    Sequence = Sequence + 1;
    writeBCIfile(bciWrFile, 'w', Protocol, numel(tapWeightsIn), numel(FFEtaps), tapWeightsIn, FFE
    %
    % Write to log file:
    logFileName = 'BCI_comm_log.csv';
    writeBCIhistory(logFileName, 'Rx', 'GetW', sampleCounter, BCIStateIn, numel(tapWeightsIn), nu

end
```

Set the training State

The last thing that needs to be done in this MATLAB function block is to update the State for the BCI_State Data Store.

To set the training state, Copy/Paste the following code into the rxBCtrainingRead MATLAB function block:

```
%% Update State Out if State In changed
if BCIStateIn == 3     % Converged
    State = ['Converged' 0];
elseif BCIStateIn == 4 % Failed
    State = ['Failed' 0];
end

if strcmpi(State,'Off') || strcmpi(State,['Off' 0])
    BCIStateOut = 1;
elseif strcmpi(State,'Training') || strcmpi(State,['Training' 0])
    BCIStateOut = 2;
elseif strcmpi(State,'Converged') || strcmpi(State,['Converged' 0])
    BCIStateOut = 3;
elseif strcmpi(State,'Failed') || strcmpi(State,['Failed' 0])
    BCIStateOut = 4;
else %Error
    BCIStateOut = 5;
end
```

Save and close this MATLAB function block.

In Simulink, type **Ctrl-D** to compile the model and check for errors. Resolve any errors before proceeding.

Run the Model and Verify results

The next step is to run the model and verify that the back-channel code is operating correctly.

Set up simulation parameters

Before running the complete model, open the Stimulus block to set the stimulus pattern used to test the model:

- Set **PRBS** to 7, so that a PRBS7 pattern will be used during simulation.
- Set the **Number of symbols** to `50000` to allow the back-channel training algorithm sufficient time to complete.

Test proper opereration of Tx and Rx models

Run the model. While the model is running, observe the time domain waveform changing as each of the tap settings is swept. When the simulation is complete the back-channel communication file, BCI_comm.csv, should look similar to:

```
Protocol,DDR5,
numDFEtaps,4,
numFFEtaps,3,
DFEtaps,0.01000,-0.00500,-0.01000,-0.00500,
FFEtaps,0.00000,0.85000,-0.15000,
Sequence,176,
State,Converged,
EyeHeight,0.612739,
```

Open the back-channel communication log file, BCI_comm_log.csv, in a spreadsheet editor. Each row in the log file shows the Sequence number, which model wrote to the file (Tx or Rx), the current

Sample Count, BCI_State and calculated Eye Height. The last 7 columns in the log show the current FFE and DFE taps values being simulated. Observe how the Eye Height changes as each value is swept, and the parameter value that gives the largest Eye Height is set after each iteration. Note that the value of FFE0 is always computed from the values of FFE-1 and FFE1.

Generate DDR5 Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model and generates IBIS-AMI compliant DDR5 model executables, IBIS and AMI files.

Open the Block Parameter dialog box for the Configuration block and click on the **Open SerDes IBIS-AMI Manager** button.

Export Models

On the **Export** tab in the SerDes IBIS/AMI manager dialog box.

- Update the **Tx model name** to `ddr5_bc_tx`.
- Update the **Rx model name** to `ddr5_bc_rx`.
- Note that the **Tx and Rx corner percentage**is set to `10`. This will scale the min/max analog model corner values by +/-10%.
- Verify that **Dual model** is selected for both the Tx and the Rx AMI Model Settings. This will create model executables that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Tx model Bits to ignore value** to `3` since there are three taps in the Tx FFE.
- Set the **Rx model Bits to ignore value** to `50000` to allow enough time for training to complete during time domain simulations.
- Set **Models to export** as **Both Tx and Rx** so that all the files are selected to be generated (IBIS file, AMI files and DLL files).
- Set the **IBIS file name** to be `ddr5_bc_txrx.ibs`
- Jitter can be added if desired.
- Press the **Export** button to generate models in the Target directory.

Update AMI files if Desired

The Tx and Rx AMI files generated by SerDes Toolbox are compliant to the IBIS 6.1 specification, so all back-channel specific parameters have been placed in the Model_Specific section of the file. If you wish to make the models compliant to the IBIS 7.0 specification, update the AMI_Version to "7.0" and move all the BCI_* parameters into the Reserved_Parameters section of the file.

The BCI_State parameter has 5 states required for complete back-channel training, however to make these models more user-friendly the end user only needs 2 states: "Off" and "Training". To make this change, update the BCI_State parameter in each AMI file as follows:

- Change **(List 1 2 3 4 5)** to `(List 1 2)`.
- Change **(List_Tip "Off" "Training" "Converged" "Failed" "Error")** to `(List_Tip "Off" "Training")`.
- Note that this will not affect the operation of the model, only to the parameter values visible to the user.

Test Generated IBIS-AMI Models

The DDR5 transmitter and receiver IBIS-AMI models are now complete and ready to be tested in any industry standard AMI model simulator.

Model Limitations

When simulating with these models in an industry standard AMI model simulator, keep the following limitations in mind:

- BCI_Protocol is not supported. These models have been hard-coded to a Protocol named "DDRx_Write".
- BCI_ID is not supported. These models have been hard-coded to a BCI_ID named "bci_comm", which means that each simulation must be run in a separate directory to avoid filename collisions during simulation.
- These models must be run with a block size of 1024 for proper operation.
- Back-channel training must be enabled on both models for training to be enabled. This is done by setting the BCI_State parameters to "Training".
- These models will operate correctly with any UI or Samples Per Bit values.

References

1 IBIS 7.0 Specification, https://ibis.org/ver7.0/ver7_0.pdf

2 JEDEC website, https://www.jedec.org/

See Also

DFECDR | FFE | PassThrough | **SerDes Designer** | VGA

More About

- “DDR5 Controller Transmitter/Receiver IBIS-AMI Model” on page 7-26
- “DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model” on page 7-15
- “Managing AMI Parameters” on page 6-2

External Websites

- https://www.sisoft.com/support/