SerDes Toolbox™

User's Guide

MATLAB®&SIMULINK®

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

Revision History

Contents

Design and Simulate SerDes System Topics

1

Customize SerDes Systems Topics

2

Industry Standard IBIS-AMI Models

7

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.

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...
"Phase Detector" on page 1-3
"Recovering Clock Signal" on page 1-6

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.
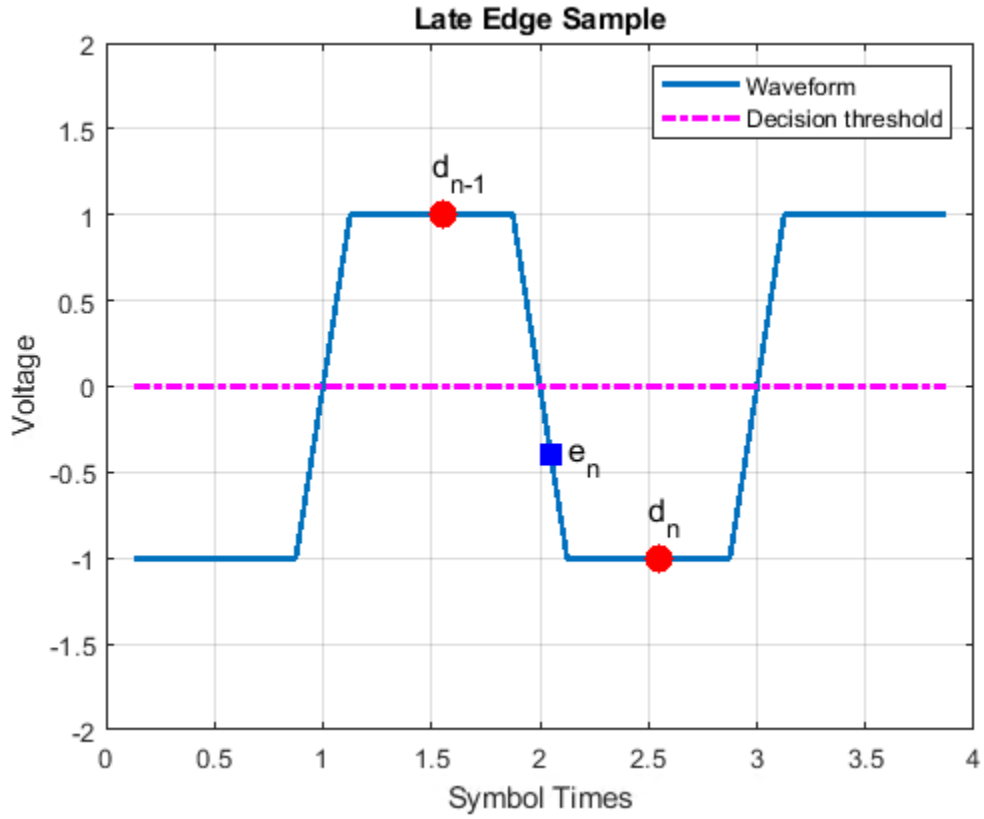
Similarly, when the binary values resolved from e_n and d_{n-1} match, the clock phase is early.
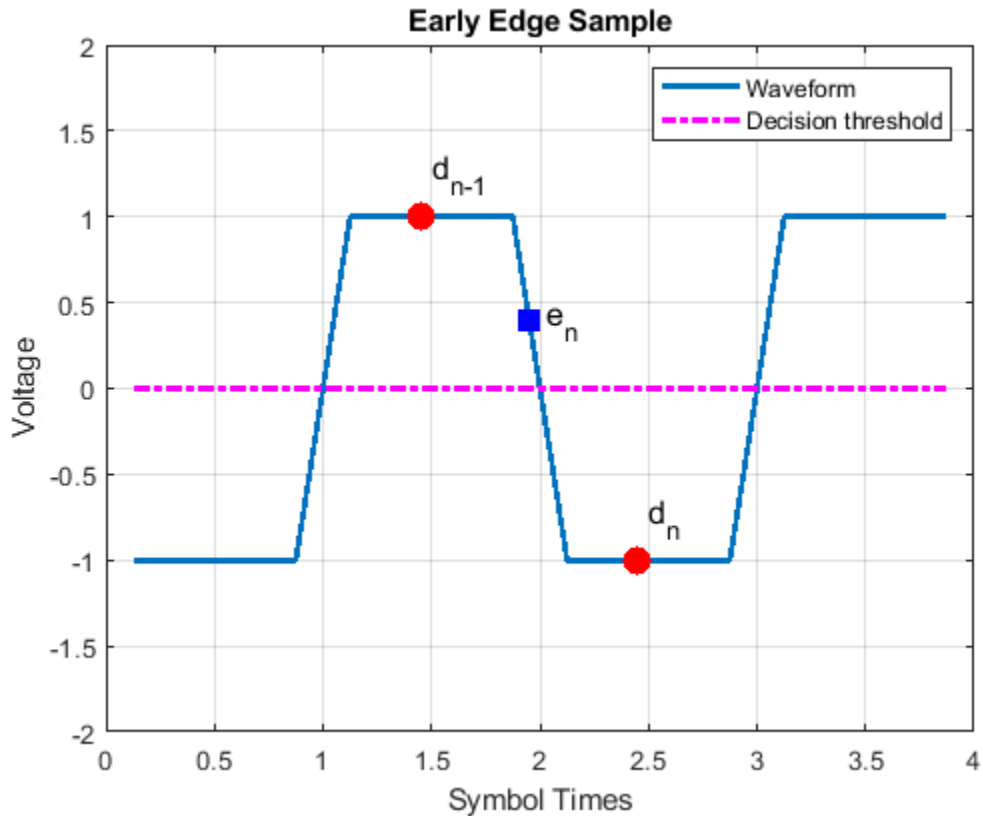
Representing the binary output of the sampler by ±1, the behavior of the phase detector for NRZ or PAM4 modulation is summarized here:

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	

For PAM3 modulation, the symbol levels are −0.5, 0, and 0.5. The default threshold levels (*th*) are ±0.25. The modified truth table thus become:

d_{n-1}	e_n	d_n	Action
−0.5	$e_n > -th$	0	late
−0.5	$e_n < -th$	0	early
−0.5	$e_n > 0$	0.5	late
−0.5	$e_n < 0$	0.5	early
0	$e_n > th$	0.5	late

d_{n-1}	e_n	d_n	Action
0	$e_n < th$	0.5	early
0	$e_n > -th$	−0.5	early
0	$e_n < -th$	−0.5	late
0.5	$e_n > th$	0	early
0.5	$e_n < th$	0	late
0.5	$e_n > 0$	−0.5	early
0.5	$e_n < 0$	−0.5	late

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.
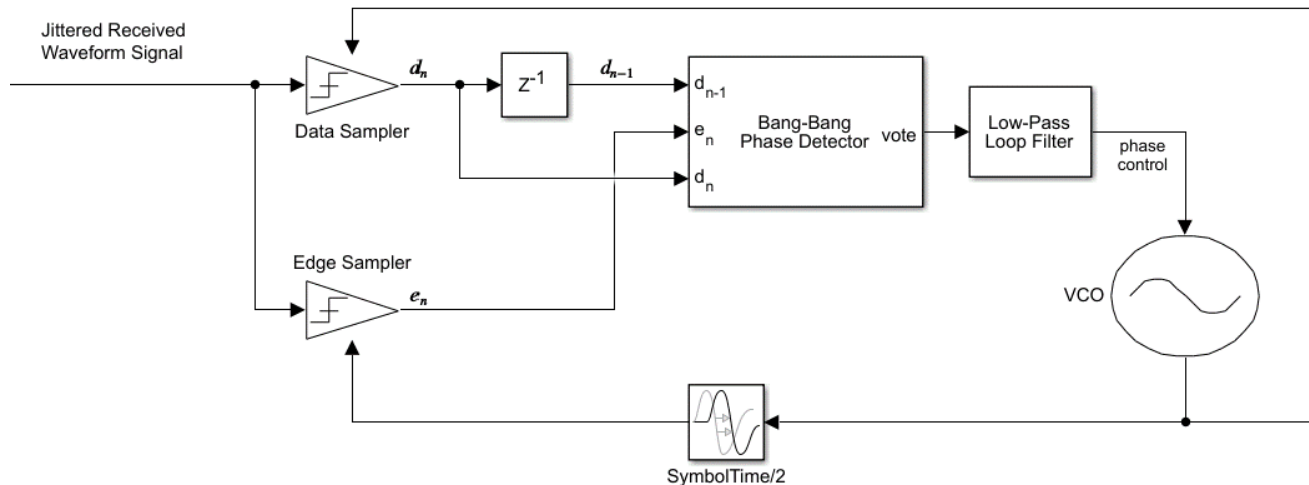
Recovering Clock Signal

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.
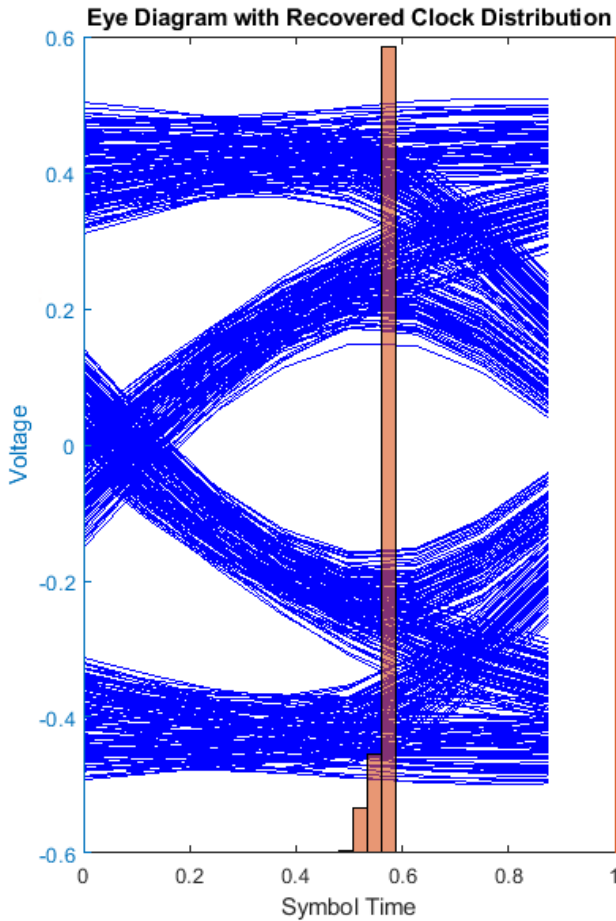
Eye Diagram with Recovered Clock Distribution

Channel Loss = 4 dB
phase step size = 0.0078125
Vote Count Threshold = 8
Phase Offset = 0
Reference Offset = 0
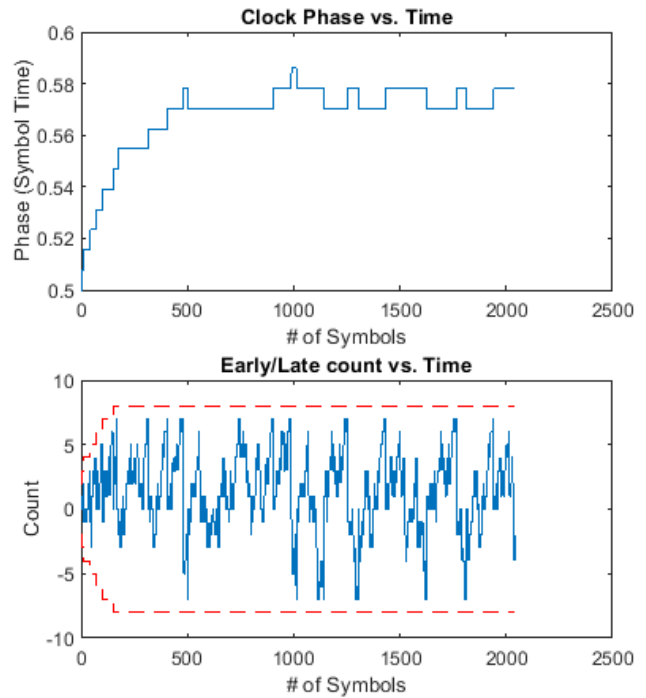
The output of the phase detector is accumulated in the early/late vote count. When the count exceeds the vote count threshold, the phase is incremented or decremented. To accelerate CDR convergence, the count threshold starts at 2, and each time the magnitude of the vote exceeds the threshold, the threshold 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.

Early/Late count vs. Time
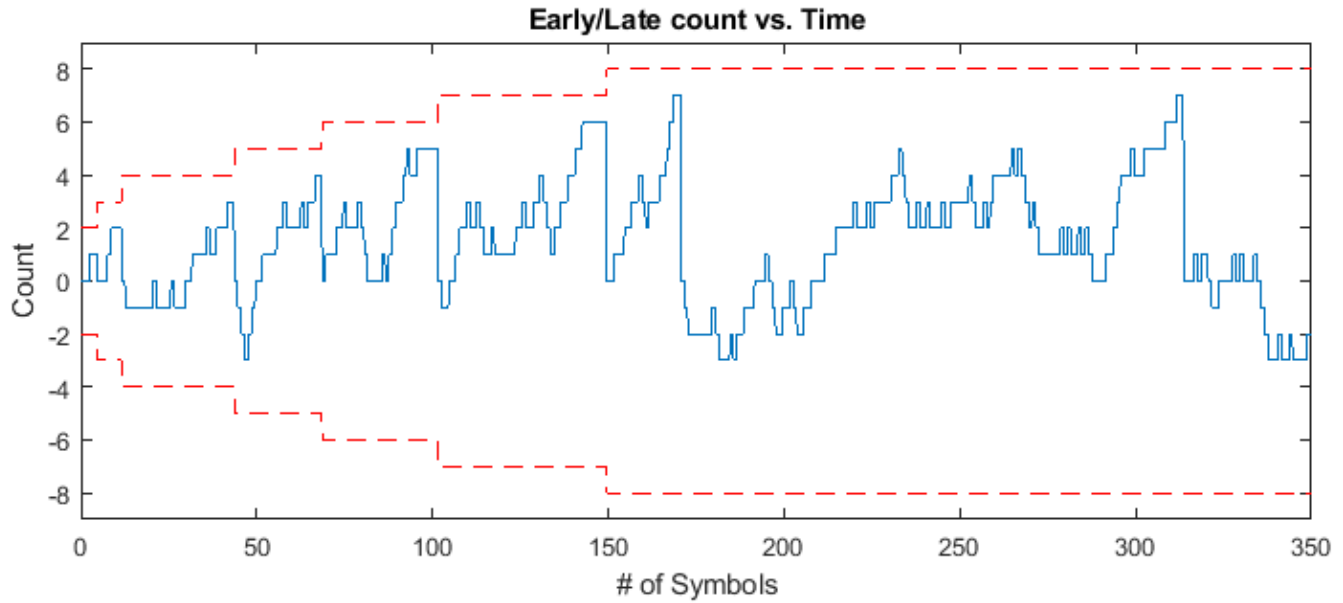
To show the clock converging to a different phase, change the channel loss to 2 dB. The clock phase now adapts to around 0.35 symbol time.

Eye Diagram with Recovered Clock Distribution
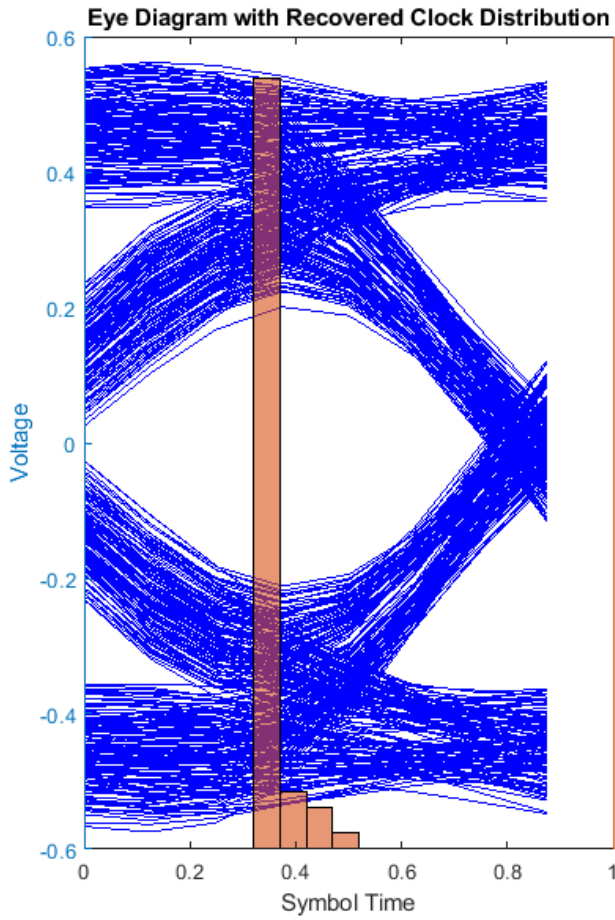
Channel Loss = 2 dB
phase step size = 0.0078125
Vote Count Threshold = 8
Phase Offset = 0
Reference Offset = 0

Clock Phase vs. Time

Early/Late count vs. Time
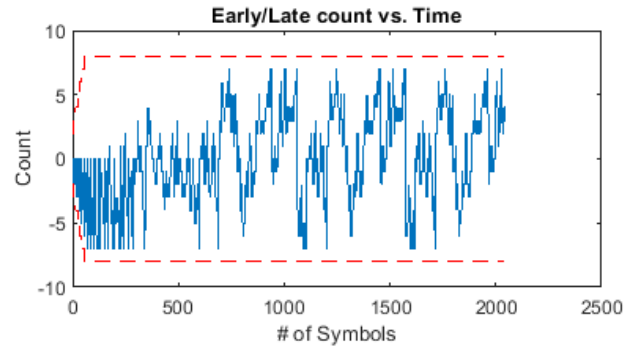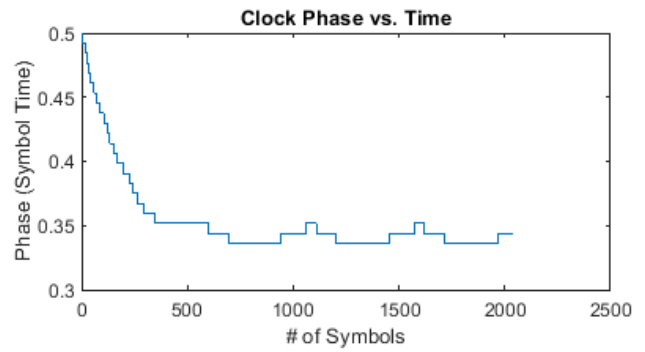
Increasing the vote count threshold to 16 results in a larger dithering period.
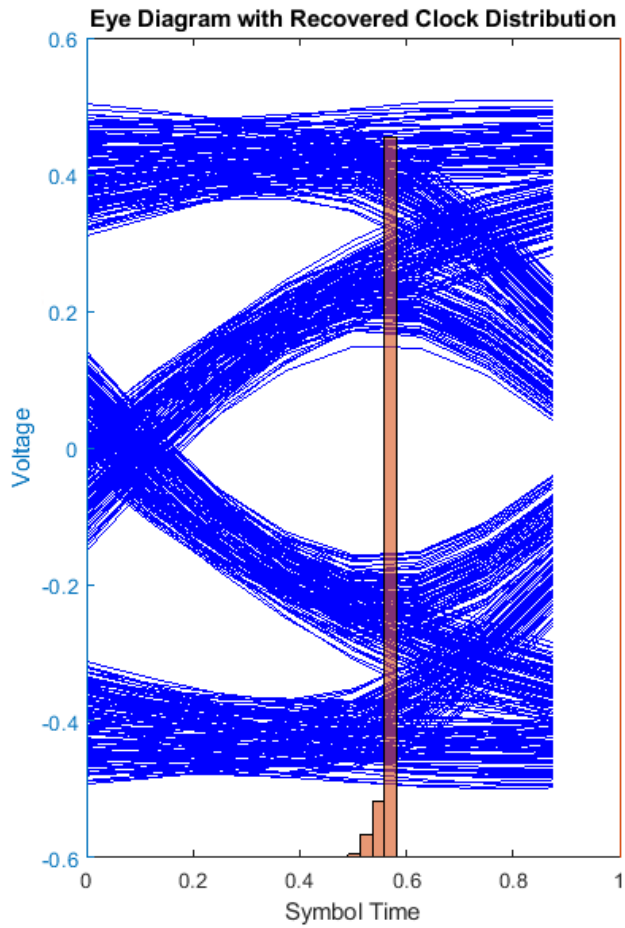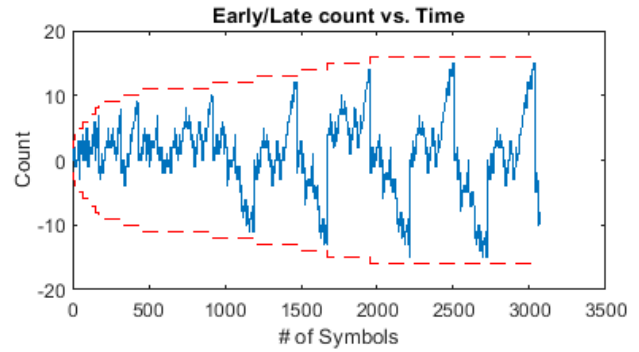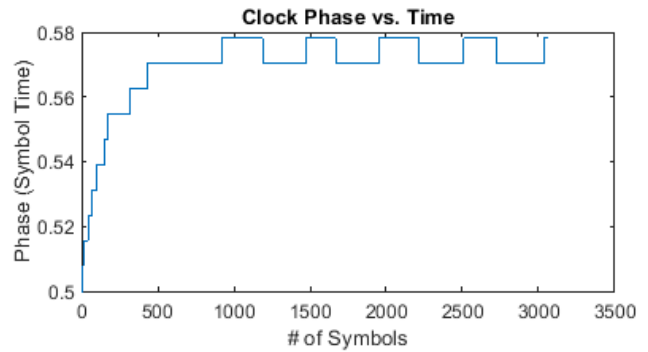
Increasing the phase step size to $\dfrac{1}{64}$ increases the dithering magnitude.

Eye Diagram with Recovered Clock Distribution
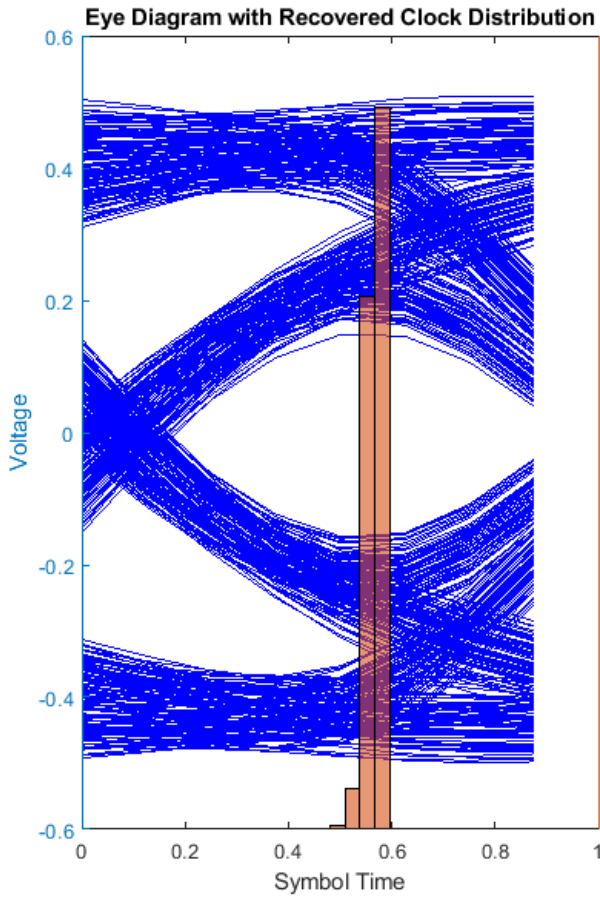
Channel Loss = 4 dB
phase step size = 0.015625
Vote Count Threshold = 8
Phase Offset = 0
Reference Offset = 0

Clock Phase vs. Time

Early/Late count vs. Time
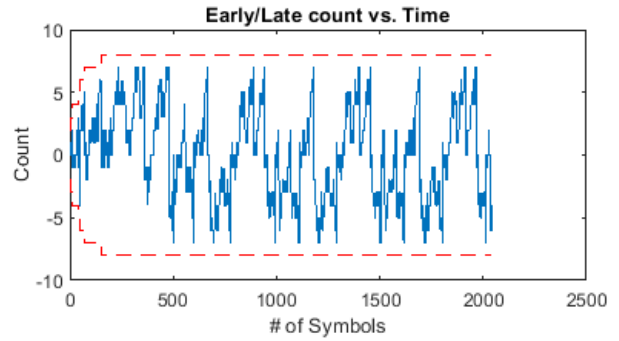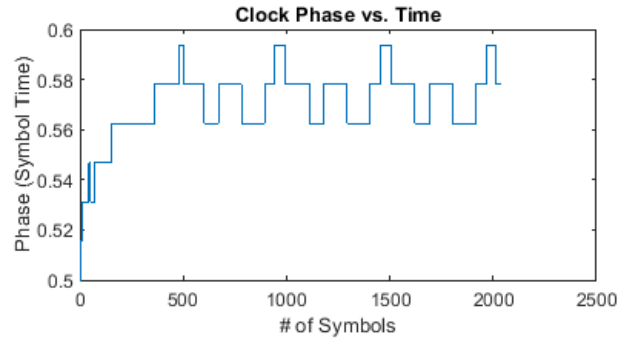
Manually shifting the data sampler location when the equalized eye does not display left/right symmetry can maximize the eye height. For example, shift the clock phase to the right by $\frac{1}{8}$ of a symbol time to shift the output clock phase from 0.57 symbol time to 0.7 symbol time.
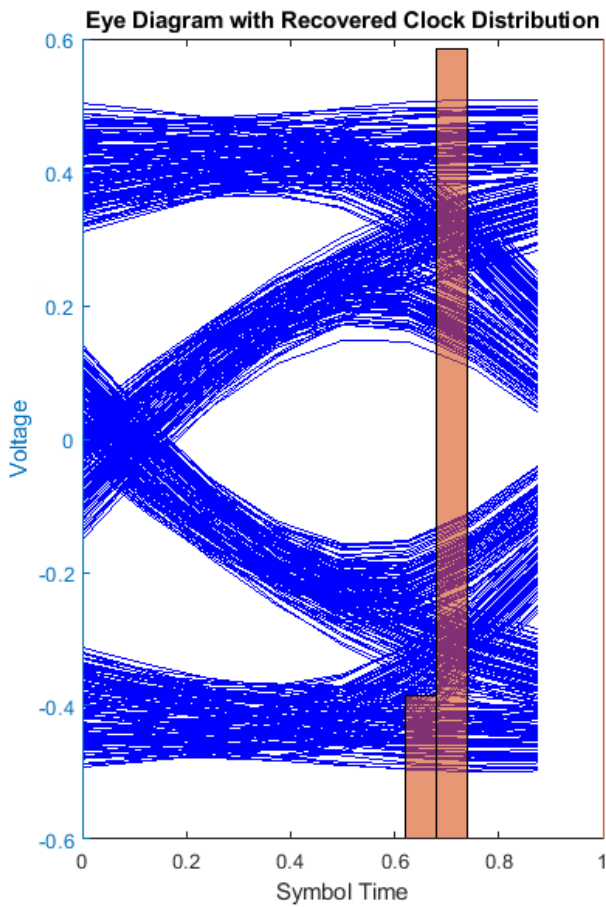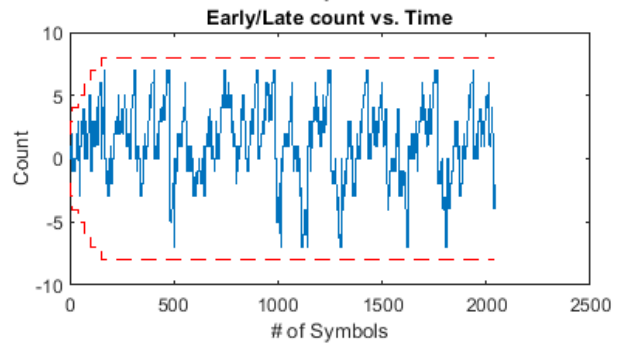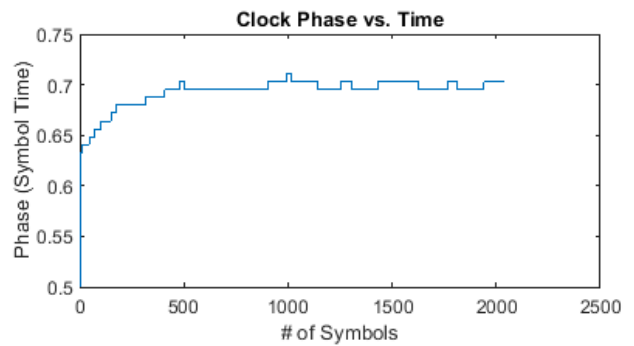
You can also inject a small amount of reference clock frequency offset impairment to implement a more realistic CDR.
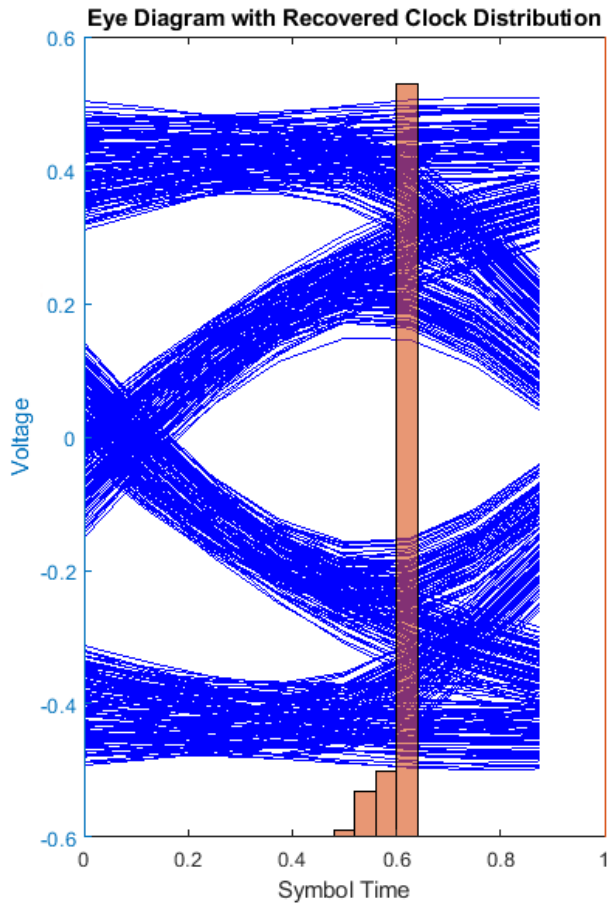
Eye Diagram with Recovered Clock Distribution

Channel Loss = 4 dB
phase step size = 0.0078125
Vote Count Threshold = 8
Phase Offset = 0
Reference Offset = 0.0003

Clock Phase vs. Time

Early/Late count vs. Time
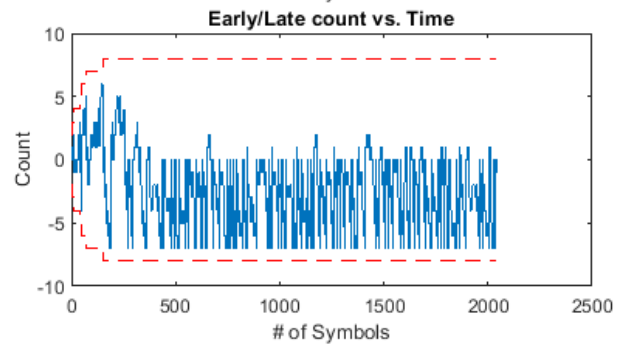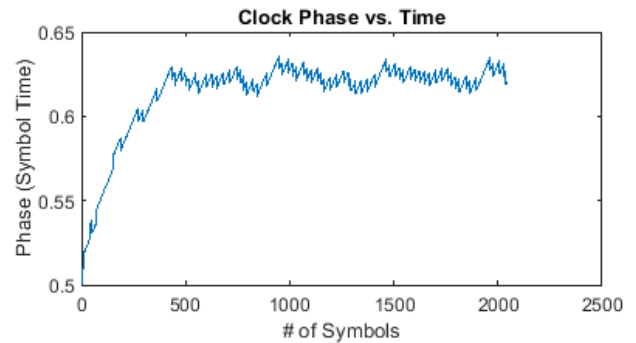
References

[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
serdes.CDR | serdes.DFECDR | DFECDR | CDR

Analog Channel Loss in SerDes System

In this section...
"Loss Model from Channel Loss Metric" on page 1-14
"Loss Model from Impulse Response" on page 1-14
"Introducing Cross Talk" on page 1-14

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.ChannelLoss` | **SerDes Designer**

Manage Contents of IBIS and AMI files

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 Model Type 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.
- **Ignore_Bits** — the number of bits ignored during time domain analysis.

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.

Customize AMI Parameters

You can define and modify the parameters of individual transmitter and receiver blocks. From the `Model_Specific` parameters, you can add new custom AMI parameters to specific blocks. The new AMI parameters references in the Simulink model are automatically maintained for you. 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 select to hide the `Model_Specific` AMI parameters and tap structures using the **Edit...** button The parameters still work the same way in Simulink models. But they are not written in the AMI files and does not show up in the `AMI_Parameters_Out` string. The hidden parameters are hard-coded to their current values in the DLL files and the end user cannot modify them.

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. Some of these reserved parameters are only used by the EDA tools. Simulink Coder™ ignores these parameters:

- `Rx_Decision_Time`
- `Ts4file (and Tx_V, Rx_R)`
- `Rx_Clock_Recovery_Mean`
- `Rx_Clock_Recovery_Rj`
- `Rx_Clock_Recovery_Dj`
- `Rx_Clock_Recovery_Sj`
- `Rx_Clock_Recovery_DCD`

Define Clock Position in Statistical Eye

You can define the timing of the clock in the statistical eye according to BIRD 205 using the reserved parameter `Rx_Decision_Time`. The corresponding Init function code, data structure, and AMI tree is automatically supported.

To add the `Rx_Decision_Time` parameter, click the **Reserved Parameters...** button in the AMI-Rx tab to open the Add/Remove dialog box. Once you add the parameter, it appears in the AMI tree. A

model workspace variable is also created You need to refresh the Init function in the receiver after you modify the Rx_Decision_Time parameter. After refreshing the Init function, the parameter appears in the Custom User Area.

You can use a CDR or DFECDR with baud-rate type-A phase detector model in the **SerDes Designer** app and then export the model to Simulink. In this case, automatically adds the Rx_Decision_Time parameter to define the clock position. If you want to change the phase detector model in Simulink, you need to manually add the Rx_Decision_Time parameter.

Note The Rx_Decision_Time parameter is only added in the receiver section.

PAMn Capabilities

The SerDes IBIS-AMI manager supports NRZ, PAM3, PAM4, PAM8, and PAM16 modulation schemes. If your model previously was created using IBIS pre-7.2 modulation levels, you can switch to IBIS 7.2 parameter Modulation_Levels by editing the **Modulation** AMI parameter under the Reserved_Parameter section. Switching to IBIS 7.2 parameter Modulation_Levels requires refreshing the Init function in the receiver side. The changes are applied both to transmitter and the receiver side.

Debug 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.

Note SerDes Toolbox does not support compilation of AMI_Wrapper.cpp files with non-inlined S-functions. As a result, you cannot export IBIS-AMI models with non-inlined S-functions. If you have a Simulink Coder or Embedded Coder® license, you can convert your S-functions to inlined to support IBIS-AMI model export. For more information, see "Inlining S-Functions" (Simulink Coder).

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.
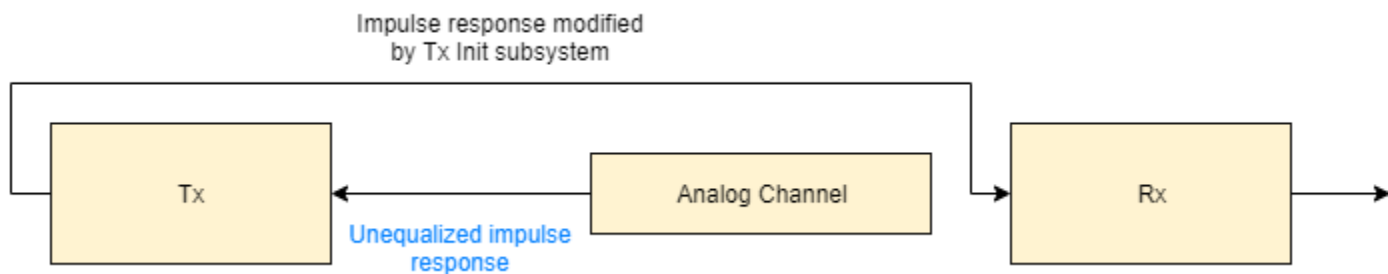
During a time-domain simulation, a digital stimulus waveform is passed to the transmitter model's time-domain function. This modified time-domain waveform is then convolved with the analog channel impulse response used in the statistical simulation. The output of this convolution is then passed to the receiver model's time-domain function. The modified output of the receiver becomes the simulation waveform at the receiver latch.
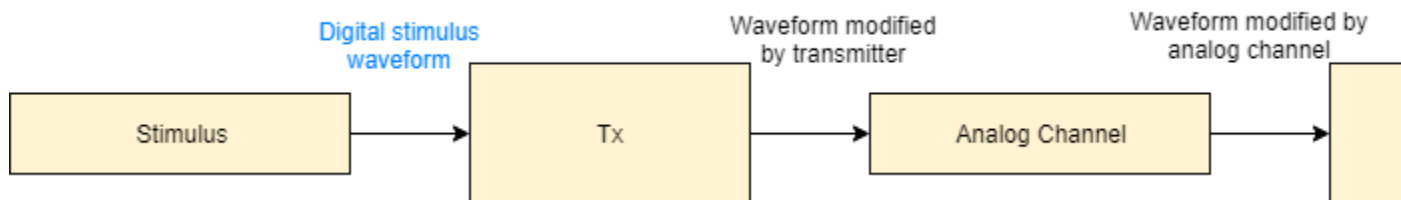
In SerDes Toolbox, the Init subsystem within both the Tx and Rx blocks uses an Initialize Function Simulink block. The Initialize Function block contains a MATLAB® function to handle the statistical analysis of an impulse response vector. The impulse response vector is generated by the Analog Channel 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.

	Name	Data
1	Eye Height (V)	0.4330
2	Eye Width (ps)	91.4896
3	Eye Area (V...	25.8097
4	COM	21.7733
5	VEC	0.7387
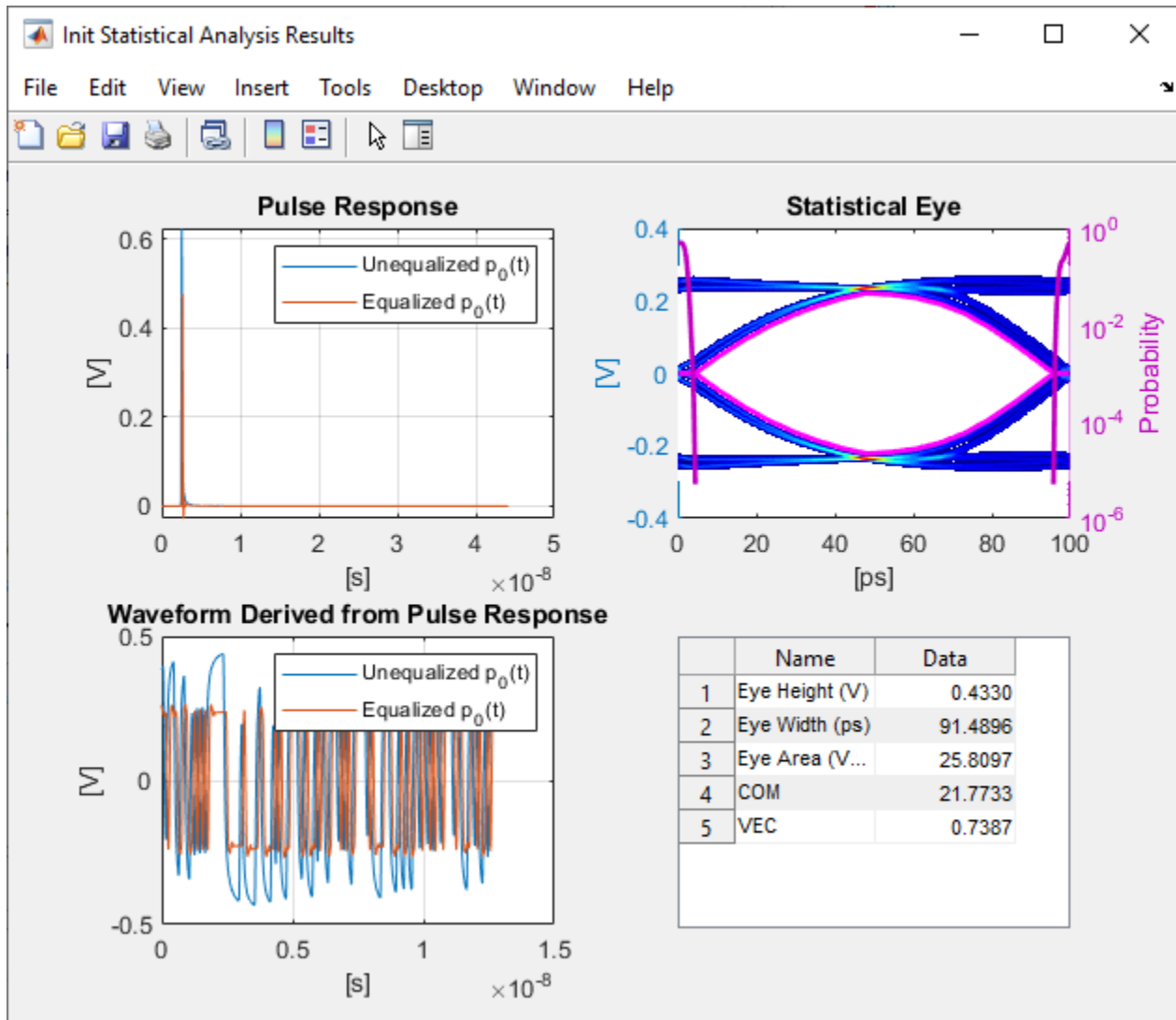
SerDes System Using Init Subsystem

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.
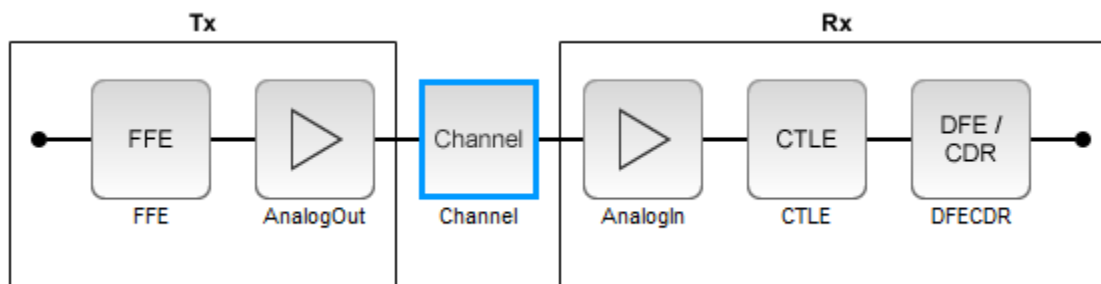
Access Init Code

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.

Reshape Impulse Response and Instantiate Tx System object

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;
```

Tx Impulse Response Processing

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;
```

Reshape Impulse Response and Instantiate Rx System object

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.

```matlab
%% 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;
```

Rx Impulse Response Processing

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.

```matlab
%% 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.

Custom User Code Area

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 5-14 and "Implement Custom CTLE in SerDes Toolbox PassThrough Block" on page 5-28.

The code generation of Init function (**Refresh Init**) can support one or multiple System objects when using the custom PassThrough block. If multiple system objects are present, they must be in series. The first input port must have a waveform as the input. If any waveform output is present, it must be the first output port.

PAMn Thresholds

If you are using a SerDes Toolbox datapath library block, PAMn thresholds in the Init function are maintained for you automatically. If you are using a custom configuration using a PassThrough, the code generation of the Init function finds the Data Store Write blocks that reference the PAMn threshold signals (`PAMn_UpperThreshold`, `PAMn_CenterThreshold`, `PAMn_LowerThreshold`) and determines connectivities. The connectivities that are supported are:

- Direct connection to System object
- Connection to System object through bus selector
- Connection to System object through Gain block
- Direct connection to Constant block

If the Init code generation cannot find a supported topology, it applies the default PAM4 thresholds.

Advance Init Options

External Init

You can export the Init code to an external MATLAB function, customize it, and then use the customized Init function for rapid analysis. To export the Init code, select the **External Init** option in the block parameters dialog box of either the Tx or Rx Init block, then click the **Refresh Init** button. This copies the contents of each of the Init MATLAB function blocks to `txInit.m` and `rxInit.m` files and links these functions back to the Simulink model. It also creates a `runExternalInit.m` file that runs these external Init files in MATLAB.

Once you have customized the Init code and you want to reintegrate the Init function back into the Simulink model, you can disable the **External Init** option and click the **Refresh Init** button again. This copies the contents of the Init function into the default Init files and deletes the external Init files.

Disable Default Impulse Response Processing

You can comment out the default impulse processing section of the Init code. This option comments out the code preforming the impulse response processing as shown in the "Tx Impulse Response

Processing" on page 1-22 and "Rx Impulse Response Processing" on page 1-23 sections. You can then customize the impulse response processing required for your system design in the "Custom User Code Area" on page 1-24.

Metrics Used in Statistical Analysis

Performance Metric	Description
Eye Height (V)	Eye height at the center of the BER contour
Eye Width (ps)	Eye width of the BER contour
Eye Area (V*ps)	Area inside the BER contour eye
Eye Linearity	Measure of the variance of amplitude separation among different levels of PAM3, PAM4, PAM8, or PAM16, given by the equation: $\text{Linearity} = \dfrac{\text{Minimum amplitude of the different eye levels}}{\text{Maximum amplitude of the different eye levels}}$
COM	Channel operating margin, given by the equation: $\text{COM} = 20\log_{10}\left(\dfrac{\text{Mean eye height}}{\text{Mean eye height - Inner eye height}}\right)$
VEC	Vertical eye closure, given by the equation: $\text{VEC} = \dfrac{\text{Mean eye height}}{\text{Inner eye height}}$

See Also

More About

- "Customizing Datapath Building Blocks" on page 5-14
- "Implement Custom CTLE in SerDes Toolbox PassThrough Block" on page 5-28
- "Managing AMI Parameters" on page 6-2
- "Customizing SerDes Toolbox Datapath Control Signals" on page 5-2

Jitter Analysis in SerDes Systems

Jitter is an important part of SerDes systems specification. You can include jitter parameters from the **SerDes Designer** app and from the Simulink model. Including jitter impairment in your link and equalization design helps calculate the required eye margins. You can also perform trade-off between different equalization schemes based on total jitter contribution. You can export the jitter values to IBIS-AMI models.

The most common types of jitter are:

Jitter Type	Description
DCD (duty cycle distortion)	Impairment from half and quarter rate clock misalignment. Also known as even-odd jitter. Duty cycle distortion is defined as the difference in symbol duration between one symbol and the next. The transmitter and receiver duty cycle distortions are half of the clock duty cycle distortion.
DJ (deterministic jitter)	Usually modeled as bounded uniform jitter. Also known as uncorrelated bounded high probability jitter. Deterministic jitter is defined as half of the peak-to-peak variation.
RJ (random jitter)	Gaussian process that models unbounded jitter events. Also known as uncorrelated unbounded Gaussian jitter. Random jitter is defined as the standard deviation of a white Gaussian phase noise process.
SJ (sinusoidal jitter)	Bounded periodic jitter that typically comes from power supply voltage variation. Sinusoidal jitter is defined as half of the peak-to-peak variation of sinusoidal phase noise amplitude.
Noise	Random voltage noise. IBIS-AMI 7.0 defines Gaussian noise and uniform noise impairments. Also known as additive white Gaussian noise (AWGN).

The expected simulation results vary depending on the type of jitter, injection site (transmitter or receiver), and analysis domain (statistical or time-domain). The **SerDes Designer** app only supports statistical or impulse-based analysis. To perform time-domain analysis, you must export the model to Simulink. The different types of jitter are injected into transmitter and receiver sites according to the IBIS-AMI specifications:

Statistical/Impulse Analysis Jitter Injection

Channel Impulse

TX

RX

Impulse2pulse

pulse2stateye

Statistical Eye

⊗

| Tx DCD |
| Tx DJ |
| Tx RJ |
| Tx SJ |
| Rx Noise |

=

Jittered Statistical Eye

Ideal Clock Eye

Jittered Statistical Eye + Clock PDF

Hula Hoop

⊗

Ideal Clock PDF

Rx Jitter	Rx Clock Recovery Jitter
DCD	DCD
DJ	DJ
RJ	RJ
SJ	SJ
	Mean
	Clock PDF

=

Clock PDF

Clocked Eye

⊗ =

Jittered Statistical Eye ⊗ Clock PDF
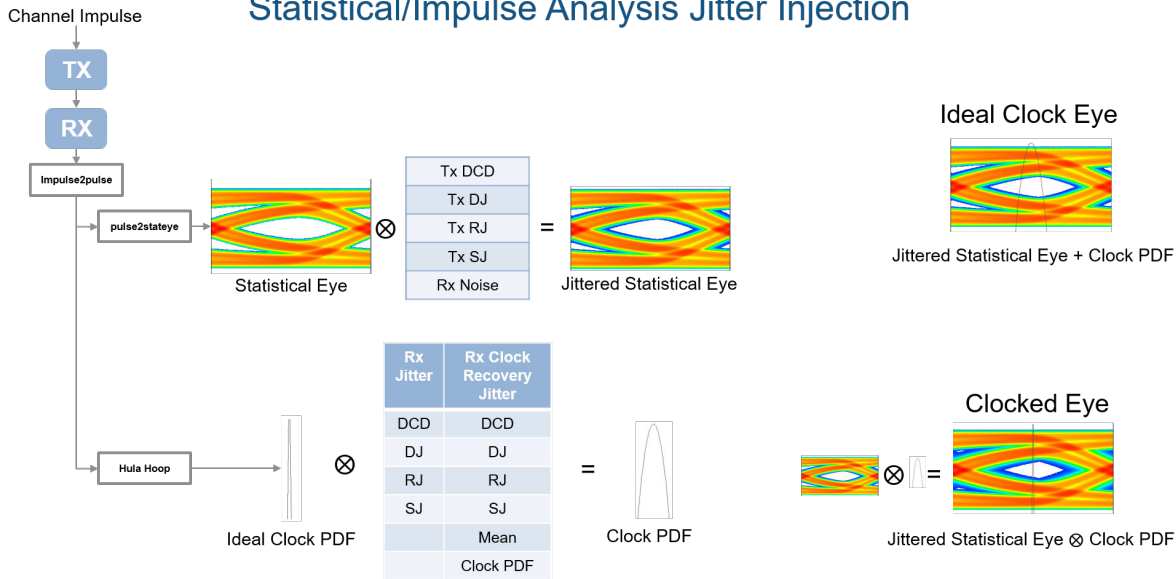
Time Domain Simulation Jitter Injection

| Tx DCD |
| Tx DJ |
| Tx RJ |
| Tx SJ |

Pattern Generator

stimulus

TX

Channel

RX

Clock times

Rx Noise

Rx Noise Injection

Waveform

| Rx Clock Jitter |
| DCD |
| DJ |
| RJ |
| SJ |

Jitter Injection

If Rx does not return clock times

Clock PDF

Trigger

| Rx Clock Recovery Jitter |
| DCD |
| DJ |
| RJ |
| SJ |
| Mean |
| Clock PDF |

Ideal Clock Eye

Accumulate persistent eye with ideal sampler

Clocked Eye

Accumulate persistent eye with jittered sampler

Ideal Clock Eye

Ideal clock eye + Clock PDF

Clocked Eye

Clocked eye with jittered sampler

Normal Mode	Statistical Analysis	Time Domain Analysis
Transmitter jitter	Convolved with eye	Injected in stimulus
Receiver jitter	Convolved with clock PDF (probability density function)	Injected in clock times
Clock recovery jitter	Convolved with clock PDF	Injected in clock times if receiver does not return clock times

See Also
SerDes Designer

External Websites
- https://ibis.org/ver7.0/ver7_0.pdf

Linux Version Compatibilities

When generating a shared object (.so) file on Linux® platform, MATLAB uses the GLIBC and GLIBCXX (GLIBC++) library versions provided by the operating system (OS). This means that when running these models on an equivalent OS from another vendor, you may encounter compatibility issues which prevent the model from running.

Supported Library Versions for Different OS

Linux OS	GLIBC Library Version	GLIBCXX Library Version	GCC Library Version
Debian® 9	2.24-11	3.4.22	6.3
Debian 10	2.28-10	3.4.25	8.3
Red Hat® 6.6	2.12	3.4.13	4.4.7
Red Hat 7.7	2.17	3.4.19	4.8.5
SUSE® 11.4	2.11.3	—	4.3.4
SUSE 12.3	2.19	—	4.8
SUSE 12.4	2.22	—	4.8

When generating a .so file, the compiler only uses the latest GLIBC/GLIBCXX version for each individual library function. So while the latest Debian 10 GLIBC version is 2.28, SerDes Toolbox only uses a sub-set of the GLIBC libraries. Depending on the blocks being used, it's possible that only v2.12 is required.

For example, a generated .so file for a random Rx AMI model on Debian 10 requires these libraries:

- 0x08922974 0x00 05 GLIBCXX_3.4
- 0x06969194 0x00 04 GLIBC_2.14
- 0x09691a75 0x00 03 GLIBC_2.2.5
- 0x09691a75 0x00 02 GLIBC_2.2.5

This shared object can run on any system with GLIBC v2.14 or later and GLIBCXX v3.4 or later. This means this shared object can run on Red Hat 7.7, but not on Red Hat 6.6.

Note This only applies to Linux shared objects. Windows® only requires the Universal C Runtime libraries to be compatible with ALL GLIBC/GLIBCXX versions.

A simple workaround to generate shared objects on an earlier Linux version than what is officially supported in MATLAB is to export the .so file on a fully supported platform, then manually run the build on an earlier version. To do this:

- Go to the transmitter or receiver build directory, denoted by `Tx_ert_rtw` and `Rx_ert_rtw`, respectively.
- From the command line, type the following command:
  - `make –f Tx.mk`
  - `make –f Rx.mk`

The generated shared objects reside one directory above the build directory (`../Tx.so` for transmitter and `../Rx.so` for receiver.)

- Copy the .so file to the full name used by SerDes Toolbox.

Note Manual build of the shared object requires a Simulink Coder or Embedded Coder license.

For a list of supported Linux versions, see Previous Releases: System Requirements and Supported Compilers.

See Also

External Websites

- Windows 10 Universal C Runtime

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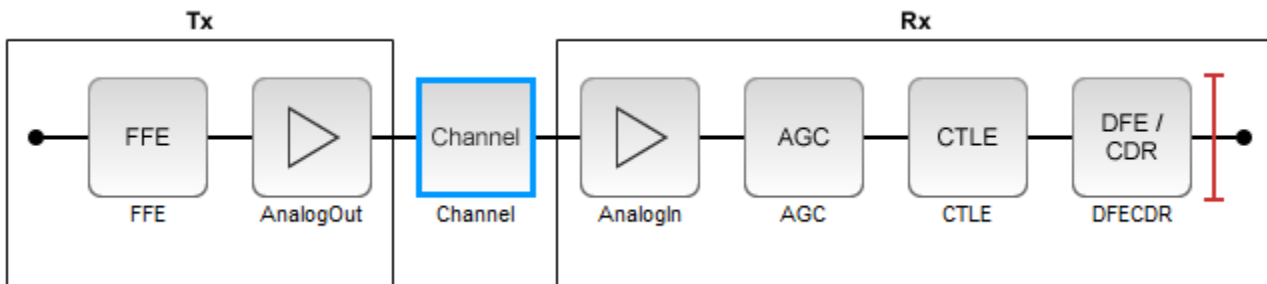
Select the channel block. Set **Channel loss (dB)** to 13.

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:
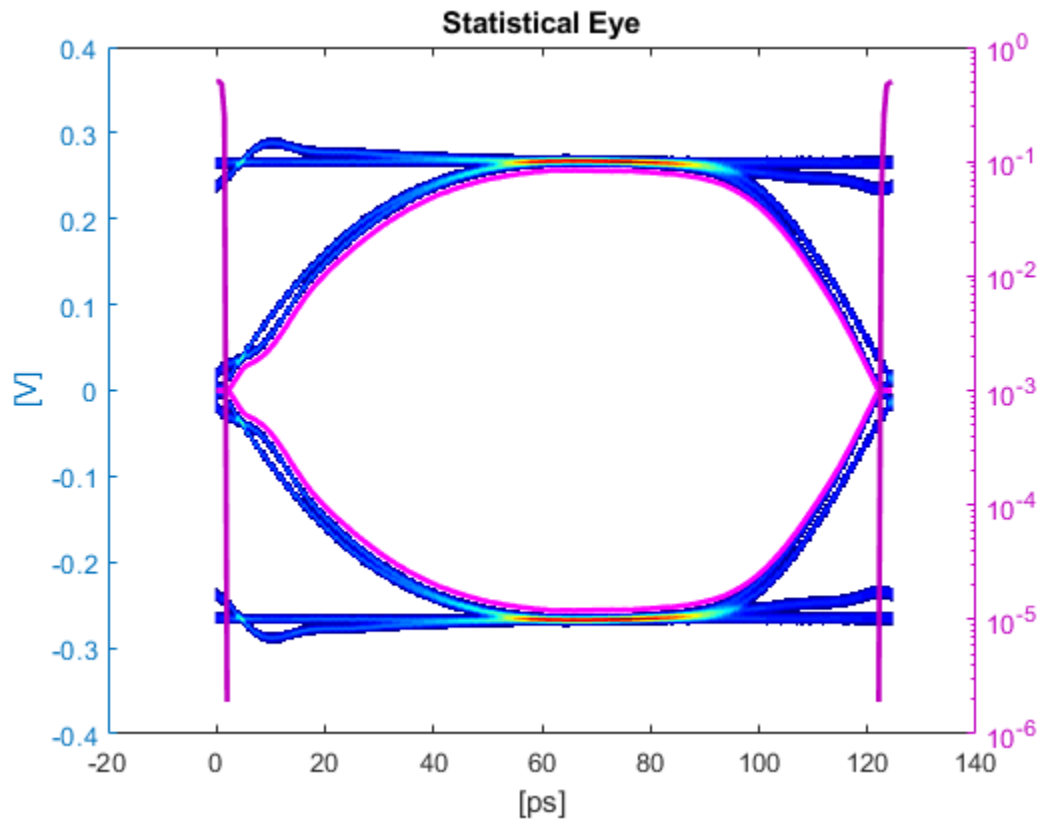
The eye diagram when the **Channel loss** is set to 16 dB:
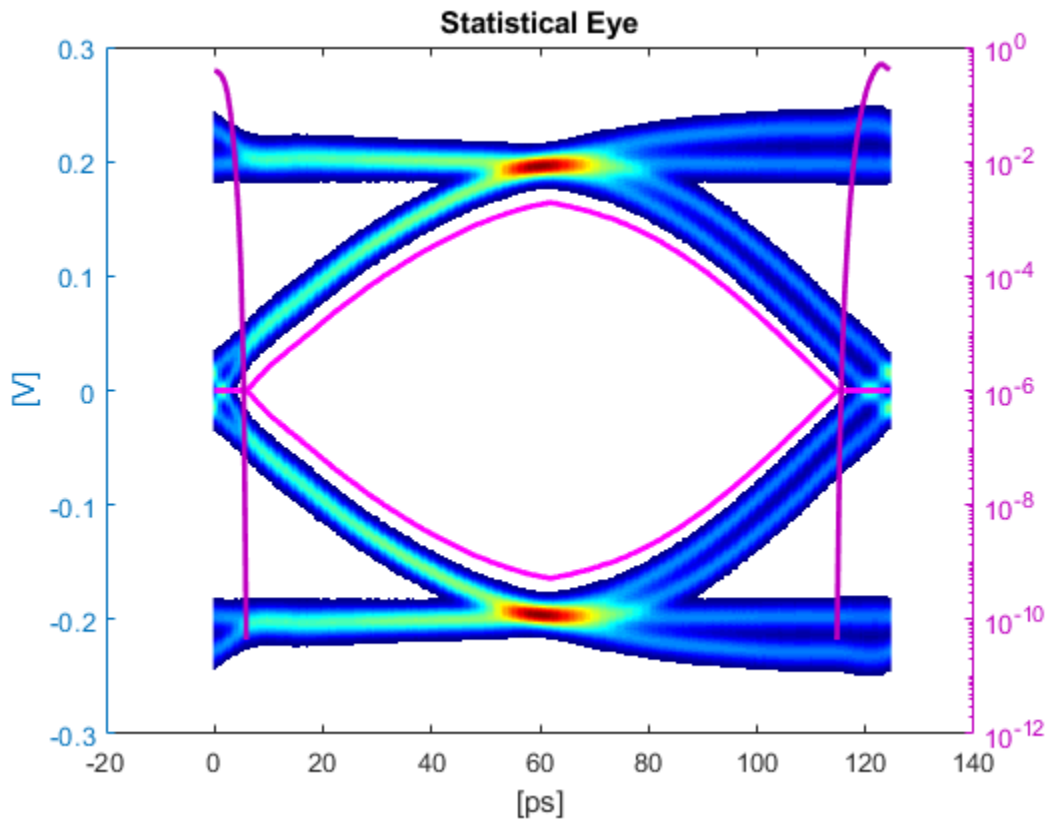
After you finalize the SerDes system with your desired Channel Loss, you can export the MATLAB script of the SerDes system as a Simulink model. From the Simulink canvas, you can perform further time-domain analysis, or export the system to a AMI model.

See Also

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

Create and Customize IBIS-AMI Models Topics

- "SiSoft Link" on page 3-2
- "SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software" on page 3-3
- "Signal Integrity Link" on page 3-11

SiSoft Link

> **Note** The SiSoft Link app is not recommended. Use "Signal Integrity Link" on page 3-11 instead.

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".

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

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.

Note SerDes Toolbox Interface for SiSoft Quantum Channel Designer and QSI Software is not recommended. Use "Signal Integrity Link" on page 3-11 instead.

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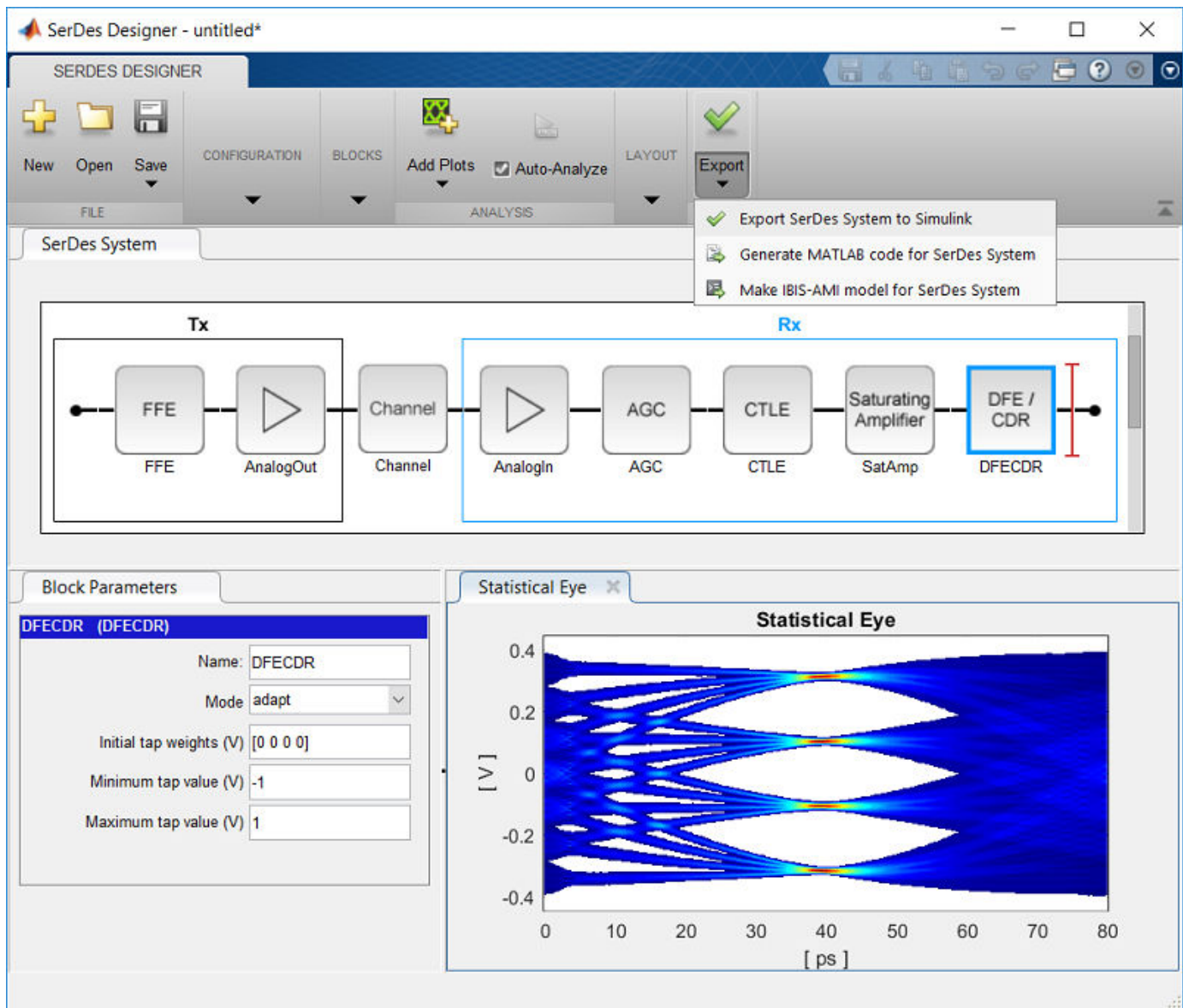
Add blocks from the Blocks gallery to the Tx and Rx sides. If you change the block parameters, the statistical eye display shows the performance changes. Click on **Export SerDes System to Simulink** from the Export dropdown menu to create a Simulink model for the system.

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.
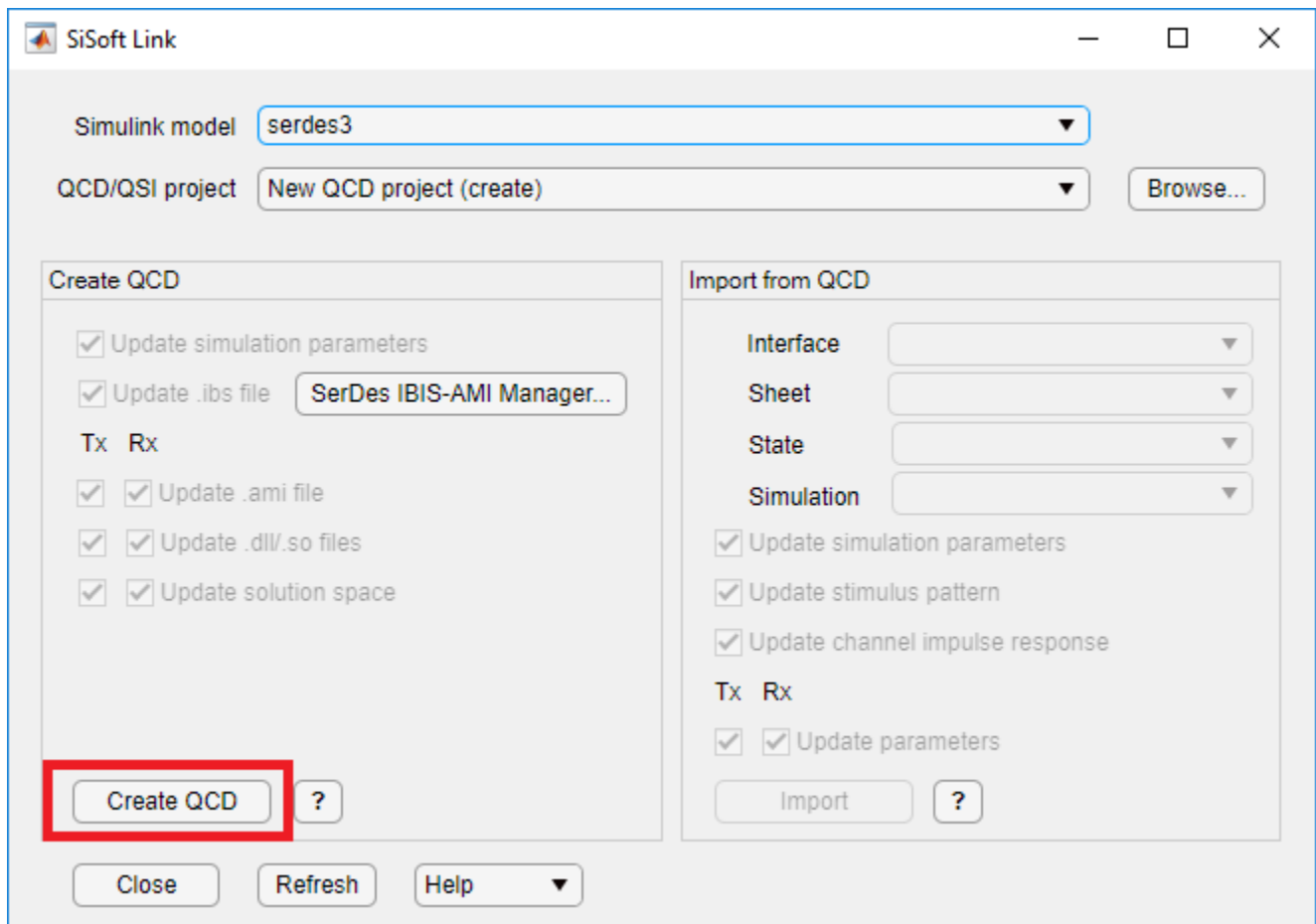
Choose a folder in which the QCD project resides and a name for the project folder. The folder path and project name must not have spaces. If you have not yet used SiSoft Link to create a project, the system asks you to locate the folder containing your SiSoft software. A report window appears and QCD 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.
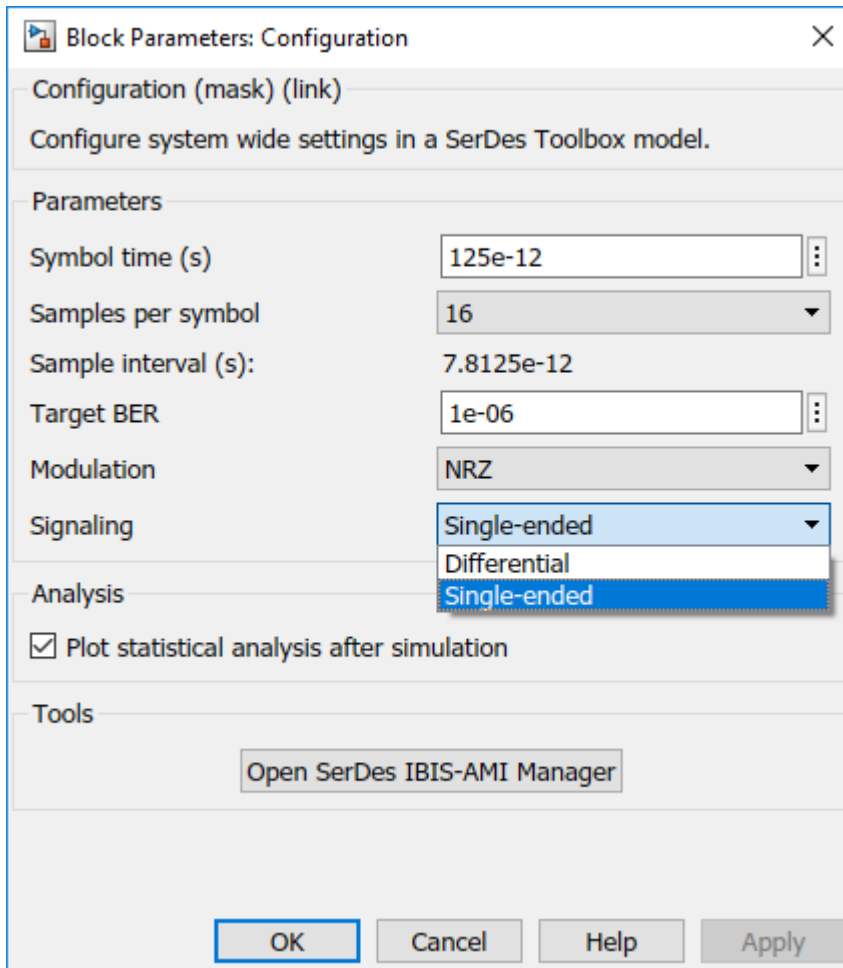
For QSI the following simulation parameters are set:

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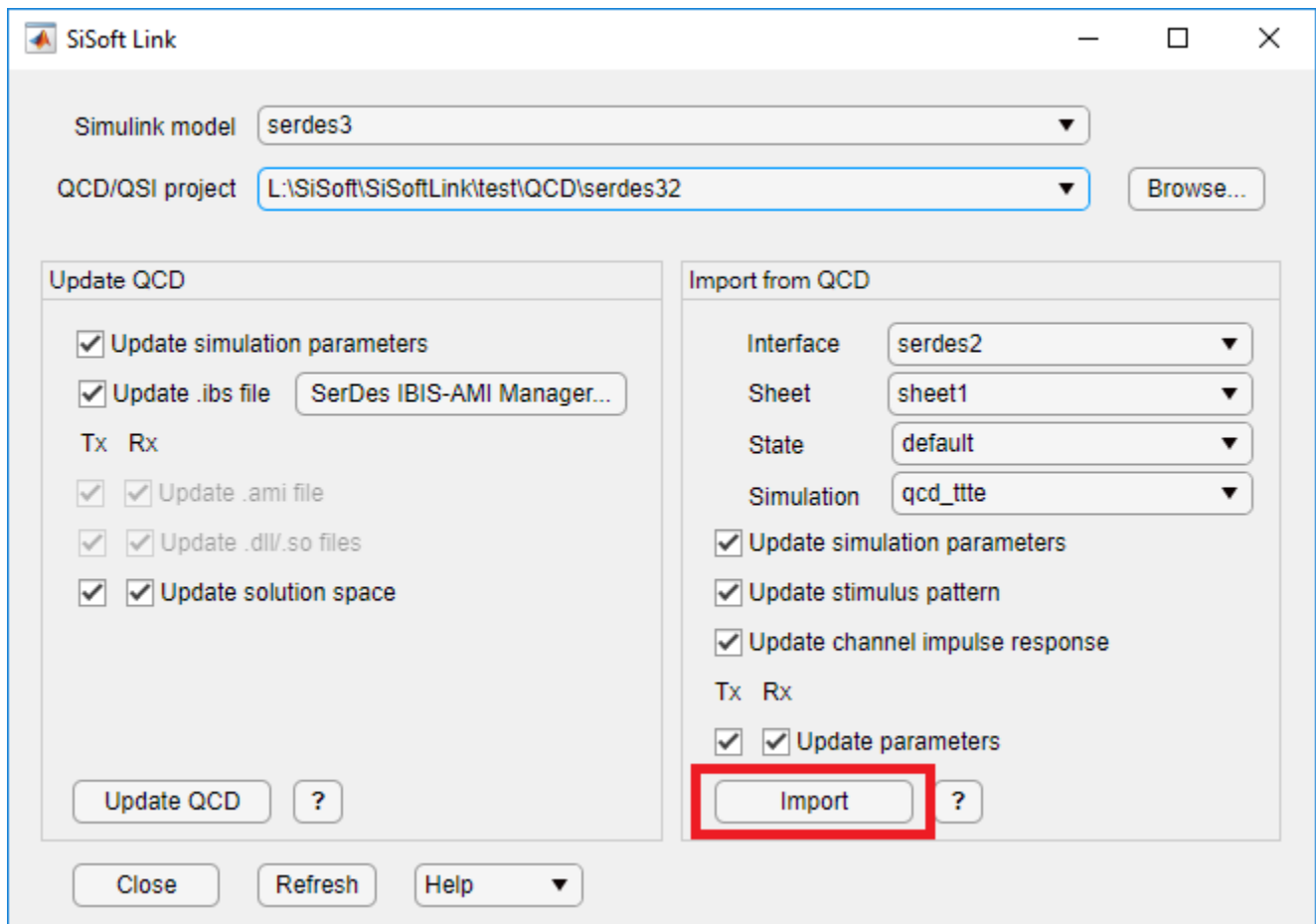
The following data are copied from QCD/QSI to Simulink, as enabled by the Import section checkboxes:

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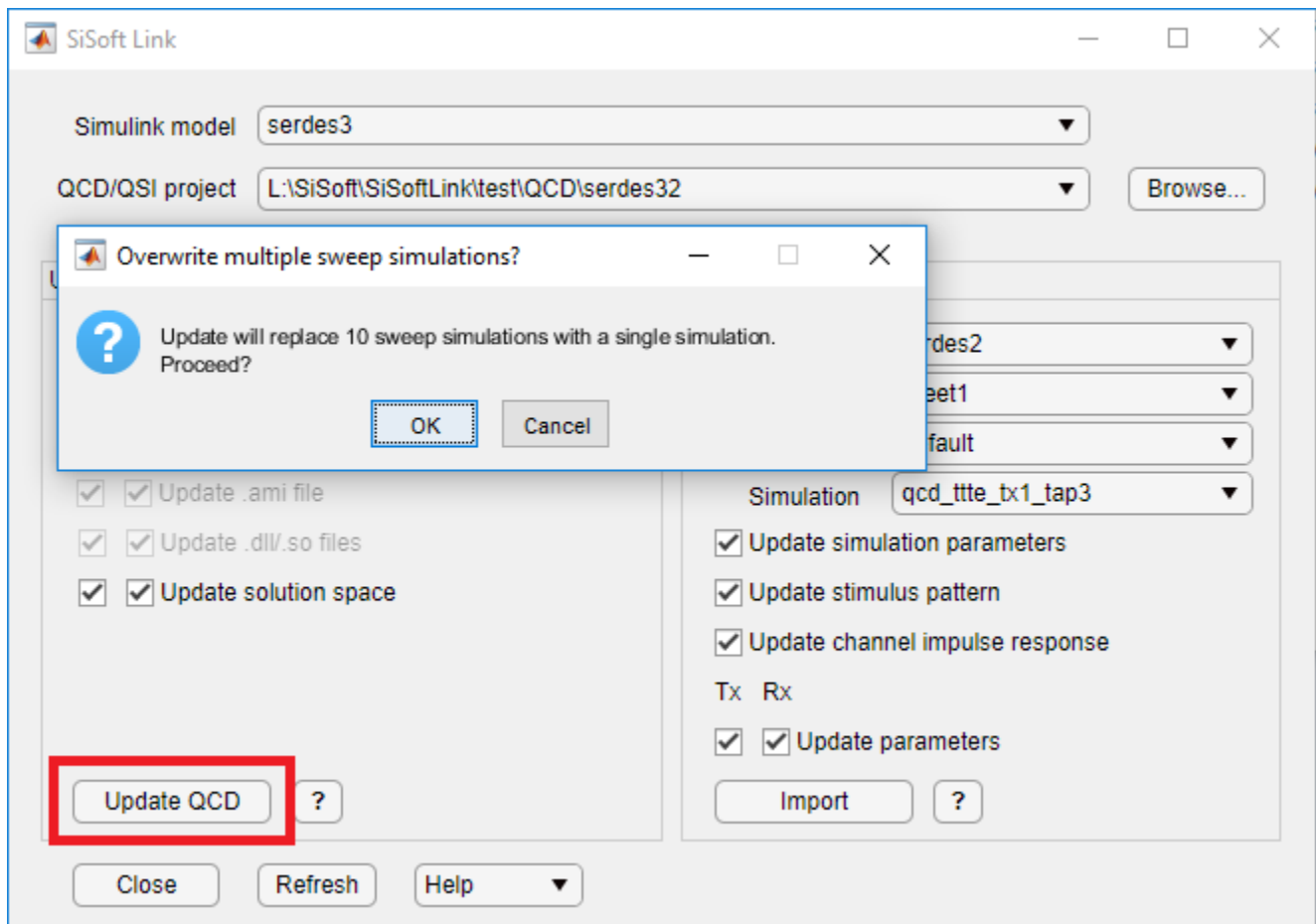
Clicking **Update QCD** (or **Update QSI**) runs the QCD (or QSI) to open the project and makes the changes. To avoid conflicts, you must close the project before updating it.

See Also

SerDes Designer | Analog Channel | Stimulus | Configuration

More About

- "SiSoft Link" on page 3-2

Signal Integrity Link

This example shows how to test the IBIS-AMI SerDes models developed in SerDes Toolbox using the Signal Integrity Toolbox™. You need a license to Signal Integrity Toolbox.

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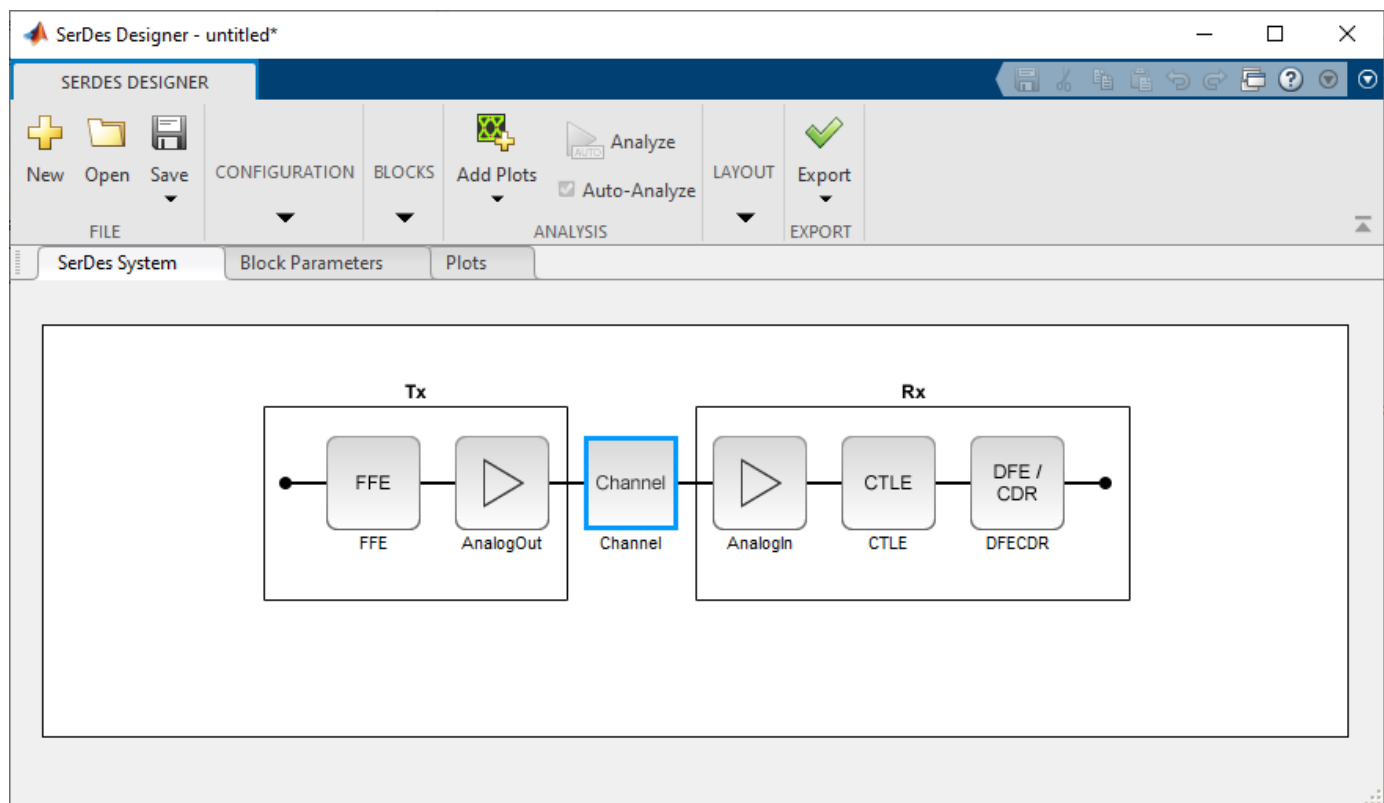
Add blocks from the Blocks gallery to the Tx and Rx sides. If you change the block parameters, the statistical eye display shows the performance changes. Click on **Export SerDes System to Simulink** from the Export dropdown menu to create a Simulink model for the system.

Prepare SerDes Simulink Model for Signal Integrity Toolbox

The Signal Integrity Toolbox 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.
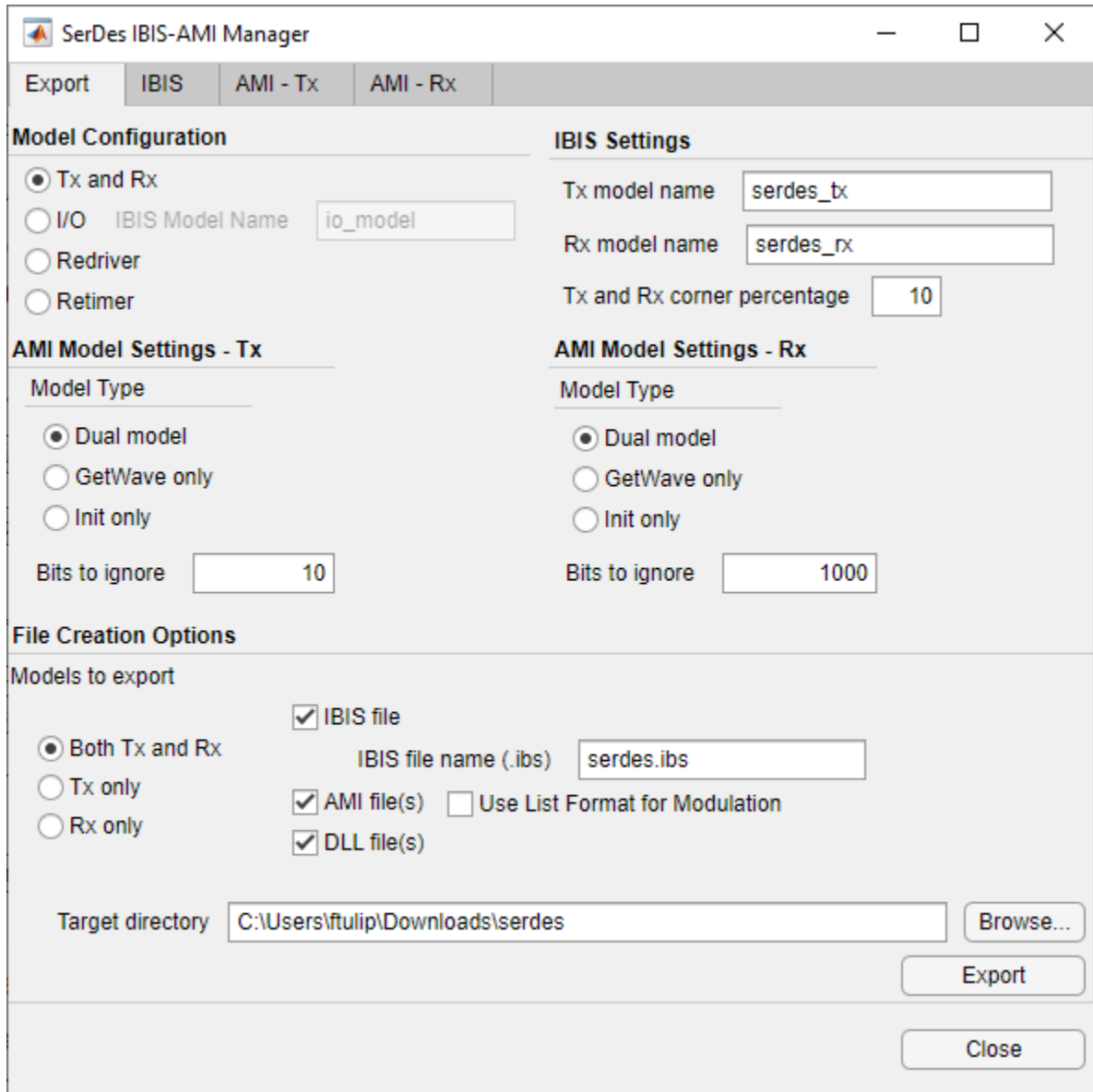
Serial Link Project

Click the "Open Signal Integrity Link" button in the Configuration block. In the newly opened dialog box, select `New serial link project` from the dropdown menu of **Link design project** parameter.

Choose a project name and destination folder. The folder path and project name must not have spaces. A report window appears and the **Serial Link Designer** app opens to create serial link project from SerDes Toolbox model.
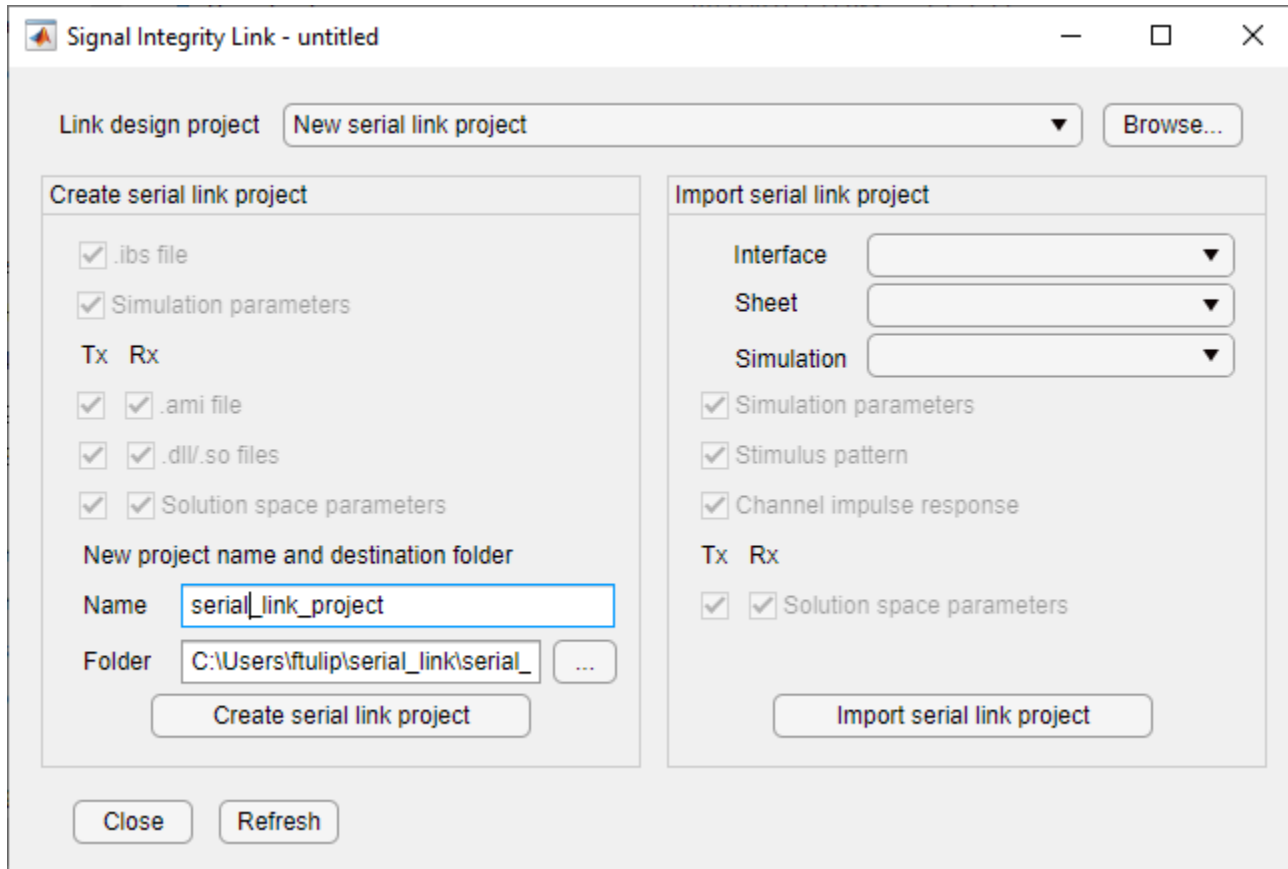
The following data are copied from Simulink to **Serial Link Designer**:

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

Once you create the serial link project, you can refresh the Signal Integrity Link to see your project from the **Link design project** parameter dropdown menu. You can make updates to existing project or import a new project from a simulation of an already simulated project.

Parallel Link Project

Click the "Open Signal Integrity Link" button in the Configuration block. In the newly opened dialog box, select `New parallel link project` from the dropdown menu of **Link design project** parameter.

Choose a project name and destination folder. The folder path and project name must not have spaces. A report window appears and the **Parallel Link Designer** app opens to create parallel link project from SerDes Toolbox model.
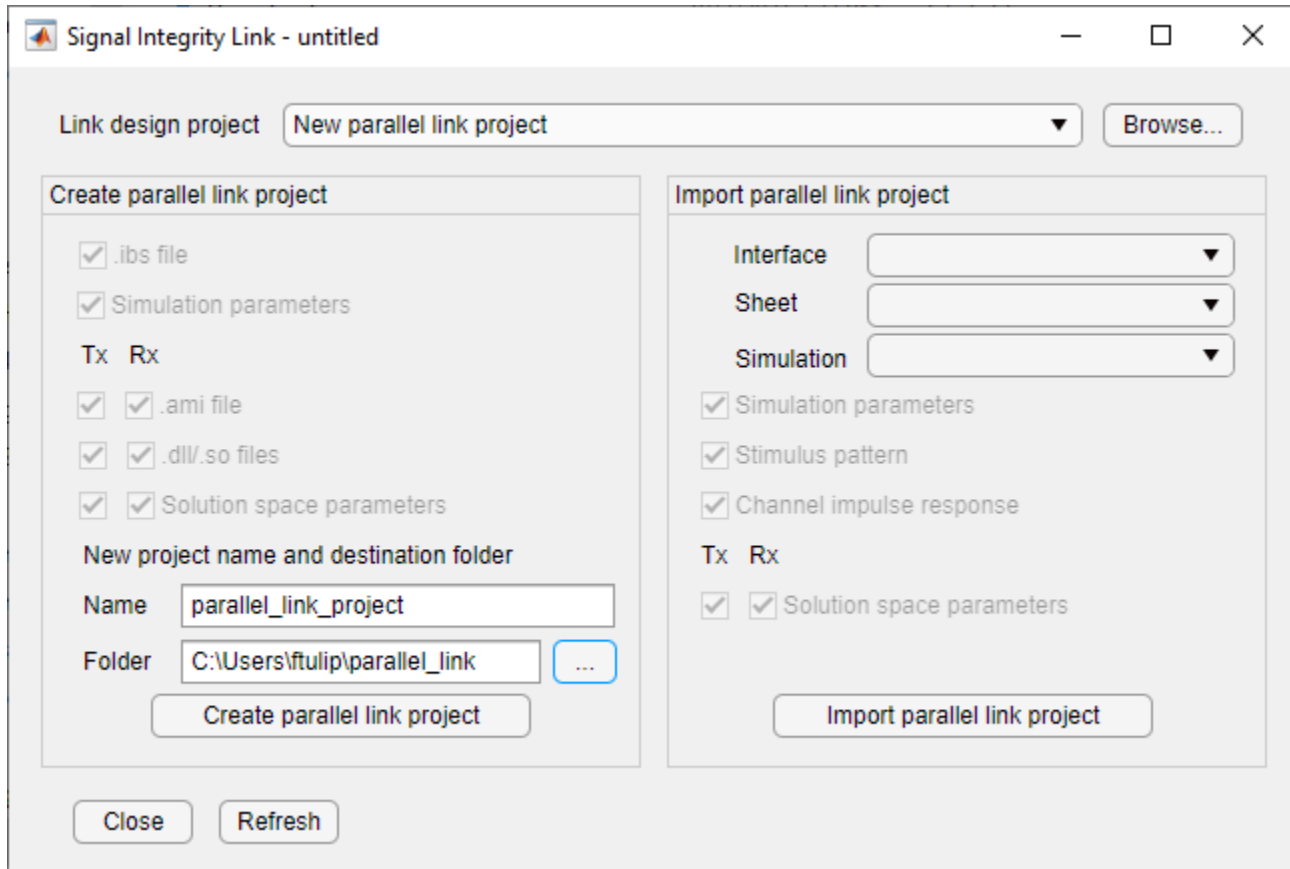
The following data are copied from Simulink to **Parallel Link Designer**:

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

Once you create the parallel link project, you can refresh the Signal Integrity Link to see your project from the **Link design project** parameter dropdown menu. You can make updates to existing project or import a new project from a simulation of an already simulated project.

See Also
SerDes Designer | Analog Channel | Stimulus | Configuration | **Serial Link Designer** | **Parallel Link Designer**

4

Design and Simulate SerDes Systems Examples

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

This example shows how to use the **CTLE Fitter** app to configure a CTLE block from SerDes Toolbox™ in the SerDes Designer app or in Simulink®. You can use the **CTLE Fitter** app to fit zeros, poles, and gains from a transfer function to create a `GPZ Matrix` and then export to your workspace. The **CTLE Fitter** app finds the `GPZ Matrix` by performing a fit comparison to a transfer function using the `rational` (RF Toolbox) function from RF Toolbox™.

Using CTLE Fitter App

You can open the **CTLE Fitter** app from SerDes Toolbox using any of the three workflows:

- From the CTLE block in the **SerDes Designer** app.
- From the CTLE block in the Simulink model.
- From the MATLAB® command window in the standalone mode.

Configure CTLE Block in SerDes Designer App

This workflow creates a variable representing a `GPZ Matrix` in the base workspace that is referenced by the CTLE block **GPZ properties** field in the **SerDes Designer** App. The steps are:

- Add a CTLE block and click the button **Launch CTLE Fitter App**.
- Import a CTLE frequency response. There can also be multiple responses in your data file.
- Adjust preprocess options for your transfer function data.
- Configure parameters of the `rational` function from RF Toolbox to optimize the fit to the transfer function.
- Visualize the fit response within the **CTLE Fitter** app using plots provided for magnitude response and pulse response.
- Close the **CTLE Fitter** app and continue with your session in the **SerDes Designer** app.

Simulink SerDes Model with CTLE block

This workflow creates a variable representing a `GPZ Matrix` in the model workspace and references this in the CTLE block mask **GPZ field.** The steps are:

- Open the CTLE block mask and click the button **Launch CTLE Fitter App**.
- Import a CTLE frequency response.
- Adjust preprocess options for your transfer function data.
- Configure parameters of the `rational` function from RF Toolbox to optimize the fit to the transfer function.
- Visualize the fit response within the **CTLE Fitter** app using plots provided for magnitude response and pulse response.
- Close the **CTLE Fitter** app and continue with your session in Simulink.

Standalone Mode

This workflow creates a variable in the base workspace representing a `GPZ Matrix.` The steps are:

- Launch the app with the MATLAB command `ctlefitter.`

- Import a CTLE frequency response.
- Adjust preprocess options for your transfer function data.
- Configure parameters of the `rational` function from RF Toolbox to optimize the fit to the transfer function.
- Visualize the fit response within the **CTLE Fitter** app using plots provided for magnitude response and pulse response.
- You have the option to both export a script and save a `GPZ Matrix` to the base workspace.
- Close the **CTLE Fitter** app and continue with your session in MATLAB

Configure CTLE Block in the SerDes Designer App

Launch the **SerDes Designer** app. Place a CTLE block after the analog model of the receiver. Then in the **Block Parameters** section, you can click on the button to launch the CTLE Fitter App.

Import One or More CTLE Frequency Responses

The app will open with some default values shown. Follow these steps to import a file containing one or more CTLE frequency responses:

- Click on the dropdown menu "Import CTLE frequency response from" and select the "CSV" option
- Click the Browse button to open a .csv file containing a transfer function. **Note:** You can use the file attached to this example "CTLEdefault1RealImag.csv" to explore the features of the `ctlefitter` app. In the screenshots below, this file has been placed in the folder "D:\data" but may be in a different location on your system.
- You will see the app loads the file and automatically updates the figure shown on the Plot tab:
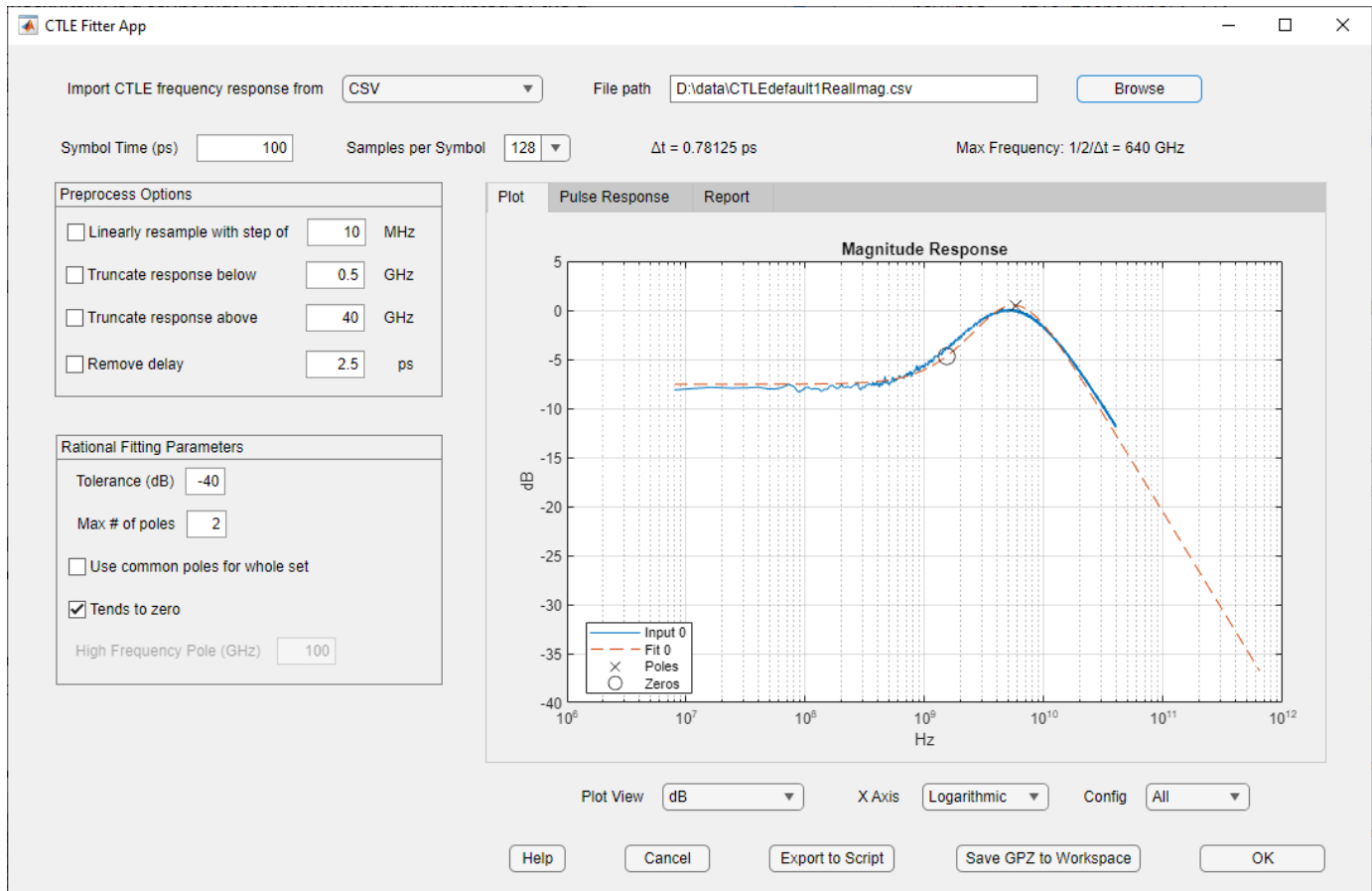
Adjust Preprocess Options

In the app, you can see many **Preprocess Options** are available. For example it is possible to truncate the data set from the transfer function used by the Fit. In the screenshot below you can see this is set to a cutoff frequency of 13 GHz:
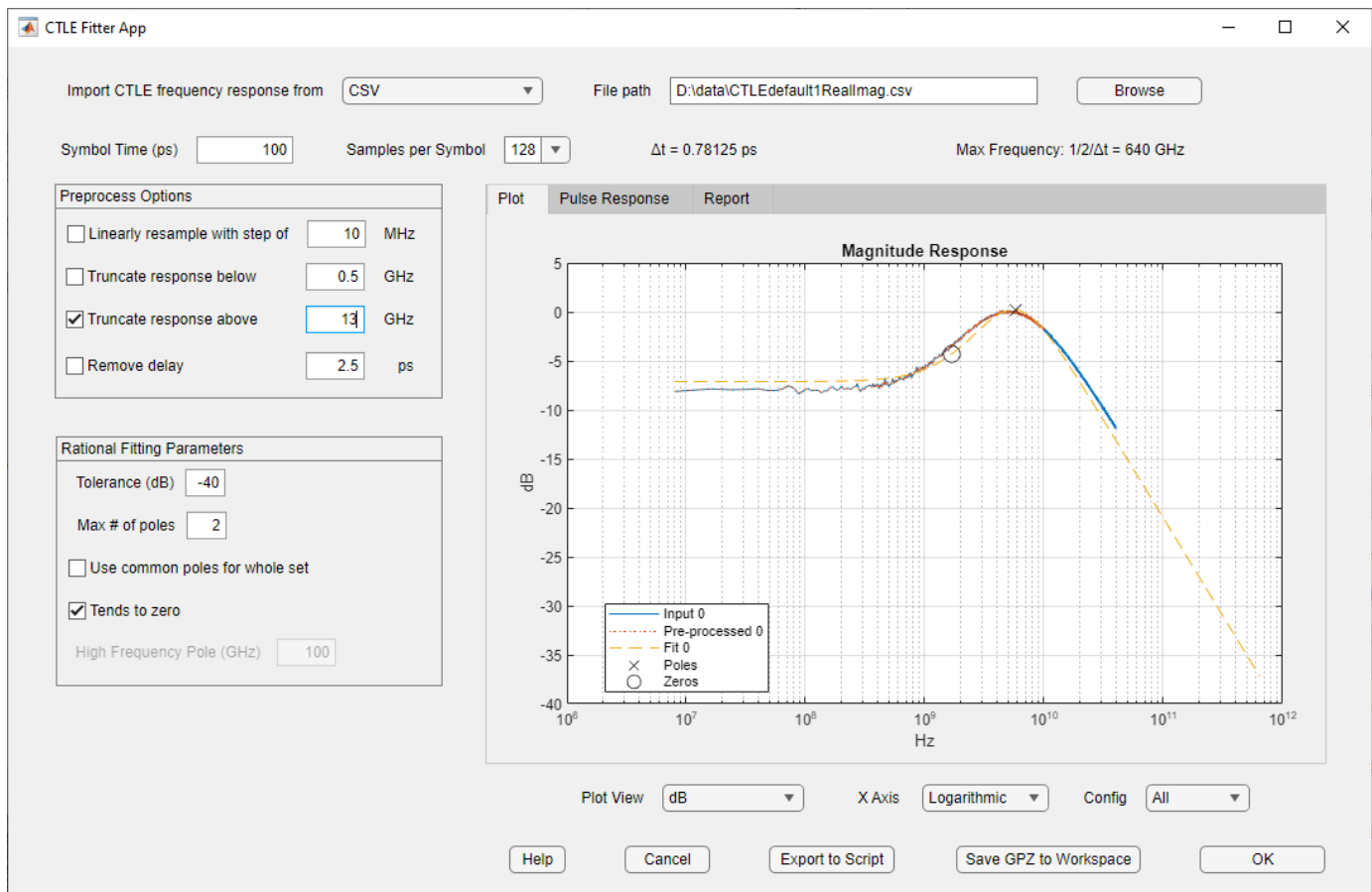
You can also adjust:

- Linearly resample with step size in MHz
- Truncate response below a specified frequency in GHz
- Truncate response above a specified frequency in GHz
- Remove delay in picoseconds

Configure Rational Fitting Parameters

You can configure the way the MATLAB function `rational` determines a fit by adjusting the following:

- Error tolerance (dB)
- Maximum number of poles
- Use common poles for whole set
- Enable or disable "tends to zero"

These parameters are explained in the documentation for the MATLAB function `rational`, which is part of the RF Toolbox.

Report on Rational Fit Results

You can view the statistical parameters of the fit reported by the MATLAB function `rational` on the "Report" tab:

Pulse Response

You can view the pulse response on the Pulse Response tab:
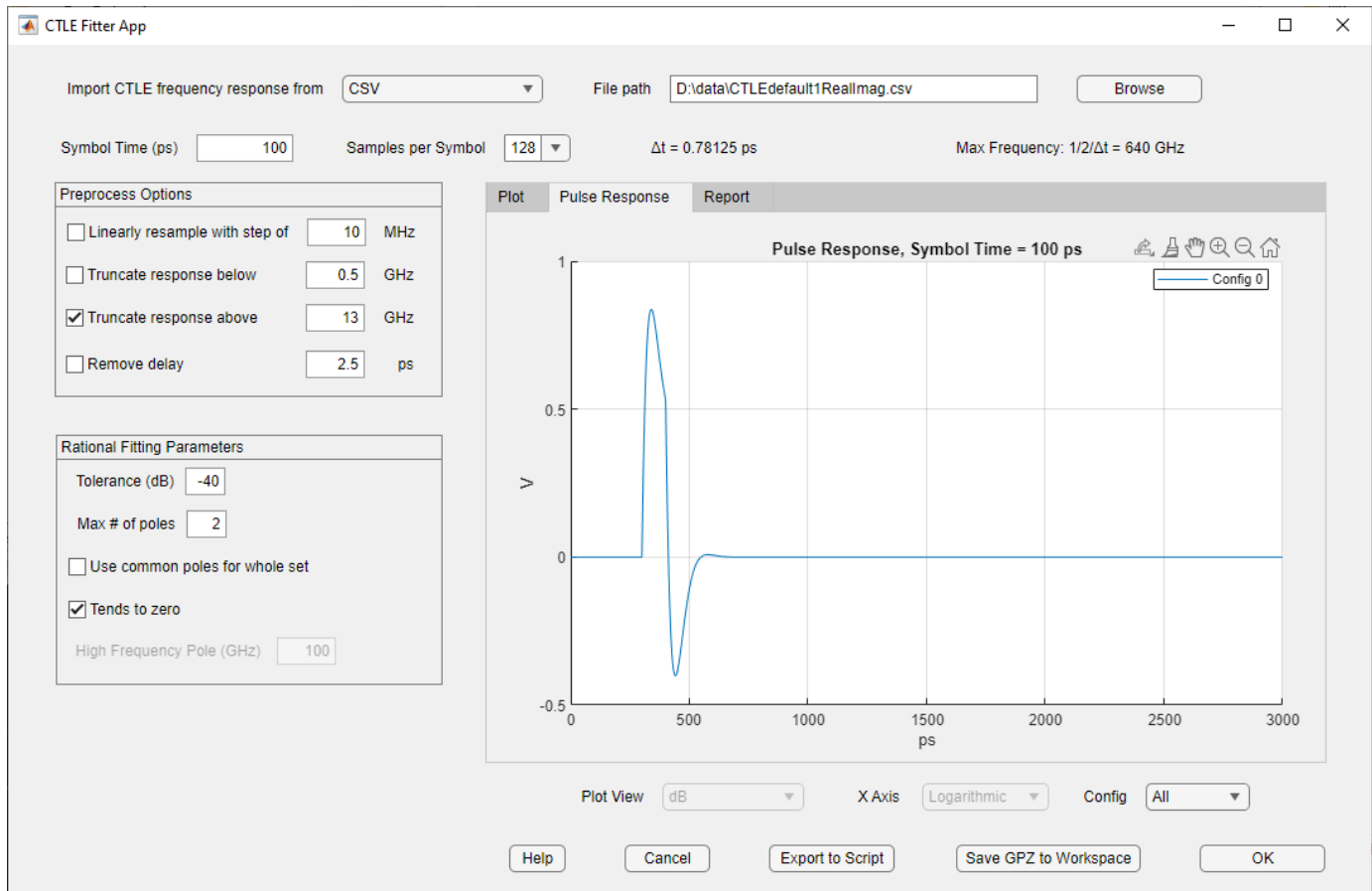
Export a GPZ Matrix for CTLE Block

You can export the `GPZ Matrix` to the Workspace by clicking on the button "Save GPZ to Workspace."

Note: If you have previously exported a `GPZ Matrix`, the name will automatically increment. For example, `gpz01` is created in the figure below, but if `gpz01` already exists in the workspace it would be automatically named `gpz02` and added to your workspace.
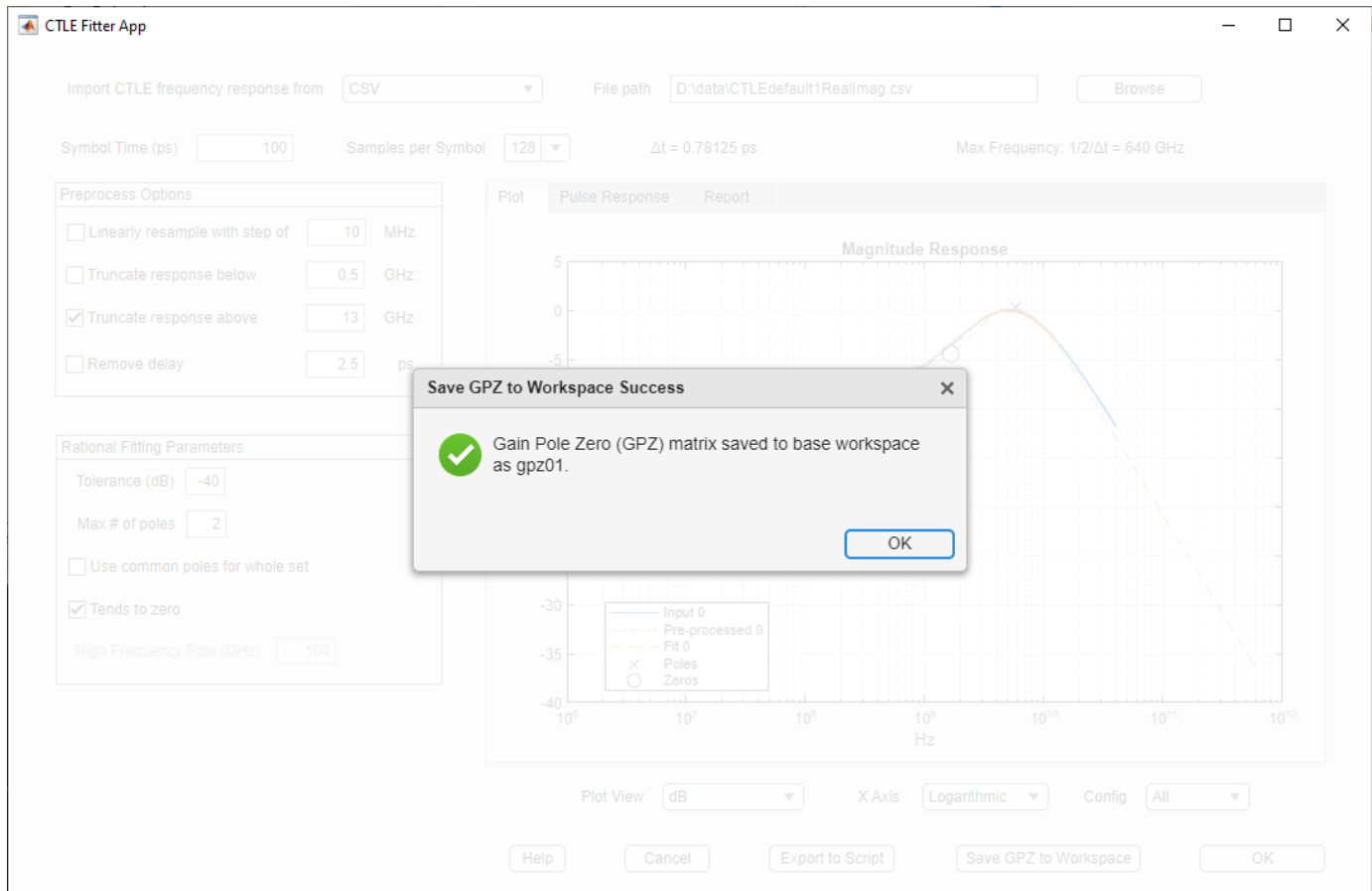
Export Script from CTLE Fitter App to Base Workspace

You can also export a script from the `ctlefitter` app to the base workspace by clicking on the button "Export to Script" and you can see example output below.

Note: The script contents you see may differ from the example below- depending on the data file being analyzed and your specific CTLE configuration options.

```
%Read in file:
fn = 'CTLEdefault1RealImag.csv';
[f,H]=ctlefit.readcsv(fn);

SymbolTime = 1e-10;

%Initialize ctleit object
obj = ctlefit(...
    'f',f,...
    'H',H,...
    'SampleInterval',7.8125e-13,...
    'MaxNumberOfPoles',2,...
    'ErrorTolerance',-40,...
    'TendsToZero',1,...
    'UseCommonPoles',0,...
    'PaddedPole',1e+11);
```

```matlab
%Preprocess transfer function waveform
df = 1e+07;
%resample(obj,df);
fcut1 = 5e+08;
%truncateBelow(obj,fcut1);
fcut2 = 1.3e+10;
truncateAbove(obj,fcut2);
delay = 2.5e-12;
%removeDelay(obj,delay);

%Get GPZ matrix
gpz = obj.GPZ;

%Visualize and create report
%TFView (Transfer function view) can be 'dB', 'Phase', 'Real/Imag',
%'Phase Delay', 'Group Delay'.
TFView = 'dB';
%ConfigSelect (CTLE Configuration Select) can be - 'All', 'Worst fit', 0 to
%N-1, where N is the number of configurations.
ConfigSelect = 'All';
%AxisStyle can be 'semilogx', 'plot', 'semilogy' or 'loglog'.
AxisStyle = 'semilogx';
figure,
plot(obj,TFView,ConfigSelect,AxisStyle)
```
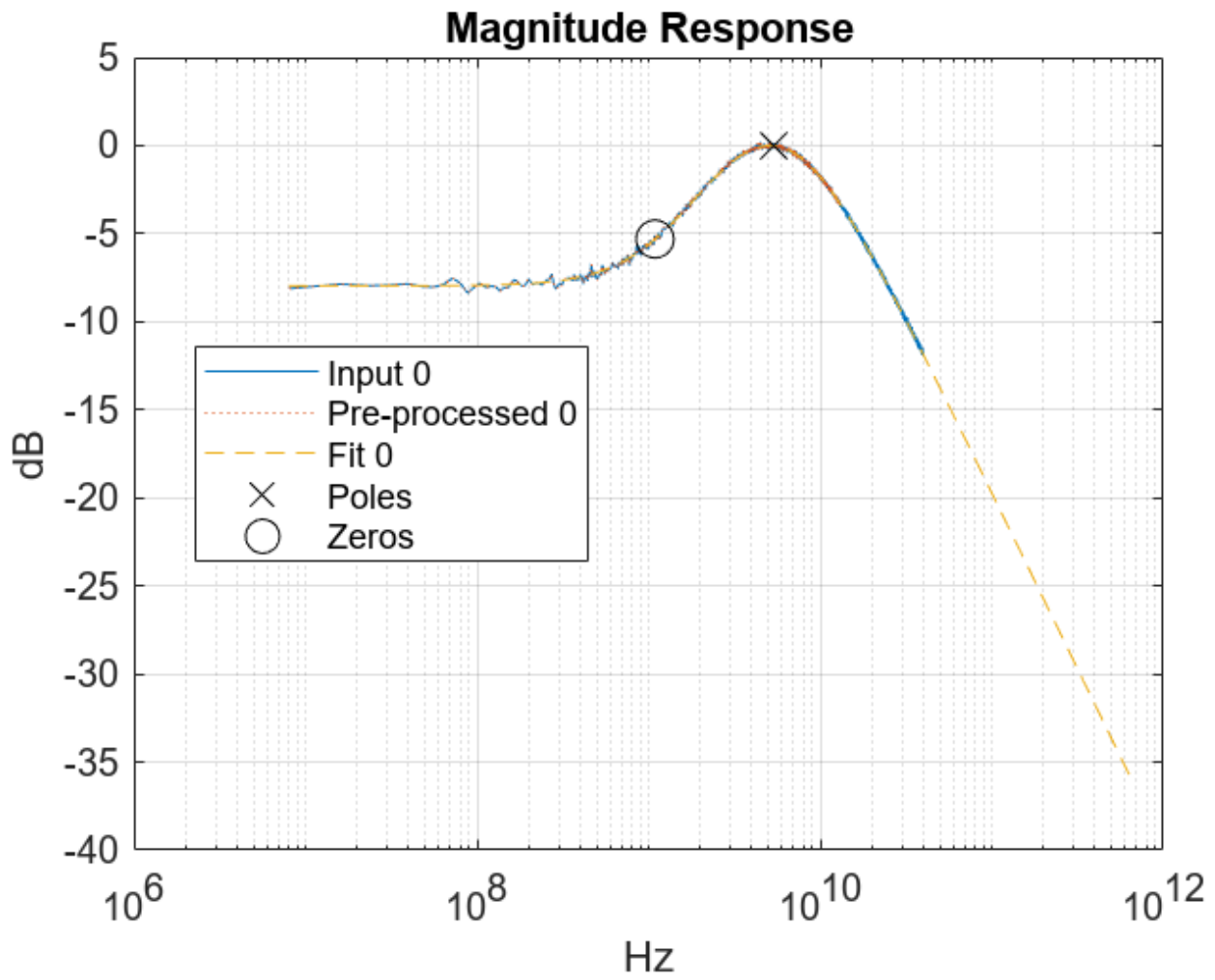
```
figure,
plotPulse(obj,ConfigSelect,SymbolTime)
```

Pulse Response, Symbol Time = 100 ps
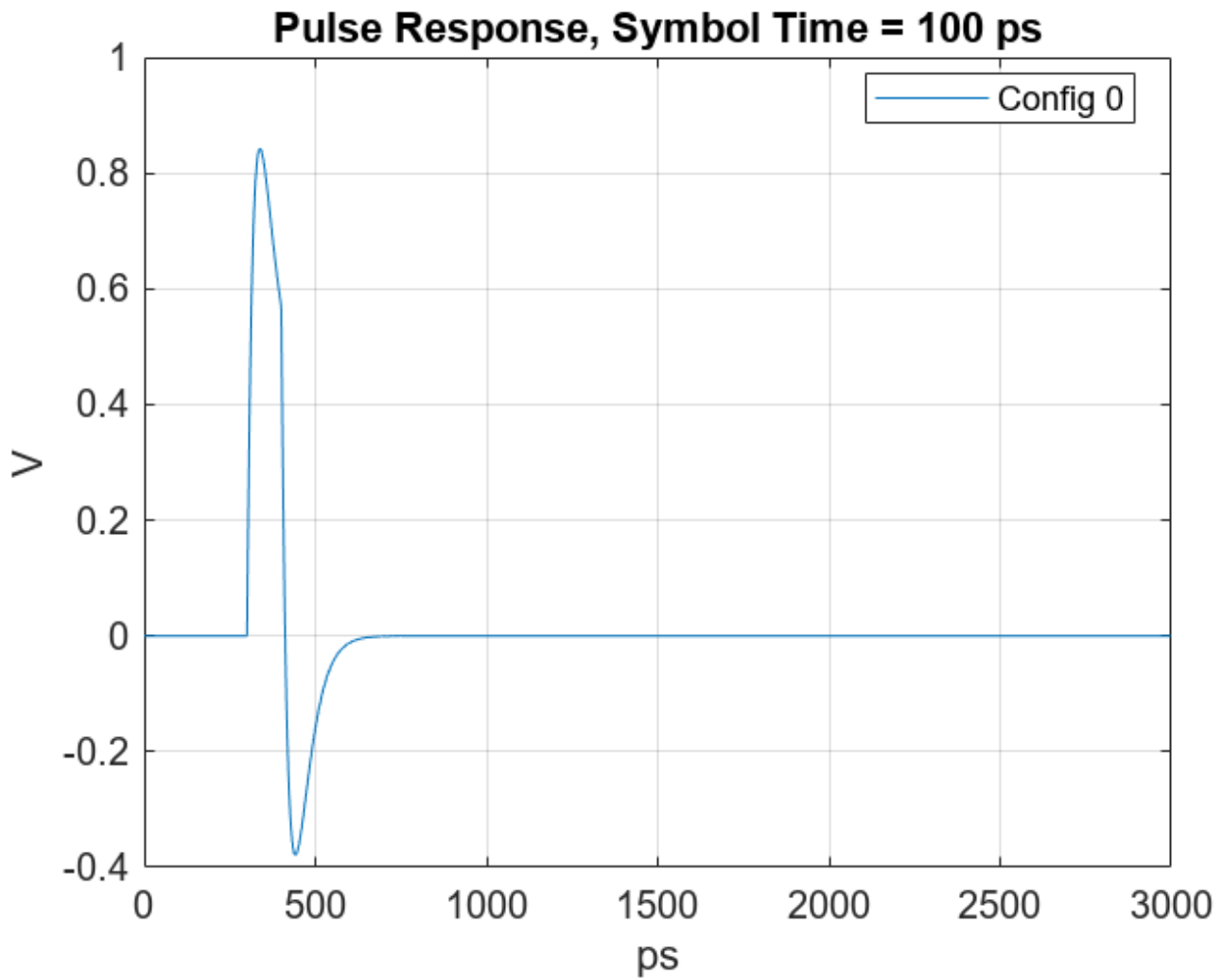

```
figure,
plotError(obj,ConfigSelect)
```

Fit Error = dB(Data - Fit)
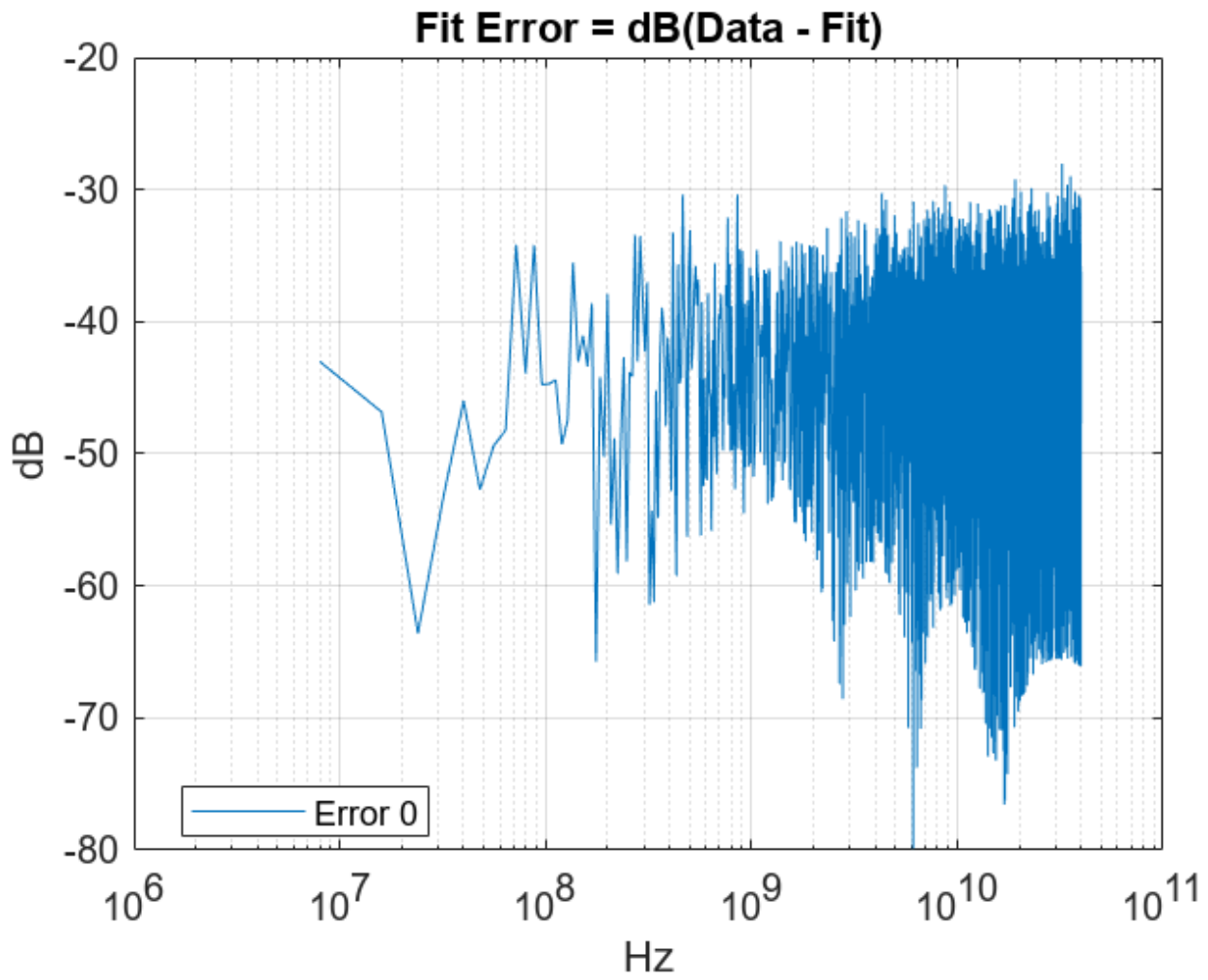
```
figure,
plotFitMetric(obj)
```
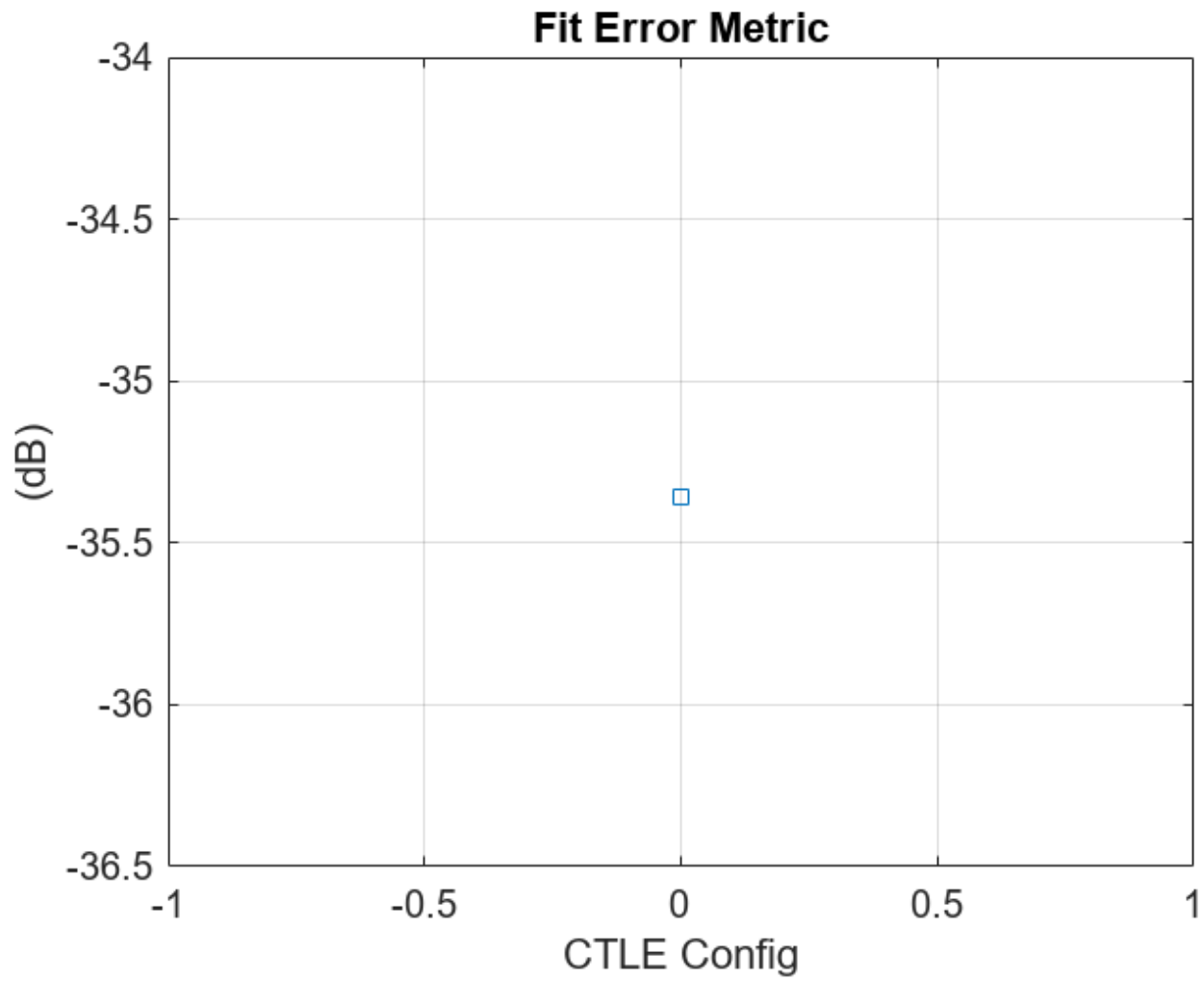
```
figure,
plotPoleZero(obj,ConfigSelect,SymbolTime)
```

Pole/Zero Plot
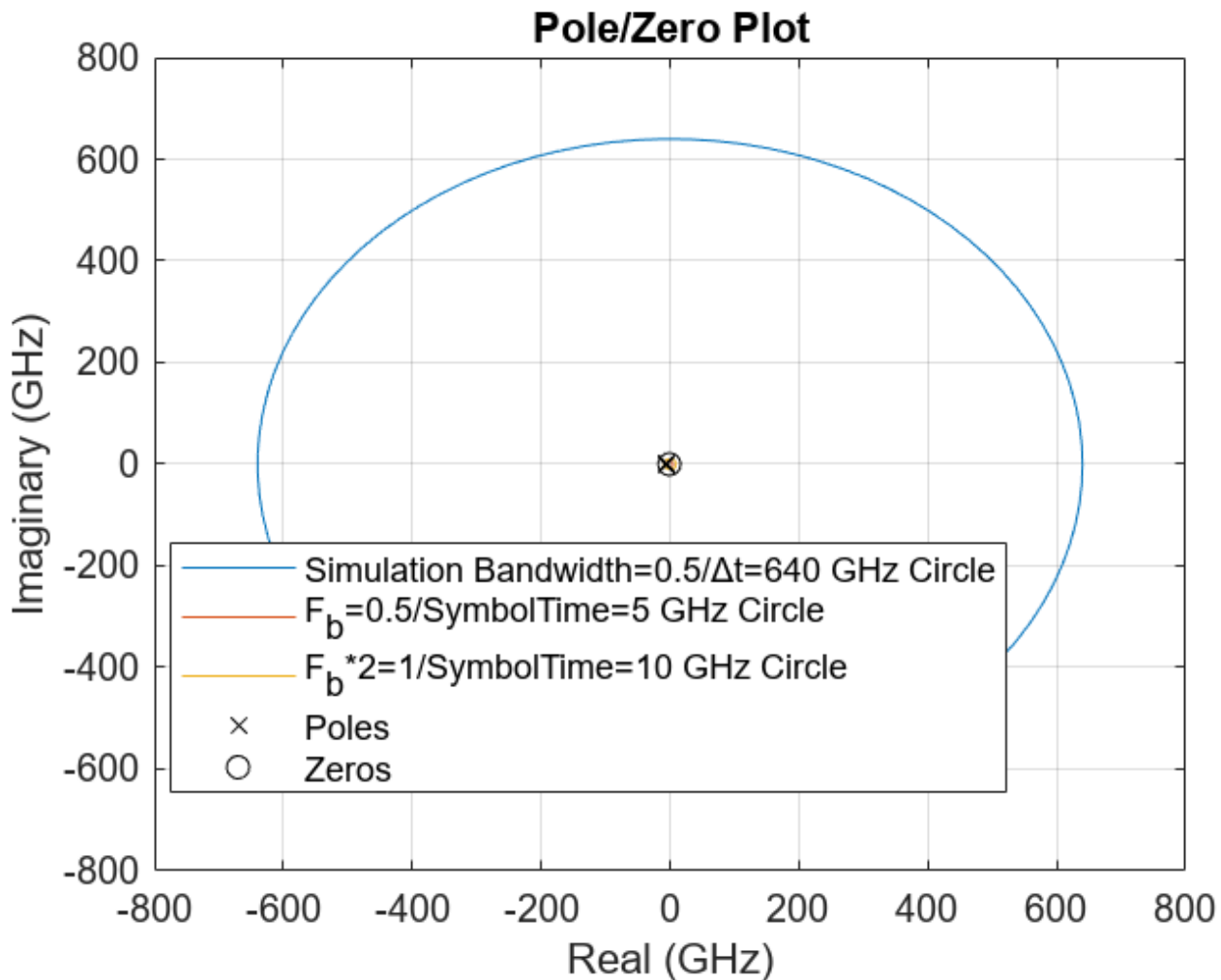

```
report(obj,'All');
```

```
15-Jul-2022 12:36:23.11
CTLE with 1 Configurations
Fit response with a maximum of 2 poles

For ConfigSelect = 0
Fit error = -35.361 dB
Gain: -7.96275 V/V or 18.0213 dB
Zeros:
    -1.09021 GHz = | -1.09021 + 0i |*1e9
Poles:
    -5.31435 GHz = | -5.2918 + 0.489137i |*1e9
    -5.31435 GHz = | -5.2918 + -0.489137i |*1e9
```

Simulink SerDes Model with CTLE block

You can configure a Simulink SerDes Model with a CTLE block by opening the CTLE block parameters and click the "Launch CTLE Fitter App" button. You can follow the same steps outlined

4-15

above in the section Configure CTLE Block in the SerDes Designer App on page 4-3to configure the **CTLE Fitter** app and export a `GPZ Matrix` for use in a CTLE block.

In the Block Parameters of the CTLE, you can click the button to launch the `ctlefitter` app:
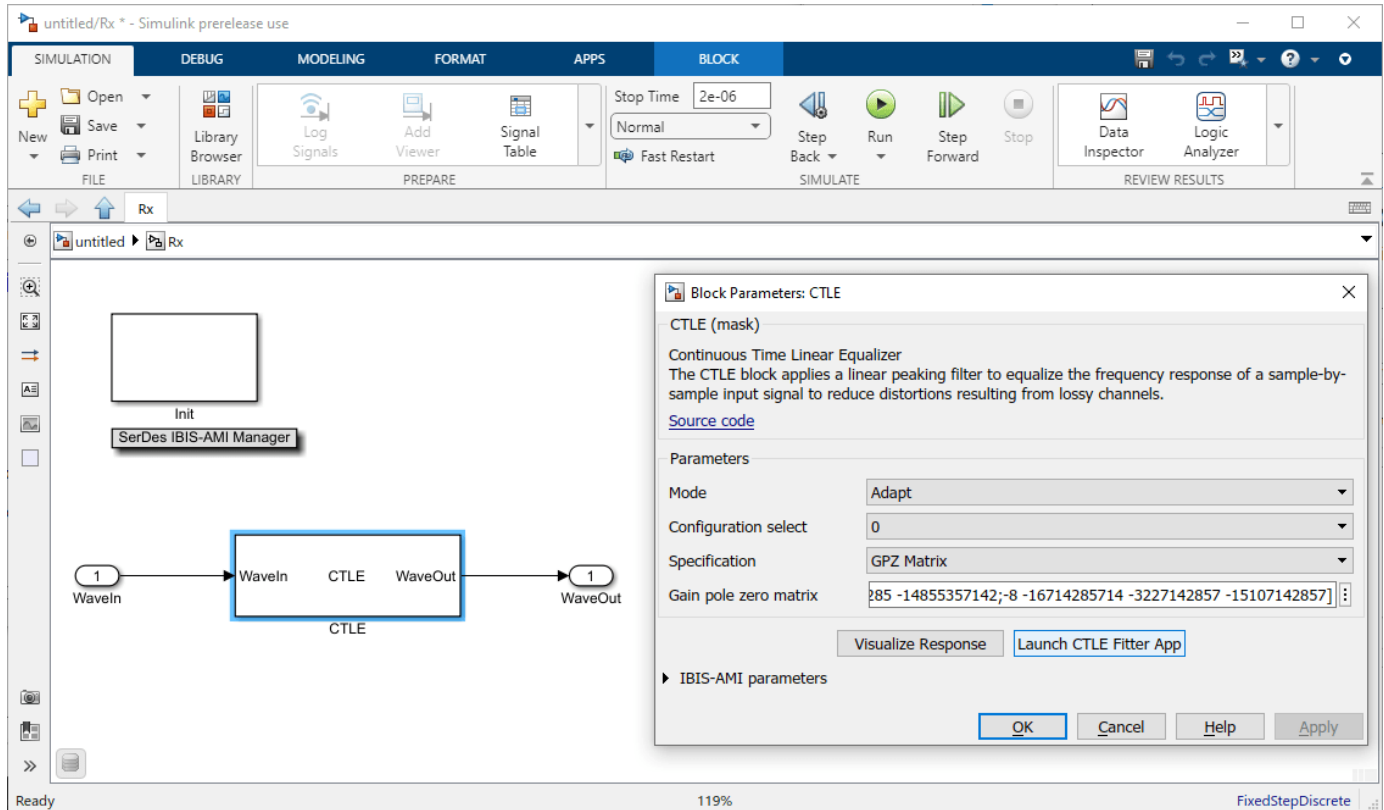
After you close the `ctlefitter` app, you will see the CTLE block is automatically configured to use the `GPZ Matrix` it created:

You can confirm the transfer function represented by the `GPZ Matrix` has a reasonable magnitude and phase response by clicking on "Visualize Response" button. These plots are also available in the SerDes Designer app workflow, and further detailed plots are provided as part of the exported script template.
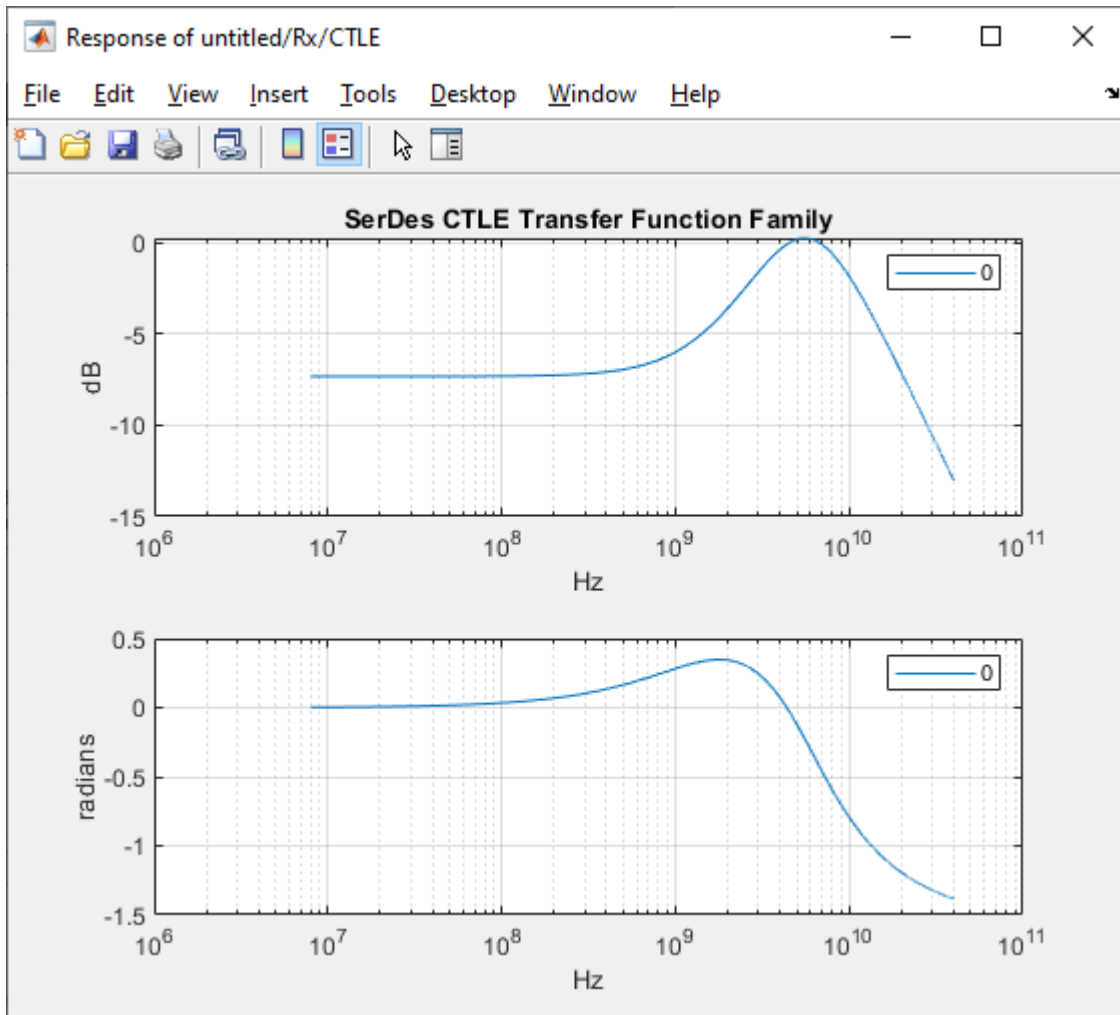
Standalone Mode

Open the CTLE fitter app from the MATLAB command window:

```
ctlefitter;
```
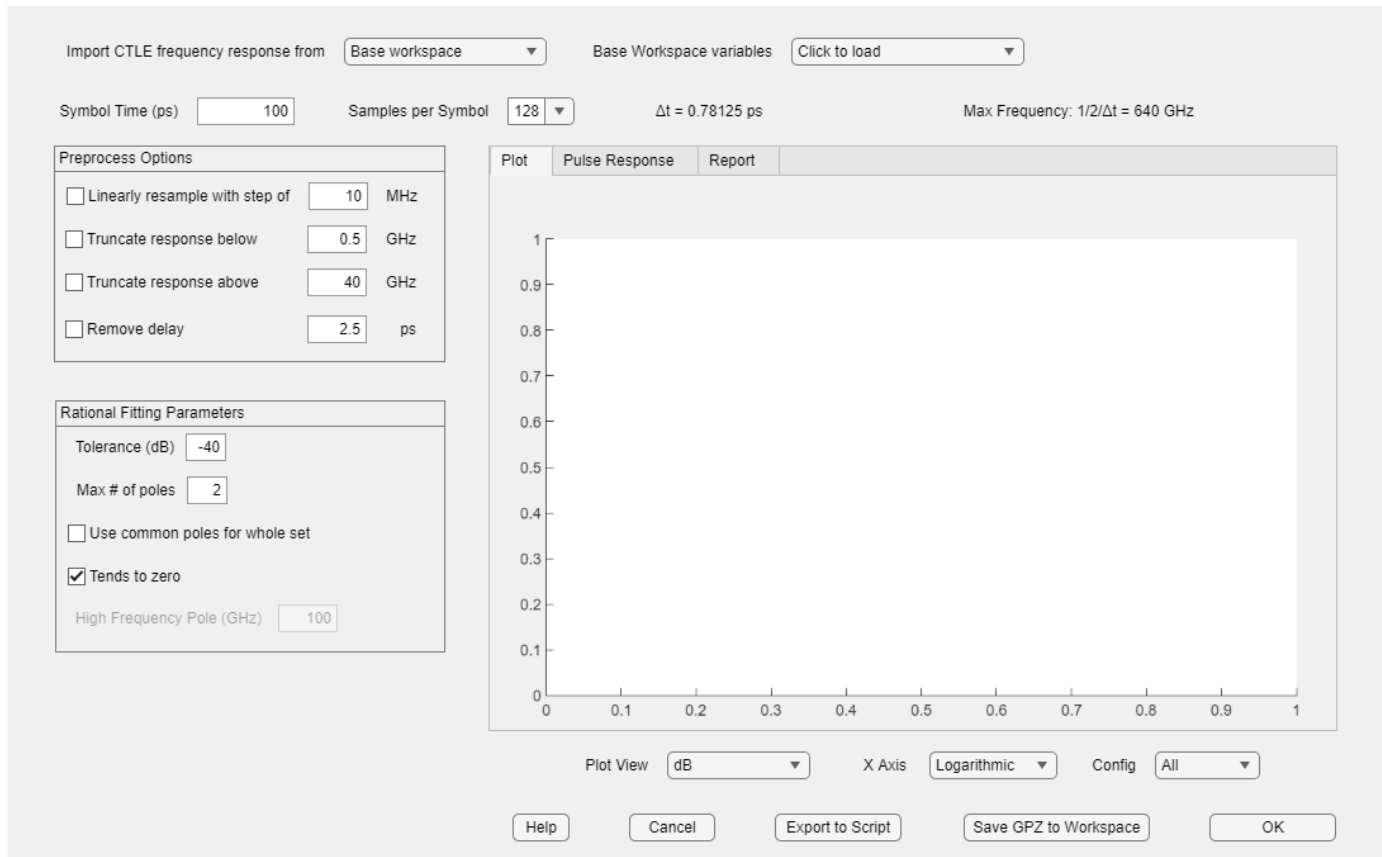
You can follow the same steps outlined above in the section Configure CTLE Block in the SerDes Designer App on page 4-3 to configure the **CTLE Fitter** app and export a `GPZ Matrix` to the base workspace in your MATLAB session.

Once you browse to and open a file containing one or more CTLE filter responses, you will see the app automatically updates the figure shown on the Plot tab:
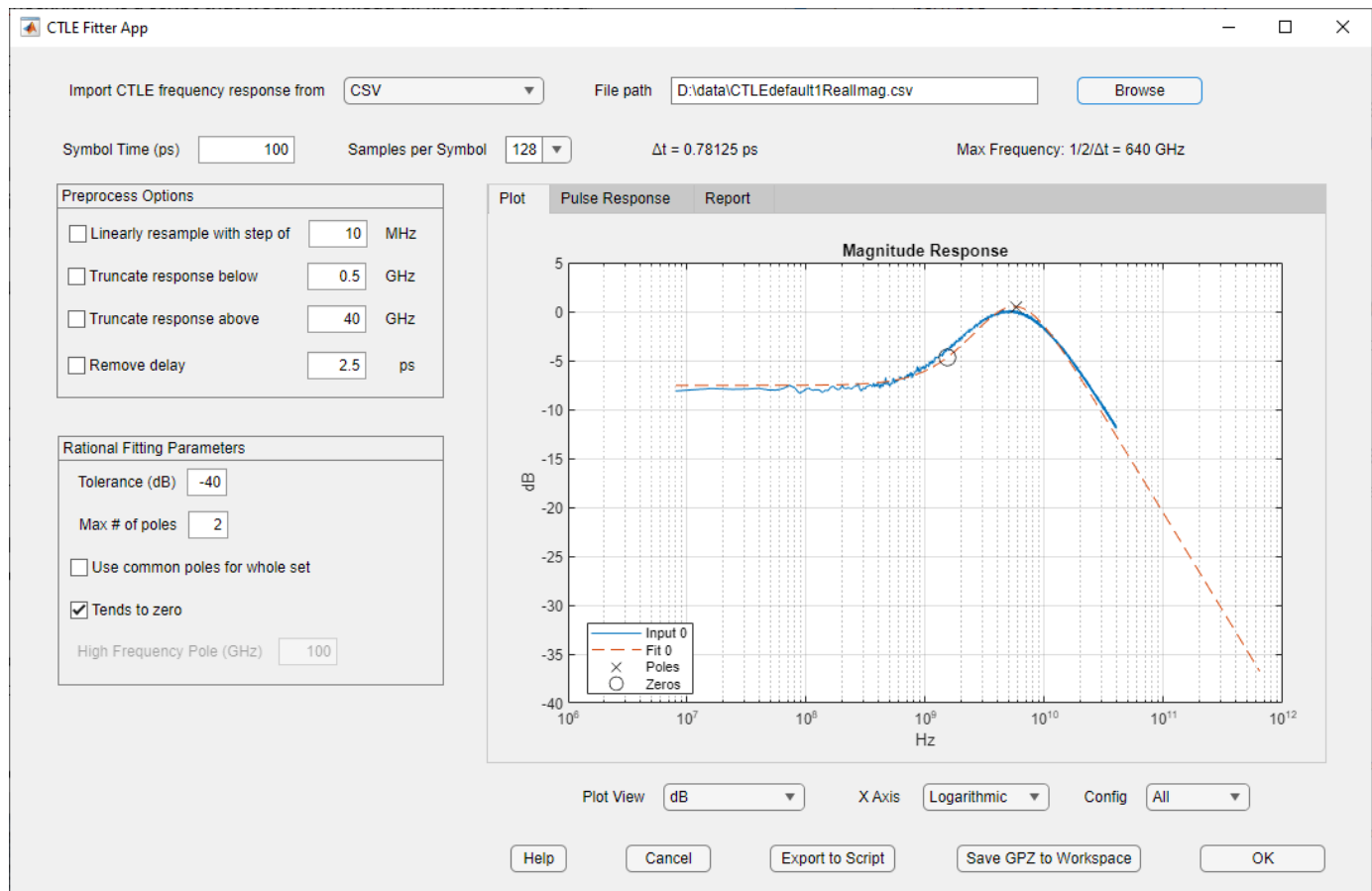
See Also

SerDes Designer | CTLE | `serdes.CTLE` | `rational`

More About

- "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance" on page 4-27

Convert Scattering Parameter to Impulse Response for SerDes System

This example shows how to find an Impulse Response by combining a Scattering-Parameter (S-Parameter) model of a baseband communication channel along with a transmitter and receiver represented by their analog characteristic impedance values. You will see how to find an Impulse Response of this network using the `sParameterFitter` app to create an `SParameterChannel` (hyperlink to sParameterChannel in doc) object in SerDes Toolbox™, which also uses some supporting functions from RF Toolbox™ such as `rational` (RF Toolbox) and `impulse` (RF Toolbox).

Configure Variables

The S-Parameter file representing the baseband channel should be a Touchstone 1.0 (.sNp) file. Typically these are extracted from a software EDA tool or laboratory VNA with a port reference impedance (50-Ohms is recommended). The following properties are the main settings you would use to extract an impulse response of the concatenated Transmitter-Channel-Receiver circuit network:

SParameterChannel Properties:

- FileName - Filename of the S-Parameter to be imported.
- PortOrder - Port order for the S-Parameter.
- MaxNumberOfPoles - Maximum number of poles to use for a fit output by the `rational` function.
- ErrorTolerance - Desired error tolerance in dB for a fit output by the `rational` function.
- SampleInterval - Sample interval in units of seconds.
- StopTime - Desired duration of the Impulse Response in units of seconds.
- TxR - Single-ended impedance value in Ohms of the analog TX IO structure.
- TxC - Capacitance value in Farads of the analog TX IO structure.
- TxAmplitude - Stimulus amplitude of the Tx output rising waveform.
- TxRiseTime - The 20-80% rise time of the Tx stimulus waveform.
- TxRTFactor - The conversion factor from 20-80% or 10-90% to 0-100% risetime, default is 20-80%.
- RxR - Single-ended resistance value in Ohms of the analog RX IO structure.
- RxC - Single-ended capacitance value in Farads of the Rx structure.
- Signalling - Specify signaling as 'single-ended' or 'differential'.
- AggressorDefinition - Method of acquiring aggressor behavior. For 'same-source' the stimulus is applied to victim input and probed at aggressor output and for 'same-load' (default) the stimulus is applied to each aggressor input and probed at the victim output. While the same load definition is more direct, the same source definition is used by methodologies that rely on time domain excitation (like HSPICE) to stimulate the system with a single pulse or step response. Consider the following 4-port single-ended system:
- Through-Ports: (1) -- (3)
- Through-Ports: (2) -- (4)
- The victim line is S31, the same-source crosstalk aggressor is S41 and the same-load crosstalk aggressor is S32.
- AutoDetectPortOrder - Boolean option to force auto-detect of port order.

Note: defaults are provided for all settings if no entries are made when calling `SParameterChannel`.

Create the S-Parameter Channel Object:

You create an `SParameterChannel` object by launching the `sParamterFitter` app from the base workspace.

```
sParameterFitter;
```
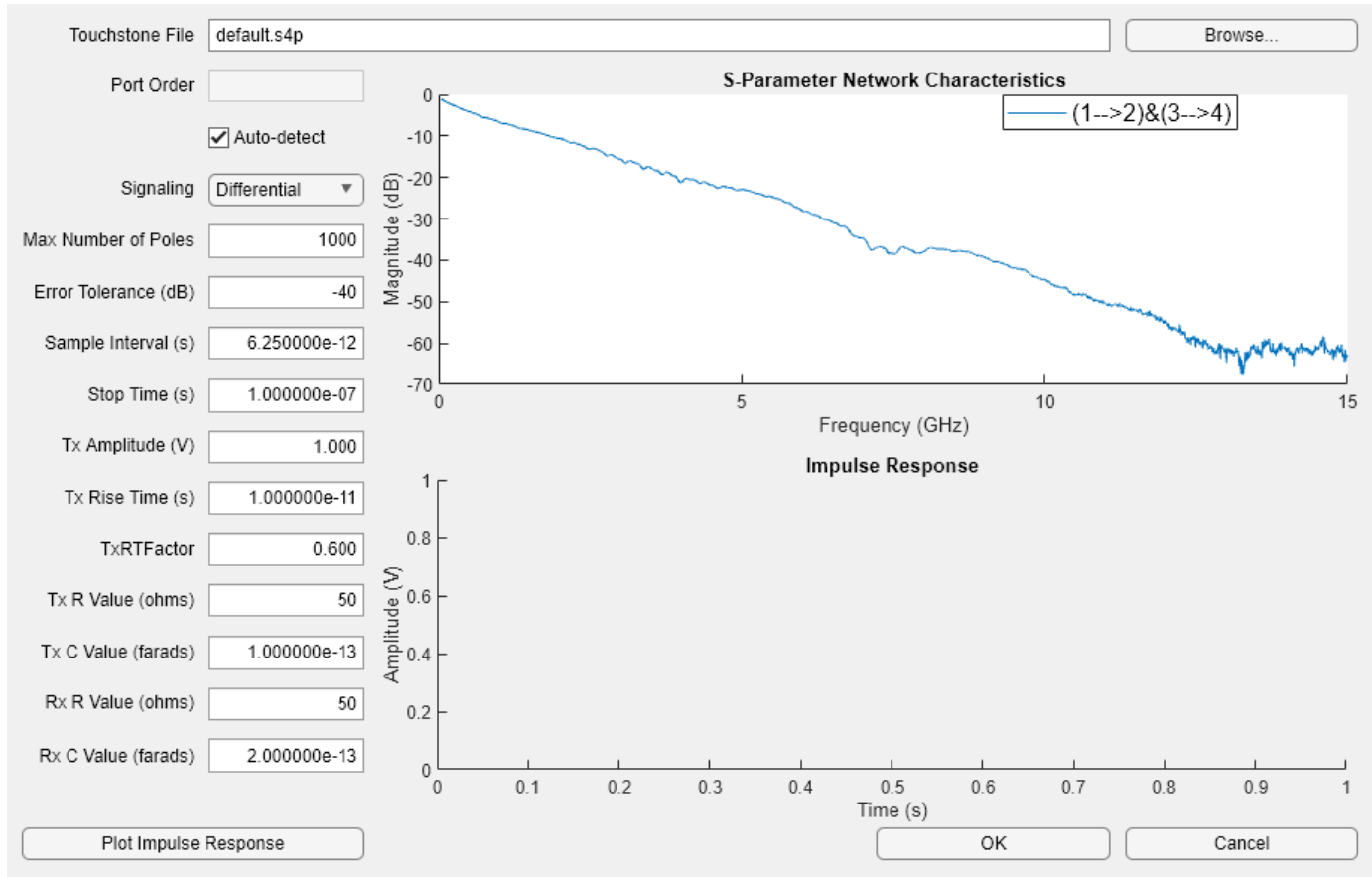

This will allow you to create an impulse response from a Touchstone S-Parameter data file. The app loads with default parameters for an `SParameterChannel` object. The equivalent command would be as follows:

```
obj = SParameterChannel('FileName','default.s4p',...
    'PortOrder', [1 2 3 4],...
    'MaxNumberOfPoles', 1000,...
    'ErrorTolerance', -40,...
    'SampleInterval', 6.25e-12,...
    'StopTime', 20e-9,...
    'TxR',50,...
    'TxC',0.1e-12,...
    'TxAmplitude', 1.0,...
    'TxRiseTime', 40e-12,...
    'TxRTFactor', 0.6,...
    'RxR',50,...
    'RxC',0.2e-12,...
    'Signaling','differential',...
    'AggressorDefinition','same-load');
```

It is important to also ensure `SampleInterval` and `StopTime` are set appropriately for the Impulse Response calculation (in this case, 6.25e-12 seconds and 20e-9 seconds, respectively), as well as the stimulus represented by `TxAmplitude` and `TxRiseTime`. Understanding S-Parameters is beyond the scope of this example, but it is important to remember the bandwidth of an S-Parameter must be sufficient to model a channel according to the Nyquist-Shannon sampling theorem.

Visualize a Plot of the Impulse Response:

You can plot the impulse response in the `sParamterFitter` app and export the object `sParameterFit` and the impulse response `sParameterFitImpulse` to the base workspace.
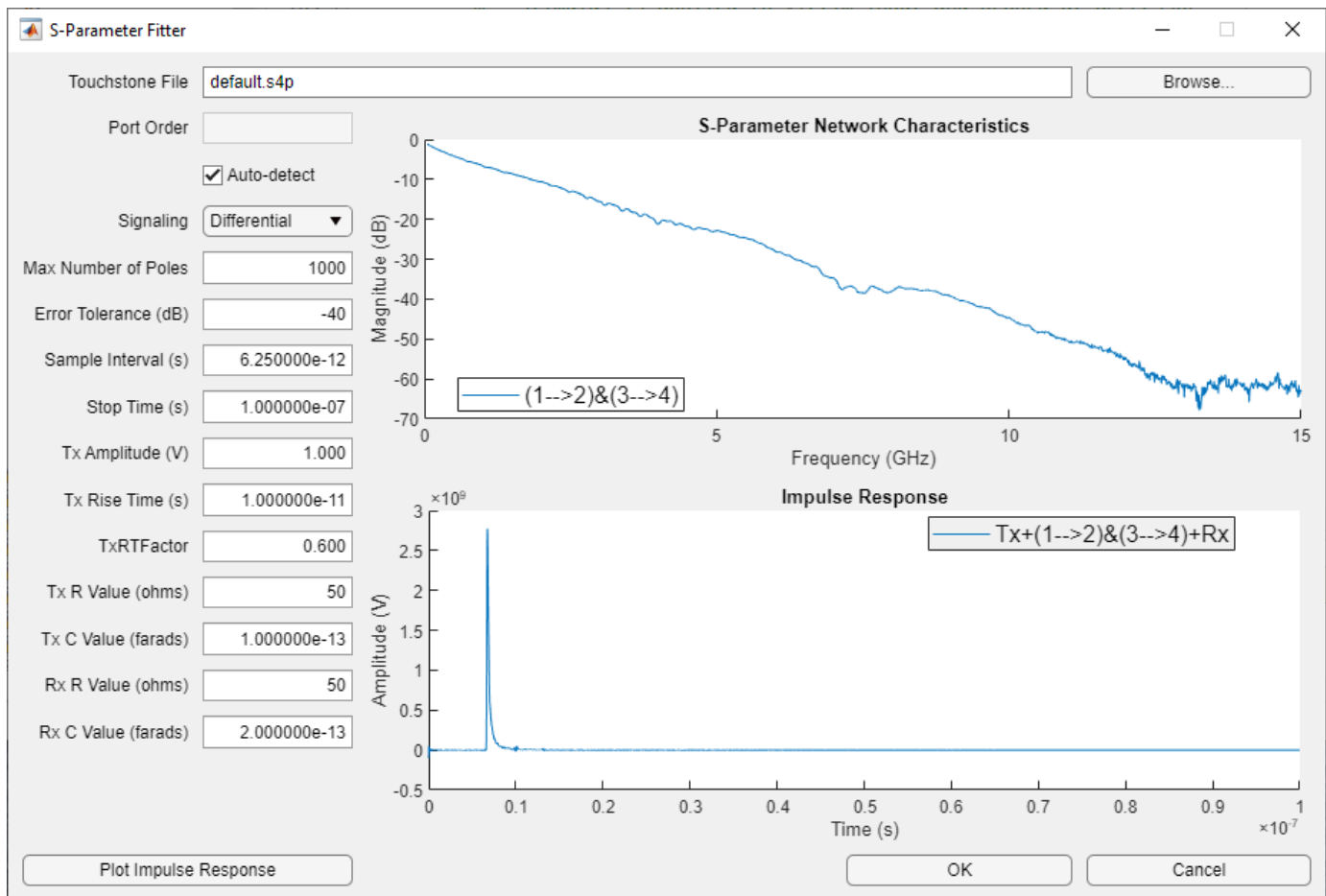
Use Impulse Response for Channel Model in Serdes Designer

You can use the impulse response of the baseband channel model within SerDes Designer by selecting "Impulse response" for channel model and enter the base workspace variable `sParameterFitImpulse` that you created with the `sParameterFitter` app.
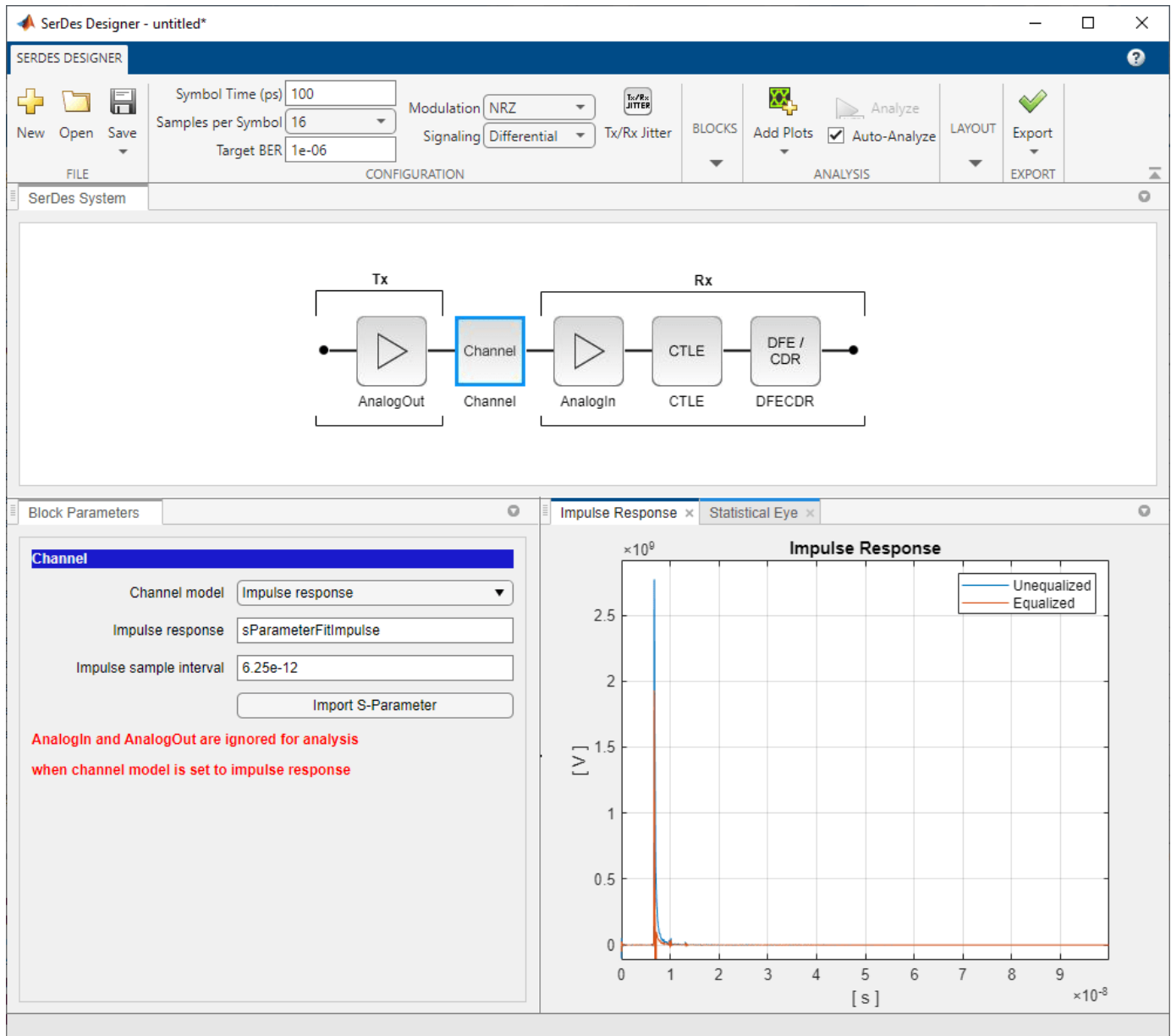
Simulate Channel Network with Transmitter and Receiver in Serdes Designer

You can use the impulse response of the baseband channel model within Serdes Designer in concert with TX and RX topologies to simulate an eye diagram.
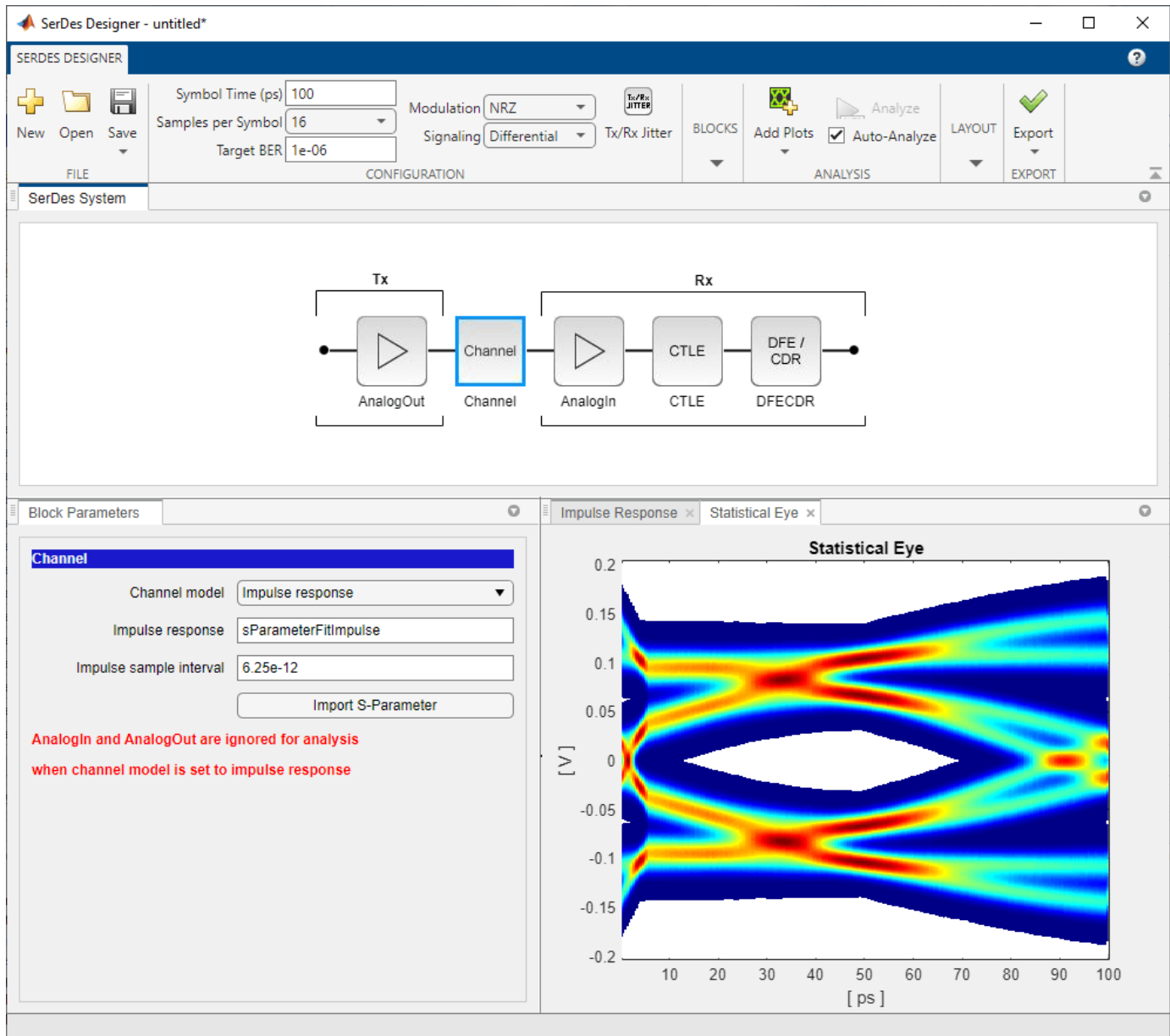
Impulse Response for Channel Model in Simulink

You can use the impulse response of the baseband channel model within Simulink by opening the **Analog Channel** block and clicking on the option "Impulse Response" under **Channel Model**. This will launch the `sParameterFitter` app.

Block Parameters: Analog Channel ✕

AnalogChannel (mask) (link)

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

Channel Model | Crosstalk

Channel model Impulse response ▼

Impulse response [zeros(1,63),1/SampleInterval,zeros(1,192)] ⋮

Impulse sample interval 6.25e-12 ⋮

Analog Model

Tx R (Ohms) 50 ⋮   Tx C (F) 100e-15 ⋮

Rx R (Ohms) 50 ⋮   Rx C (F) 200e-15 ⋮

Rise time (s) 10e-12 ⋮   Voltage (V) 1 ⋮

Plot Channel Responses   Import S-Parameter Touchstone File...

OK   Cancel   Help   Apply

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')
```
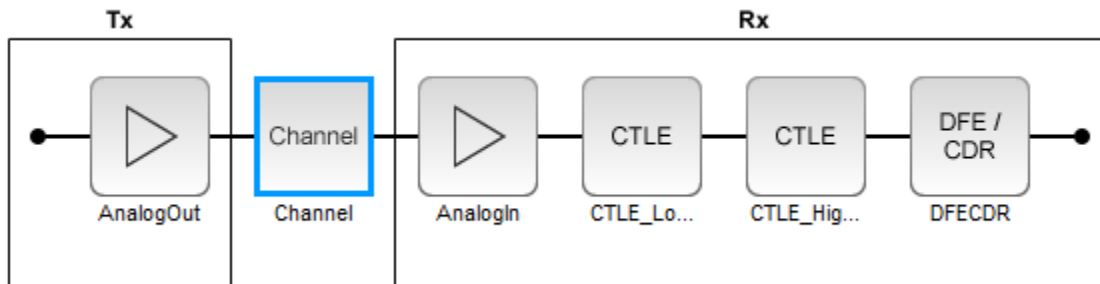

This project contains a receiver section with two CTLE blocks followed by a DFECDR block. In their default configuration, these blocks optimize individually. The goal of this example is to optimize the blocks as a system.

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. You can place code in the **Custom user code area** from the following steps and save the model. The code is broken down below in several subsections for easy comprehension.

Note: To complete the example, you can also reference the attached file `'rx_init_custom_user_code.m'` and place in the **Custom user code area** inside the Init

subsystem. For more information about Init subsystem, see "Statistical Analysis in SerDes Systems" on page 1-19.

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 the 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.

Note: The function `optPulseMetric` is designed to analyze a single response, not a matrix of responses, so you can use `sweepPulse(:,1)` to trim the main response from an impulse matrix or pulse matrix.

```
            % Convert impulse after DFE to pulse then calculate eye metrics
            sweepPulse = impulse2pulse(sweepImpulse,SPB,SampleInterval);
            eyeMetric = optPulseMetric(sweepPulse(:,1),SPB,SampleInterval,1e-6);
            % Select eye metric to evaluate results
            sweepMetric = eyeMetric.maxMeanEyeHeight;
%              sweepMetric = eyeMetric.maxEyeHeight;
%              sweepMetric = eyeMetric.maxCOM;
%              sweepMetric = eyeMetric.centerMeanEyeHeight;
%              sweepMetric = eyeMetric.centerEyeHeight;
%              sweepMetric = eyeMetric.centerCOM;
```

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.
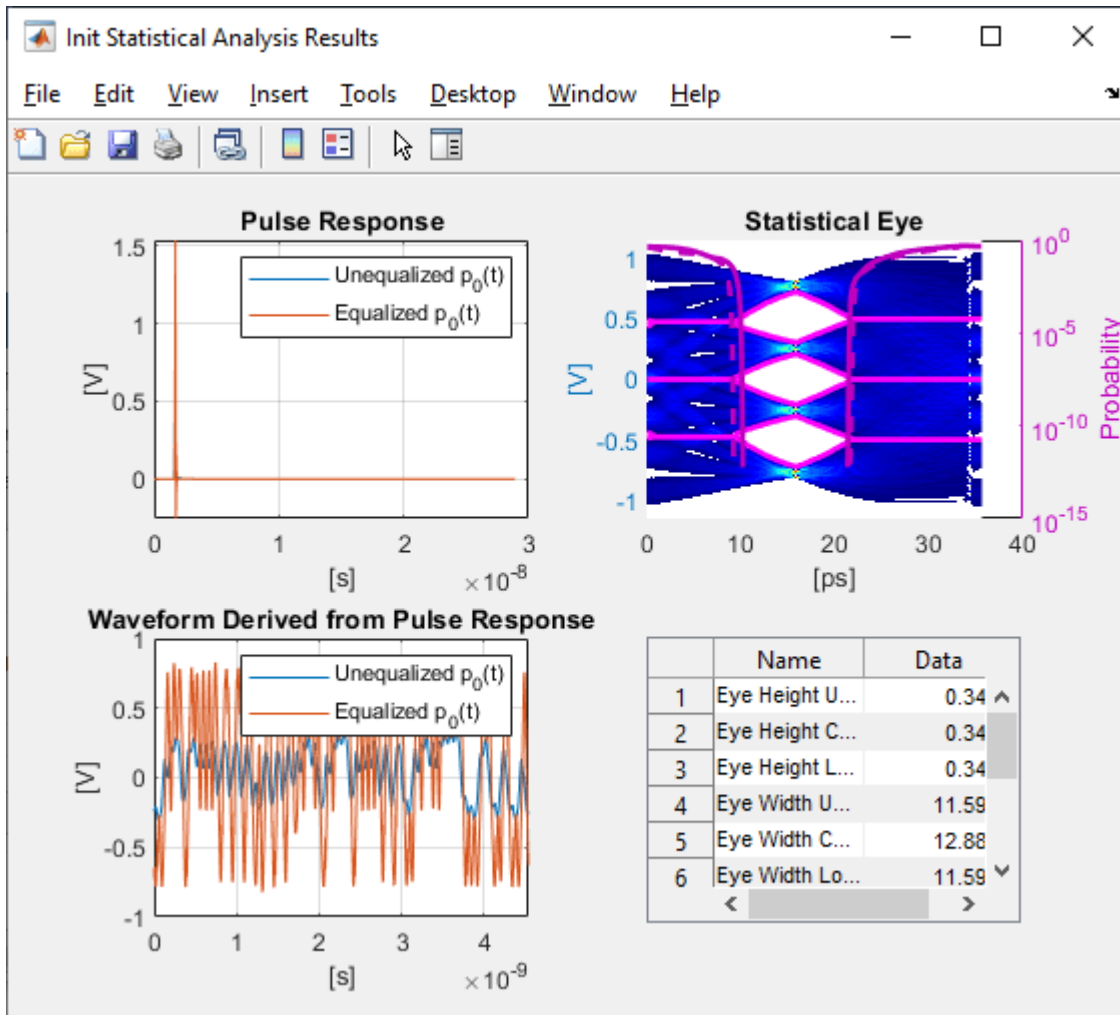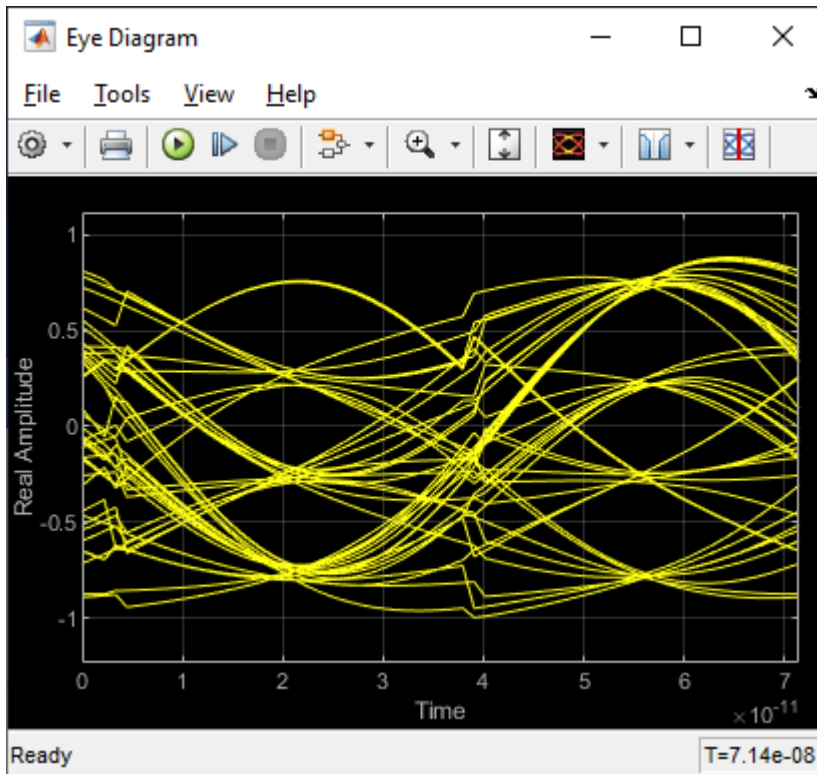
See Also
optPulseMetric | DFECDR | CTLE

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 5-28
- "Statistical Analysis in SerDes Systems" on page 1-19

Globally Adapt Receiver Components in Time Domain

This example shows how to perform optimization of a set of receiver components as a system during Time Domain (GetWave) Simulation. You will see how to setup a CTLE and a DFECDR Block so their settings adapt together globally during simulation. This is a follow on to the example "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance."

Receiver Component Global Adaptation Overview:

The receiver components for CTLE and DFECDR can work together to perform adaptation in Time Domain simulation. Normally, they operate independently as follows:

- CTLE adapts in Statistical (Init), then sets to this value when Time Domain simulation begins
- DFECDR adapts in Statistical (Init), then sets to these tap values for Time Domain and the Block proceeds to continuously train tap values

You can follow these steps to customize the CTLE and DFECDR to share signals within the RX system to adapt together globally during Time Domain simulation:

Part 1: Determine A Method for Optimizing RX Waveform vs. Equalization

You will see how equalization can affect RX waveforms to be either over-equalized, under-equalized, or critically-equalized (e.g. similar to how filter responses can be defined as over-damped, under-damped, or critically-damped).

Note: The concept of data words (3 symbols per word) from Communications Theory is used in this example.

Low Frequency (LF) and a High Frequency (HF) data word are defined for this example as follows:

- A LF word retains same logical value across 3 symbol UI (e.g. 111 or 000) to represent non-changing bit-to-bit values within a word.
- A HF word changes during the 3 symbol UI (e.g. 101 or 010) to represent changing bit-to-bit values within a word.

Note: a CTLE block optimizes for inner-eye (HF content only).

Part 2: Customize Simulink Blocks in Receiver Section

- Disable CTLE internal adaptation by connecting output to a terminator
- Change CTLE input to use a signal from the DFECDR (which allows adaptation together globally)
- Customize the DFECDR by creating a MATLAB function block that evaluates Eye metrics, depending on Interior bus and CTLE parameters, then outputs a value to use for CTLE configuration.

Part 3: Implement Custom MATLAB Function to Adapt Equalization during Time Domain Simulation

- In the DFECDR, code the MATLAB function to operate on input signals during UI boundaries rather than on each sample interval
- Add conditional statements to compare Eye metrics between Low Frequency (LF) data words and High Frequency (HF) data words, to determine next best equalization value.

• The equalization value output by the MATLAB function is a Signal in Simulink. This means the CTLE block will use this as its input- so every time it changes, the RX waveform will be equalized with this new value during Time Domain simulation.

Note: Blocks within the RX system can share signals. This is also true within the TX system. However, no signals can be shared between RX and TX systems.

Part 1: Determine a Method for Optimizing Receiver Waveform vs. Equalization

Initialize SerDes System with CTLE and DFECDR in the Receiver

Open the system by typing serdesDesigner('TDadapt.mat'). You will see a system with a basic TX, 100-ohm channel with a loss of 16dB at 5GHz, and an RX containing a CTLE followed by a DFECDR. In this example, the CTLE is customized to use "fixed" mode (because this example shows how to programatically adapt by evaluating fixed values in a MATLAB function), and **specification** set to "DC Gain and Peaking Gain," and set "DC gain" to a range of 0 to -15 dB (in increments of -1dB), and set "Peaking gain" to a range of 0 to +15dB (in increments of +1dB) as shown below. Also note that the DFECDR should be set to "adapt" mode.
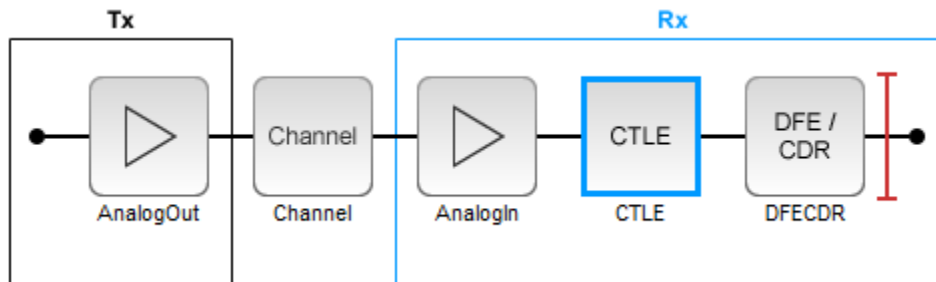
You can click on the CTLE and set the mode to "fixed."
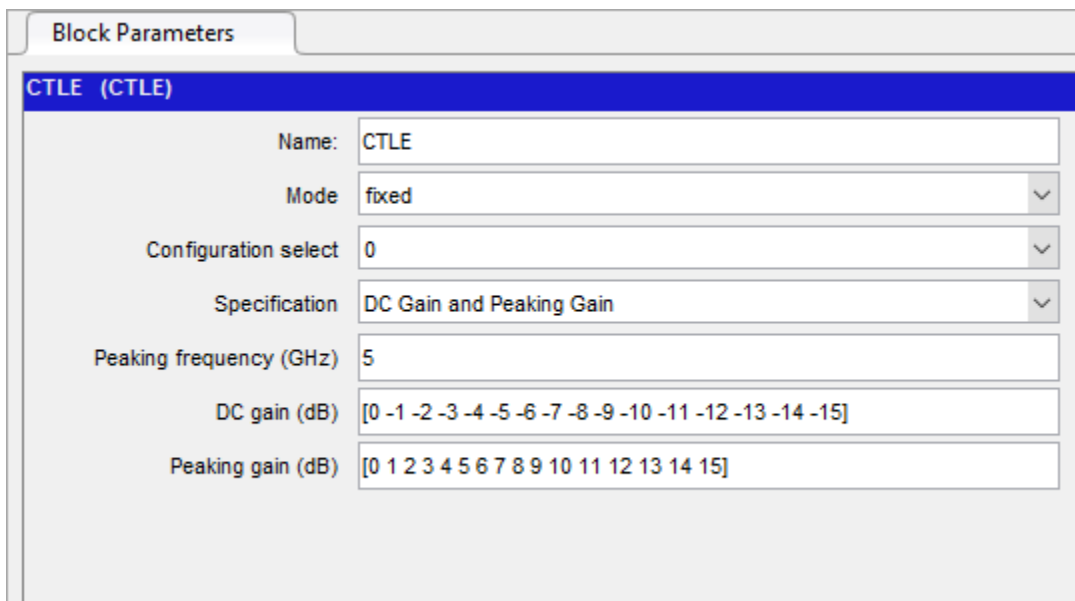
Then you can cycle through different values for **Configuration Select** for the CTLE and observe the effect of different equalization values on the receiver eye diagram:
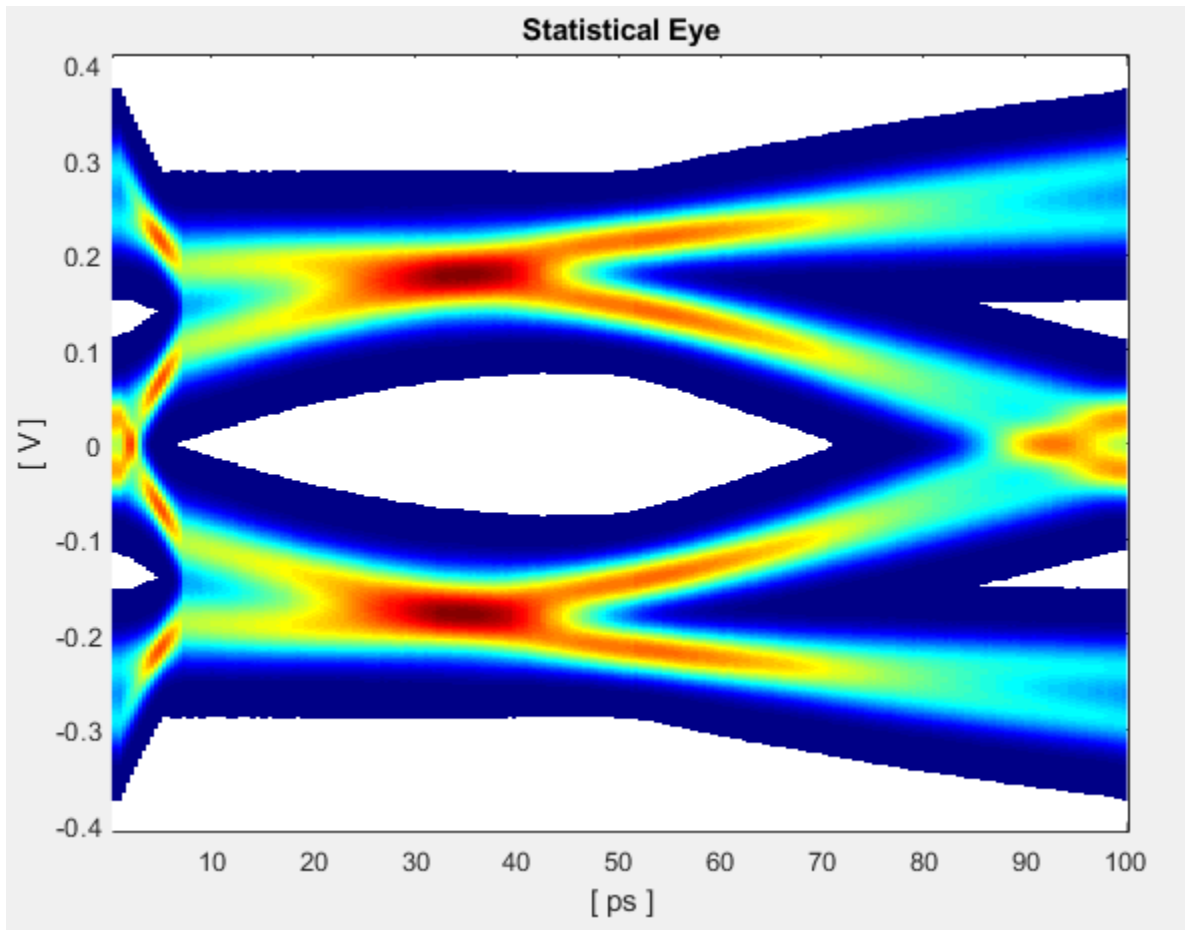
Figure Above: Under-equalized RX waveform: CTLE **Configuration Select** set to 0 (minimum).
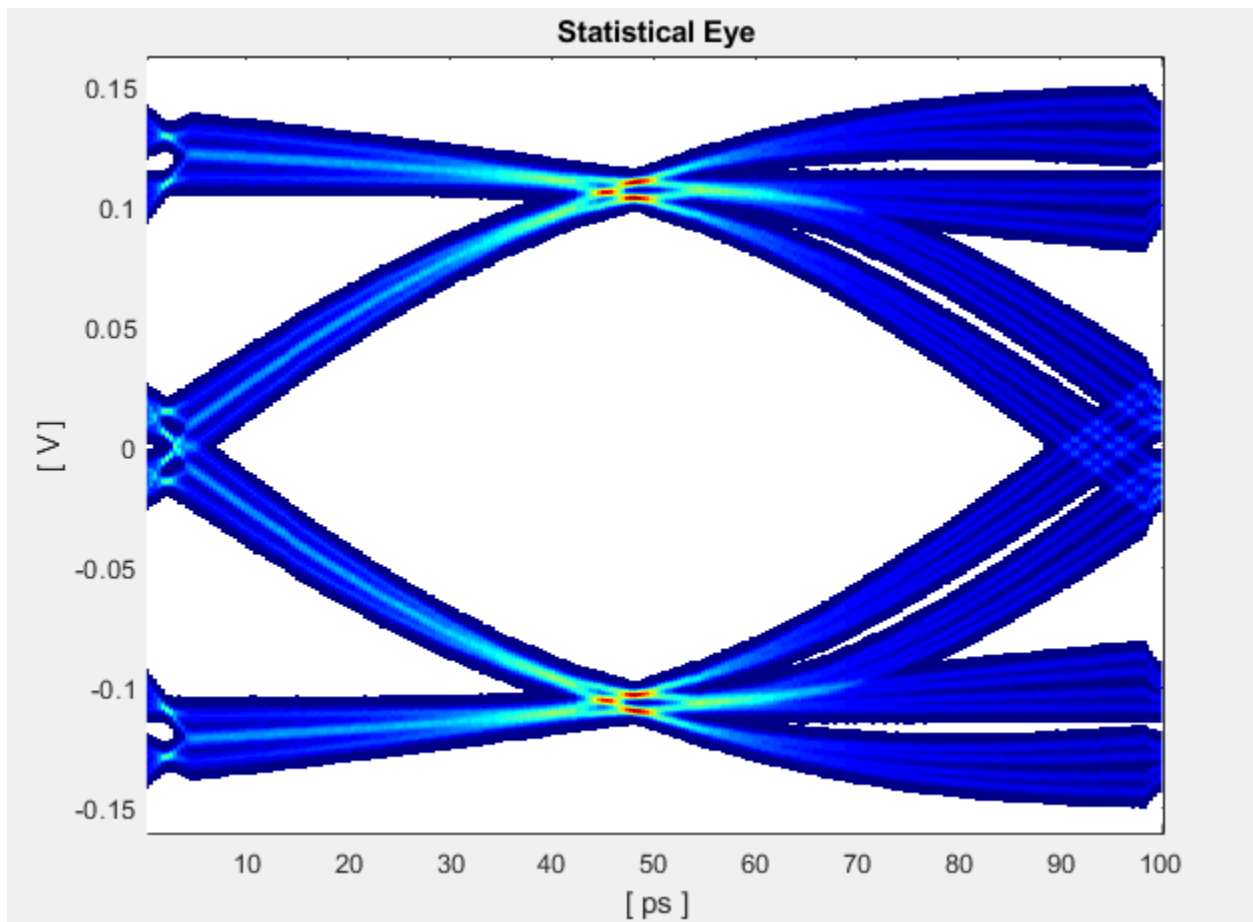
Figure Above: Over-equalized RX waveform: CTLE **Configuration Select** set to 15 (maximum).
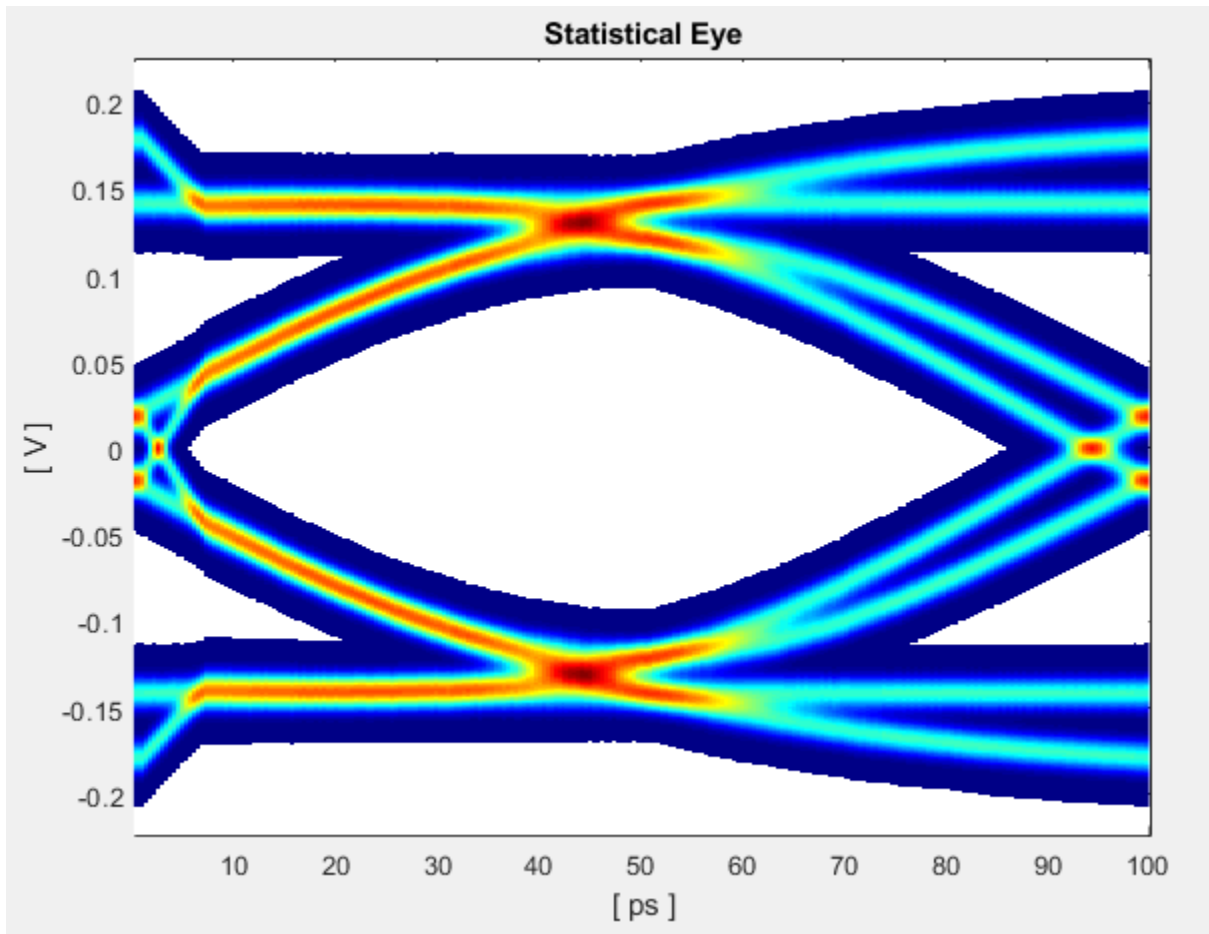
Figure Above: Critically-equalized signal: CTLE **Configuration Select** set to 7 (e.g. a midrange value).

You can develop an algorithm to optimize the receiver signal by using the concept of Equalization. For example, an RX signal can be considered as being over-equalized, under-equalized, or critically-equalized:

Part 2: Customize Simulink Blocks in Receiver System

Export the system to Simulink.
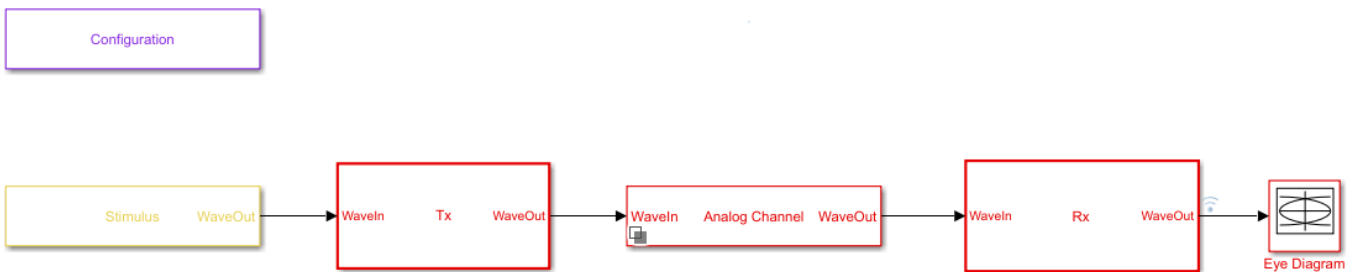
Inside the Receiver section, you can modify the CTLE and DFECDR to share values using Signals in order to enable global adaptation during Time Domain simulation.
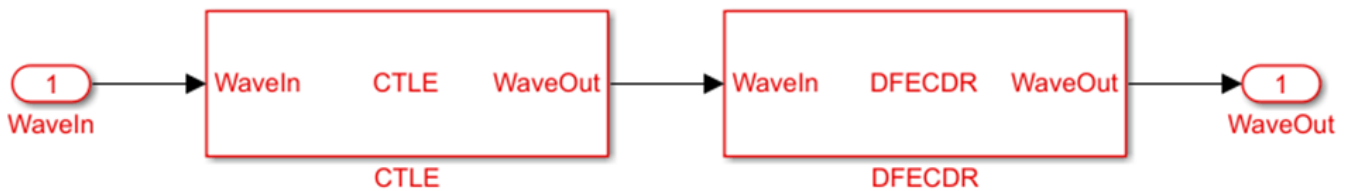
Modify the CTLE

You can look under the CTLE mask (CTRL-U), then change the **ConfigSelect** output to a **Terminator** instead of a DataStoreWrite. By placing a **Terminator** at the **ConfigSelect** output, the CTLE is no longer in feedback mode, and any other block in the RX system can take control of this CTLE by writing to its **ConfigSelect** input signal.
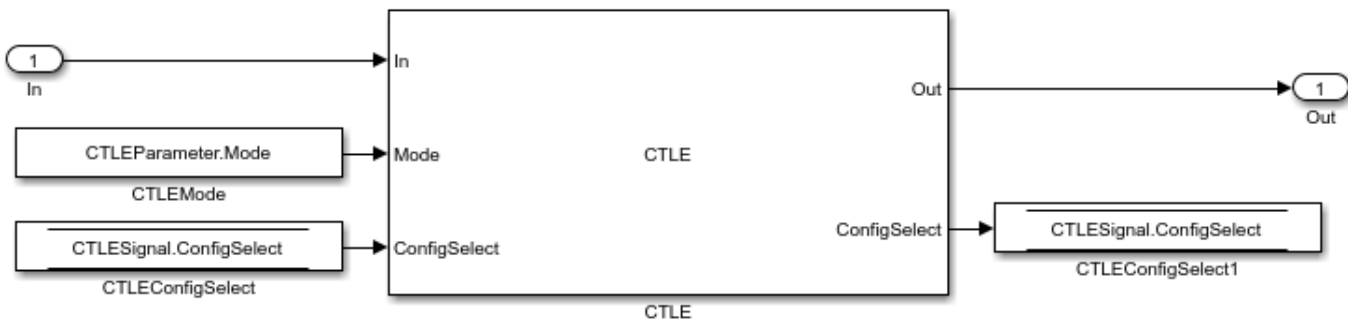
You will change the CTLE.ConfigSelect output to connect to a terminator:
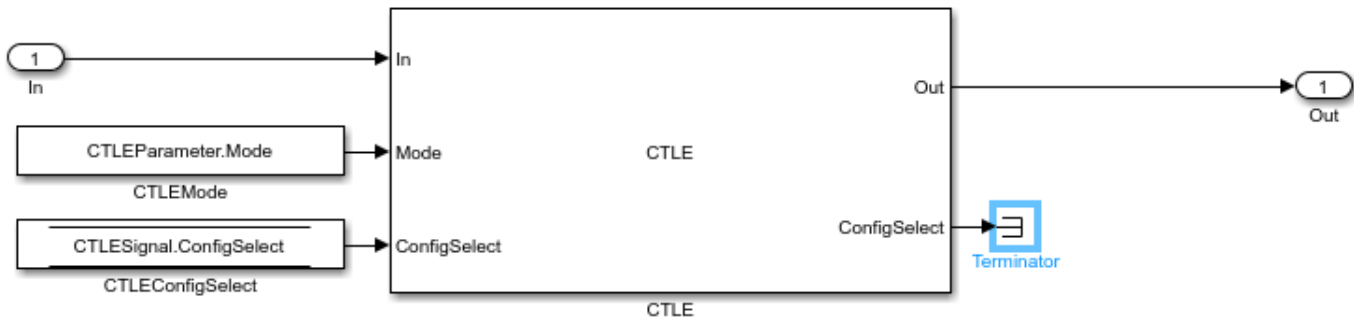
Add CTLE Adaptation to the DFECDR

You can now modify the DFECDR to control the value of **ConfigSelect** input used by the CTLE during Time Domain simulation. This can be accomplished by adding a MATLAB function that uses the following parameters to evaluate a CTLE configuration to adapt to the next best equalization value:

- Mode
- ConfigIn
- symbolRecovered
- voltageSample

Next, you will see how to change the DFECDR to add this MATLAB function.
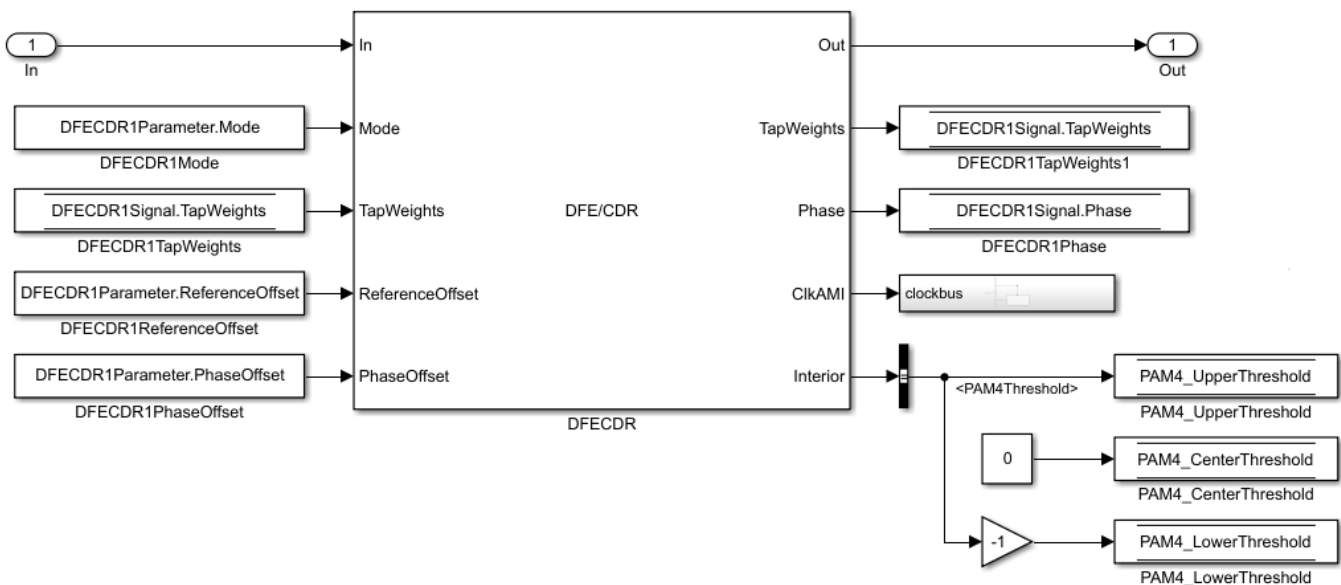
You will see a bus selector at the output port **Interior**. Double click to open its Block Parameters menu and make the following changes:
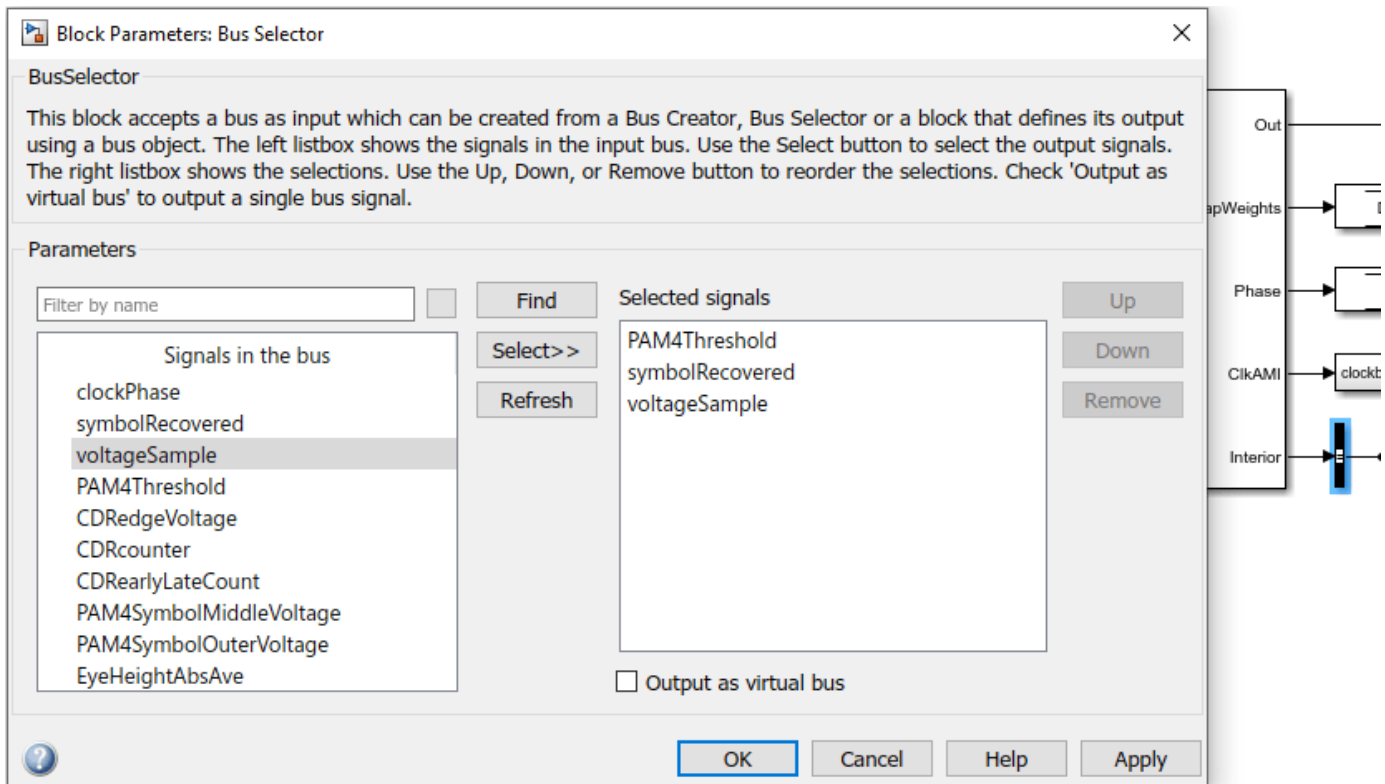
Click on signal **symbolRecovered** and click button marked "Select>>" and repeat this for **voltageSample**. These are optional outputs which is why you have to enable these. Also, you need to change the configuration of and **symbolTime** and **sampleInterval** from "signal" to "parameter." You can do this from the "edit data" toolstrip while in the function editor in MATLAB:
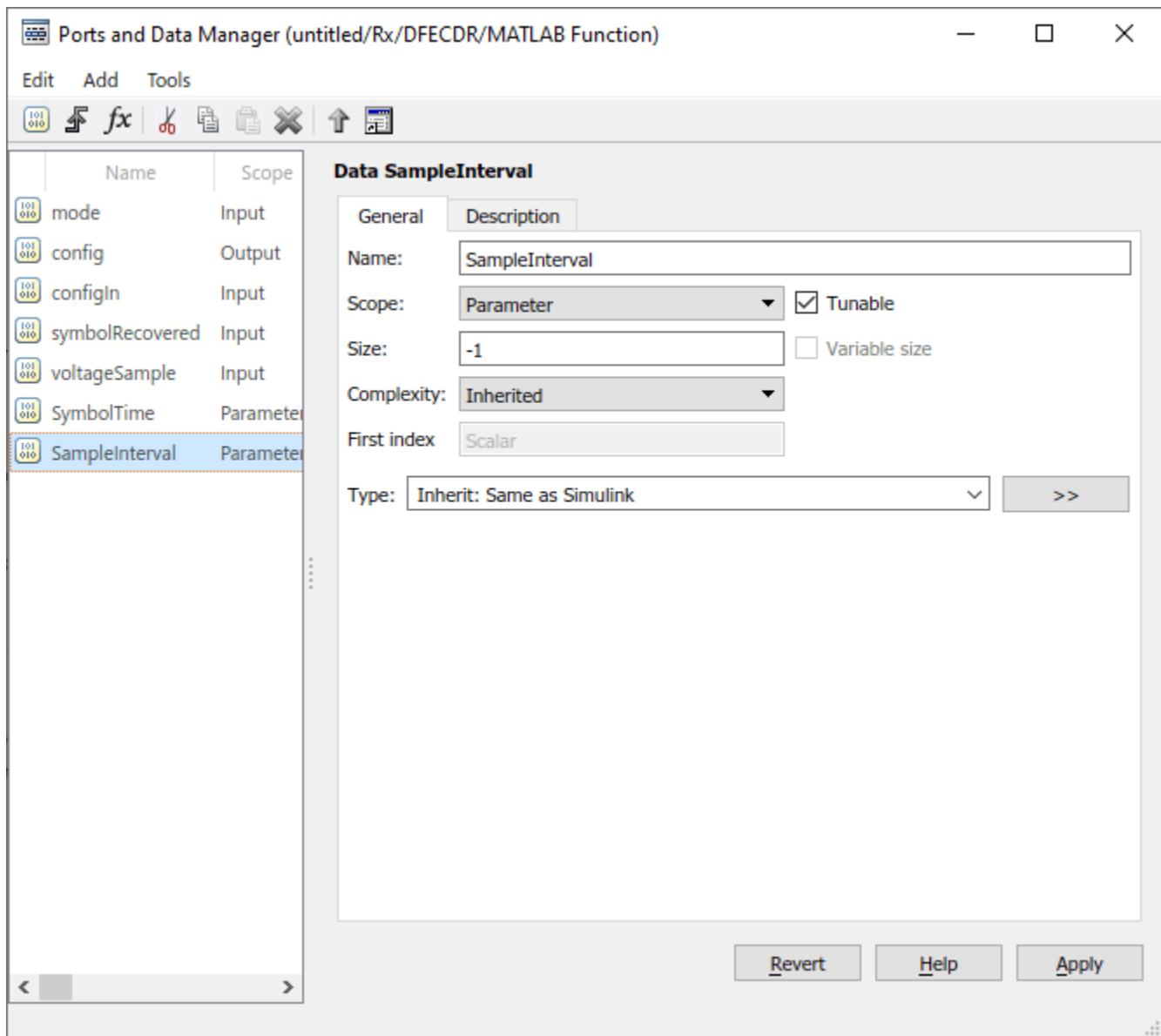
Add a MATLAB function to the Simulink canvas. You can get Simulink to automatically generate the ports by defining the function statement line of code as follows:

```
%  ctleTimeDomainAdapt - Simple adaptation algorithm to optimize a CTLE configuration
%                        in the time domain for NRZ signals.  Current serdes.CTLE
%                        adapts in Init/Statistical only.
%
%  Goal is to monitor the symbol decisions and voltage level of decisions from the
%  serdes.DFECDR.  From this, calculate the high and low frequency voltage averages to
%  adjust the CTLE config bringing the averages together. Once settled, detect toggle
%  of config and lock adapted configuration.  The config adjustment operates under the
%  assumption that the CTLE 'boost' increases with increasing config from 0 to X and
%  modulation is NRZ only.

%   Copyright 2020 The MathWorks, Inc.
```

```
function config = ctleTimeDomainAdapt(mode, configIn, symbolRecovered, voltageSample, SampleInter
```

Note: You can either use the code snippets concatenated as they are explained in this example, or you can use the attached MATLAB function "ctleTimeDomainAdapt.m" for reference.

Alternatively you can use the canvas tools in Simulink to create the ports. Using either method, the ports should appear as follows:

Inputs:

- mode
- configIn
- voltageSample
- symbolRecovered
- **Note**: **symbolRecovered** and **voltageSample** are optional outputs of the DFECDR block as shown in the Bus Selector above.

Output (same as function name):

- config

Create a constant block and configure its Element Assignment to **CTLESignal.Mode.** Then connect it to the function input port for mode.

Next, create a datastore read block, and configure the Element Assignment to **CTLESignal.ConfigSelect.** Then connect it to the function input port for configin.

Next, create a datastore write block, and configure the Element Assignment to **CTLESignal.ConfigSelect.** Then connect it to the function output port for config.
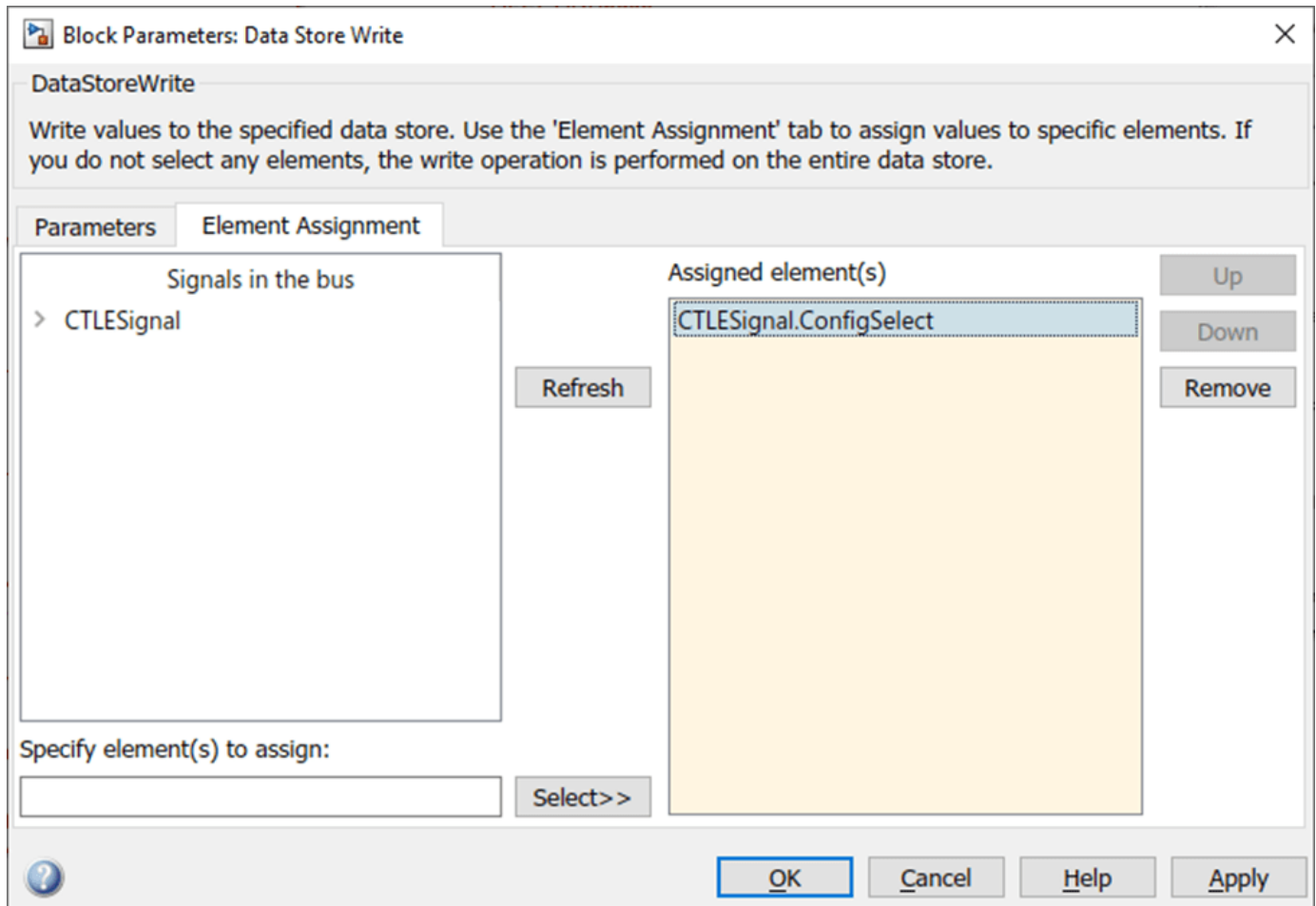
Note: you can add a Scope to observe the adapting values of **CTLESignal.ConfigSelect** during simulation.

When you are finished connecting the signals, the DFECDR will appear as follows:
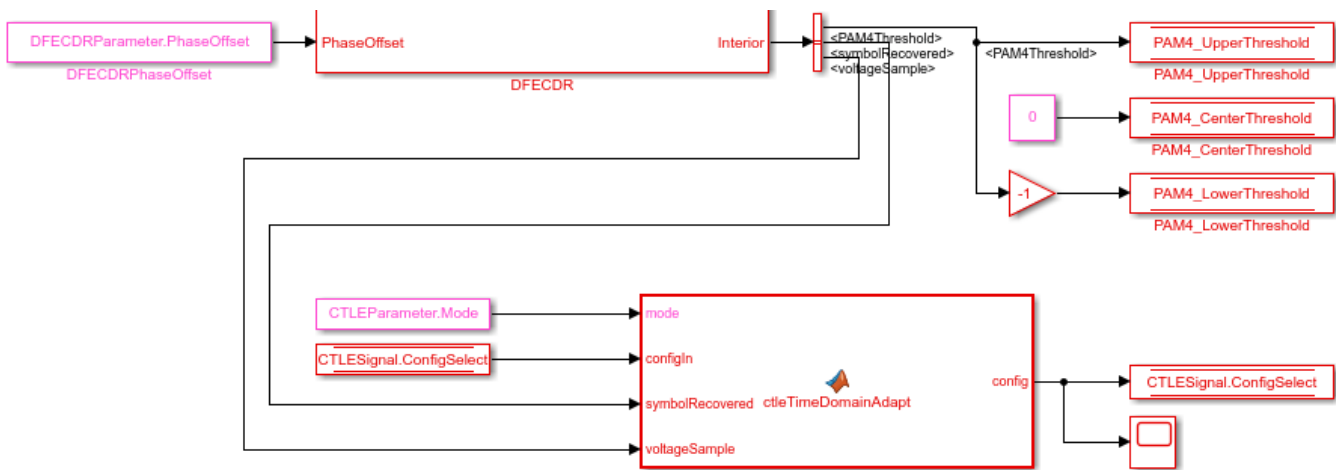
Part 3: Implement Algorithm to Adapt Equalization during Time Domain Simulation

You can edit the file "ctleTimeDomainAdapt.m" attached to this example as a starting point for your adaptation algorithm. This example makes use of Persistent variables to keep track of values each time the MATLAB function is called. As a starting point, you will evaluate if the variables are non-zero (e.g. using function isempty()), so that when the first time the function is called, they can be initialized. After this point, their values will be configured by the CTLE and DFECDR blocks working together.

```
    persistent sps sampleCounter symbolCounter
    persistent internalConfig updateConfig symbols voltages
    persistent lowFreqCount highFreqCount lowFreqVoltage highFreqVoltage
    persistent preventToggle toggling

    if isempty(sps)
        sps = SymbolTime/SampleInterval;
        sampleCounter = 0; % Total samples
        symbolCounter = 0; % Total symbols
        internalConfig = configIn; % Take config from Init and set for initial config
%         internalConfig = 0;  % Use this instead to ignore value from Init
        updateConfig = false;
        symbols = [0 0 0]; % Symbol history (3)
        voltages = [0 0 0]; % Voltage at each symbol (3)
        lowFreqCount = 0; % Low frequency event count
        highFreqCount = 0; % High frequency event count
        lowFreqVoltage = 0; % Voltage sum at low frequency events
        highFreqVoltage = 0; % Voltage sum at high frequency events
        preventToggle = [0 0 0 0]; % Toggle tracker; last 4 config updates -1/+1
        toggling = false; % Toggle detected flag
    end
```

Note: When a variable is Persistent, that variable retains its value. Otherwise they are instantiated as undefined for each time a MATLAB function is called.

Implement Watchdogs such as a Toggle Detector

You can implement many types of watchdog testing, but this example implements a toggle detector. If the CTLE is at a given value and begins to increment or decrement by 1 (e.g. 4-5, 5-4, 4-5) from word to word, the program will test for a toggle condition by detecting 3 repetitions. If true, it exits the loop, thus the CTLE retains its trained optimal value.

Note: You can see an example implementation of such an algorithm in the attached file.

```
    if mode == 2 && ~toggling
```

Understanding Data Slicers

You can use the signals **symbolRecovered** and **voltageSample** to process the RX waveform. But first, it is important to understand Data Slicer operation:

1. Each time the clock time occurs at the start of a UI,

2. The DFECDR block applies its taps,

3. The data slicer is triggered at +0.5 UI later,

4. Then the tap value decision occurs for that bit.

A data slicer outputs both a symbol and a voltage. For example, if the data slicer operates at the 0.5 UI location, the slicer outputs a symbol as +0.5 or -0.5 and voltage value the symbol has reached.

Find UI Boundaries from Data Slicer

The system runs on a sample-based time step, so you can keep track of UI boundaries by setting up a sample and symbol counter. When a sample count is divisible by this setting for "samples per bit," this defines a symbol boundary. This way, you can find combinations of HF (e.g. 010, 101) or LF (e.g. 111, 000) data words to optimize for critical equalization:

```
% How often to update CTLE Config
updateFrequencySymbols = 1000;
% Range of CTLE configurations
minCTLEConfig = 0;
maxCTLEConfig = 15;
% Set up a sample and symbol counter to track overall progress
sampleCounter = sampleCounter+1; % Every call to this function is a sample
% When sample count is divisible by samples per bit, there is a symbol boundary
if mod(sampleCounter,sps) == 0
    symbolCounter = symbolCounter+1;
    updateConfig = true; % Flag to keep from looping in update section
    % Maintain bit/voltage history
    symbols = [symbols(2:3) symbolRecovered]; % -0.5 or 0.5
    voltages = [voltages(2:3) voltageSample]; % Voltages at each symbol
    % Keep count of low/high frequency events and sum voltages across those events
    % Low frequency = Steady high or low
    % High frequency = Rapid tranition
    if isequal(symbols, [0.5 0.5 0.5]) || isequal(symbols, [-0.5 -0.5 -0.5]) % 1 1 1 OR (
        lowFreqCount = lowFreqCount + 1;
        lowFreqVoltage = lowFreqVoltage+ abs(voltages(2)); % keep middle voltage sample
    elseif isequal(symbols, [-0.5 0.5 -0.5]) || isequal(symbols, [0.5 -0.5 0.5])% 0 1 0 (
        highFreqCount = highFreqCount + 1;
        highFreqVoltage = highFreqVoltage+ abs(voltages(2)); % keep middle voltage sample
    end
end
```

Find 3-Symbol Combinations to Sort HF vs. LF Signal Content

You can use MATLAB function Mod to find when the samples/s has reached Mod 0, which defines the UI boundary. Once the symbol counter has reached modulo 0, you can accumulate these locations as bits. Then after accumulating sufficient bits (e.g. 1000), calculate the average of the voltages across this population, and update the CTLE Configuration.

Create an if statement to perform the following test and decision:

*   If signal is 111 or 000, increment count variable for LF

*   If signal is 010 or 101, increment count variable for HF

*   For each case, take the voltage at that symbol and increment variable for voltage counter

```
% When symbol count is divisible by update frequency, check if CTLE update is needed
if mod(symbolCounter,updateFrequencySymbols) == 0  && updateConfig
    % Calculate low/high voltage average
    lowFreqAvg = lowFreqVoltage/lowFreqCount;
    highFreqAvg = highFreqVoltage/highFreqCount;
```

Update the CTLE

You can implement any algorithm you wish, but in this example the CTLE begins with configSelect value from Init, and the function performs an increment. Each time the DFECDR is evaluated and compared to a Persistent variable. Depending on this results, the CTLE is incremented or decremented.

Note: It is important for your code to test that CTLE is not set to an invalid **configSelect**.

```matlab
        % Increase CTLE config if low freq is above high freq
        if lowFreqAvg > highFreqAvg
            % Prevent exceeding maximum CTLE config
            if internalConfig < maxCTLEConfig
                % If toggle is detected, disable adaptation
                if ~isequal(preventToggle,[1 -1 1 -1])
                    internalConfig = internalConfig + 1;
                    % Add current action to toggle tracker
                    preventToggle = [preventToggle(2:4) 1];
                else
                    toggling = true;
                end
            end
        % Decrease CTLE config if high freq is above low freq
        elseif lowFreqAvg <= highFreqAvg
            % Prevent exceeding minimum CTLE config
            if internalConfig > minCTLEConfig
                if ~isequal(preventToggle,[-1 1 -1 1])
                    internalConfig = internalConfig - 1;
                    % Add current action to toggle tracker
                    preventToggle = [preventToggle(2:4) -1];
                else
                    toggling = true;
                end
            end
        end
        % Reset variables associated with averaging every updateFrequencySymbols
        lowFreqCount = 0;
        lowFreqVoltage = 0;
        highFreqCount = 0;
        highFreqVoltage = 0;
        updateConfig = false; % Lock updates until next symbol boundary
    end
end

config = internalConfig;
end
```
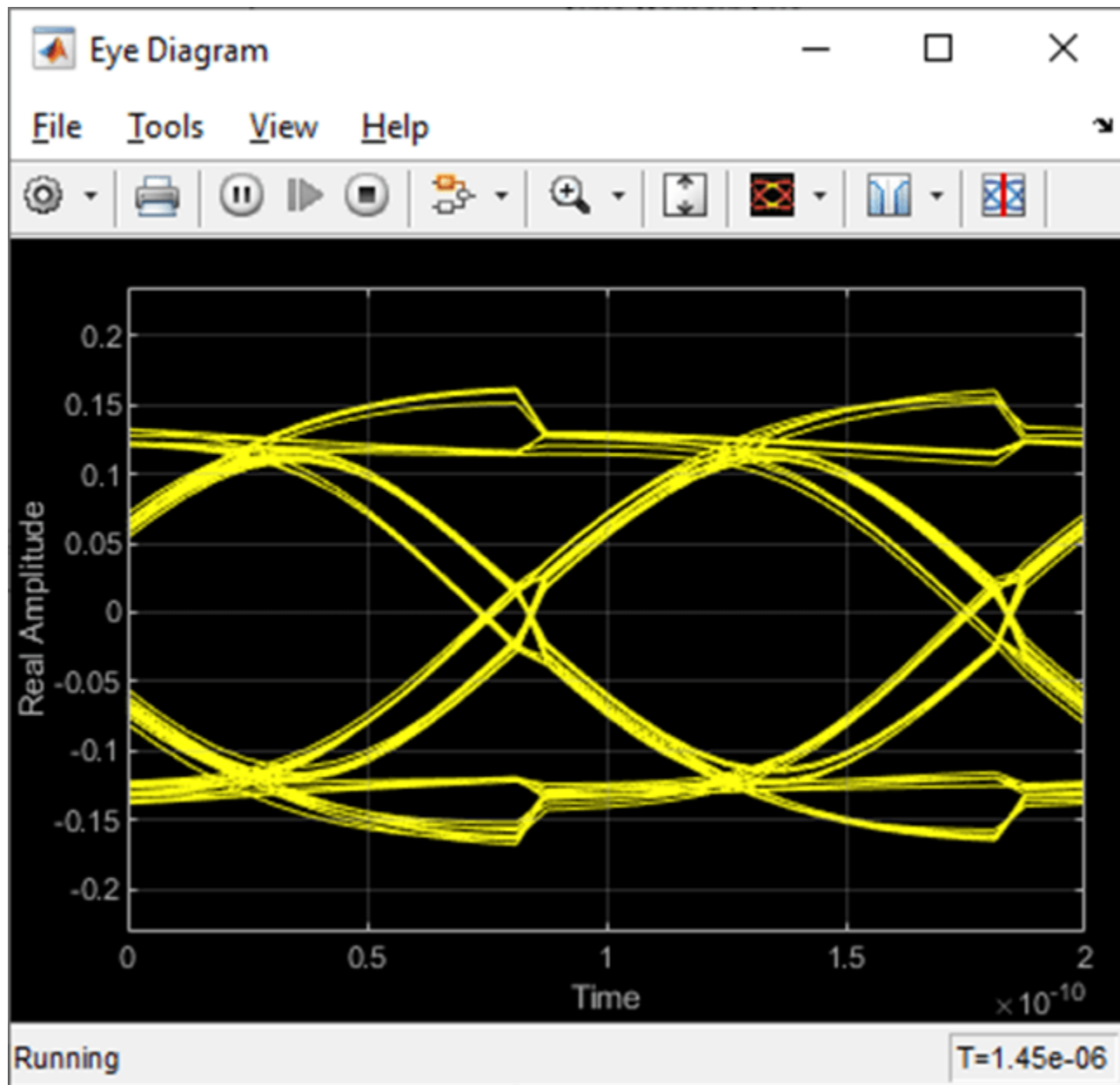
Figure: Eye Diagram during Time Domain simulation.

You can see on the Scope that the CTLE started with the value from Init, and the toggle-detector code "locks" the CTLE configuration after a few iterations:
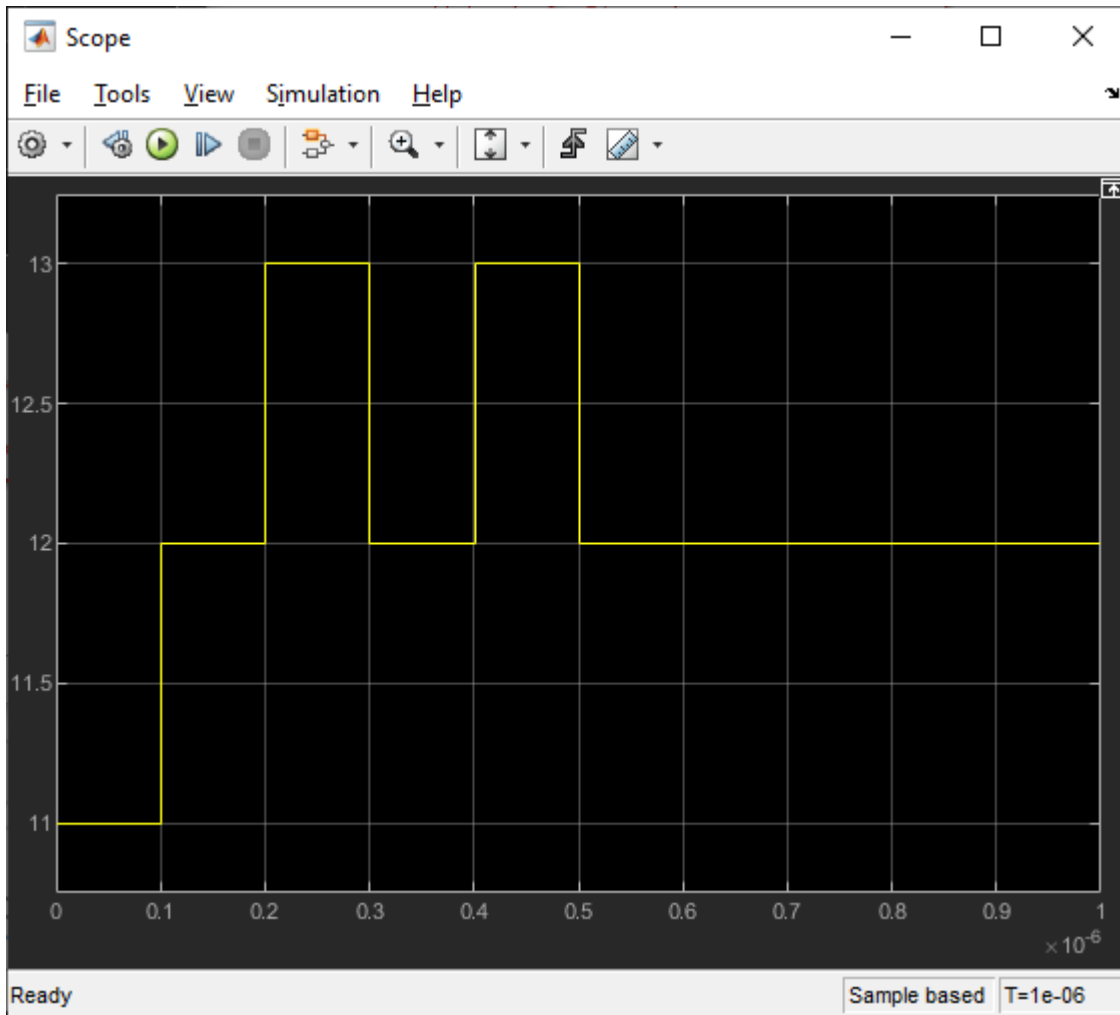
Figure: When Time Domain simulation begins, the CTLE starts with the value from Init.

You can set the code to start from CTLE configuration 0 and see that the algorithm increments the configuration until it toggles:
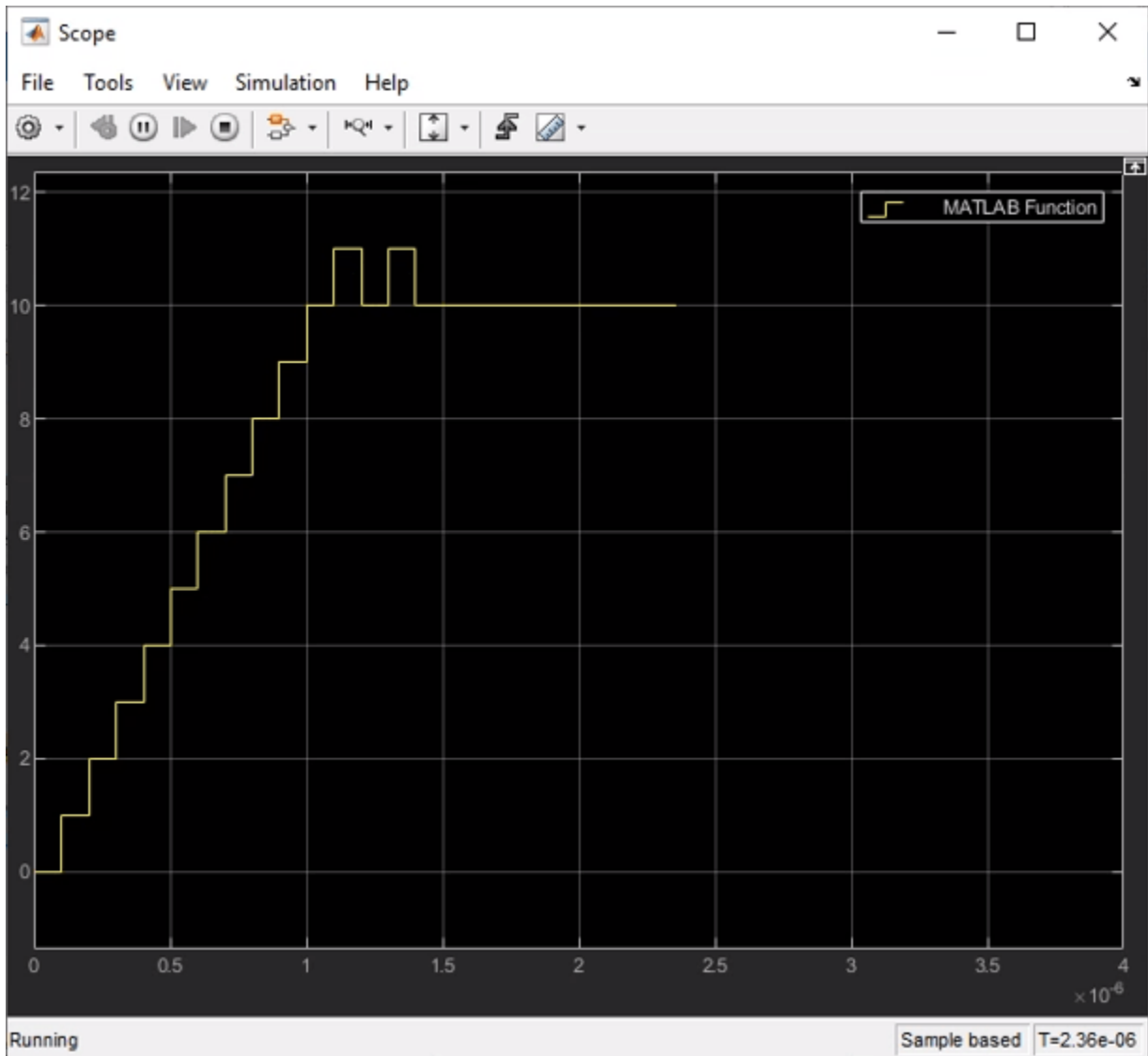
Figure: Scope output from the signal **CTLESignal.ConfigSelect** if the function is programmed to start from zero instead of the value from Init.

To find the value for Ignore Bits for the receiver, you can evaluate how many UI it takes for a CTLE to settle. In this case, it would be equal to the number of CTLE configurations available.
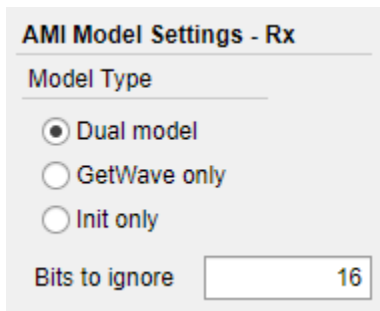
Figure: You can set the value for Ignore Bits to 16, which is the number of CTLE configurations available in this example.

When the simulation completes, you can see the Time Domain Eye has a valid Bathtub Curve if the simulation uses sufficient Ignore Bits for the receiver.
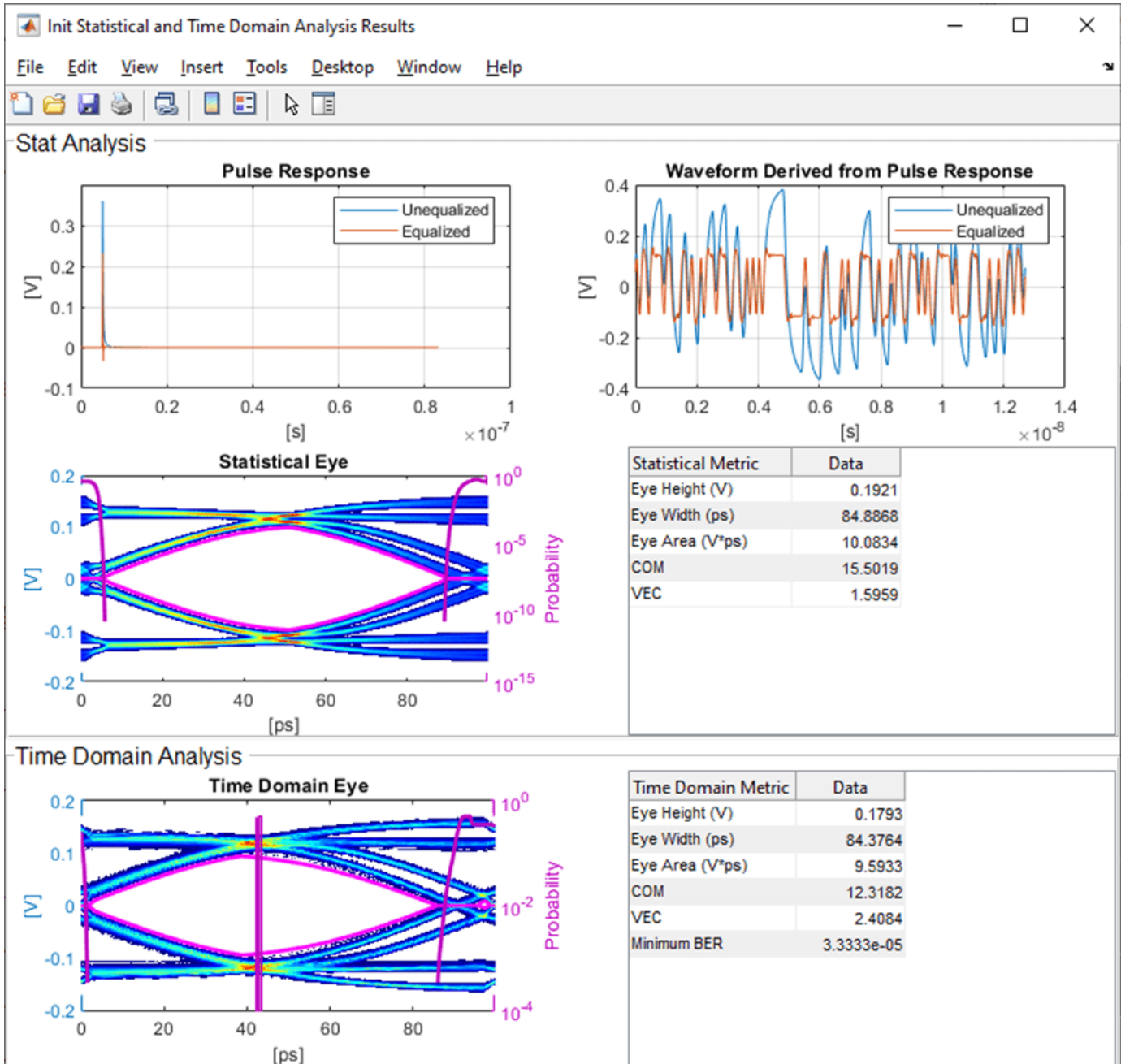
Figure: Statistical and Time Domain Results with sufficient Ignore Bits.

You can test the effect of Ignore Bits by setting the value to zero and re-running the simulation:

AMI Model Settings - Rx

Model Type

⦿ Dual model
○ GetWave only
○ Init only
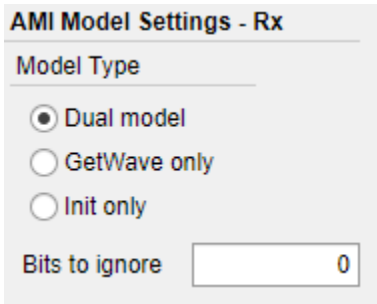
Bits to ignore   0

Figure: You can set the value for Ignore Bits to 0 from 16 to test its effect of Time Domain results.
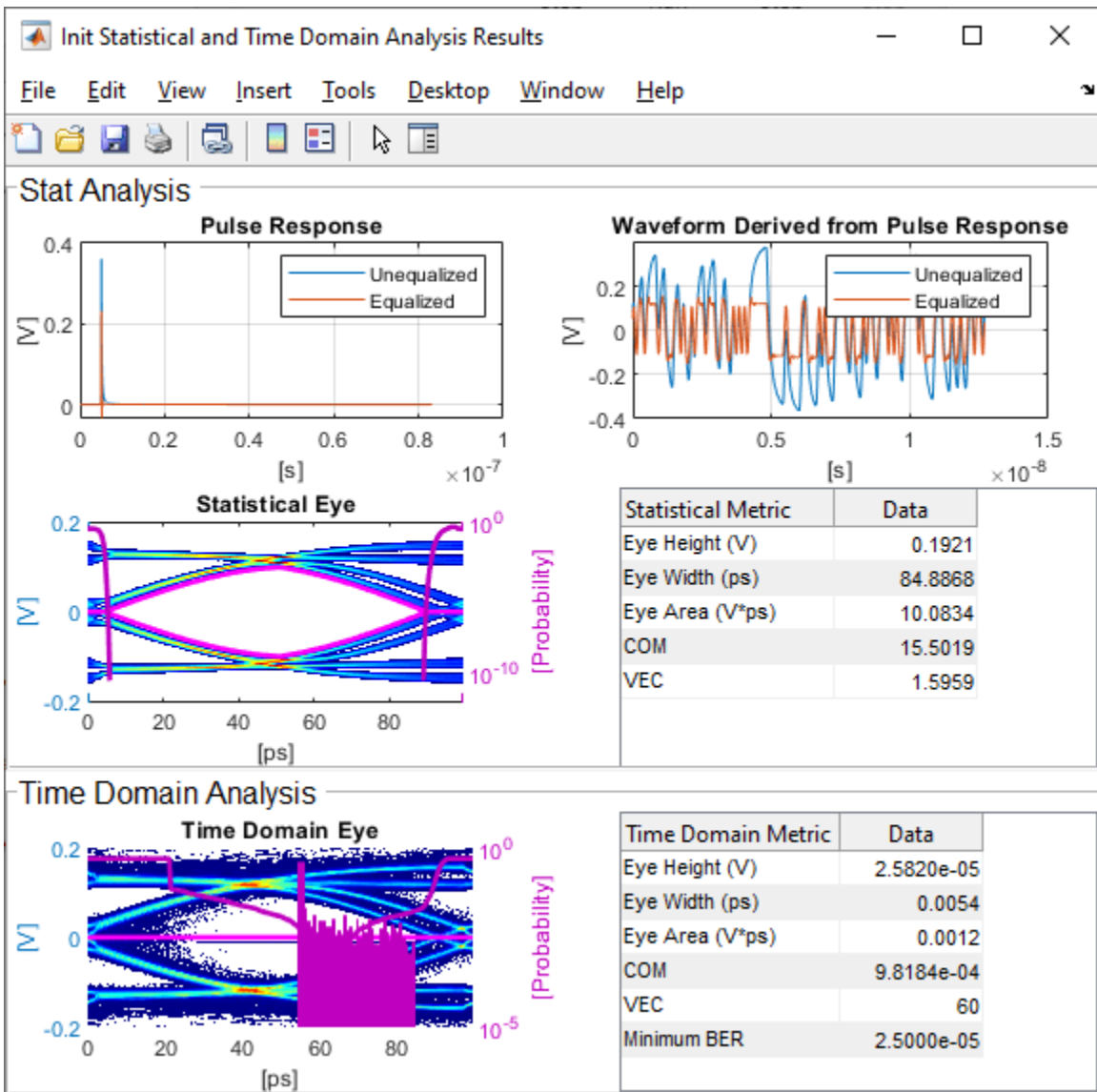
Figure: Statistical and Time Domain Results with insufficient Ignore Bits.

See Also

optPulseMetric | DFECDR | CTLE

More About

* "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance" on page 4-27

Model Clock Recovery Loops in SerDes Toolbox

This example shows how to create detailed models of different types of serial channel clock recovery loops such as Alexander (bang-bang), Meuller-Muller, and Hogg & Chu.

Clock Recovery Model Structure

To model a clock recovery loop accurately, the representation of the clock edge times and the associated sampling of the data signal must be as precise as possible. This example demonstrates a method for accomplishing that within a model that uses a fixed step discrete sample time. This method is packaged inside the Clock Generator and Signal Sampler blocks from the Mixed-Signal Blockset. The Clock Generator models the voltage controlled oscillator (VCO) of the clock recovery loop by maintaining an exact calculation of the clock edge time, including an accurate model of phase noise, and provides the exact clock edge time, along with a saturated clock, to the Signal Sampler. The Signal Sampler models the data decision latch or sample and hold circuit triggered by the clock edge. At the first fixed step sample time after a clock edge, the Signal Sampler applies linear interpolation to its input signal and outputs the resulting estimate of the input signal value at exactly the clock edge time.

The behavior of the clock oscillator and data sampling latch are very similar for different types of clock recovery loops. But the behavior and implementation of the phase detector and loop filter can vary much more widely. For example, for an Alexander clock recovery loop, the phase detection is based on comparison of logic values latched at the rising and falling edges of the clock. In contrast, Hogg & Chu phase detection compares the timing of the clock falling edge with the data threshold crossing time, and Mueller-Muller phase detection depends solely on voltage sampling at the baud rate. The clock recovery loop model treats the loop filter as a separate block, making it as easy as possible to accommodate these differences.

This example also demonstrates the design of a second order clock recovery loop. The design process is applied to a Mueller-Muller loop filter, but could be applied to the other loop filter types as well.

Example Model Structure

You can develop a model similar to the model in this example by exporting a Simulink model from the SerDes Designer app. In the receiver model, include a PassThrough block where the clock recovery loop should go, and then manually modify the PassThrough block in Simulink.

To find the clock recovery loop in the model SerdesClockRecovery, open the top level model, then open the receiver block within the example model, and then open the PassThrough block (renamed as DFE_CDR).

```
open_system('SerdesClockRecovery.slx');
toplevel = gcs;
```
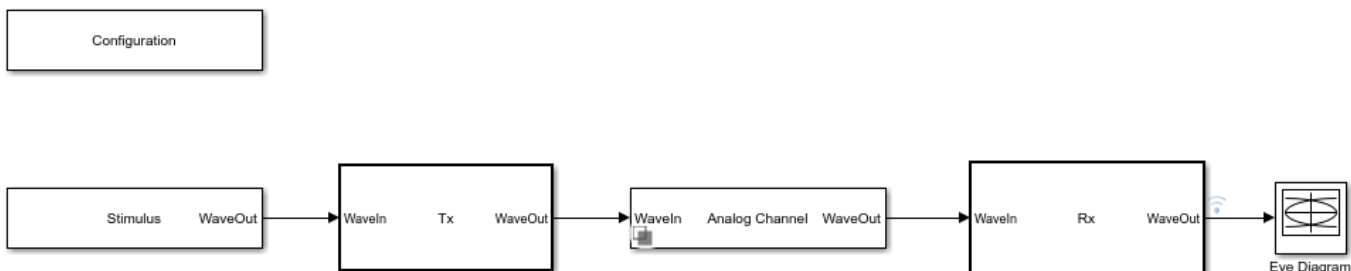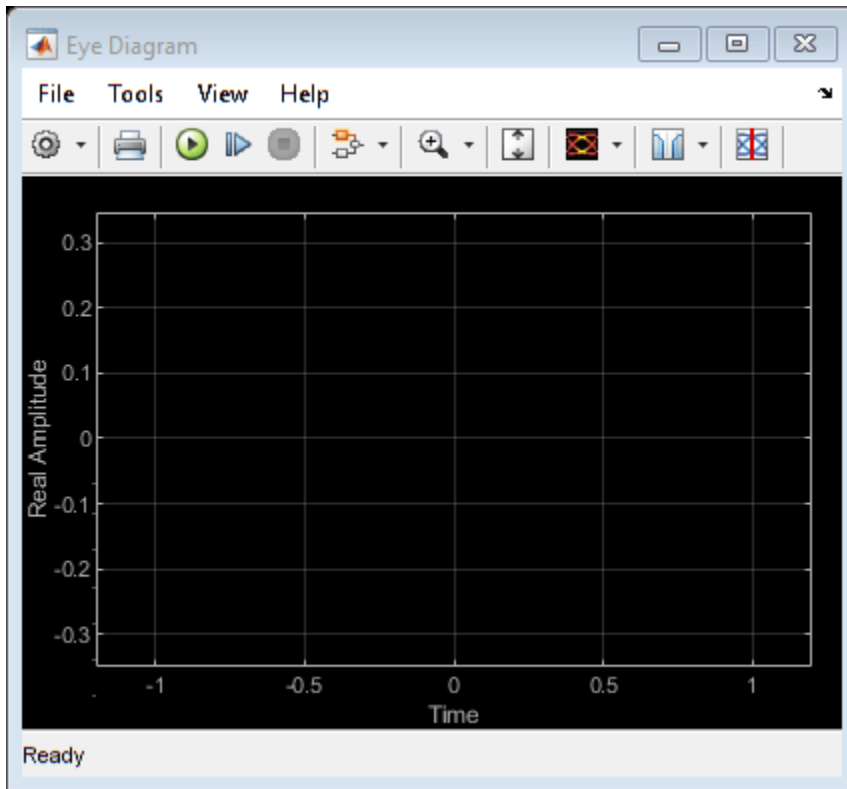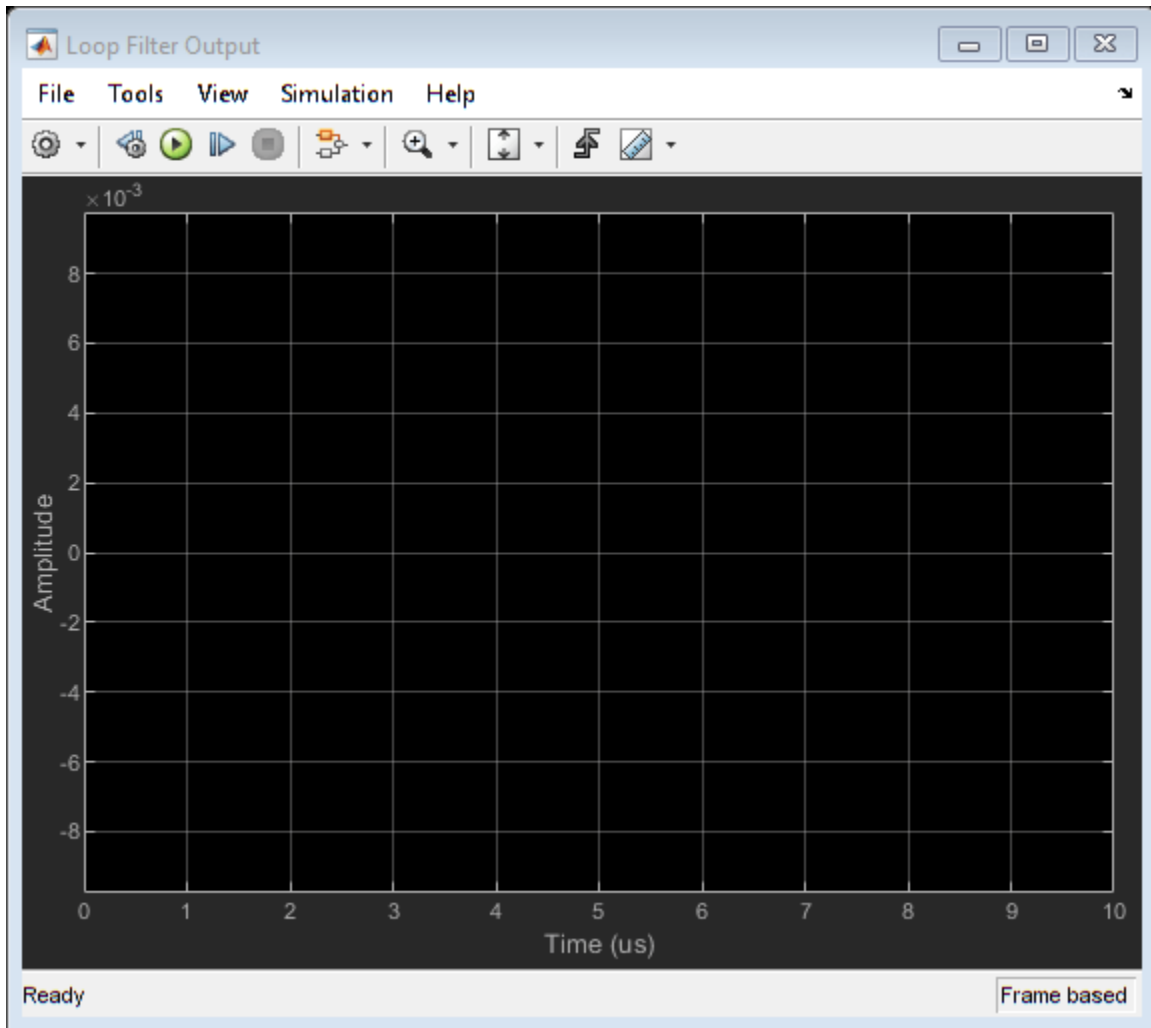
The model exported from the SerDes Designer app consists of a Non-Return to Zero (NRZ) stimulis generator, a transmitter, passive analog channel, receiver and eye diagram display.

Use scope displays to view the data signal and the clock recovery feedback signal.

Open the receiver model to view its internal structure.

```
open_system('SerdesClockRecovery/Rx');
```
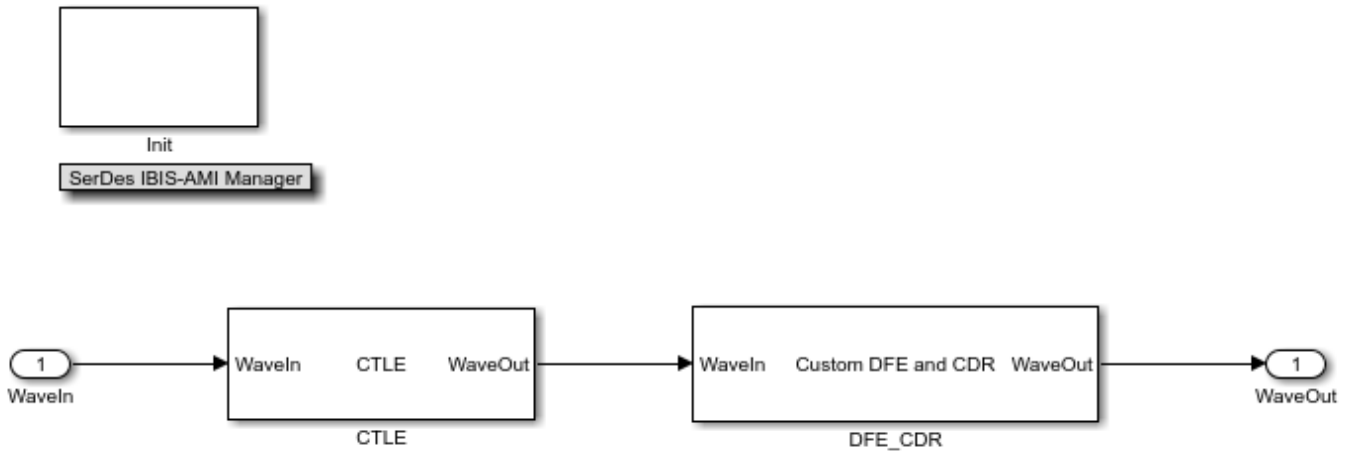
The receiver consists of a Continuous Time Linear Equalizer (CTLE) and a custom block that models the combination of a Decision Feedback Equalizer (DFE) and a Clock/Data Recovery (CDR) block. The CTLE supplies much of the equalization of the received signal, and the DFE adaptively refines the equalization while the CDR adaptively recovers the clock phase.

The DFE and CDR adaptive loops are combined in a single block because the adaptive loops are coupled. Acquisition of the correct clock phase helps the DFE loop configure the optimum equalization, and the DFE equalization helps the CDR loop acquire the correct clock phase.

Open the DFE/CDR block to view its internal structure.

```
open_system('SerdesClockRecovery/Rx/DFE_CDR','force');
```
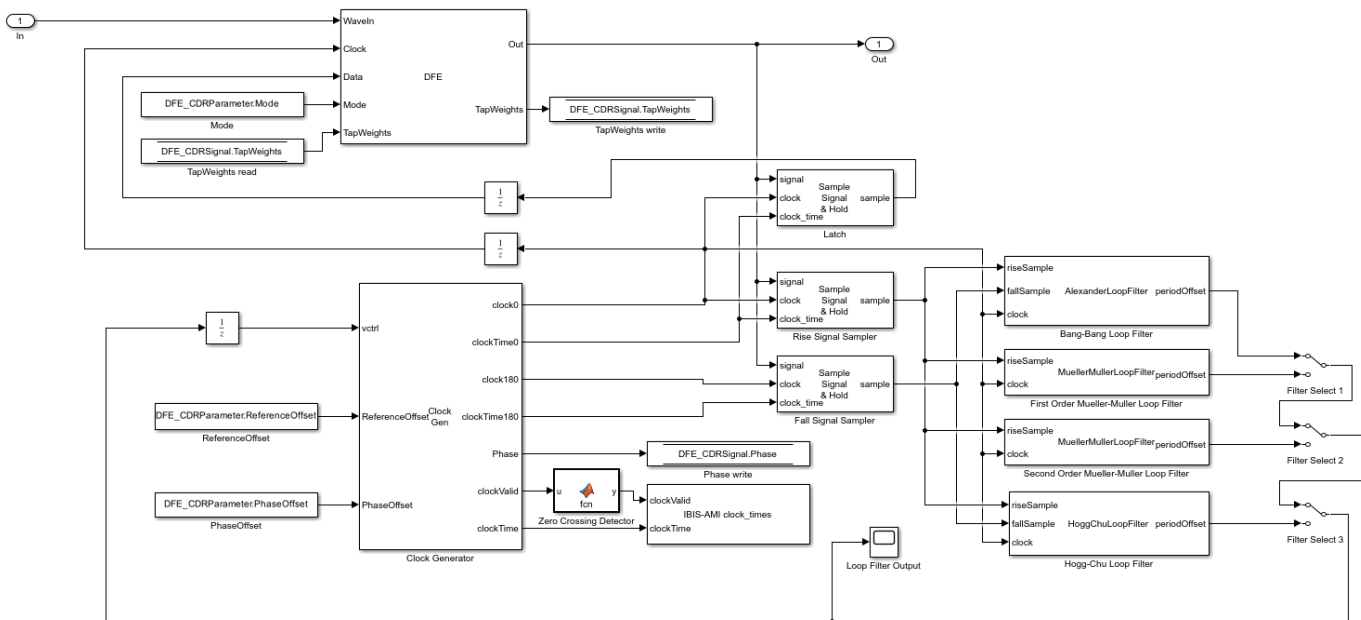
DFE/CDR Model

The DFE is modeled as a single system object that takes the input waveform as an input signal but also requires a recovered clock signal and a signal containing the detected data. The clock is provided by the Clock Generator in the CDR loop while the detected data is provided by a Signal Sampler block ("Latch") connected to the output of the DFE and configured as a latch to model the data decision latch in the receiver. A single sample delay is inserted in both the clock and detected data paths to avoid creating an algebraic loop.

The CDR is modeled using a Clock Generator block to model the VCO in the clock recovery loop and two Signal Sampler blocks ("Rise Signal Sample" and "Fall Signal Sample") to sample the DFE output signal at both the rising and falling clock edges. This configuration reflects typical hardware design practice and does mean, in particular, that the loop filter must output a frequency control voltage rather than a desired phase or similar signal.

The CDR model offers the ability to select between any of four loop filters: Alexander (bang-bang), first order Mueller-Muller, second order Mueller-Muller, and Hogg & Chu. A single sample delay is inserted in the VCO control path to avoid creating an algebraic loop. The loop filters are all implemented as system objects and the example contains the source code for these classes.

The timing in the loop filter is not critical, and loop filter processing can be performed at the sample time when the loop filter receives a clock transition. In particular, the loop filters are coded in a way that does not require knowledge of either the symbol time or the sample interval, thus avoiding problems when generating an IBIS-AMI model through SerDes Toolbox.

Alexander (Bang-Bang) Clock Recovery

The Alexander clock recovery loop detects the clock phase by determining whether the sign of the data signal at the falling edge of the clock matches the sign of the data signal at the rising edge of the clock that occurred either before or after the falling edge. If the sign at the falling edge matches the sign at the previous rising edge but not the subsequent rising edge, then the clock is early. Conversely, if the sign at the falling edge matches the sign at the subsequent rising edge but not the sign at the previous rising edge, then the clock is late. The loop filter is an up-down counter that produces either a positive (early) or negative (late) pulse when it overflows. For a detailed explanation of an Alexander clock recovery loop, see "Clock and Data Recovery in SerDes System" on page 1-3.

The initial configuration of the SerDesClockRecovery model selects the output of the Alexander loop filter to control the clock phase in the Clocked Sampler.

Run the simulation and plot the time history and the histogram of the recovered clock phase. Save the time history of the recovered clock phase to the base workspace so that you can analyze it as you choose.

```
simout = sim(toplevel);
ctBB = plotClockTimes(simout,toplevel);
```

Stat Analysis

Pulse Response

Waveform Derived from Pulse Response

Statistical Eye

Statistical Metric	Data
Eye Height (V)	0.1086
Eye Width (ps)	74.4364
Eye Area (V*ps)	5.1656
COM	7.2033
VEC	4.9798

Time Domain Analysis

Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.1054
Eye Width (ps)	71.1233
Eye Area (V*ps)	4.9915
COM	6.8605
VEC	5.2548
Minimum BER	1.2500e-05
Ignore Symbols	20000
Total Symbols	100000
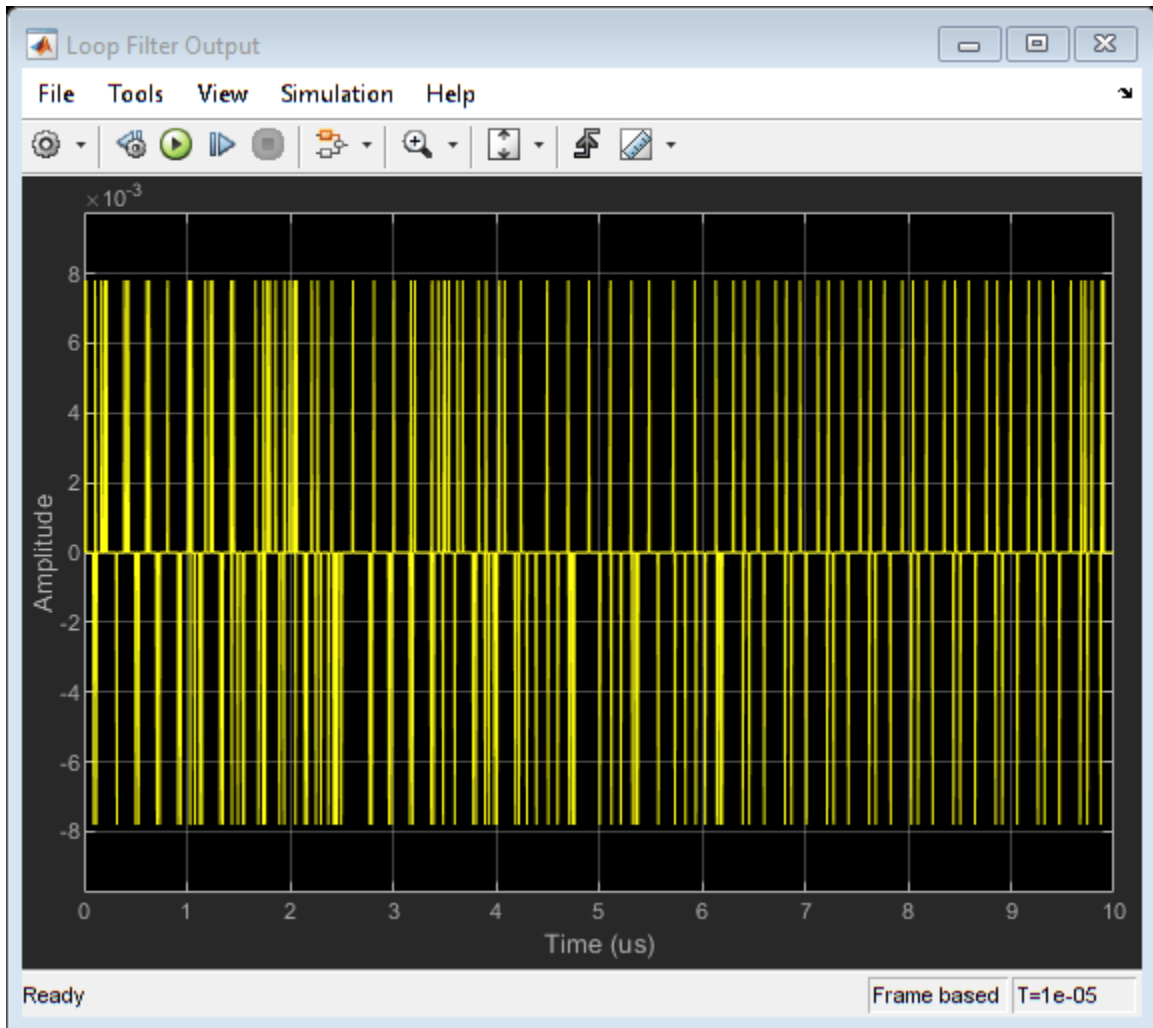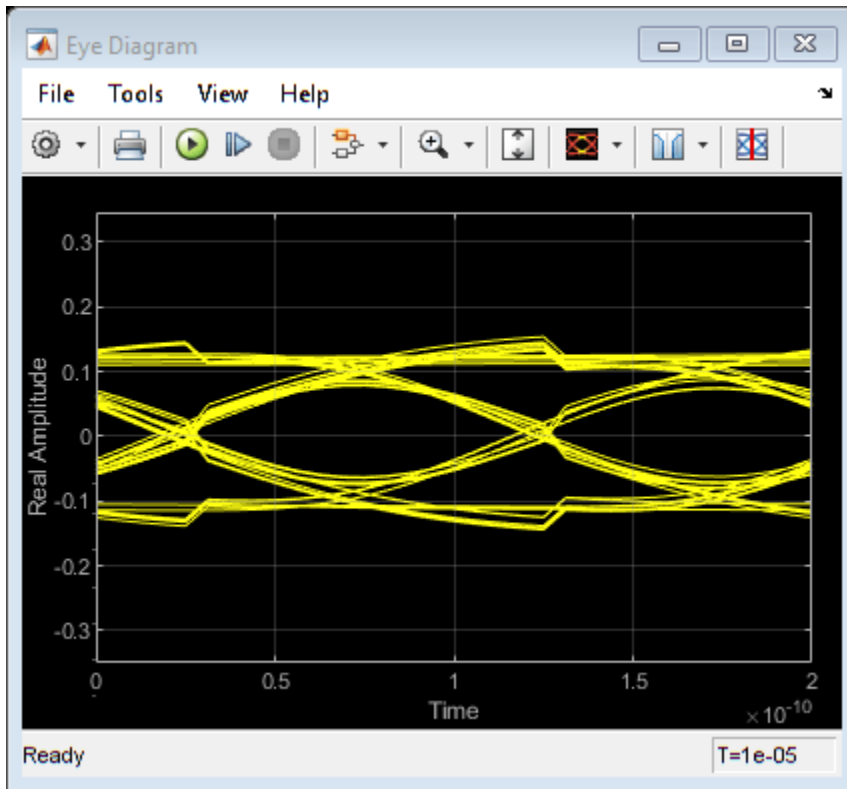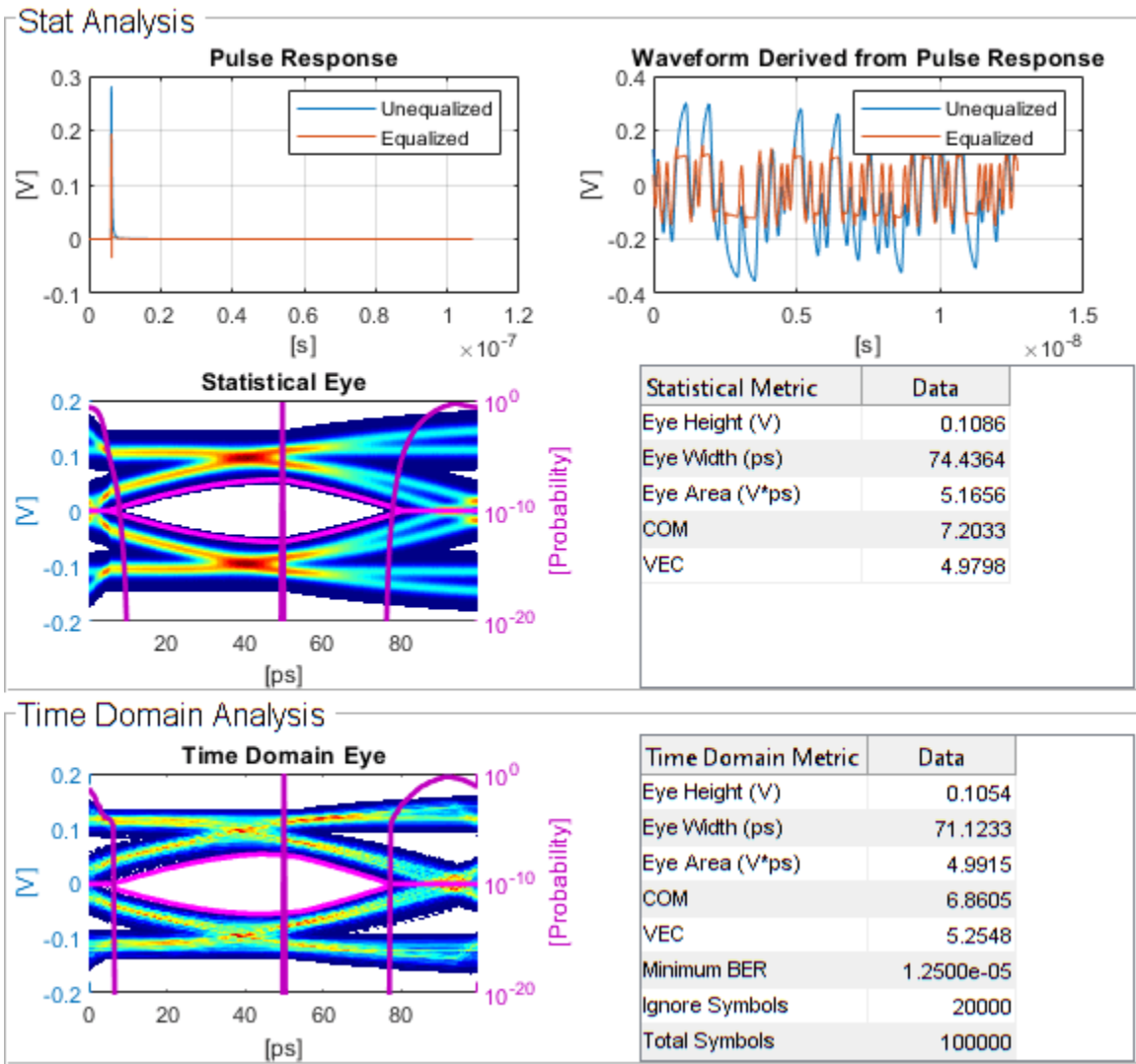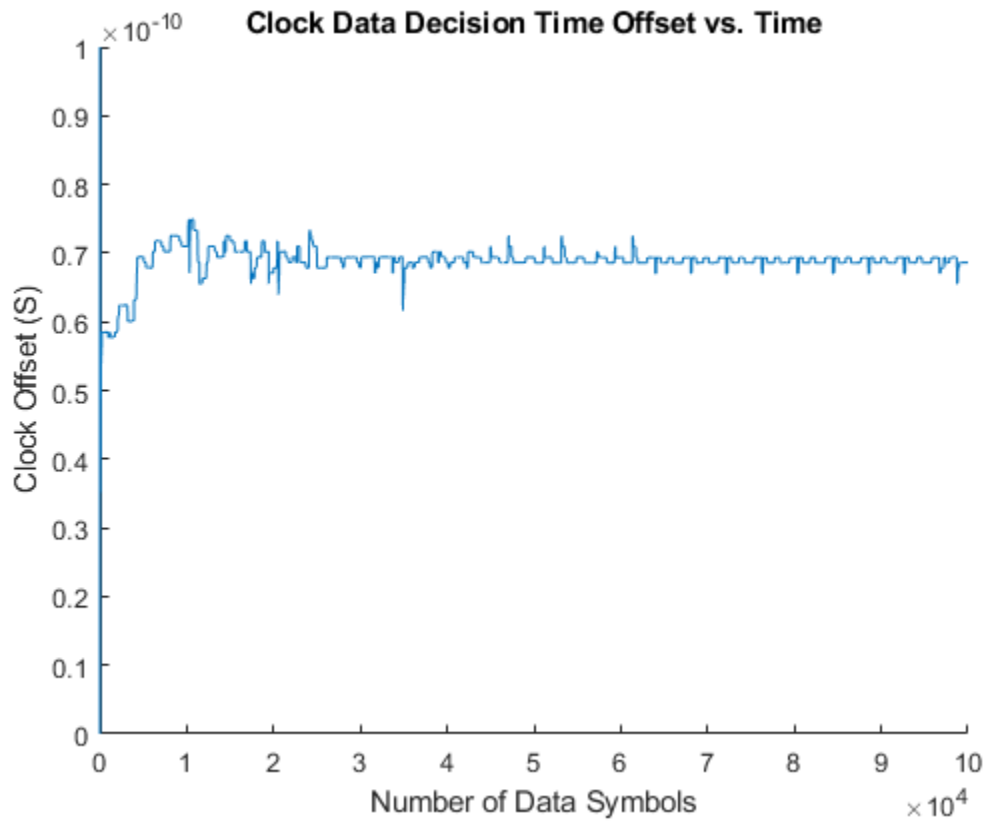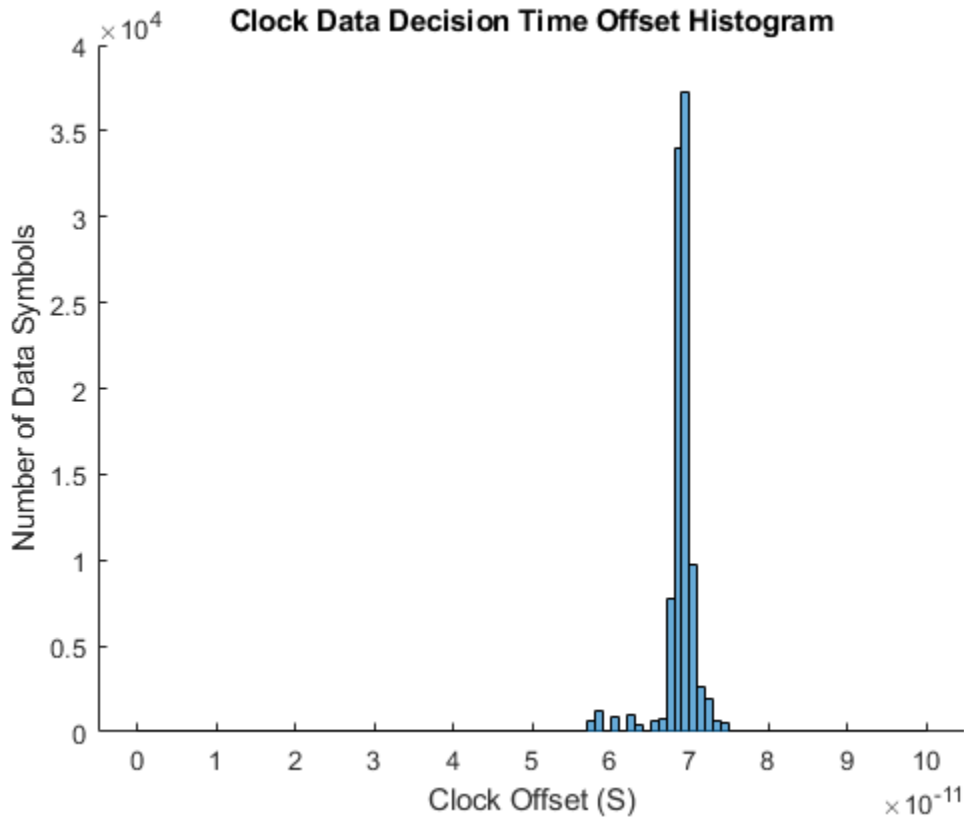
Meuller-Muller Clock Recovery

The Meuller-Muller clock recovery algorithm assumes that the data waveform changes fastest when there is a transition between data symbol values, such as a transition from a one to a zero for an NRZ data signal. This assumption enables the clock recovery loop to use one quantitative voltage per symbol, which is an advantage at high data rates. The time error estimate for the example's Meuller-Muller Loop Filter is drawn from CLOCK AND DATA RECOVERY FOR HIGH-SPEED ADC-BASED RECEIVERS, section 2.3.1

$$\tau_A = (y_{i-1}\hat{y}_i) - (y_i\hat{y}_{i-1})$$

where y_{i-1} is the previous voltage sample, y_i is the current voltage sample, $\hat{y}_{i-1}$ is the previous latched symbol value and $\hat{y}_i$ is the current latched symbol value.

To evaluate the response of the Meuller-Muller clock recovery loop, move the Filter Select 1 switch to its second input port. Run the simulation and add the time history of the recovered clock phase and clock phase histogram to the figures that have already been created for the Alexander clock recovery loop. Save the time history of the clock phase to the base workspace so that you can analyze it later.

```
set_param([gcs '/Filter Select 1'],'sw','0');
simout = sim(toplevel);
ctMM1 = plotClockTimes(simout,toplevel);
```
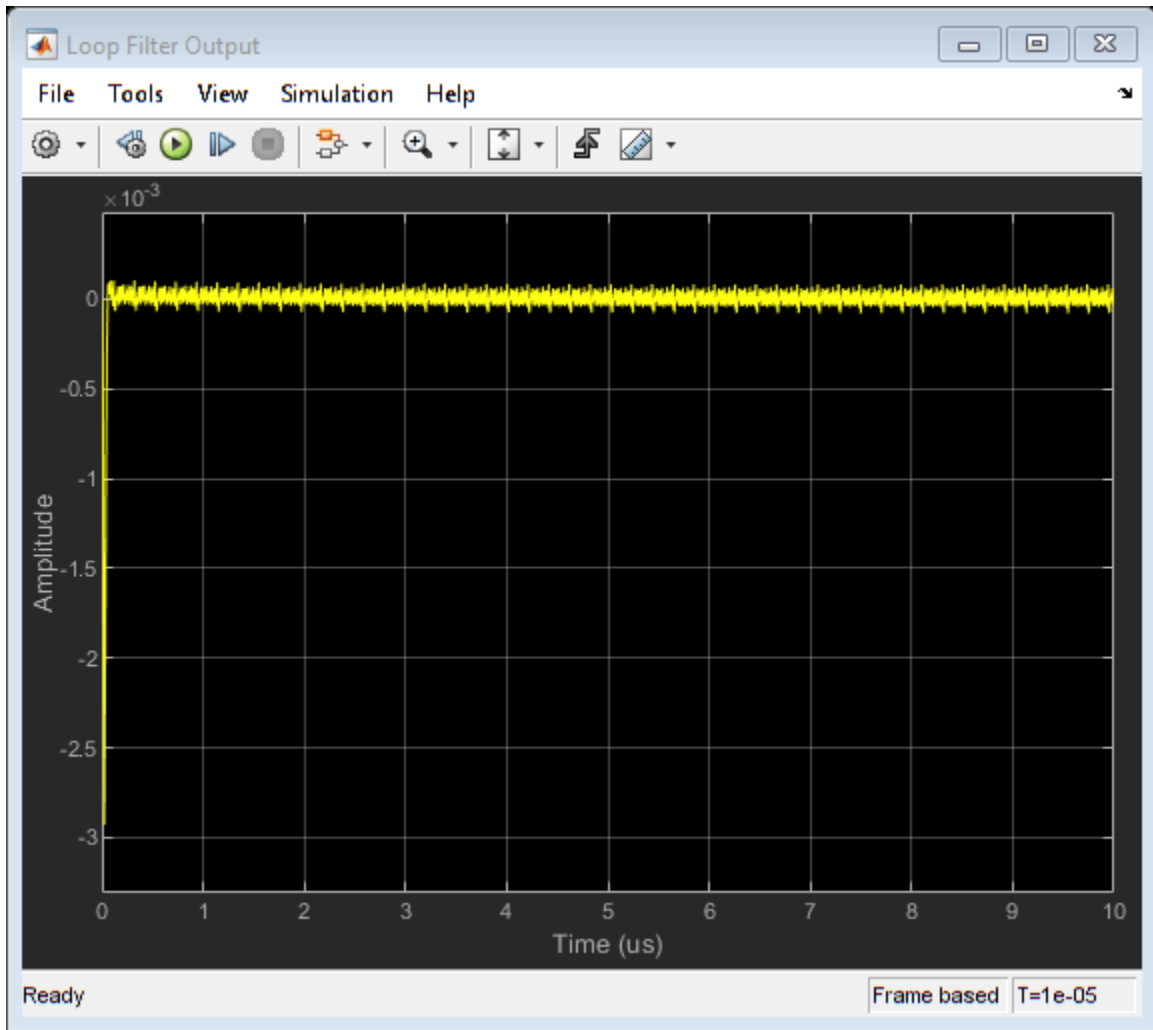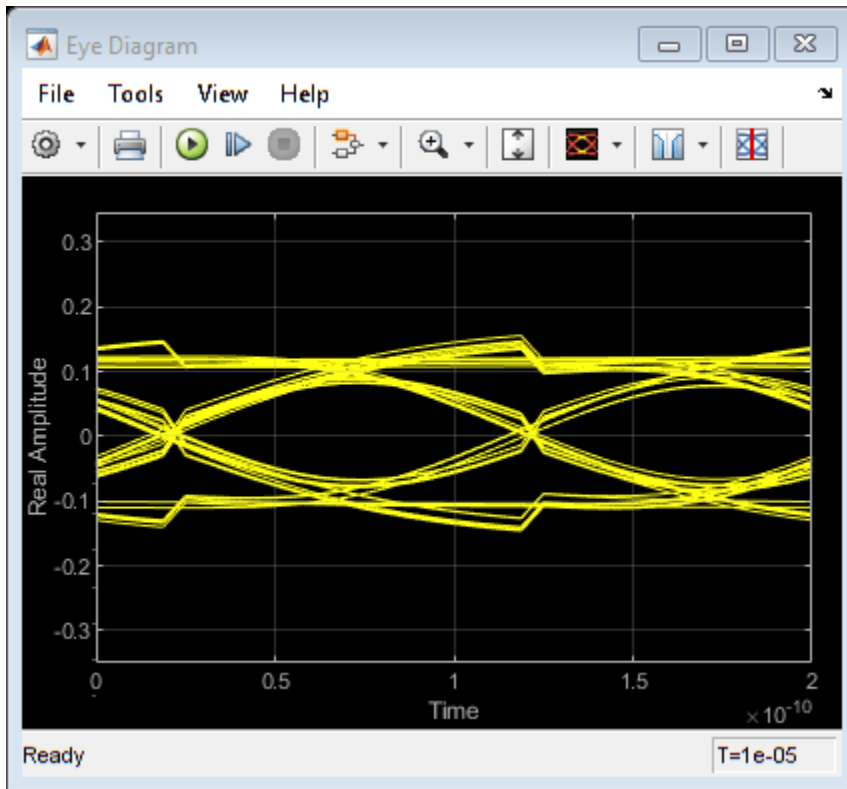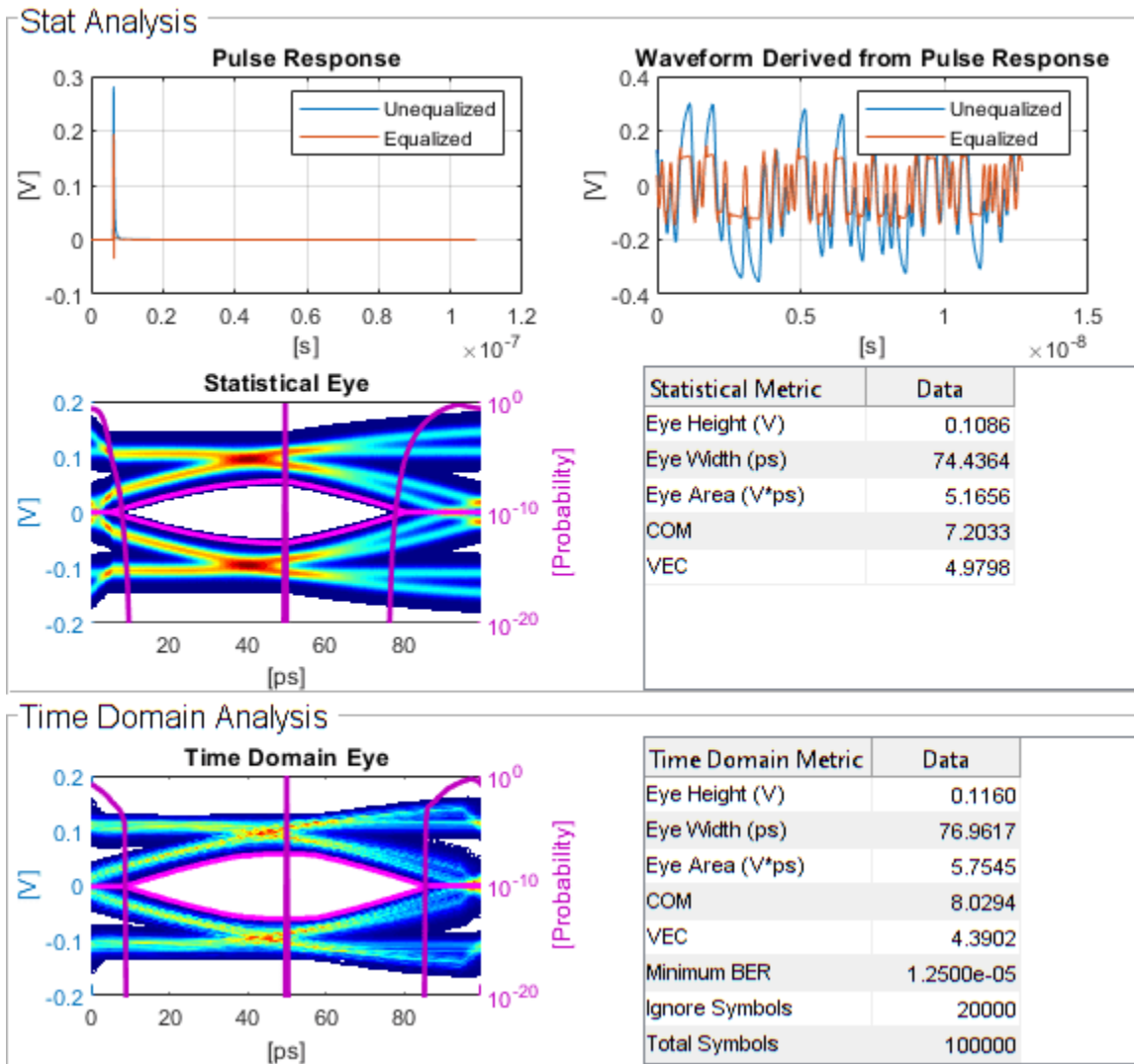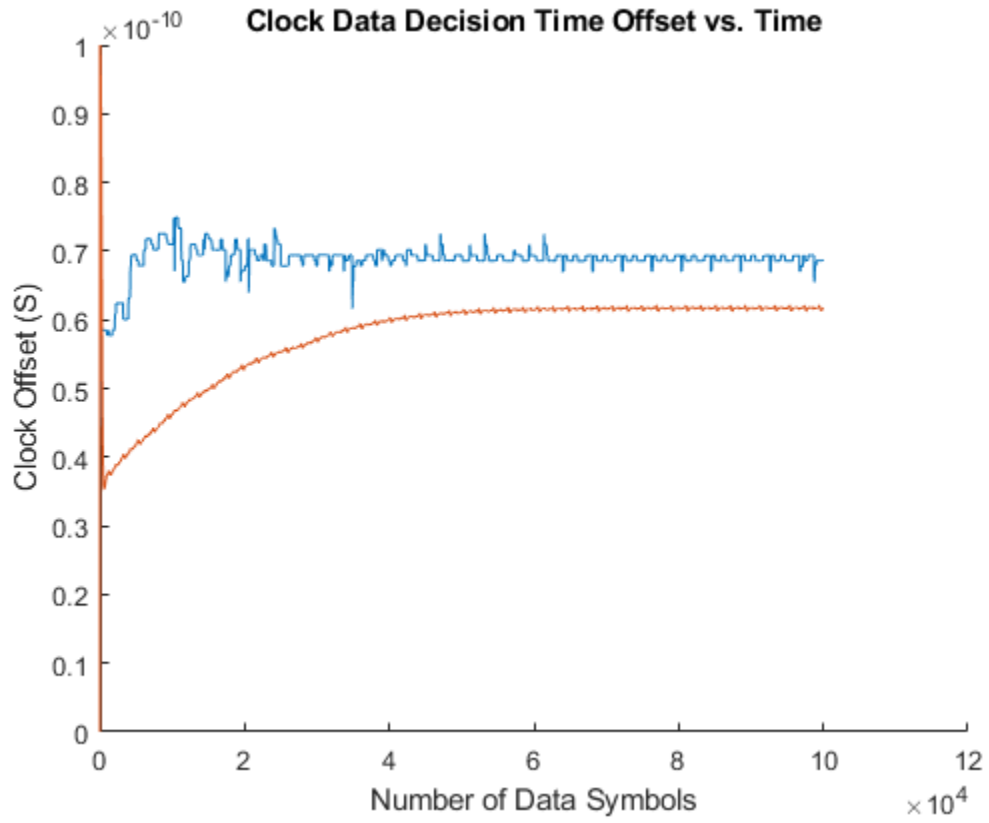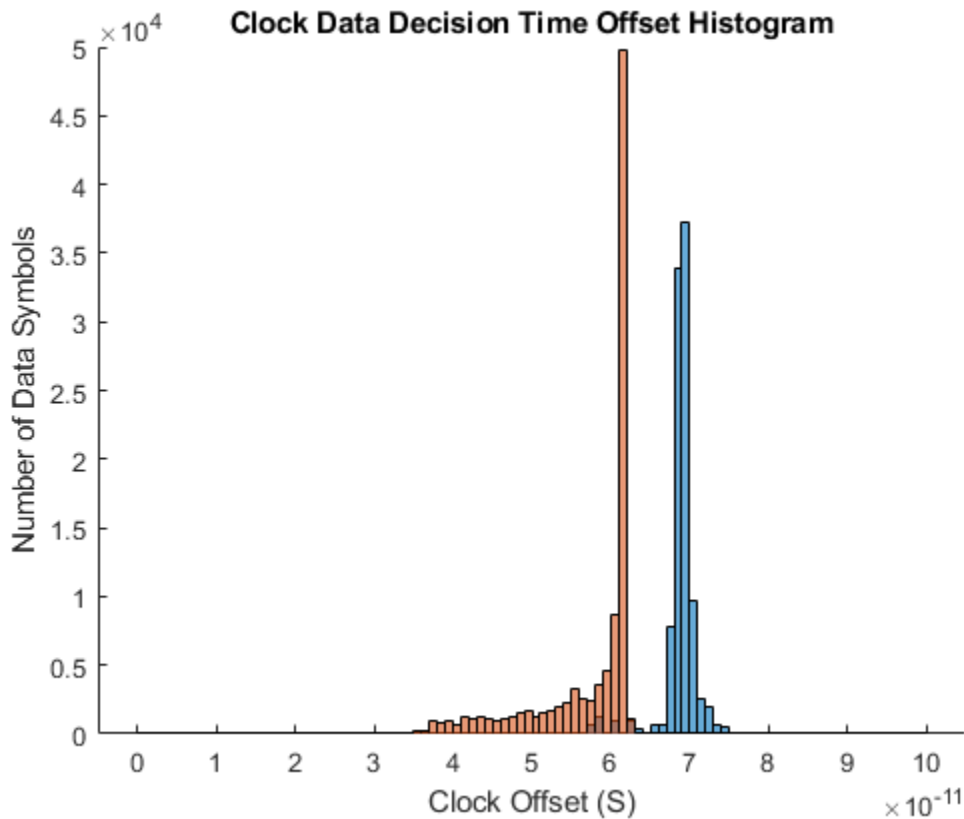
Hogg & Chu Clock Recovery

The Hogg & Chu clock recovery algorithm performs a relatively direct measurement of the clock phase by measuring the time between the threshold crossing of the data signal and the falling edge of the recovered clock. While blocks could be added to the example model to measure the data signal threshold crossing time directly, the Hogg & Chu Loop Filter in this example uses the simplifying approximation that the data signal slope in the threshold crossing region is constant. As estimated once a threshold crossing has been confirmed by the the sample at the next clock edge, the time error is

$$\tau_A = \frac{\hat{y}_{i-1} v_{f,i}}{v_{max}}$$

where $\hat{y}_{i-1}$ is the previously detected data symbol value, $v_{f,i}$ is the voltage recorded on the previous clock edge, and v_{max} is the maximum data signal amplitude.

To evaluate the reponse of the Hogg & Chu clock recovery loop, move the Fiter Select 3 switch to its second input port. Run the simulation, and add the time history of the recovered clock phase and clock phase histogram to the figures that have already been created for the Alexander and Meuller-Muller clock recovery loops. Save the time history of the clock phase to the base workspace so that you can analyze it later.

```
set_param([gcs '/Filter Select 3'],'sw','0');
simout = sim(toplevel);
ctHC = plotClockTimes(simout,toplevel);
```

Stat Analysis

Pulse Response

Waveform Derived from Pulse Response

Statistical Eye

Statistical Metric	Data
Eye Height (V)	0.1086
Eye Width (ps)	74.4364
Eye Area (V*ps)	5.1656
COM	7.2033
VEC	4.9798

Time Domain Analysis

Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.1068
Eye Width (ps)	74.6321
Eye Area (V*ps)	5.5107
COM	6.9952
VEC	5.1444
Minimum BER	1.2500e-05
Ignore Symbols	20000
Total Symbols	100000

Clock Data Decision Time Offset Histogram

Second Order Clock Recovery

A CDR loop is a phase-locked loop (PLL) for which the clock reference is provided by a received data signal and the phase detector is designed to work with this form of reference. As such, most early CDR loops were first order PLLs; however second order CDR loops are now common. A first order PLL/CDR control loop attempts to minimize the phase error directly while in a second order PLL/CDR control loop includes an integrator that zeros out the frequency offset directly.

The design2ndOrderCDR() function supplied with this example uses the equations from https://www.ti.com.cn/cn/lit/ml/snaa106c/snaa106c.pdf, Chapter 38. In addition to open loop transfer function gain, poles and zero, this function calculates a set of biquad filter coefficients that can be used directly in a dsp.BiquadFilter block.

The only additional step required is to estimate the gain of the phase detector.

For an Alexander loop filter, the phase detector gain is inversely proportional to the time axis width of the threshold crossing region in the eye diagram. One reasonable approximation is to assume that the density of the threshold crossings is a parabolic function of the timing offset. For this approximation, the phase detector gain varies with threshold crossing time and the maximum phase detector gain (in the center of the threshold crossing region) is

$$K_\phi = \frac{3Vt_s}{8\pi t_{max}}$$

volts per radian, where V is the voltage amplitude of a loop filter's "up" or "down" output pulse, maintained over one symbol time, t_s is the symbol time and t_{max} is the maximum deviation of the

4-71

threshold crossing time from the average threshold crossing time. (The total width of the threshold crossing region is $2t_{max}$.)

For a Mueller-Muller loop filter the output varies linearly from plus one half the received signal amplitude V to minus one half the received signal amplitude over a range of 2π radians. The phase detector gain (in volts per radian) is therefore

$$K_\phi = \frac{V}{4\pi}$$

You can use the demonstrate2ndOrderCDRDesign script supplied with this example to see how the design2ndOrderCDR() function is used. This scipt also uses Control System Toolbox methods, if available, to evaluate the loop dynamics.

The Second Order Mueller-Muller Loop Filter in this example has been configured to use the biquad filter coefficients produced by the demonstrate2ndOrderCDRDesign script. To run the CDR loop with this loop filter, set the Filter Select 2 switch to its second input port and set the Filter Select 3 switch to its first input port. Run the simulation, and add the time history of the recovered clock phase and clock phase histogram to the figures that have already been created for the other clock recovery loops. Save the time history of the clock phase to the base workspace so that you can analyze it later.

```
set_param([gcs '/Filter Select 2'],'sw','0');
set_param([gcs '/Filter Select 3'],'sw','1');
simout = sim(toplevel);
ctMM2 = plotClockTimes(simout,toplevel);
```
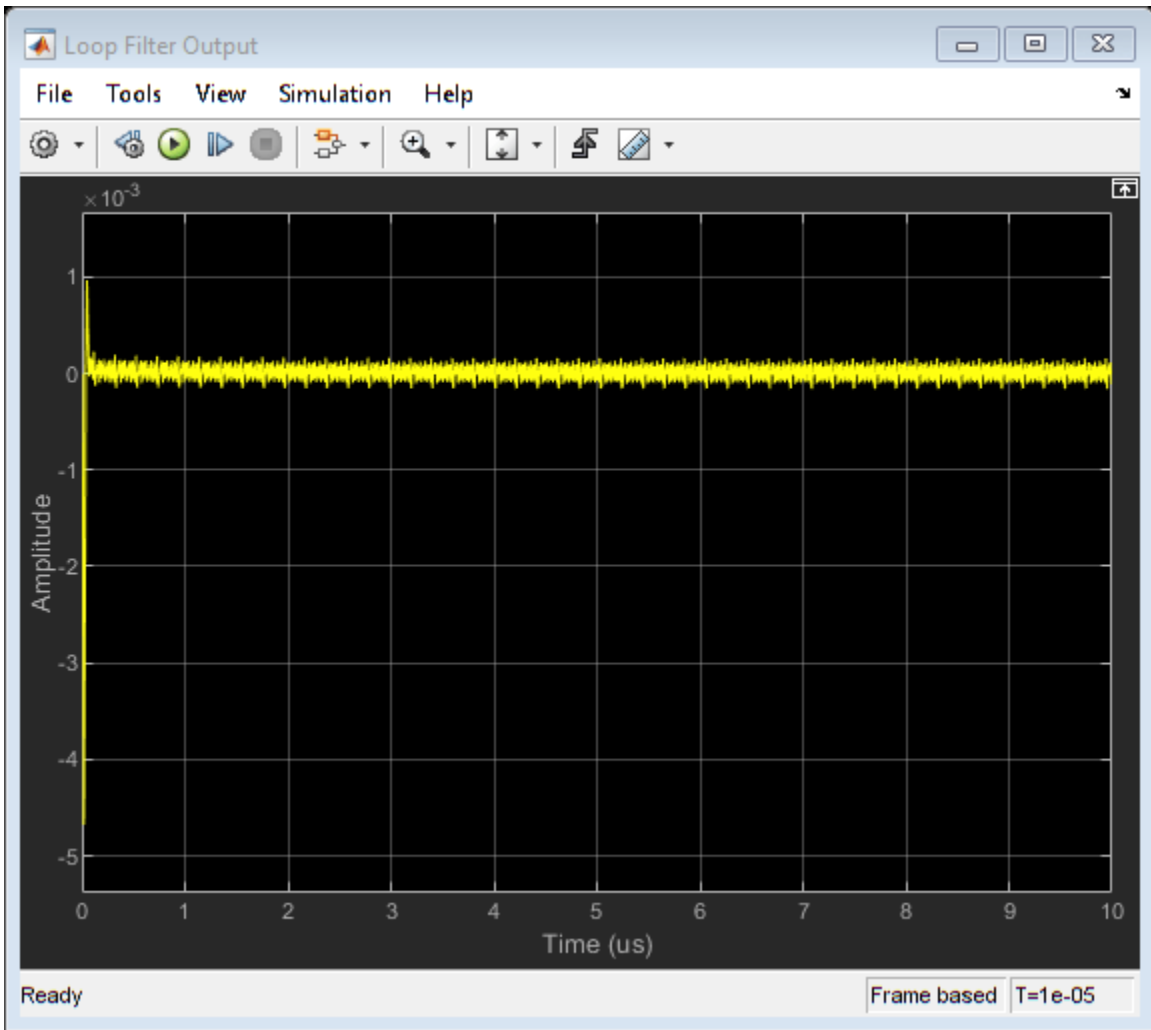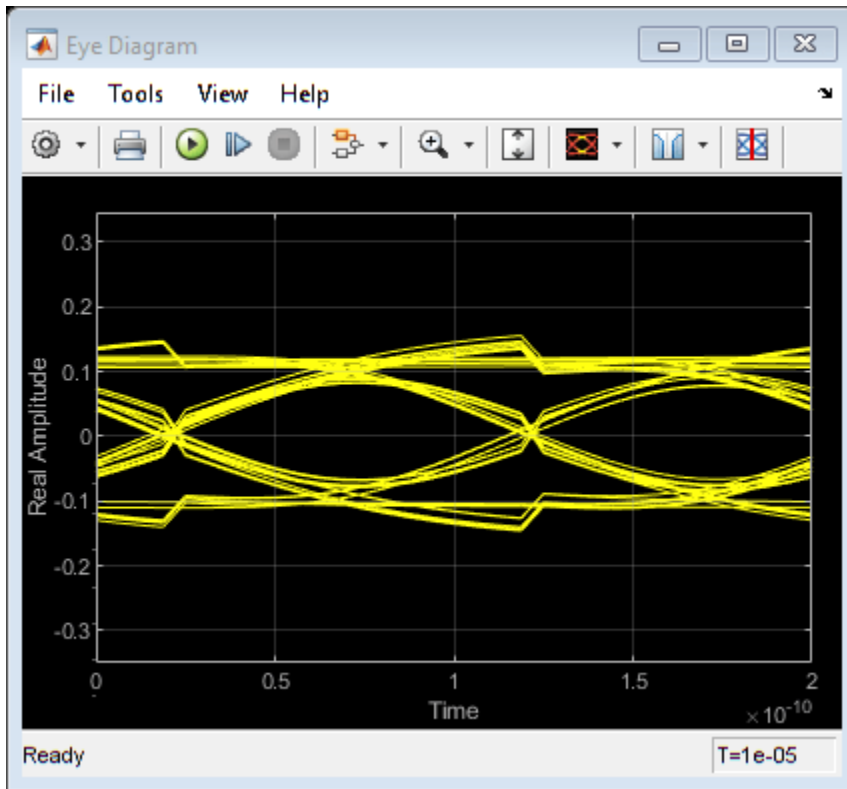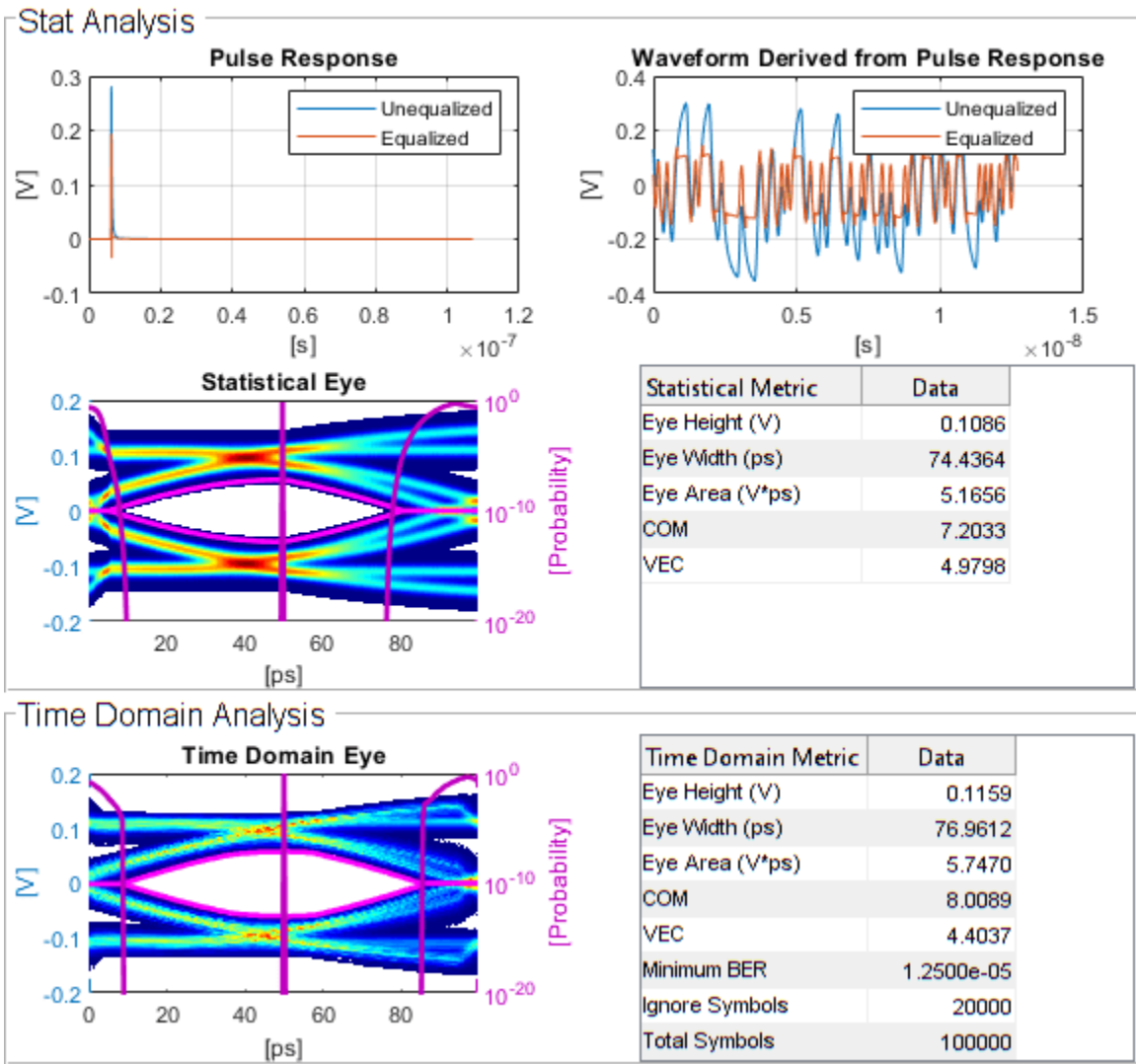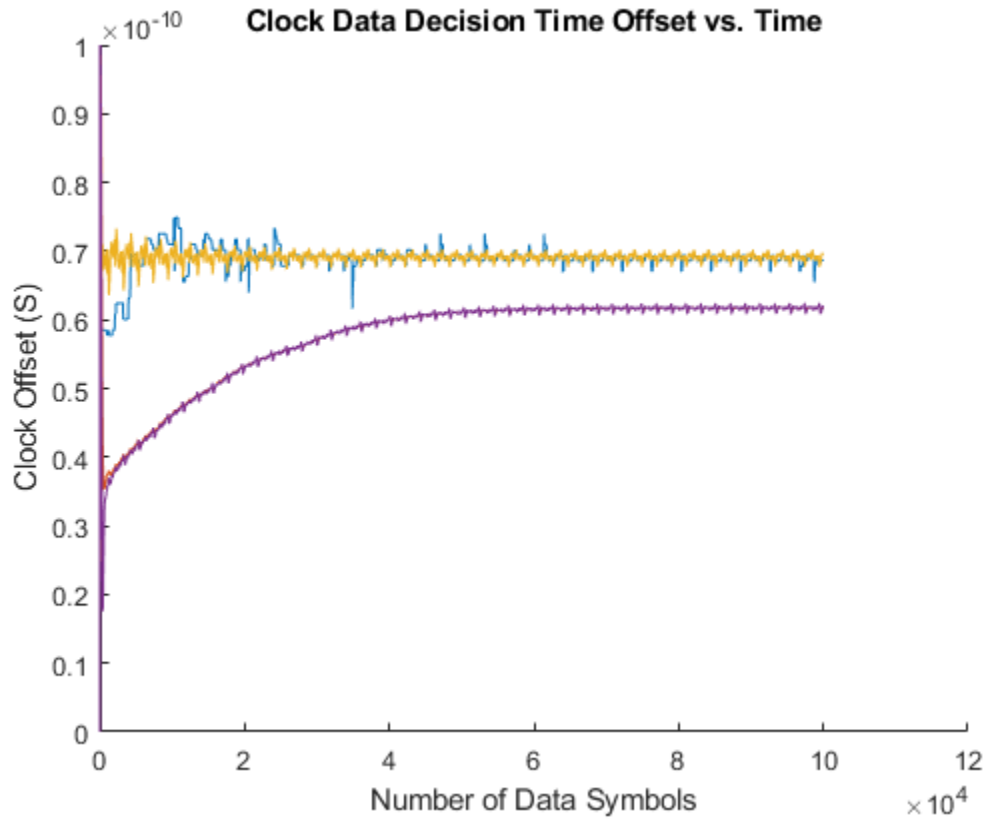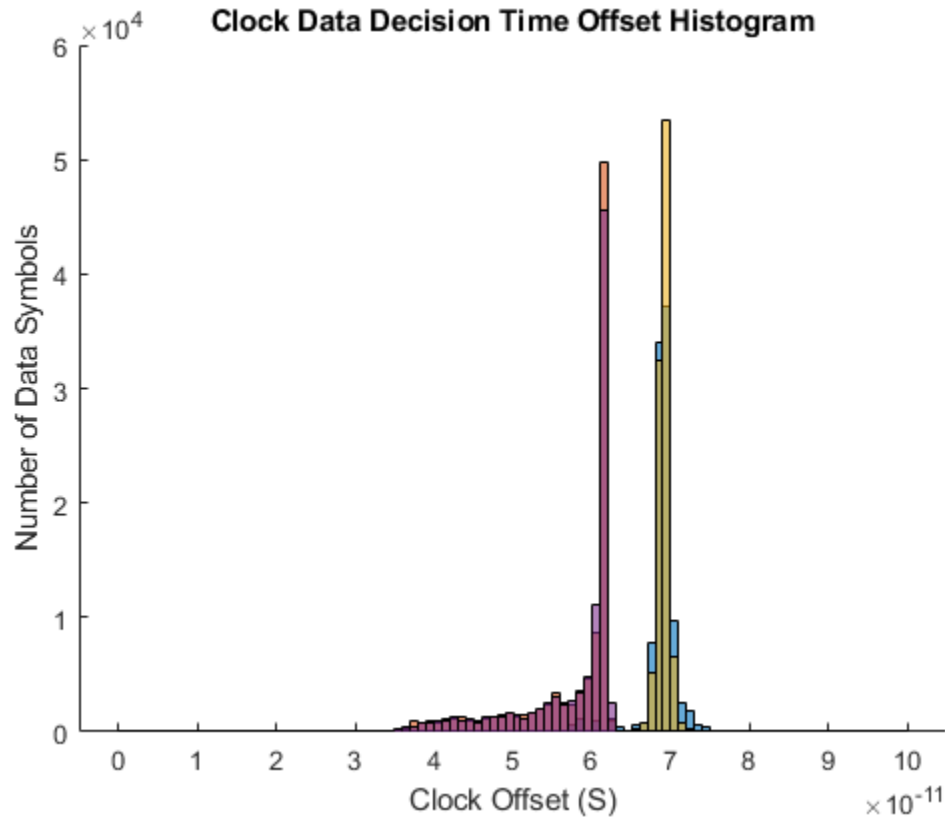
Observe the Effect of VCO Frequency Offset

The Clock Generator block has both a reference offset and a phase offset input port. You can use the reference offset input port to observe the performance improvement that a second order CDR loop provides compared to a first order CDR loop.

In the SerDes IBIS-AMI Manager accessed from the top level of the receiver block, set the DFE_CDR.ReferenceOffset IBIS-AMI parameter value to 250 (ppm) and rerun the simulations for the First Order Mueller-Muller Loop Filter and the Second Order Mueller-Muller Loop Filter. In the results you observe that the clock time for the First Order Mueller-Muller Loop Filter occurs 5ps later in the eye diagram, and the eye height at the clock time is reduced from 114mV to 107mV.

Phase Detectors: Baud-Rate Type-A versus Bang-Bang

Introduction

To successfully send and receive data between a SerDes transmitter and receiver requires a myriad of conditions to be satisfied. Chief among these conditions is a synchronization such that the receiver samples the voltage waveform at the correct time instants to recover the message. Typically, the transmitter and receiver have independent oscillators which set the operating frequency of the system within a specified tolerance of each other (e.g. 300 parts per million), but the receiver still needs to recover the clock phase and frequency from the waveform signal itself. A clock and data recovery function is accomplished by utilizing a phase detector algorithm and filtering the phase error information to recover the signal phase and frequency.

This example illustrates how first order-phase tracking is accomplished by bang-bang and baud-rate type-A phase detector models. The bang-bang phase detector is typically found in analog-based SerDes systems where an edge of the eye sampler and a data sampler at the middle of the eye are used together to determine if the data sample is early or late. A sampler is a circuit that can determine if the input voltage is greater than or less than a voltage threshold. The baud-rate type-A (or Mueller-Muller) phase detector is found in ADC-based SerDes systems where the voltage sample, determined with several bits of precision, and the resolved symbol is used to determine if the sample is early or late. These two types of phase detectors are not really competitors since they are used in very different types of SerDes systems, therefore, this example can be seen as an introduction to these types of phase detectors and their respective advantages and disadvantages.

This example first illustrates how the phase detector models operate in conjunction with decision feedback equalization (DFE) on a pulse response for Statistical analysis then on sample-by-sample time domain simulations in MATLAB and Simulink.

Pulse Response Clock Recovery

When the response of a SerDes system can be reasonably approximated by an impulse response, there is much utility in estimating the system performance through creating the Statistical eye. But first, the clock recovery position of the pulse response needs to be determined so that the DFE taps can be properly applied, and the Statistical eye can be properly centered.

Baud-Rate Type-A Phase Detector

To illustrate how the pulse response clock recovery "hula-hoop" algorithm works, this example uses an example system with a receiver with a DFE which utilizes a baud-rate type-A phase detector. The un-equalized and equalized pulse responses are obtained for visualization purposes. The baud-rate type-A phase detector looks to center the eye such that the pre-cursor and post-cursor ISI are equal. This can be thought of as placing a two-unit interval (UI) wide or two symbol times wide hula-hoop over the unequalized pulse response and sliding it down until it is flat; the clock center is found as the middle of the hula hoop. This is illustrated in the following code and figure.

```
% System parameters
SymbolTime = 100e-12;
ModulationLevels = 2;
SamplesPerSymbol = 16;

%Channel parameter
Loss = 8;
```

```matlab
%Number of DFE Taps
nTaps = 2;

%Define Receiver and system with BaudRate Type A DFE
rxBaudRate{1} = serdes.DFECDR(...
    'PhaseDetector','BaudRateTypeA',...
    'TapWeights',zeros(1,nTaps));
systemBaudRate = SerdesSystem(...
    'RxModel',Receiver('Blocks',rxBaudRate),...
    'SamplesPerSymbol',SamplesPerSymbol,...
    'ChannelData',ChannelData('ChannelLossdB',Loss),...
    'SymbolTime',SymbolTime, ...
    'Modulation',ModulationLevels);

%Get unequalized pulse response
pulse_uneq = systemBaudRate.Wave.pulse.uneq;

%Get equalized pulse responses
pulse_BaudRate = systemBaudRate.Wave.pulse.eq;

%Get clock location
phase_BaudRate = systemBaudRate.Wave.outparams{1}.DFECDR.Phase;

%Get DFE tap values
tapsBaudRate = systemBaudRate.Wave.outparams{1}.DFECDR.TapWeights;

%Calculate hula hoop indicies and voltages
ndx_k = 1:length(pulse_uneq);

ndx_BaudRateTypeA = phase_BaudRate*SamplesPerSymbol;
ndx_BaudRateHoop = ndx_BaudRateTypeA + [-1 1]*SamplesPerSymbol;
v_BaudRateHoop = interp1(ndx_k,pulse_uneq,ndx_BaudRateHoop);

%DFE Tap index
ndx_TapsBaudRate = round(ndx_BaudRateTypeA)+SamplesPerSymbol*(1:nTaps);

%Create lines to visualise DFE taps
v_TapsBaudRate = [pulse_uneq(ndx_TapsBaudRate)'; pulse_uneq(ndx_TapsBaudRate)'+tapsBaudRate; nan

%Find pulse max value, max index and min value (Used to frame the pulse response plot)
[maxValue,maxIndex] = max(pulse_uneq);
minValue = min(pulse_BaudRate);

%Visualize DFE + Baud-rate clock recovery pulse
figure,
plot(ndx_k,pulse_uneq,...
    ndx_k,pulse_BaudRate,'r',...
    ndx_BaudRateTypeA*[1 1],[minValue,maxValue],'c--',...
    ndx_BaudRateHoop,v_BaudRateHoop,'-sc',...
    [1;1;1]*ndx_TapsBaudRate,v_TapsBaudRate,'gd-')
axis([maxIndex-2*SamplesPerSymbol,maxIndex+(2+nTaps)*SamplesPerSymbol,minValue-0.05 maxValue + 0
xlabel('Index'),ylabel('V')
title('Pulse Response: DFE + baud-rate phase detector')
grid on
legend('Unequalized','Equalized','Baud-rate type A clock center','Baud-rate type A 2 UI hoop','Ze
```

Pulse Response: DFE + baud-rate phase detector
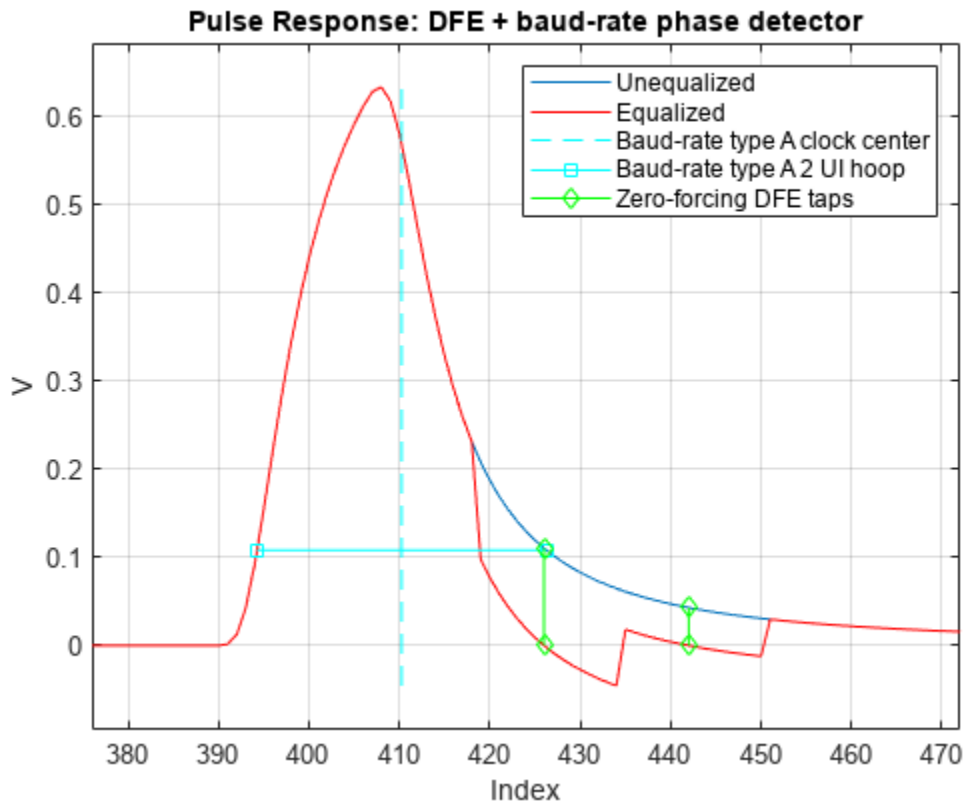

```
%Plot bathtubs, Statistical Eye and contours for the DFE with a baud-rate
%phase detector.
figure, plotStatEye(systemBaudRate)
title('Statistical Eye: DFE + baud-rate phase detector')
```
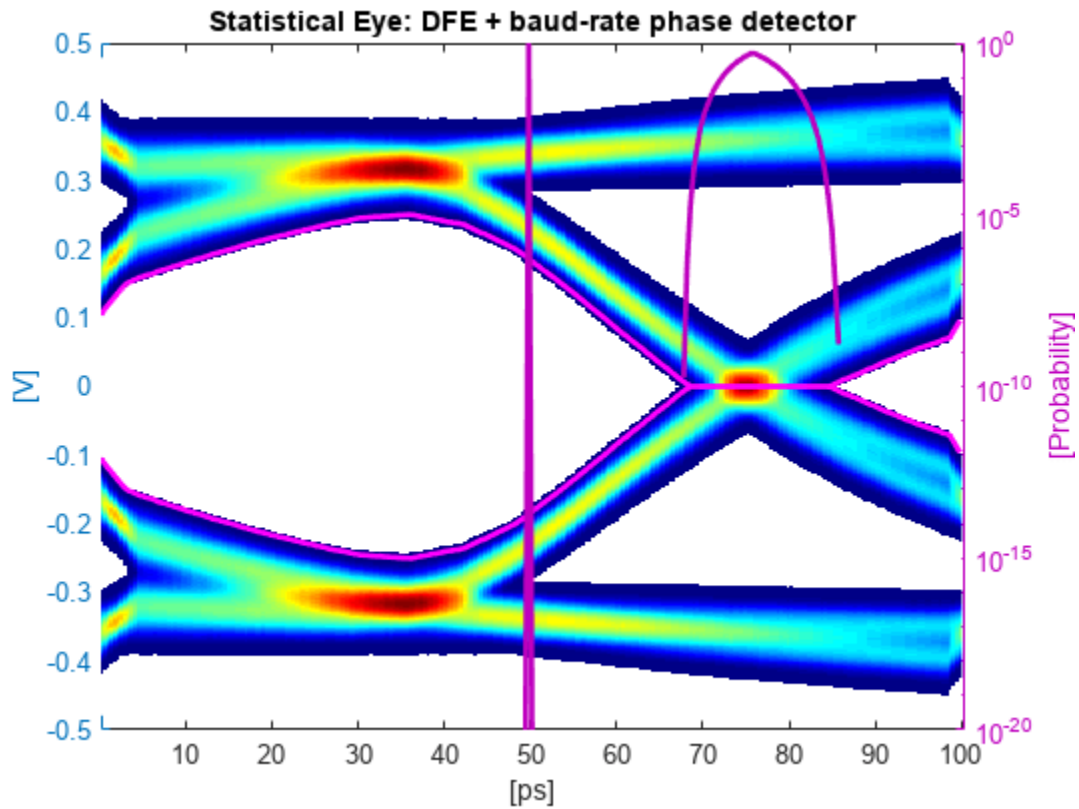
Bang-Bang Phase Detector

The pulse clock recovery is very similar for the bang-bang phase detector but the hula-hoop width is only one unit interval (UI) wide.

```
%Define Receiver and system with BangBang DFE
rxBangBang{1} = serdes.DFECDR(...
    'PhaseDetector','BangBang',...
    'TapWeights',zeros(1,nTaps));
systemBangBang = SerdesSystem(...
    'RxModel',Receiver('Blocks',rxBangBang),...
    'SamplesPerSymbol',SamplesPerSymbol,...
    'ChannelData',ChannelData('ChannelLossdB',Loss),...
    'SymbolTime',SymbolTime, ...
    'Modulation',ModulationLevels);

%Get equalized pulse responses
pulse_BangBang = systemBangBang.Wave.pulse.eq;

%Get clock location
phase_BangBang = systemBangBang.Wave.outparams{1}.DFECDR.Phase;

%Get DFE tap values
tapsBangBang = systemBangBang.Wave.outparams{1}.DFECDR.TapWeights;

%Calculate hula hoop indicies and voltages
ndx_BangBang = phase_BangBang*SamplesPerSymbol;
```

```
ndx_BangBangHoop = ndx_BangBang + [-0.5 0.5]*SamplesPerSymbol;
v_BangBangHoop = interp1(ndx_k,pulse_uneq,ndx_BangBangHoop);

%DFE Tap index
ndx_TapsBangBang = round(ndx_BangBang)+SamplesPerSymbol*(1:nTaps);

%Create lines to visualise DFE taps
v_TapsBangBang = [pulse_uneq(ndx_TapsBangBang)'; pulse_uneq(ndx_TapsBangBang)'+tapsBangBang; nan

% 1b) vs Bang Bang in Init
%Plot bathtubs, Statistical Eye and contours
figure,
plot(ndx_k,pulse_uneq,...
    ndx_k,pulse_BangBang,'r',...
    ndx_BangBang*[1 1],[minValue,maxValue],'c--',...
    ndx_BangBangHoop,v_BangBangHoop,'-sc',...
    [1;1;1]*ndx_TapsBangBang,v_TapsBangBang,'gd-')
axis([maxIndex-2*SamplesPerSymbol,maxIndex+(2+nTaps)*SamplesPerSymbol,minValue-0.05 maxValue + 0
grid on
title('Pulse response: DFE + bang-bang phase detector')
legend('Unequalized','Equalized','Bang-bang clock center','Bang-bang 1 UI hoop','Zero-forcing DF
xlabel('Index'),ylabel('V')
```


Pulse response: DFE + bang-bang phase detector
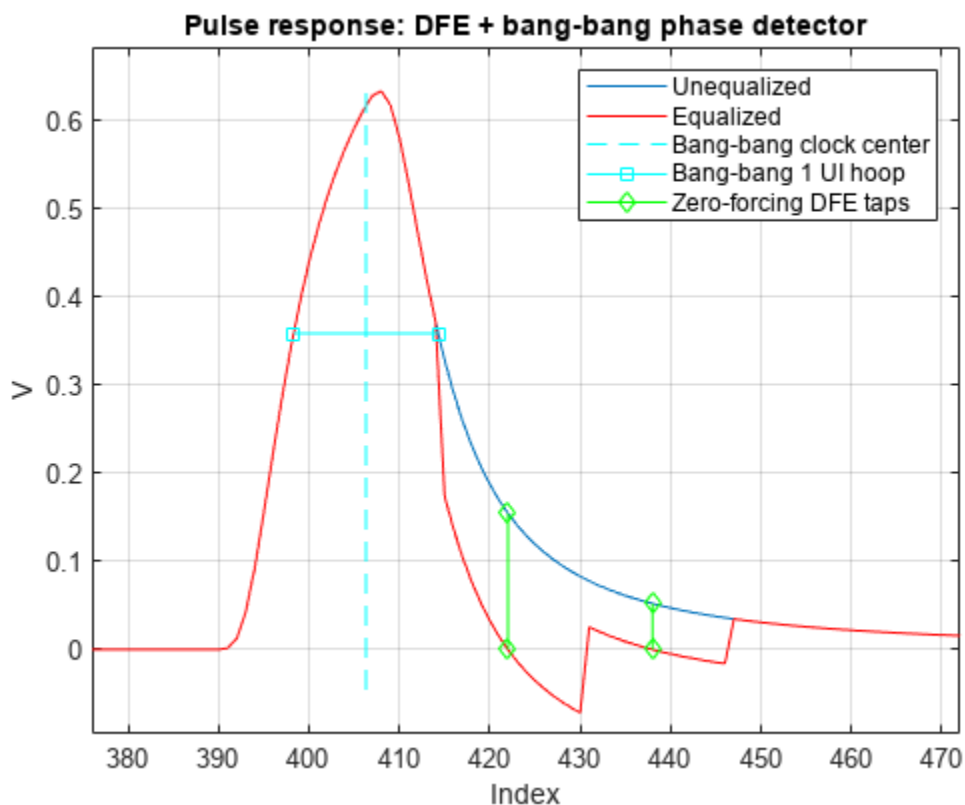
```
%Plot bathtubs, Statistical Eye and contours for the DFE with a bang-bang
%phase detector.
figure, plotStatEye(systemBangBang)
title('Statistical Eye: DFE + bang-bang phase detector')
```
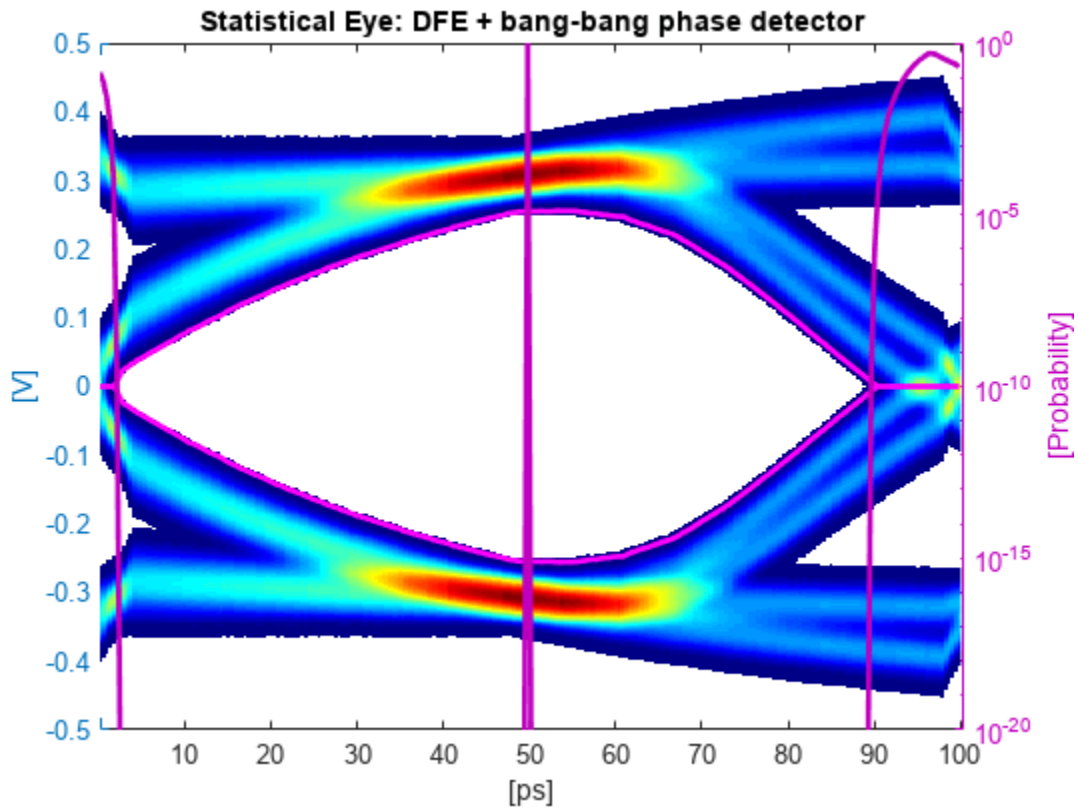
Comparison of Pulse Based Clock Recovery Methods

Comparing the pulse response clock locations for the bang-bang and the baud-rate type-A phase detectors, you can see that the baud-rate type-A clock location is to the right of the bang-bang clock location. This is typically due to the way capacitive loads shape the pulse response. The clock location also impacts the location and value of the zero-forcing DFE tap weights, resulting in significantly different looking Statistical eye diagrams.

Phase Detection of Sample-by-Sample Time Domain Simulation

In an actual system, or in a time domain simulation, the system pulse response is not typically available. Therefore, the phase detector must determine if the sampling location is early or late only with the samples of the waveform that it has available. For information on how bang-bang phase detector uses edge and data samples, see . The baud-rate type-A phase detector uses voltage sample, $v[k]$,(with the precision of a few bits), and the resolved data sample, $d[k]$. The phase error is as follows:

Phase Error = $v[k]*d[k-1] - v[k-1]*d[k]$

To visualize how minimizing this phase error results in an optimal sampling location, we create a PRBS7 waveform from the unequalized pulse response. With Samples per Symbol equal to 16, we can calculate the baud-rate phase error at 16 different phase locations and compare the sum of each phase error.

```
%Create PRBS waveform
prbsOrder = 7;
```

```matlab
dataPattern = prbs(prbsOrder,2^prbsOrder-1)-0.5;
waveform = pulse2wave(pulse_uneq,dataPattern,SamplesPerSymbol);

wavelength = length(waveform);

%Initialize phase error sum
PhaseErrorSum = zeros(1,SamplesPerSymbol);
for ii = 1:SamplesPerSymbol
    %Index of sample, i.e. "k"
    ndx_k = ii:SamplesPerSymbol:wavelength;

    %Index of prior sample, i.e. "k-1"
    ndx_km1 = mod(ndx_k-SamplesPerSymbol-1,wavelength)+1;

    %Calculate baud-rate type A phase eeror
    PhaseError = waveform(ndx_k).*sign(waveform(ndx_km1)) - waveform(ndx_km1).*sign(waveform(ndx_

    %Summarize phase error with its sum
    PhaseErrorSum(ii) = sum(PhaseError);
end

%Determine the index with the minimum phase error sum
[~,minPhaseErrorSumNdx]=min(abs(PhaseErrorSum));

%View the phase error sum for each sample index
figure,
plot(1:SamplesPerSymbol,PhaseErrorSum,'-o',...
    minPhaseErrorSumNdx*[1 1],[min(PhaseErrorSum),max(PhaseErrorSum)],'--')
legend('Sum of phase error','Phase with the minimum phase error')
xlabel('Sample position within eye')
grid on
```
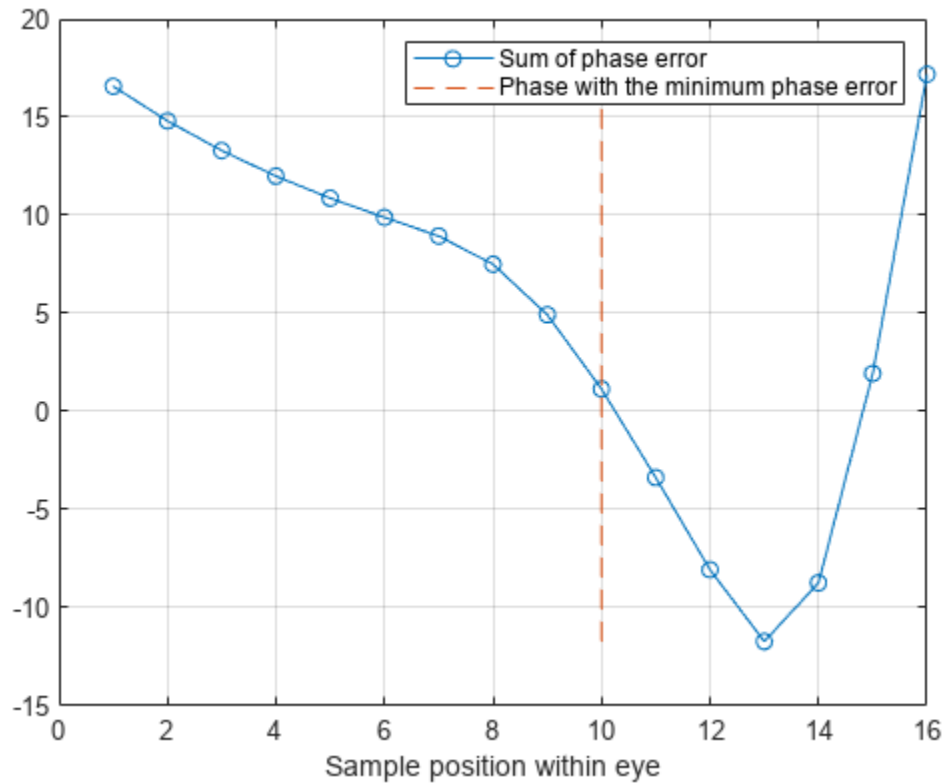
Now let's visualize the waveform eye diagram and mark the baud-rate type-A phase position with the minimum phase error.

```
%Visualize eye and highlight baud-rate type A sample lock location
figure,
    ndx = minPhaseErrorSumNdx:SamplesPerSymbol:wavelength;
    plot(1:SamplesPerSymbol,reshape(waveform,SamplesPerSymbol,[]),'b',...
        minPhaseErrorSumNdx,waveform(ndx),'ro')
    xlabel('Sample position within eye')
    ylabel('V')
    grid on
```
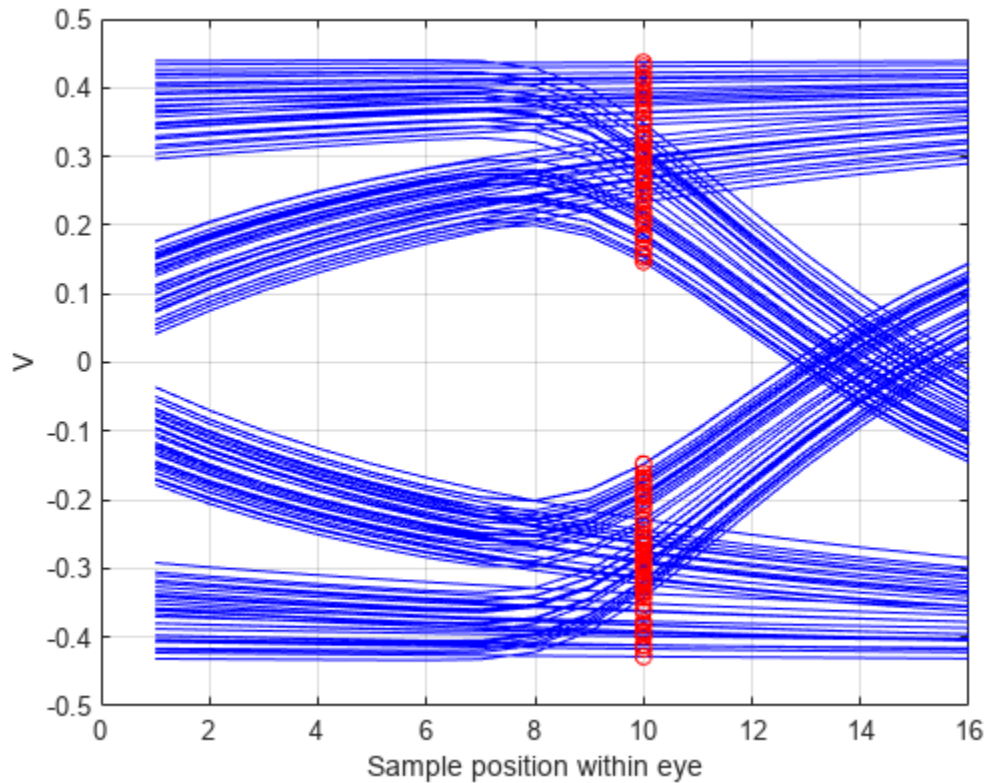
Simulink time-domain simulation

With some intuition about the baud-rate type-A phase detector, export the SerDes system to Simulink.

`exportToSimulink(systemBaudRate);`

Change the receiver Bits to ignore to 10000

- Open the Configuration block and click the "Open SerDes IBIS-AMI Manager" button
- On the Export tab, change the Rx Bits to Ignore from 1000 to 10000

Run the model and note the similarities and differences between the Statistical Eye and the Time Domain Eye.
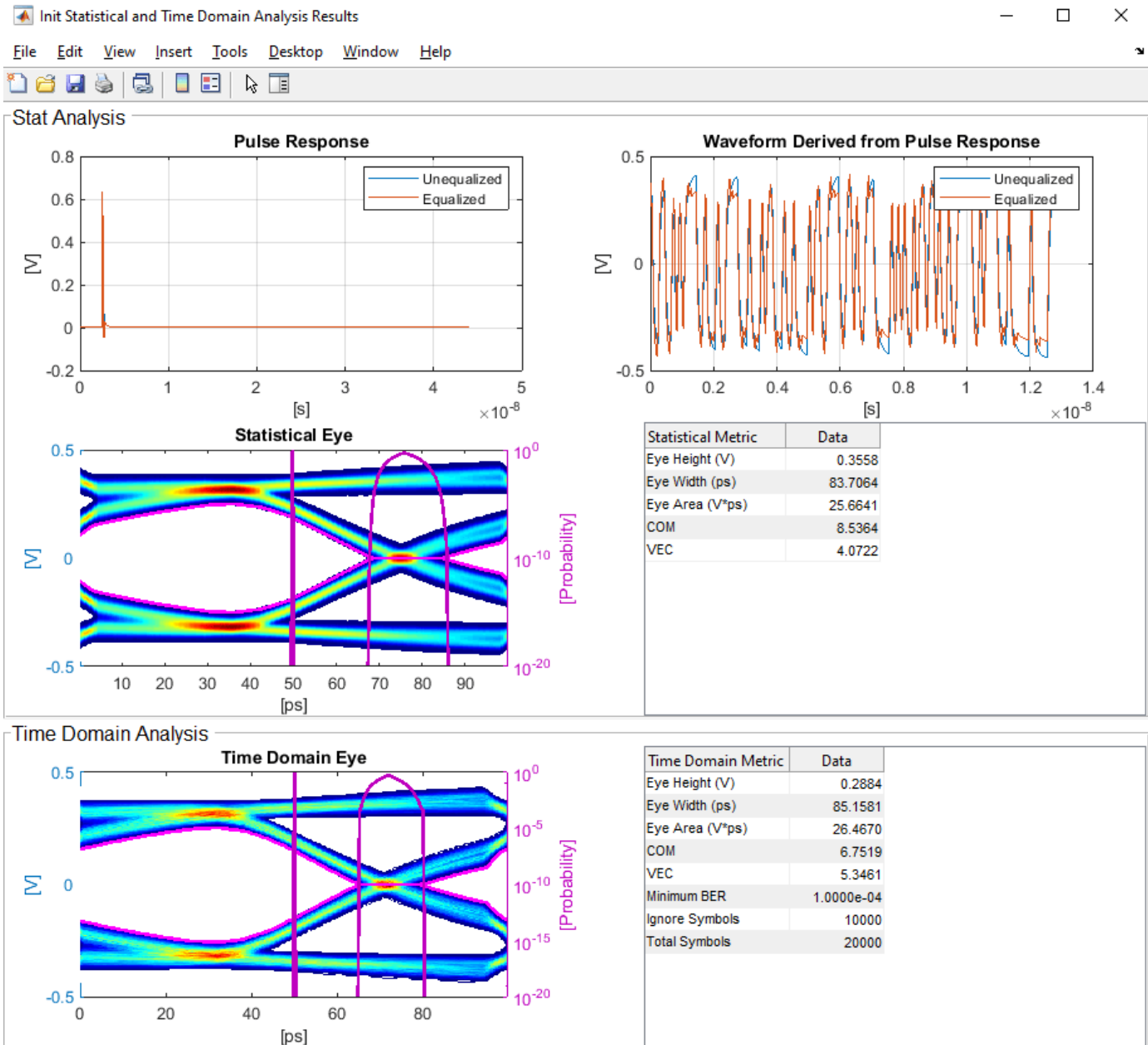
Down sides to the baud-rate phase detector

Counterintuitively, the buad-rate phase detector requires some residual inter-symbol interference (ISI) to operate properly. If the channel is really clean or if a clock pattern is used, the baud-rate phase detector does not work. To observe this, open the Stimulus block and change the Waveform creation method to "Symbol Pattern" with a "[0 1]" symbol pattern. Additionally, include a 65 ps delay to initialize the phase to the edge of the eye.

Block Parameters: Stimulus ✕

Stimulus

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

Number of symbols   20000

Recommended simulation stop time (s): 2e-06

☑ Use recommended simulation stop time

Waveform creation method   Symbol Pattern

Symbol pattern   [0 1]

Sub symbol time delay (s)   65e-12

OK   Cancel   Help   Apply
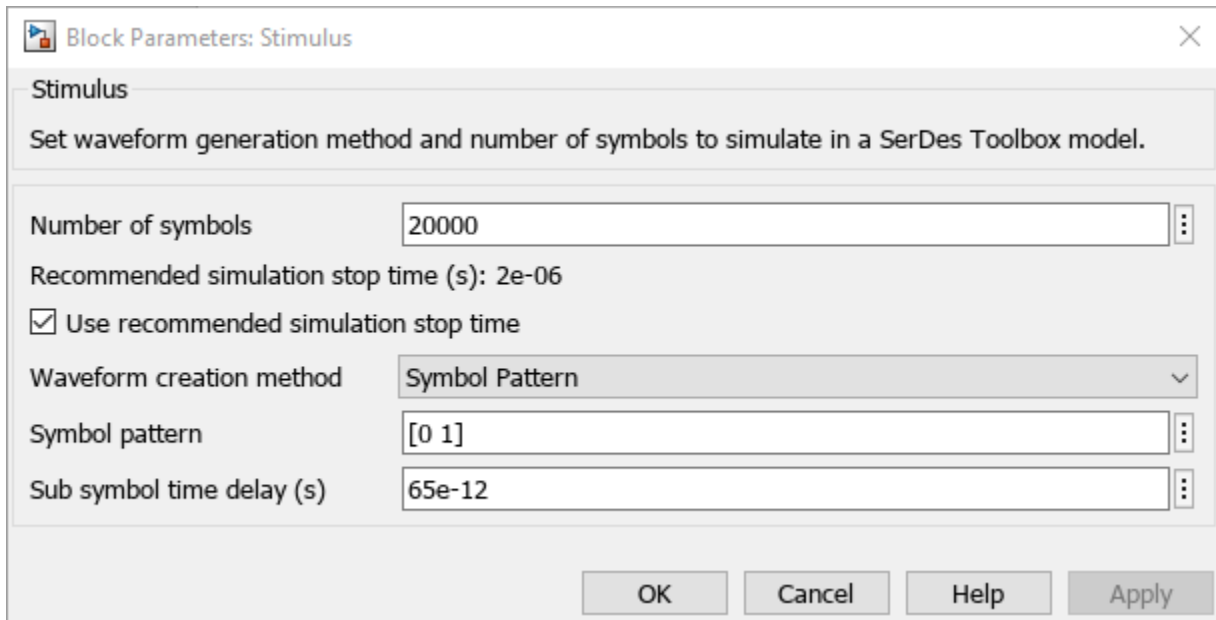
In addition to the phase detector being misaligned, the clock pattern invalidates the DFE adaptation assumption of random data.

Observe how the clock within post-simulation time domain analysis eye is nowhere near the optimal position. This shows how the baud-rate phase detector needs some amount of residual ISI for proper operation.
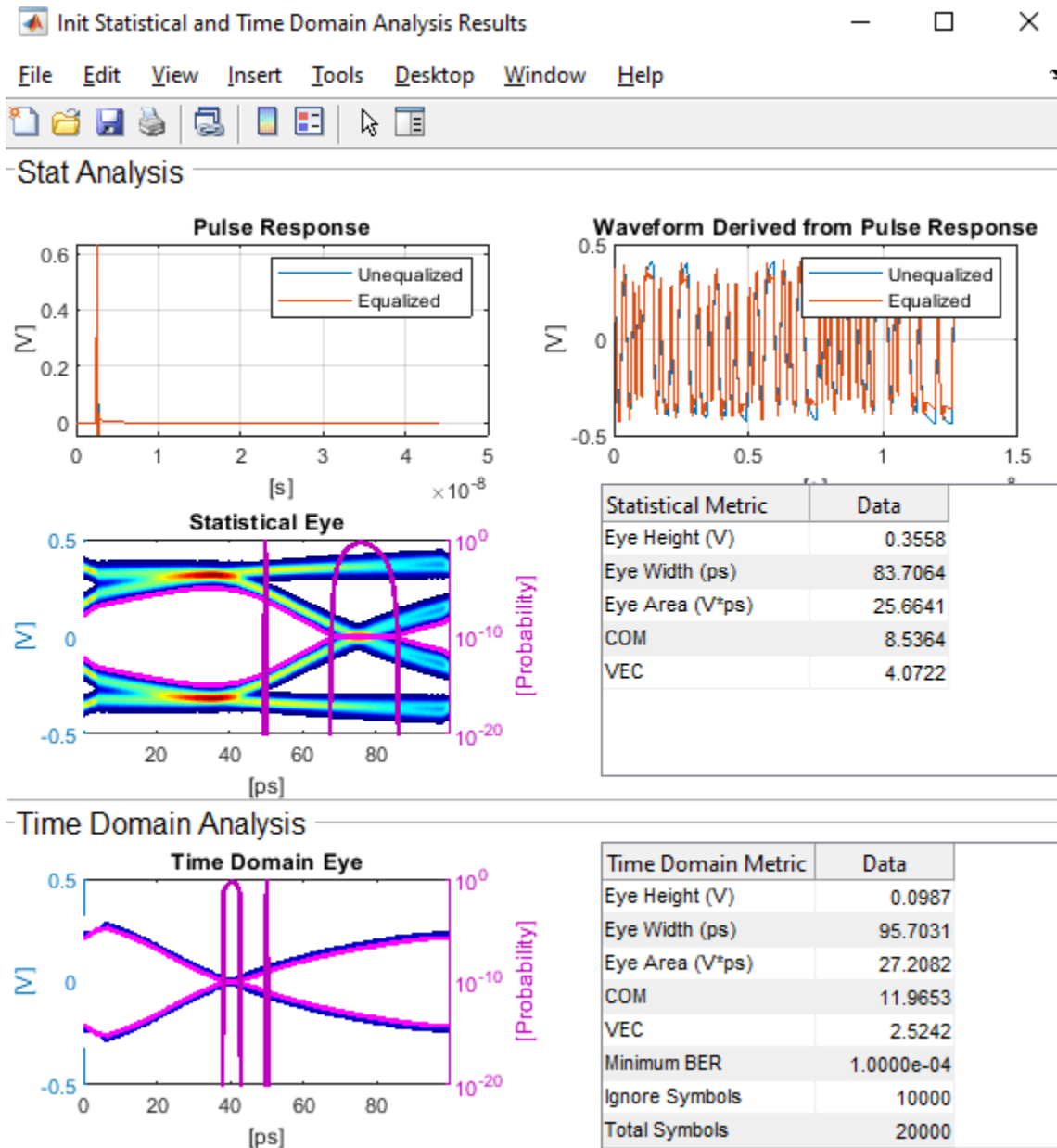
Summary

Analog-based SerDes typically utilize bang-bang phase detectors while ADC-based SerDes utilize baud-rate phase detectors. Each type of phase detector utilizes different types of inputs, the bang-bang phase detector uses two binary samplers 1/2 symbol time apart and the baud-rate phase detector uses a voltage measurement (with several bits of precision) and the resolved symbol value. Each phase detector determines a phase error so that an good phase location can be used for recovering the sent information.

References

- D. Telian, M. Steinberger, B. Katz, "New SI Techniques for Large System Performance Tuning" DesignCon 2016 Conference, San Jose, CA.

- Tyshchenko, O. (2011). Clock and Data Recovery for High-speed ADC-based Receivers (Doctoral dissertation).
- Kurt H. Mueller and Markus Muller. Timing Recovery in Digital Synchronous Data Receivers. IEEE Transactions on Communications, 24(5):516–31, May 1976.

See Also

More About

- "Clock and Data Recovery in SerDes System" on page 1-3

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')
```
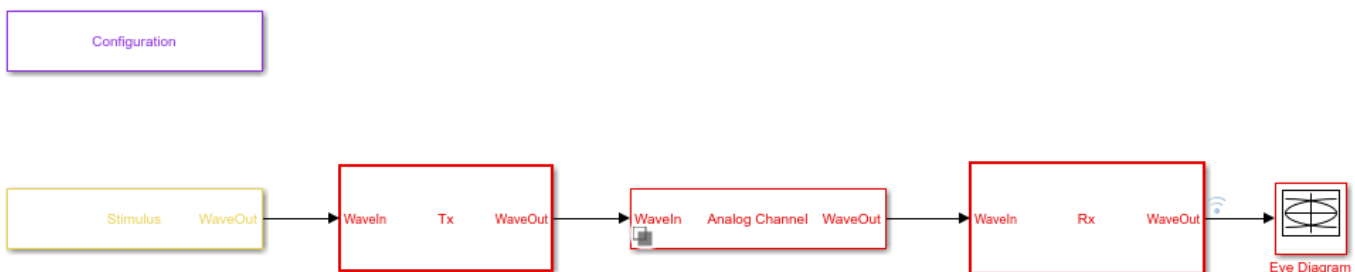

Copyright 2019 The MathWorks, Inc.

- 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.

Event Listener

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. **Note:** The Time Domain waveform will not be correct until you wire the Constant block for the new parameter **ConfigSelect** in the canvas for the FFE. You will see how to do this in the next section.
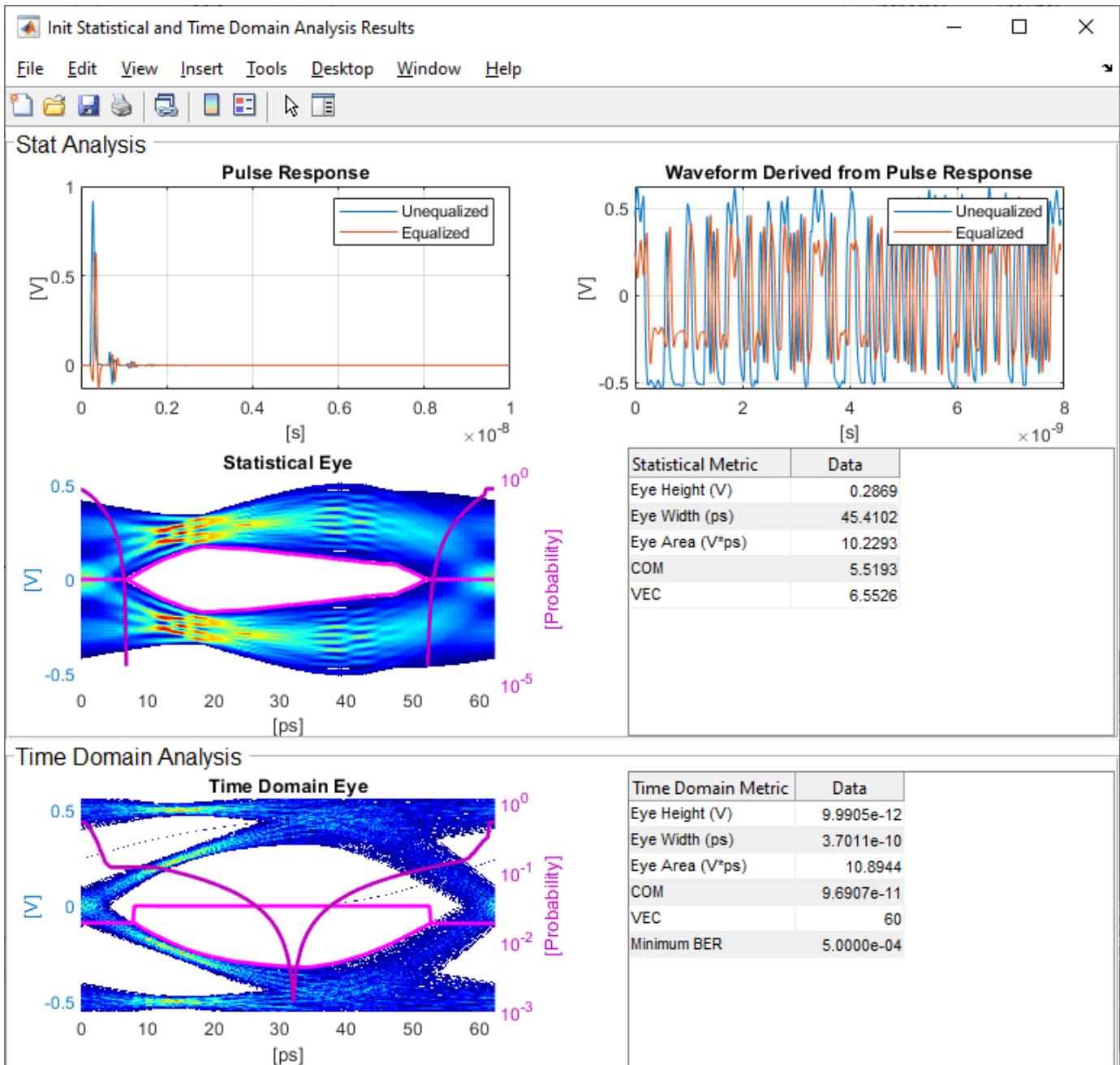
Next, set the **Current value** of the **ConfigSelect*** parameter to `User Defined`. This corresponds to user-defined tap weights: Pre = 0.000, Main = 1.000 and Post = 0.000.

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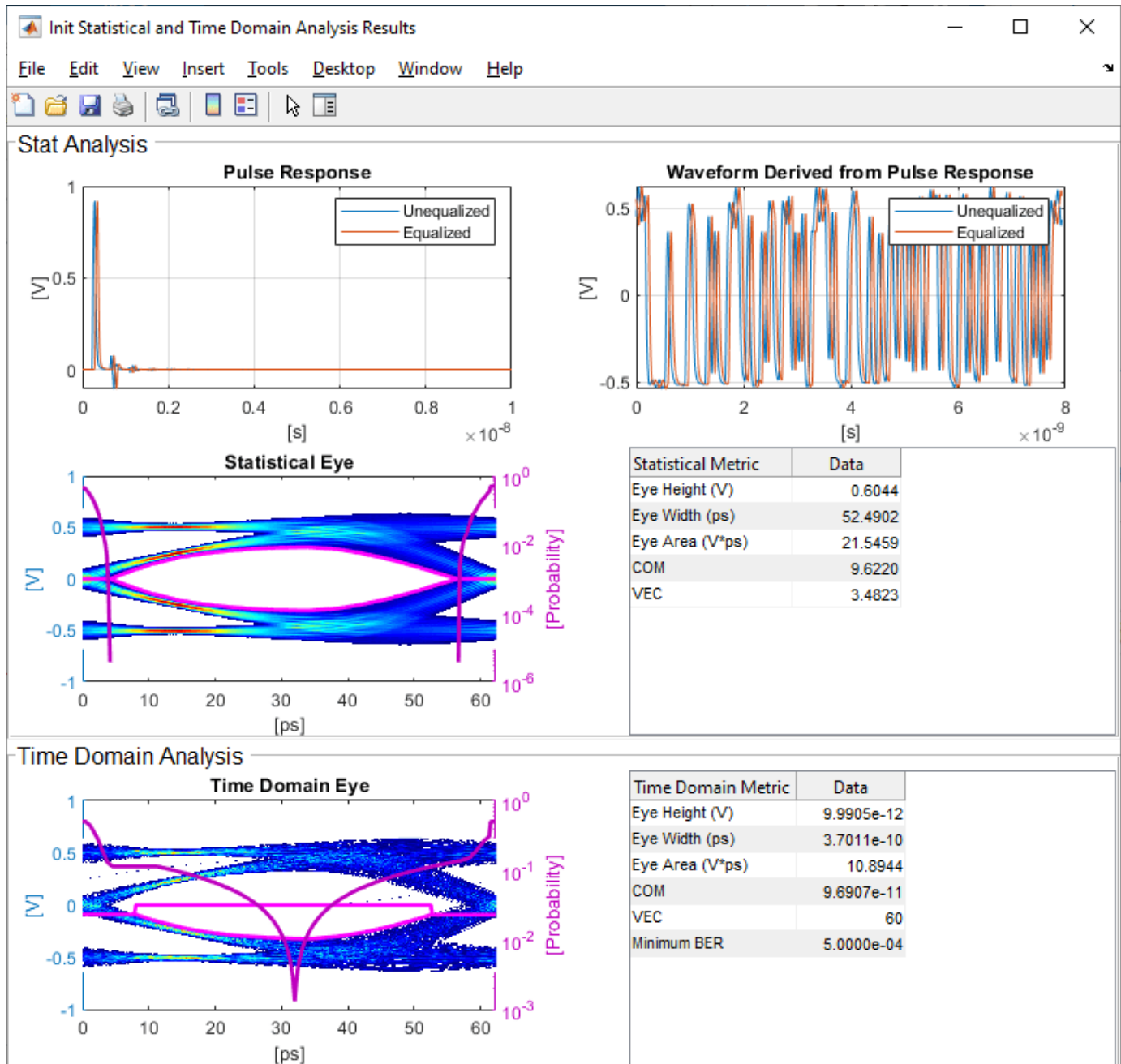
Try different values of **ConfigSelect*** to verify proper operation. The statistical eye opens and closes based on the amount of equalization applied by the FFE. How much the eye changes, and the tap values that create the most open eye varies based on the loss defined in the Analog Channel block.

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.
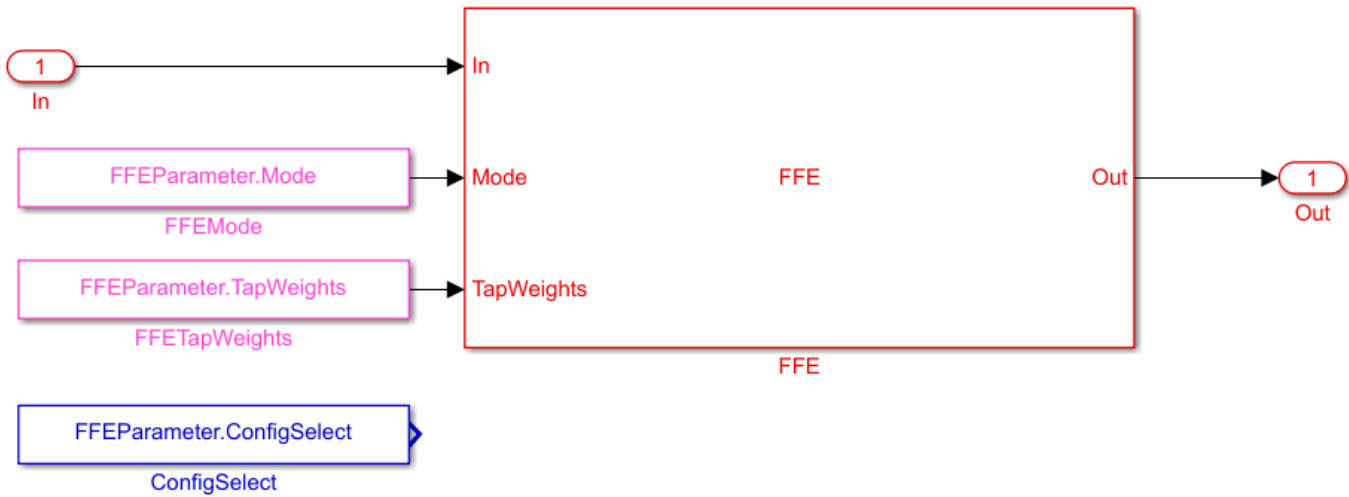
- You can see that a Constant block was automatically added by the IBIS-AMI manager to the canvas with the **Constant value** set to `FFEParameter.ConfigSelect`.
- Add a MATLAB Function block to the canvas from the Simulink/User-Defined library.
- Rename the MATLAB Function block to `PCIe4FFEconfig`.
- 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.
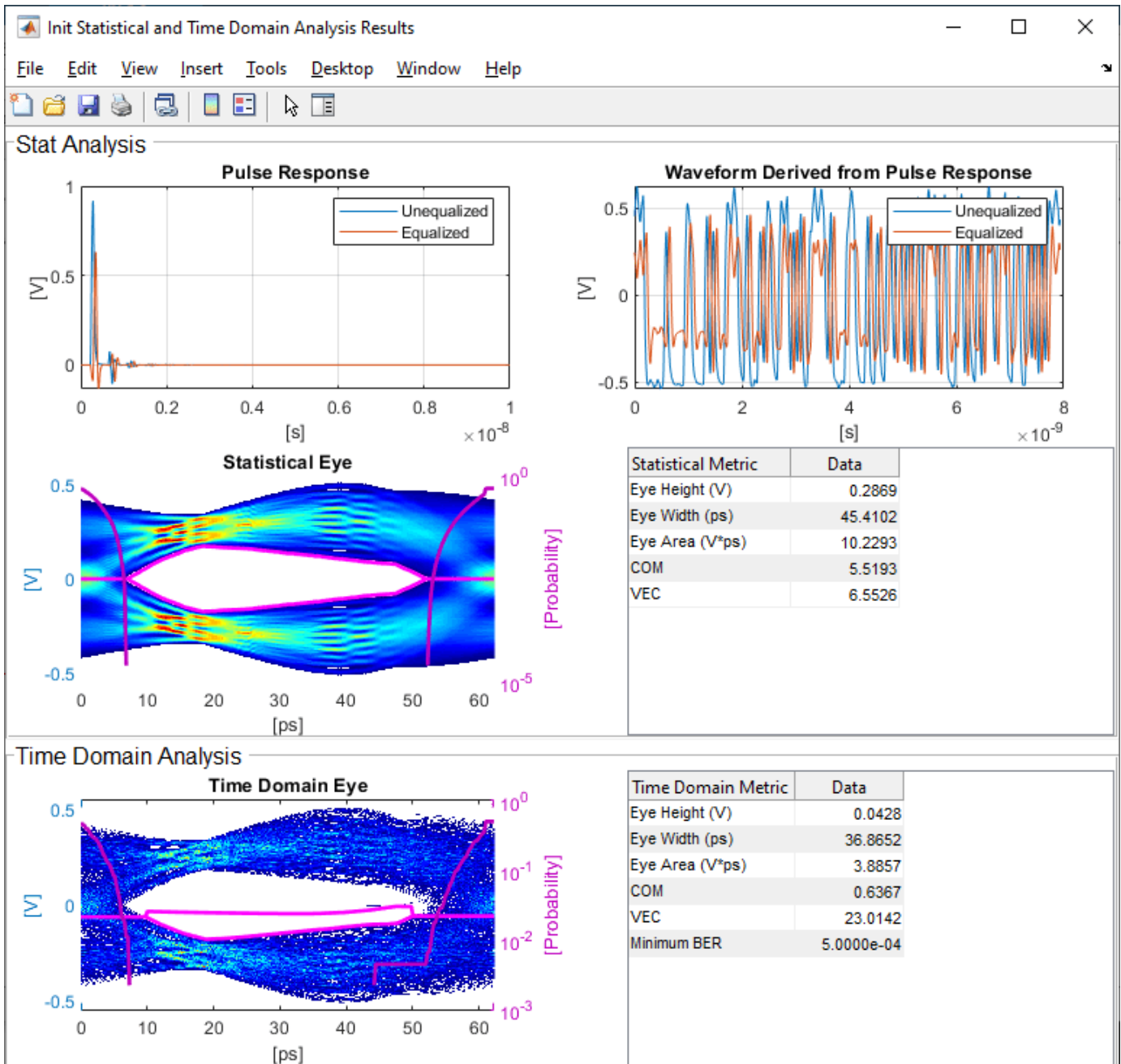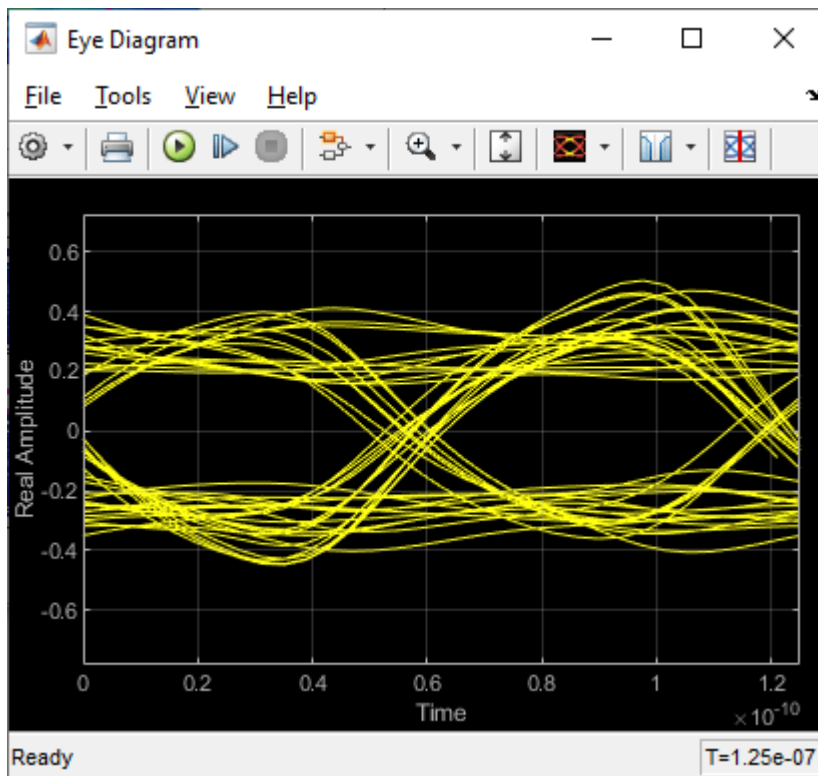
Next, set the **Current value** of the **ConfigSelect*** parameter to `User Defined`. This corresponds to user-defined tap weights: Pre = 0.000, Main = 1.000 and Post = 0.000. Observe how the output waveform changes.
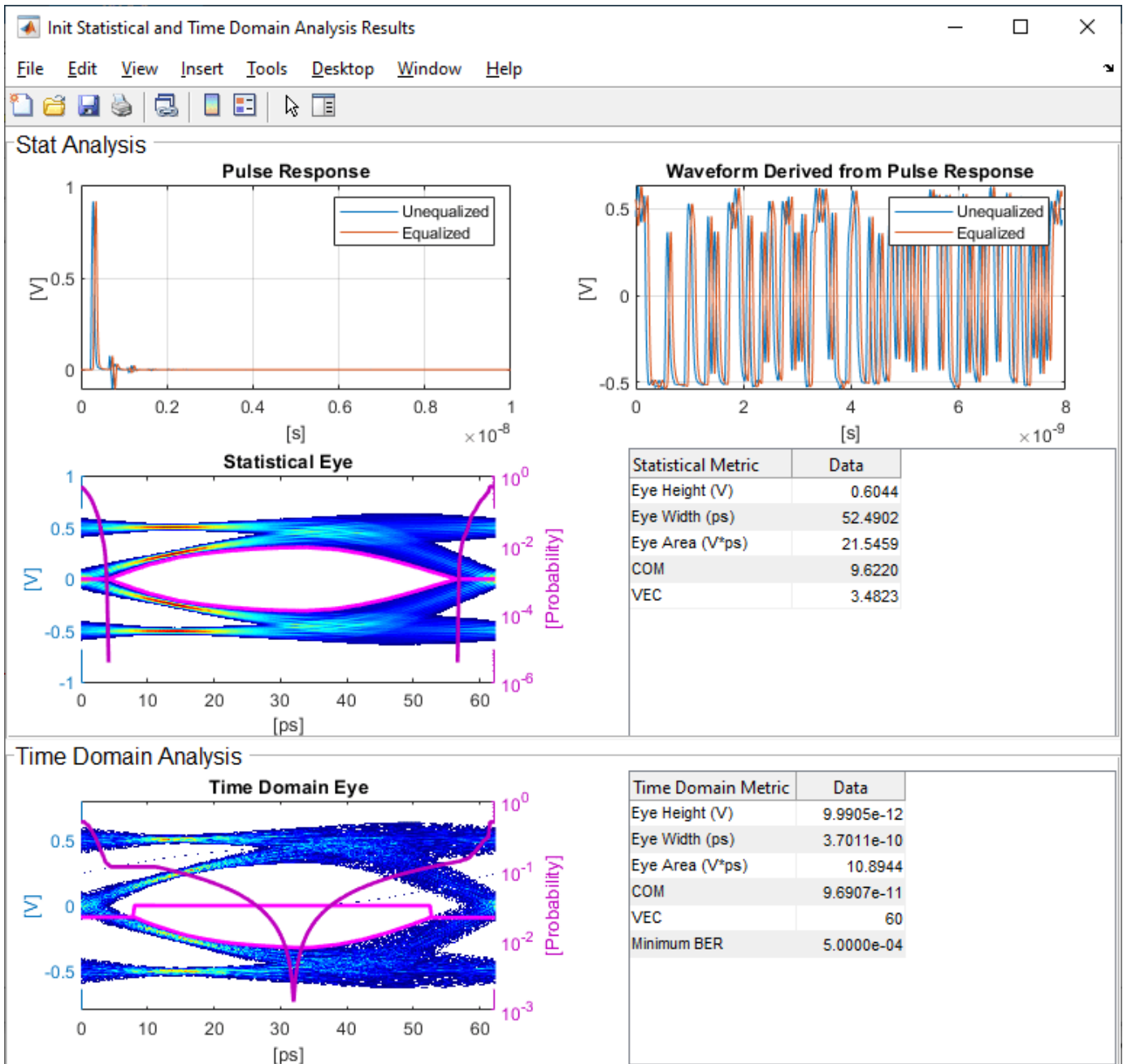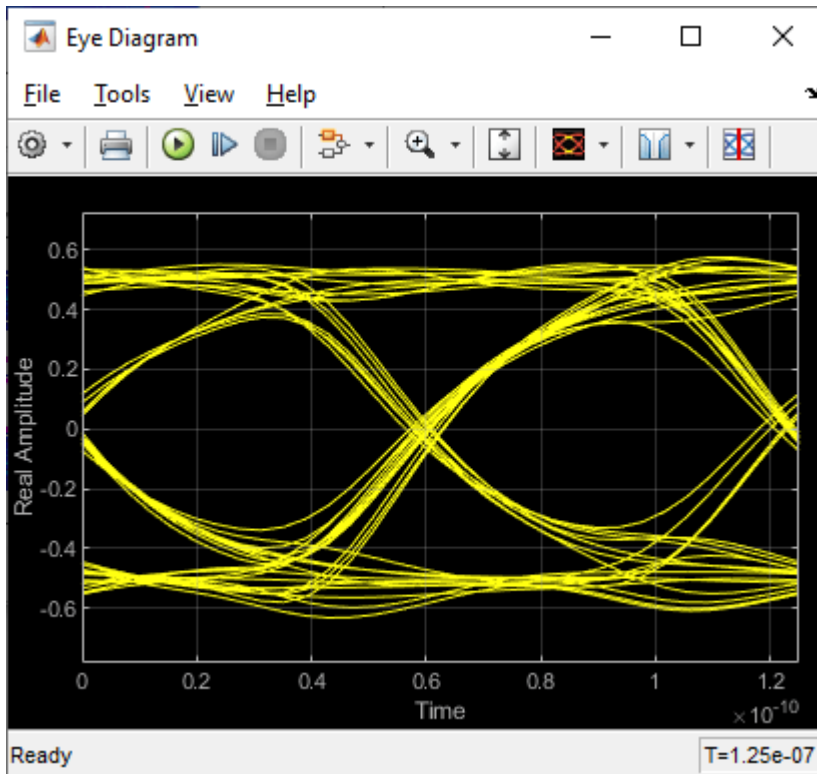
Try different values of **ConfigSelect*** to verify proper operation. The time-domain eye opens and closes based on the amount of equalization applied by the FFE. How much the eye changes, and the tap values that create the most open eye varies based on the loss defined in the Analog Channel block.

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® to use a MATLAB® function block or other Simulink library blocks. You will see how the implementation of a receiver gain or attenuation stage is controlled by an IBIS-AMI parameter, and this example provides 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 5-28.

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.`
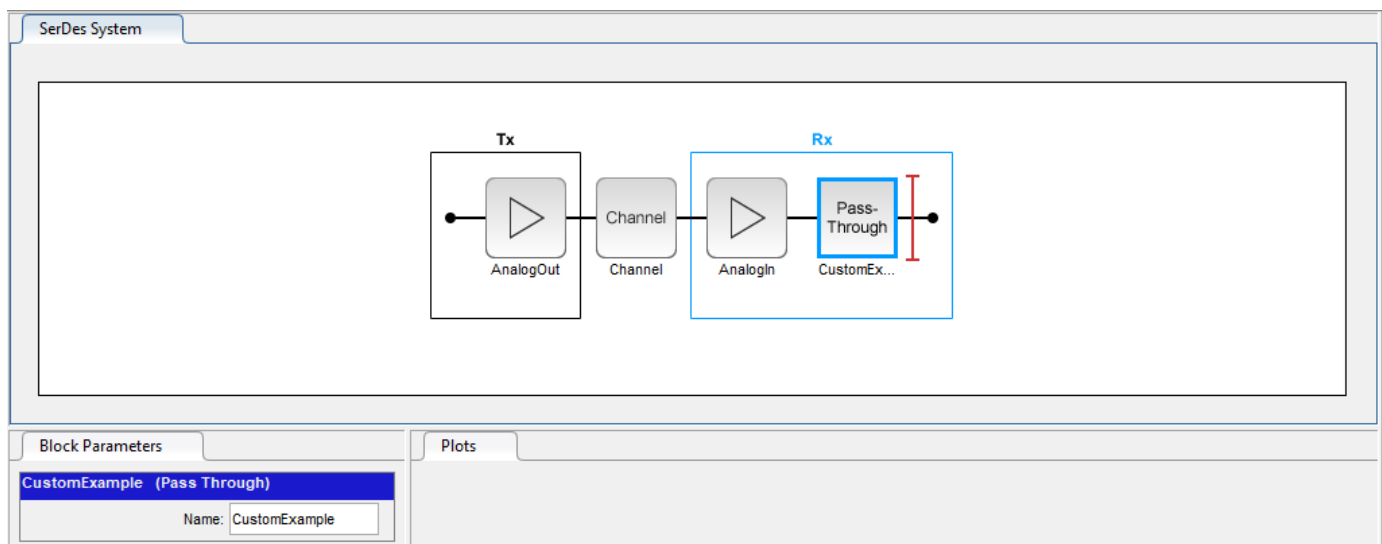
Export the SerDes system to Simulink.
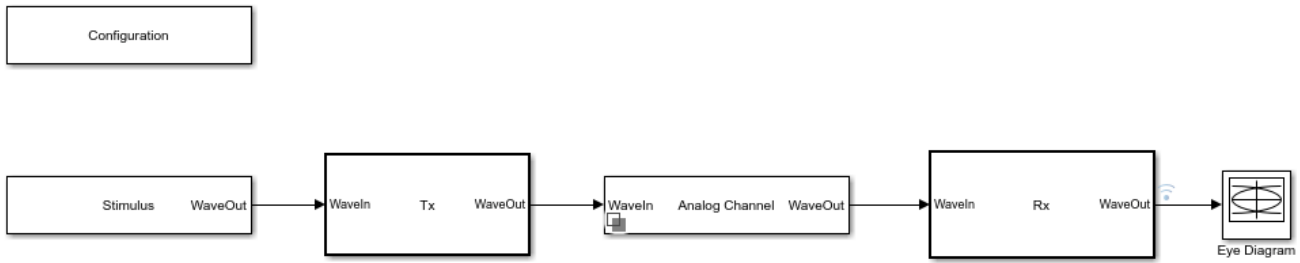
Add AMI Parameter to Control Gain

Double click on the Rx block to look inside the Rx subsystem and open the SerDes IBIS-AMI Manager dialog box.
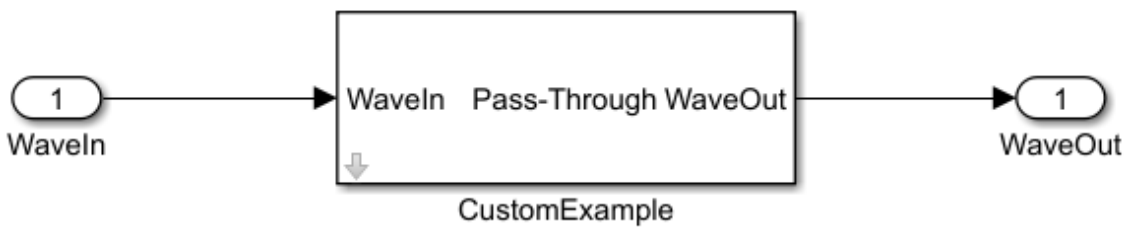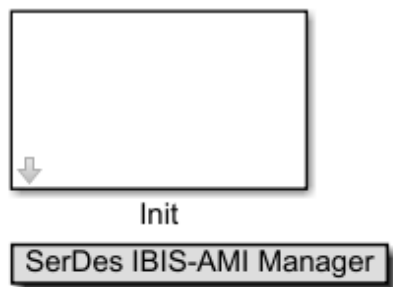
In the **AMI-Rx** tab, select the **CustomExample** node. Click on the **Add Parameter** button and set the variables:

- **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.

Confirm settings and click **OK**.

You will see a parameter automatically generated on the canvas as shown below.
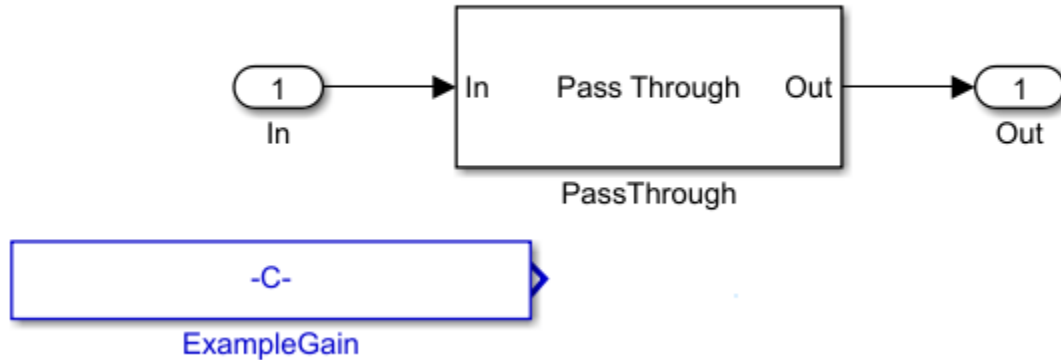
Change PassThrough to a MATLAB Function Block

You can create a `MATLAB` function block and add code to use the **ExampleGain** parameter as a modifier to the **In** signal. To illustrate the workflow, this example will show how to implement a gain (using multiplication) but any `MATLAB` function may be implemented for your system.

```
function out = fcn(in,ExampleGain)

gainSignal = ExampleGain*in;

out=gainSignal;
```

Then you can delete the PassThrough block, and wire up the `MATLAB` block with input signals **In, ExampleGain** and output signal **Out** as shown:
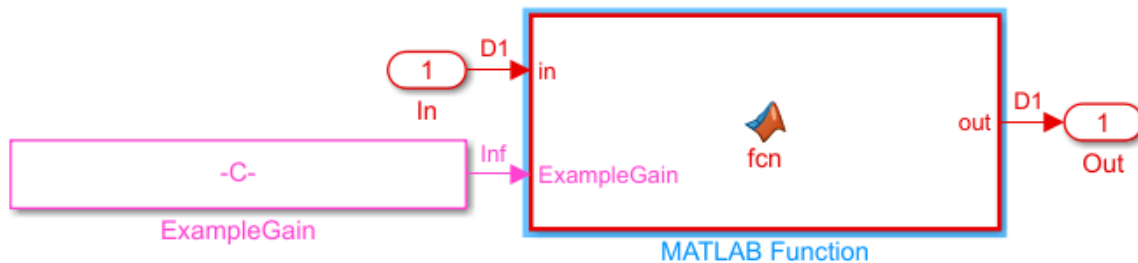
Remember to go back to the Rx subsystem, double-click on Init and click the button **Refresh Init**. You can see the affect of the value of the parameter ExampleGain by opening the IBIS AMI Manager and changing the Current value of **ExampleGain** to `0.8`.
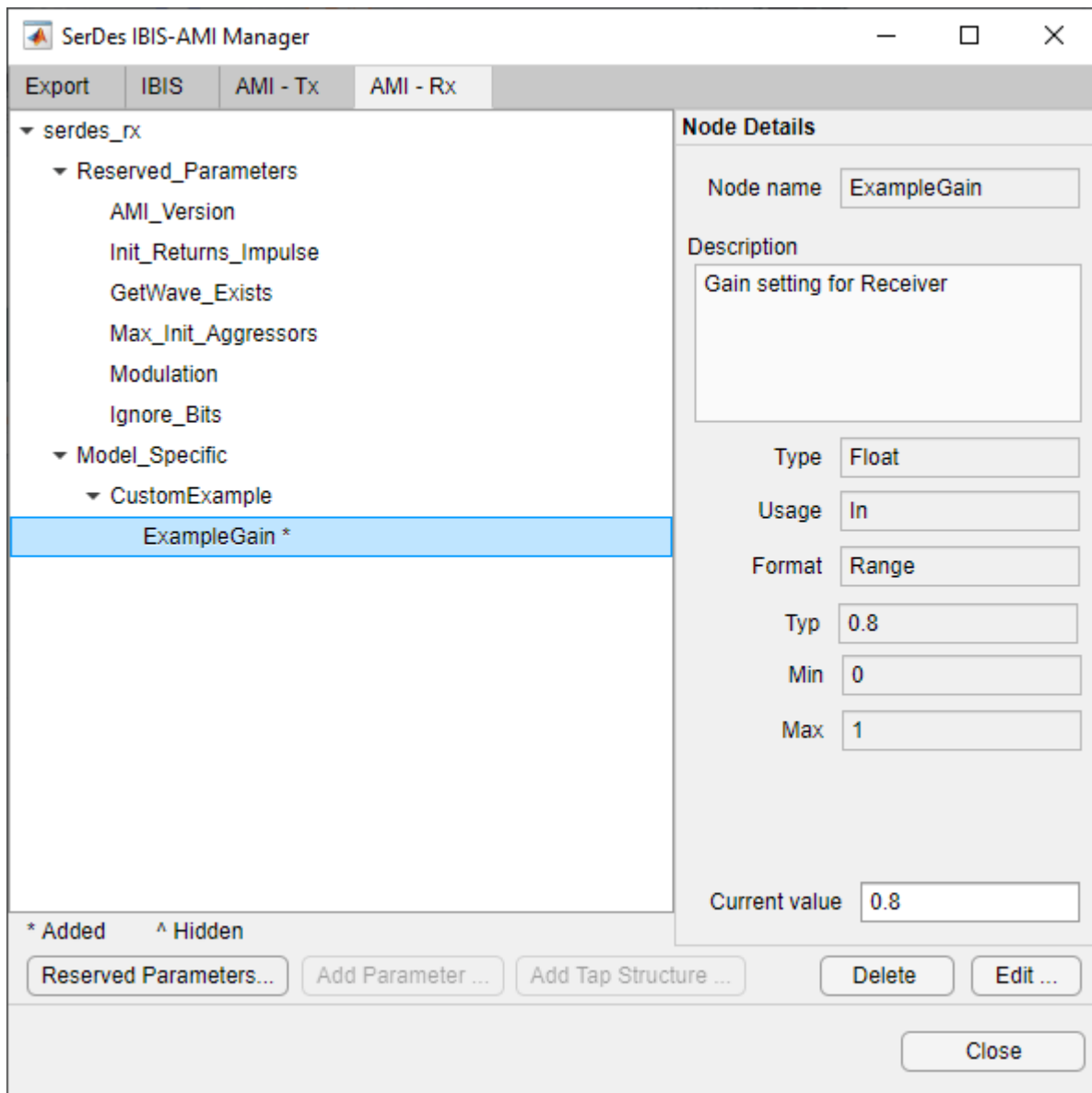
Run the simulation and observe the results.
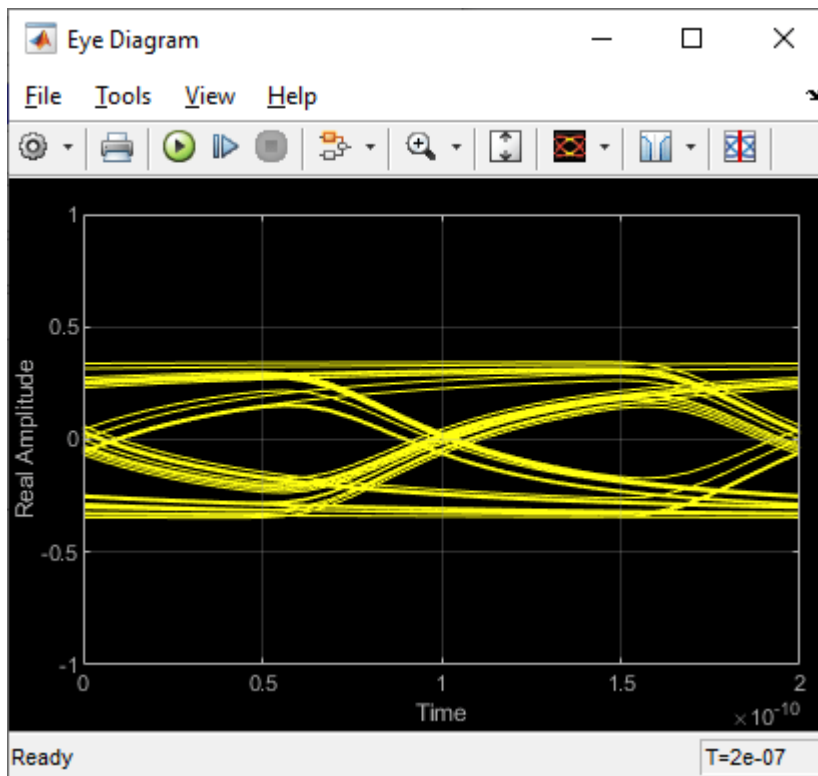
Change the Current value of **ExampleGain** to `1.0` and re-run the simulation to confirm **ExampleGain** parameter is modifying the Receiver signal.
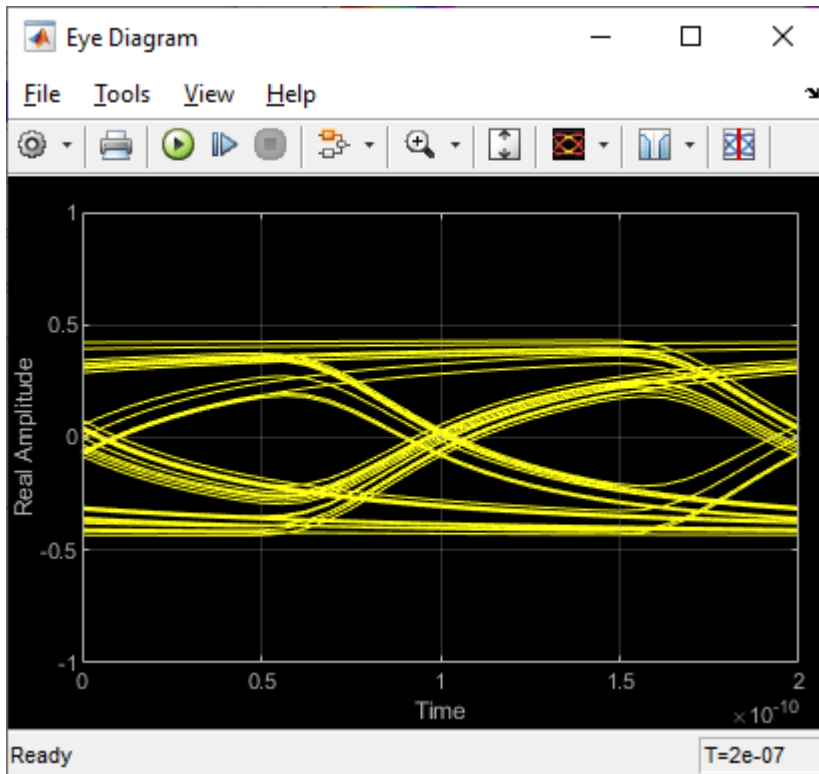
These steps showed you how to implement an AMI parameter called **ExampleGain** using a MATLAB function block in your system. You can also use built-in blocks to customize a PassThrough block as explained in the section "Change PassThrough to Gain Block or Other Built-in Block."

Change PassThrough to Gain Block or Other Built-in Block

Another way to configure a custom PassThrough block for your model is to use a built-in block. For example, a Gain block can be added within the PassThrough block. Instead of creating a MATLAB function block, look under the mask of the "CustomExample" block after the parameter **ExampleGain** is created from the steps in section "Add AMI Parameter to Control Gain" above:
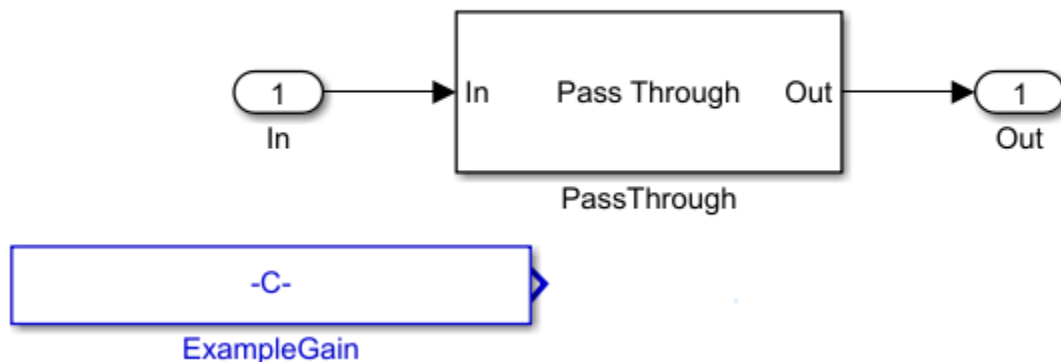
Delete the parameter **ExampleGain**. You should see the canvas now looks like the default `serdes.PassThrough` System Object:
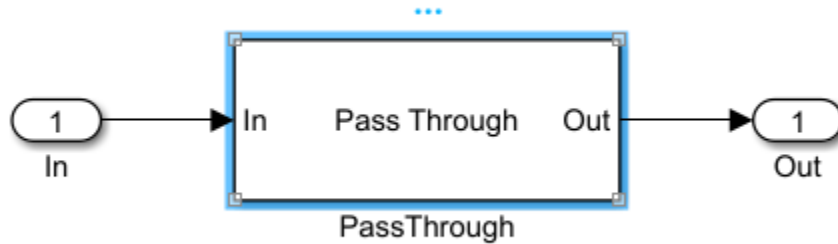
Next, delete the MATLAB System block that points to the `serdes.PassThrough` System Object:
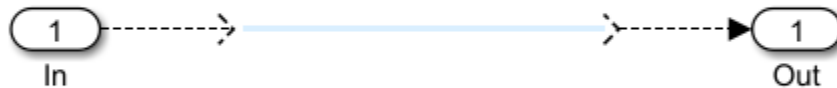
Add a Gain block from the Simulink > MathOperators library and connect the Gain block between the input and output ports:

Note: While this example uses a Gain block to illustrated workflow, you can use any built-in block (as well as a `MATLAB` function).
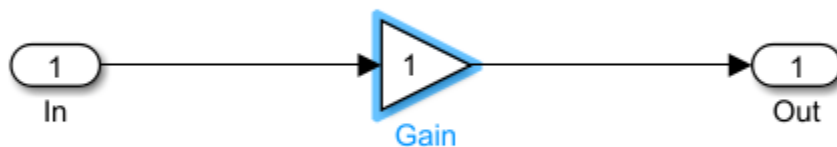
Connect Block Parameters of Gain Block to Added AMI Parameter

Constants are represented as Simulink parameters. Double click the Gain block to open the Block Parameters dialog box. Set **Gain** value to `CustomExampleParameter.ExampleGain`.
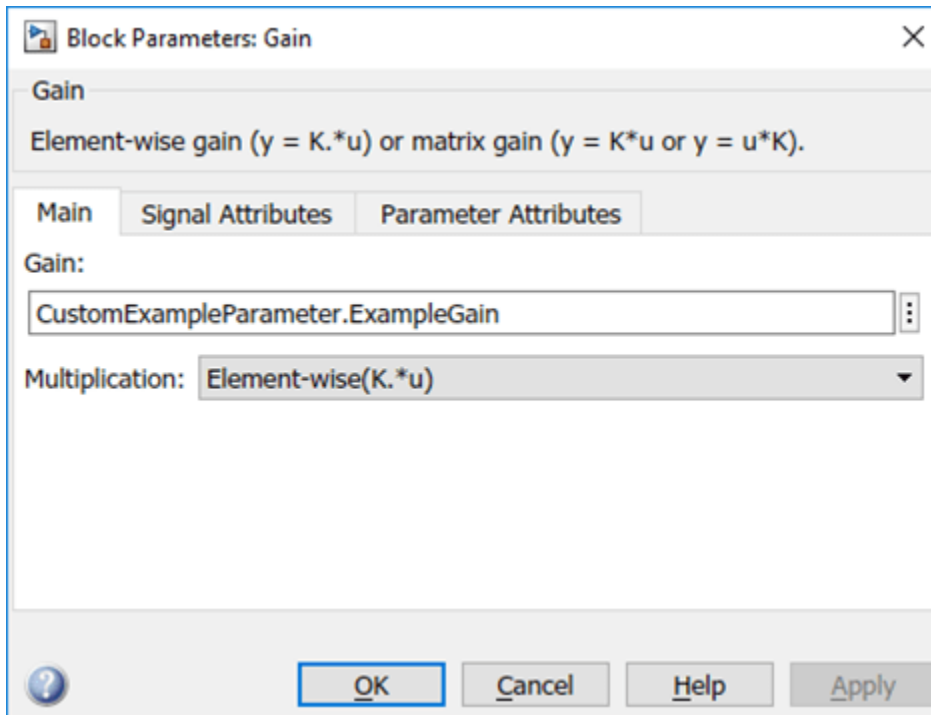
Update Code that Runs During Statistical Analysis

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;`

The **Custom user code area** should appear as below:

```
%% 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)
```

Save the changes.

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.
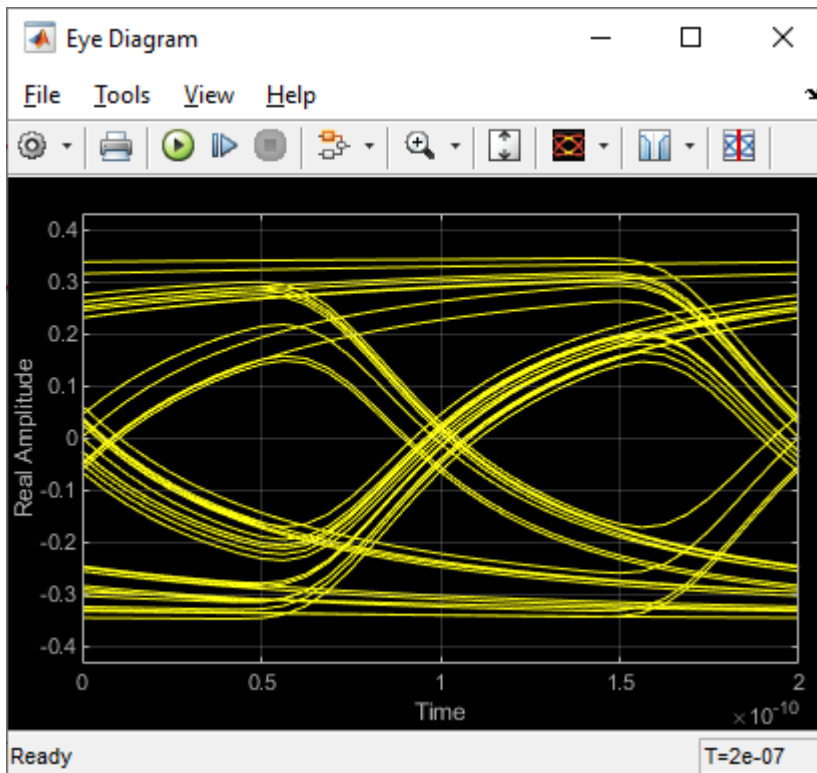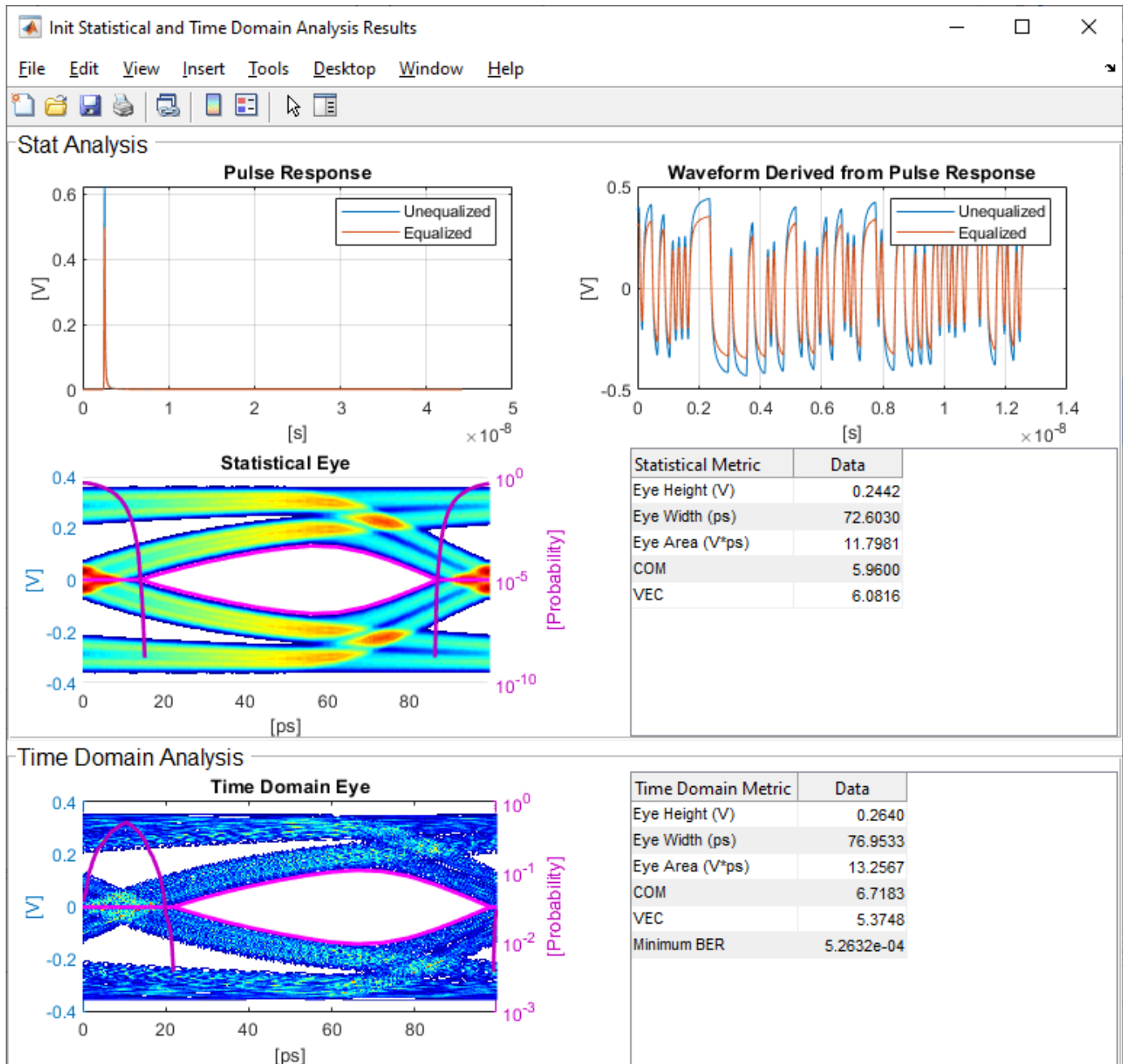
Change Gain Setting and Observe Change

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).

Init Statistical and Time Domain Analysis Results — □ ✕

File   Edit   View   Insert   Tools   Desktop   Window   Help

Stat Analysis

Pulse Response

Waveform Derived from Pulse Response

Statistical Eye

Statistical Metric	Data
Eye Height (V)	0.1221
Eye Width (ps)	72.6030
Eye Area (V*ps)	5.8991
COM	5.9600
VEC	6.0816

Time Domain Analysis

Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.1320
Eye Width (ps)	76.9533
Eye Area (V*ps)	6.6283
COM	6.7183
VEC	5.3748
Minimum BER	5.2632e-04
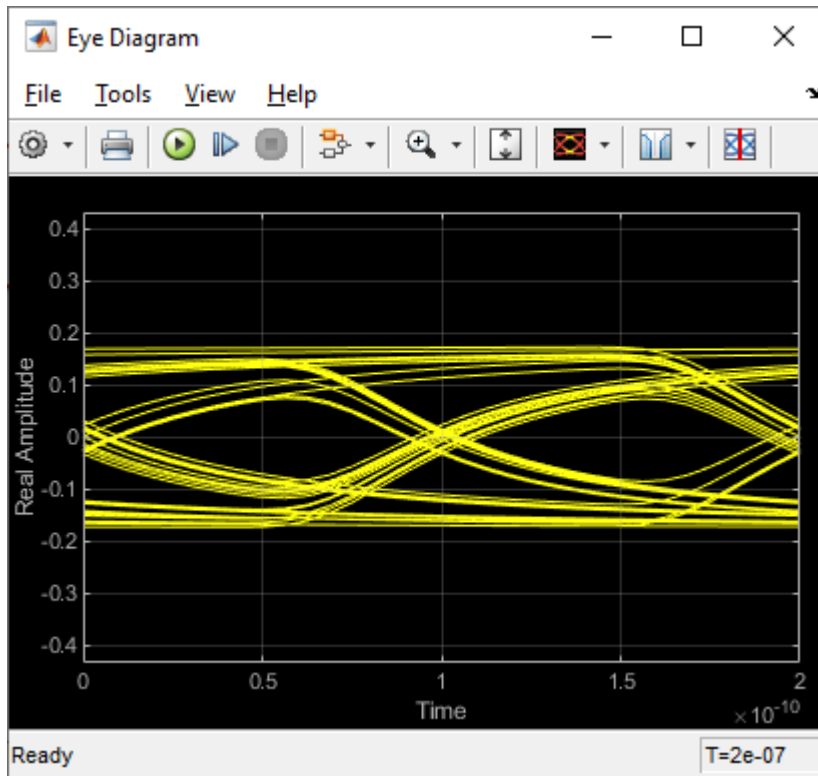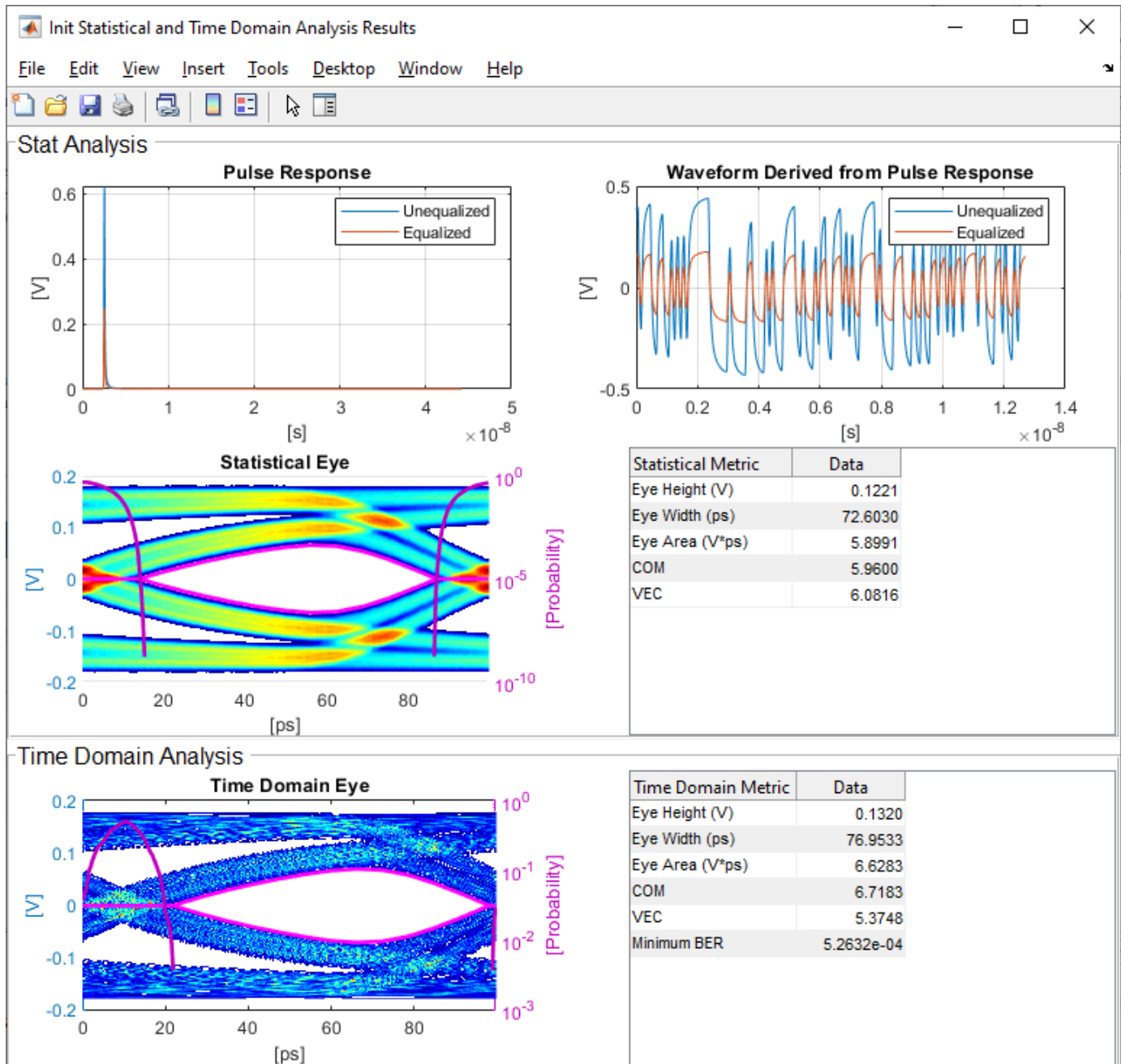
These steps showed you how to implement an AMI parameter called **ExampleGain** using a built-in block to customize a PassThrough block. You can also implement an AMI parameter using a `MATLAB` function block in your system as explained in the section "Change PassThrough to a MATLAB Function Block."

See Also

PassThrough | Configuration | **SerDes Designer**

More About

- "Implement Custom CTLE in SerDes Toolbox PassThrough Block" on page 5-28

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 5-14.

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`.
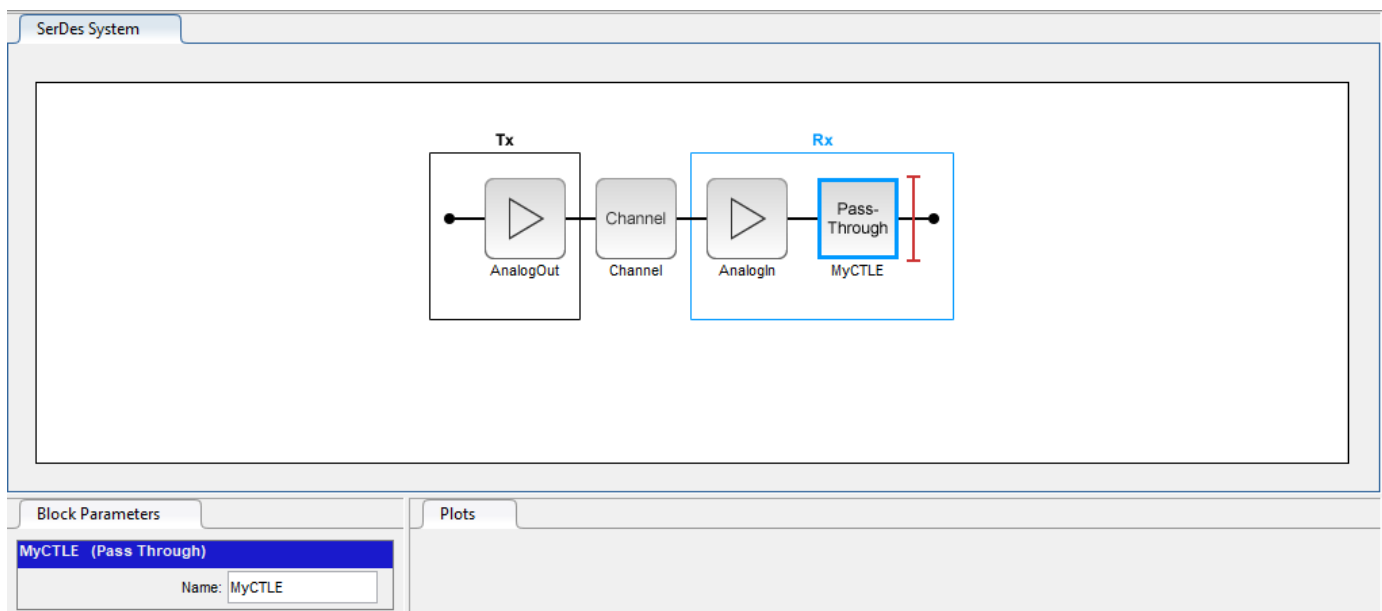
Export the SerDes system to Simulink.
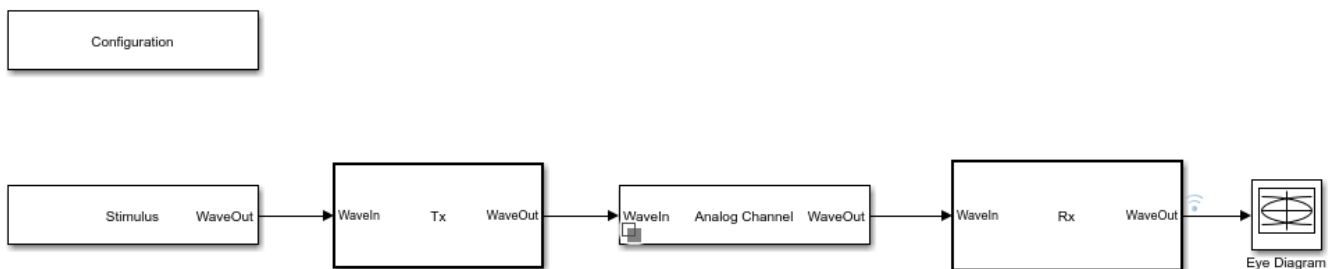
Modify PassThrough Block to Implement CTLE

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`.
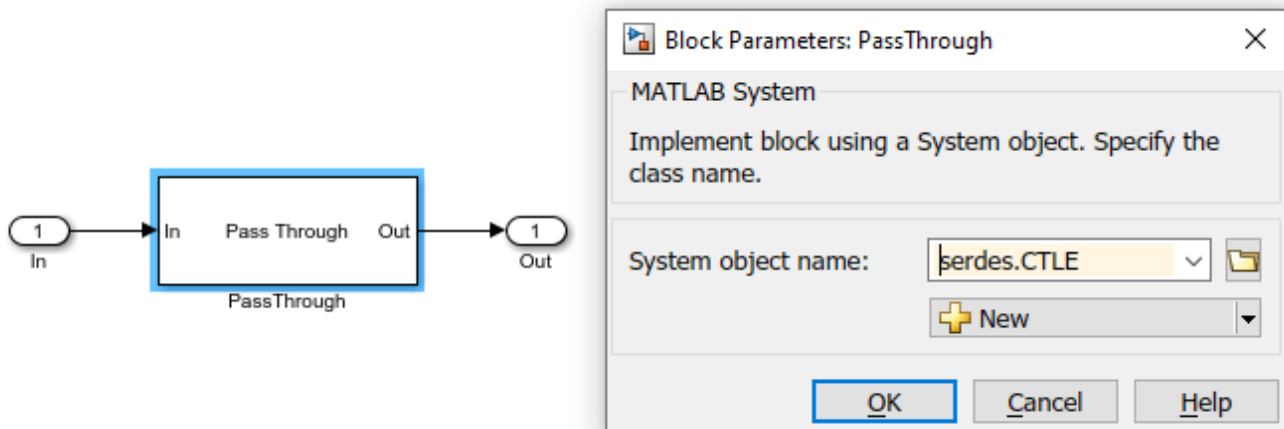
Click **OK** to save the changes, and you will see the block change from Pass Through to a CTLE:
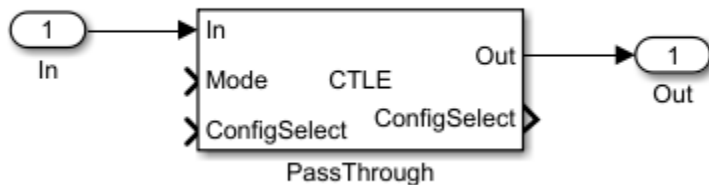
Note: You can use your own custom System object as well. For example, if you wanted to create a custom CTLE with a change in the adaptation algorithm:

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.
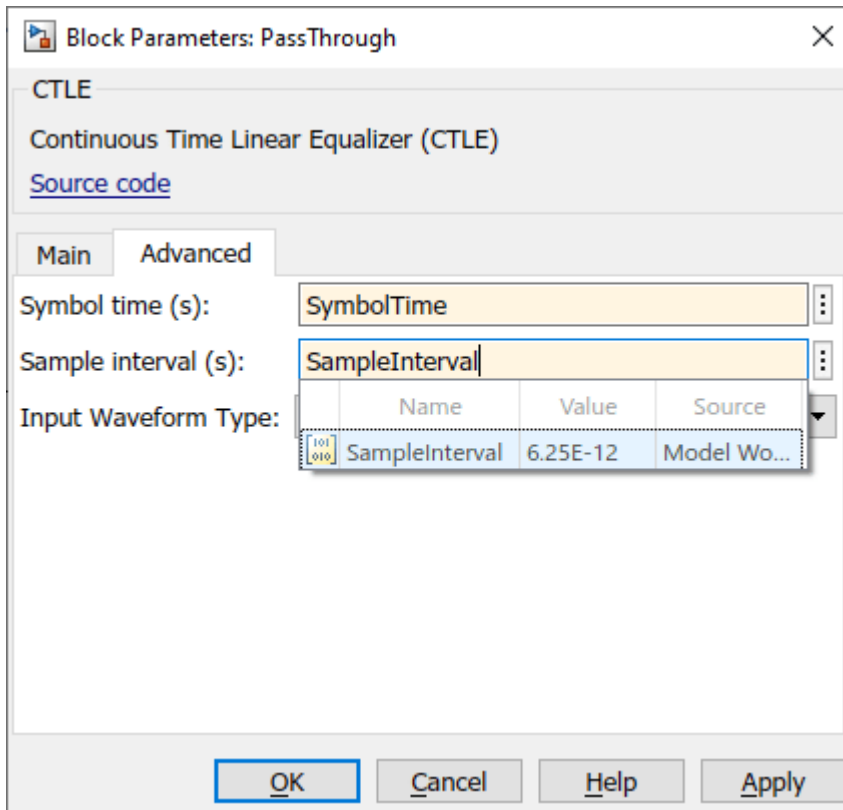
Add AMI Parameters to PassThrough Block

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`.

Press Ok to save the changes. You will see the parameter automatically added to the canvas:
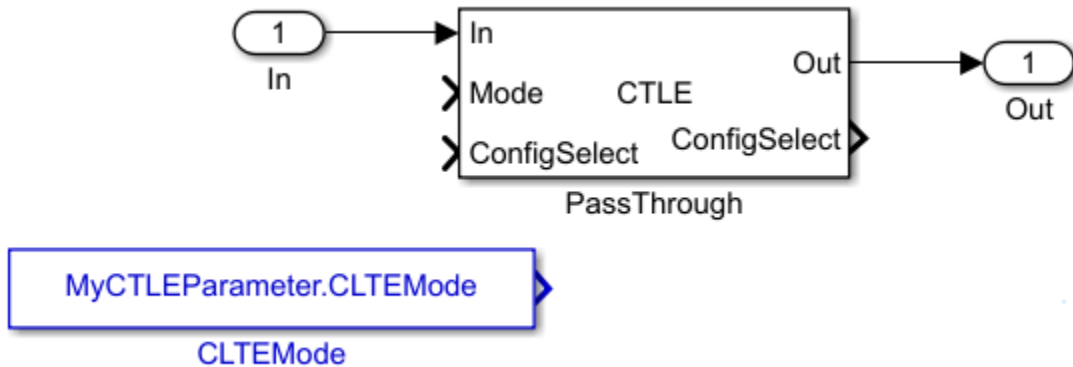
To add **CTLEConfigSelect** parameter, select the **MyCTLE** node again, click on the **Add Parameter** button and set the variables:

- **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`.

Press Ok to save the changes. Again, you will see the parameter automatically added to the canvas.

Implement AMI Parameters

Connect the blocks `MyCTLEParameter.CTLEMode` to the **Mode** input and `MyCTLESignal.CTLEConfigSelect read` to the **ConfigSelect** input of the PassThrough block. Connect the **ConfigSelect** output of the PassThrough block to the `MyCTLESignal.CTLEConfigSelect write` block.

For more information, see "Managing AMI Parameters" on page 6-2.
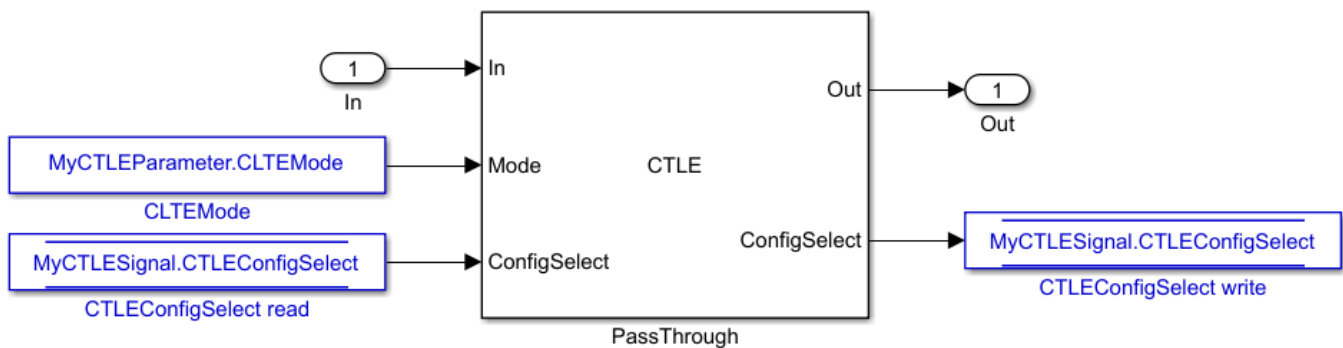
You can double-click on the blocks to confirm connectivity. For example double click on the block `MyCTLESignal.CTLEConfigSelect read` to confirm connectivity of the Data Store Read:
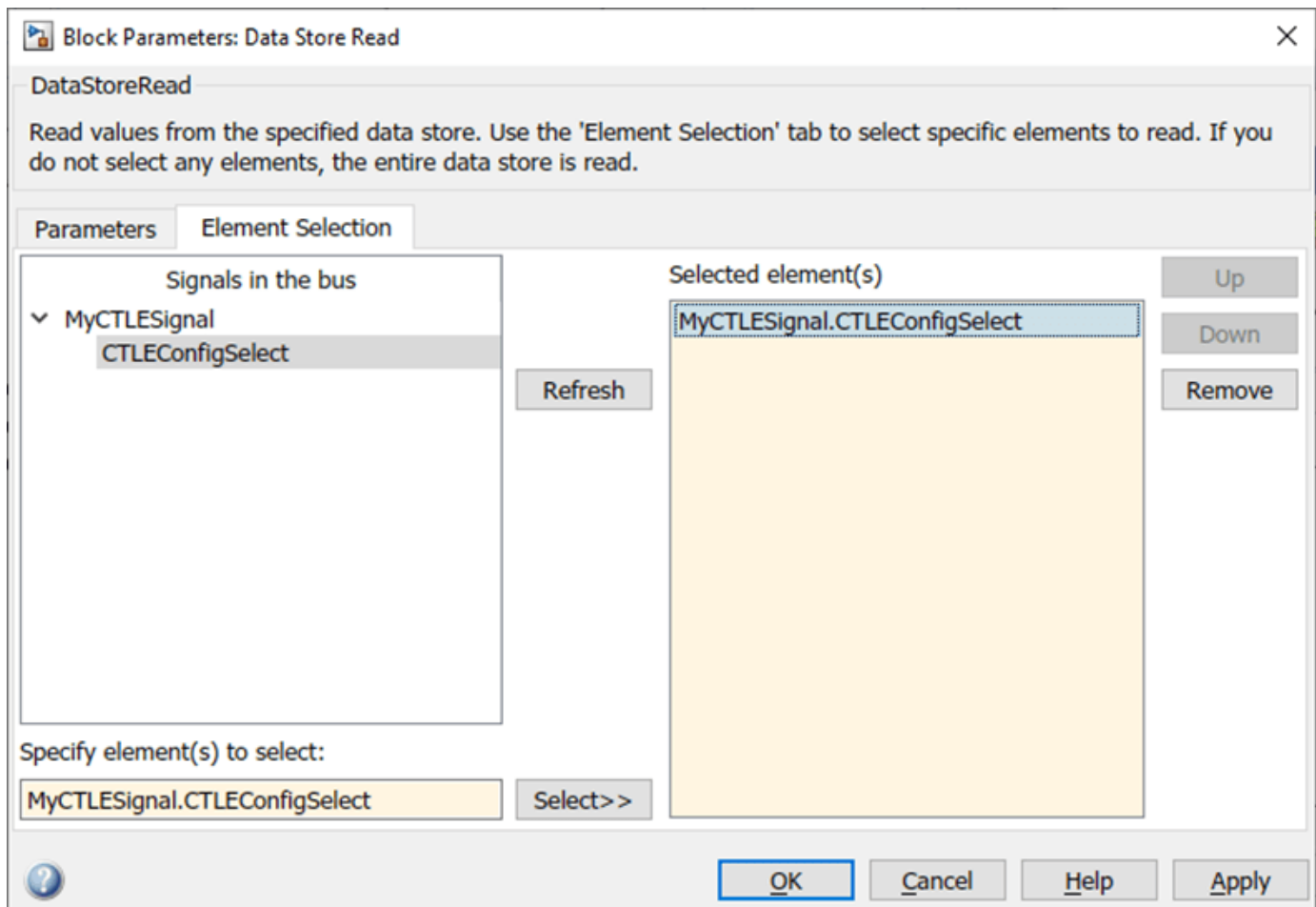
This completes setup for the time-domain (GetWave) simulation.

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.

Stat Analysis

Pulse Response

Waveform Derived from Pulse Response
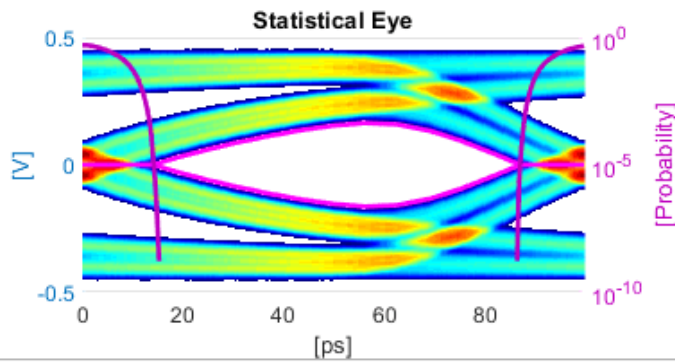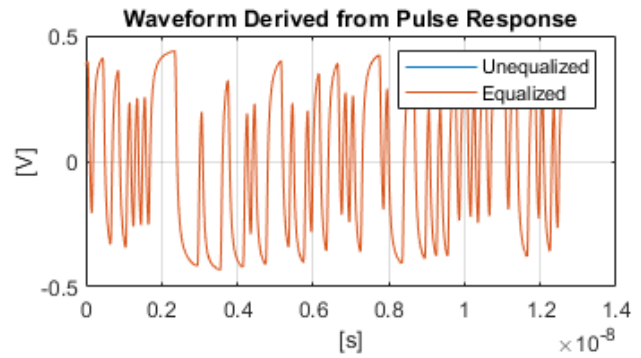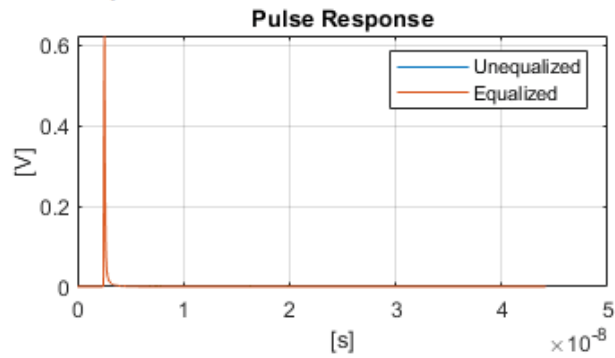
Statistical Eye

Statistical Metric	Data
Eye Height (V)	0.3053
Eye Width (ps)	72.6030
Eye Area (V*ps)	14.7477
COM	5.9600
VEC	6.0816

Time Domain Analysis
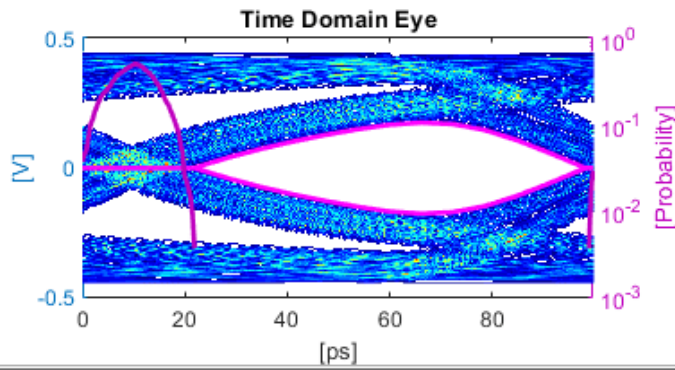
Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.3300
Eye Width (ps)	76.9533
Eye Area (V*ps)	16.5709
COM	6.7183
VEC	5.3748
Minimum BER	5.2632e-04
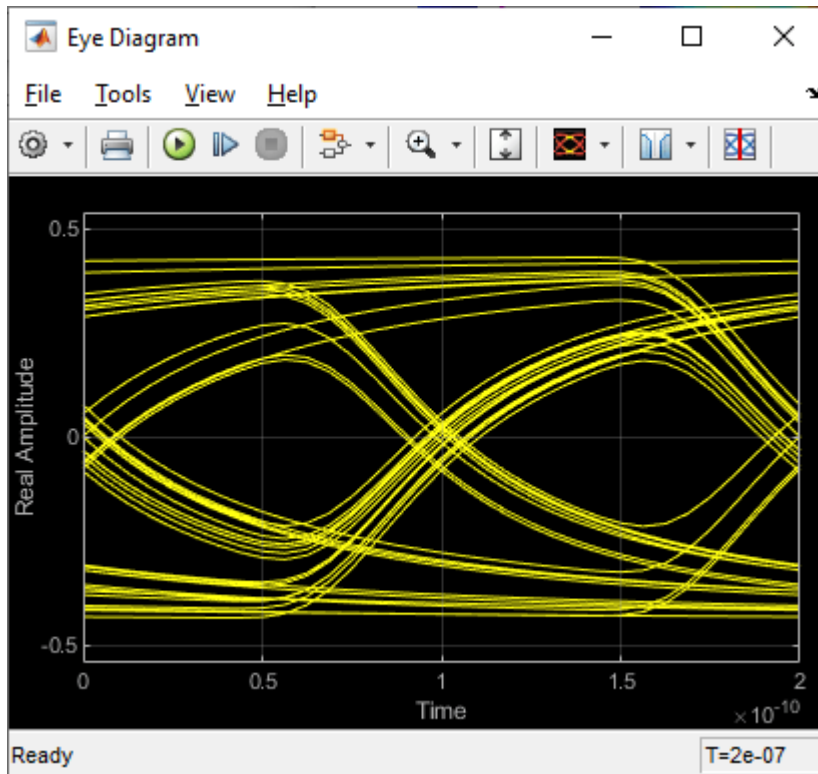
To evaluate the effect of the CTLE on output waveforms, open the SerDes IBIS-AMI manager dialog box. In the **AMI-Rx** tab, set **Current value** of **CTLEMode*** parameter to 1 to use fixed mode operation, and set **Current value** of **CTLEConfigSelect*** parameter to 4. Re-run the simulation.

Stat Analysis

Pulse Response
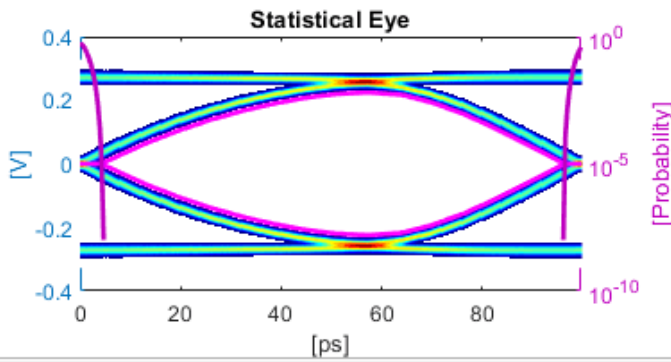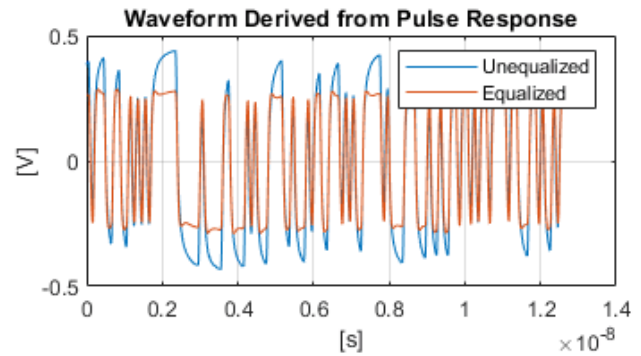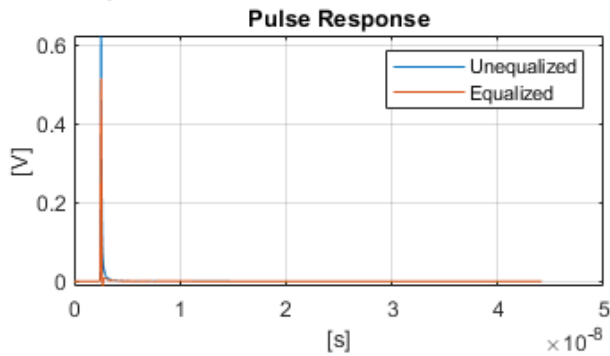
Statistical Metric	Data
Eye Height (V)	0.4335
Eye Width (ps)	92.1840
Eye Area (V*ps)	26.1905
COM	16.3293
VEC	1.4382

Time Domain Analysis
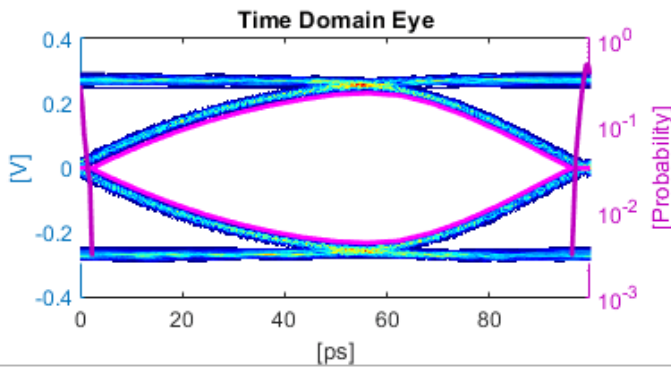
Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.4470
Eye Width (ps)	93.7503
Eye Area (V*ps)	27.3829
COM	18.0681
VEC	1.1589
Minimum BER	5.2632e-04
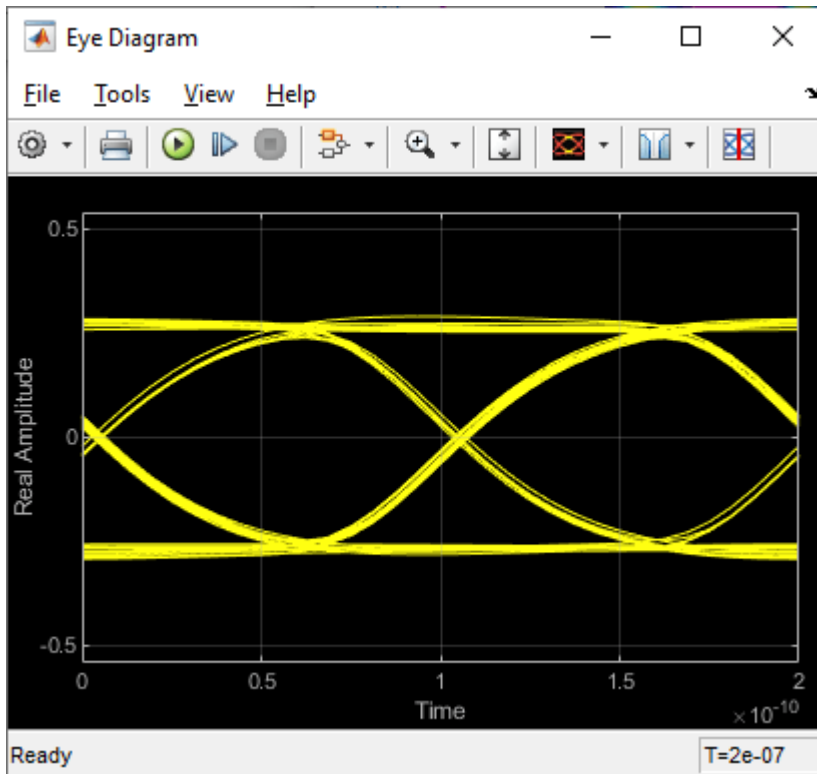
See Also

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

More About

- "Customizing Datapath Building Blocks" on page 5-14
- "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance" on page 4-27

Step Response Based CTLE

This example shows how to create a custom step response-based CTLE block in Simulink® to model wired communication links of your own specifications. The custom CTLE block exhibits equivalent behavior as the default pole/zero based CTLE block from the SerDes Toolbox™. This example also illustrates:

- how to use the MATLAB function blocks to model custom algorithms in Simulink,
- how to include large data files into your model,
- how to create custom Initialization Subsystem algorithms to perform Init (impulse based) optimization before the Simulink simulation starts, and
- how to validate the model using the Simulation Data Inspector.

The first time you call the step response-based CTLE, it loads a data table in the memory. The data table contains a reference step response for each filter configuration. The CTLE resamples or interpolates the step response to the simulation time step, differentiates the step to obtain the impulse response, and then convolves this with the input waveform. It is easier to resample step response than impulse response due to the difficulty of properly capturing the peak of an impulse response.

Characterize CTLE with Step Responses

Typically, the data table with the reference step responses is obtained from circuit simulations. But for this example, extract the step response from the default CTLE from the SerDes Toolbox to characterize its behavior. It is important to ensure that the time step of the characterization data is fine enough so that all relevant step response behavior is captured. Use 32 samples per symbol, which results in a time step size of 3.125 ps and is more than sufficient for this CTLE.

Create a CTLE object with the default peaking characteristics.

```
SymbolTime = 100e-12;
SamplesPerSymbol = 32;
dt = SymbolTime/SamplesPerSymbol;

CTLE1 = serdes.CTLE(...
    'SymbolTime',SymbolTime,...   %Duration of a single symbol
    'SampleInterval',dt,...       %time step size
    'DCGain',0:-1:-8,...          %DC Gain
    'PeakingGain',0:8,...         %Peaking Gain
    'PeakingFrequency',5e9,...    %Peaking Frequency
    'Mode',1);                    %Mode is fixed
```

For each configuration of the CTLE, stimulate the CTLE with an ideal step response excitation to extract the reference CTLE step responses and observe the output waveforms.

```
stimulus = ones(25*SamplesPerSymbol,1);
stimulus(1:SamplesPerSymbol) = 0;

numberOfConfig = CTLE1.ConfigCount;

stepResponse = zeros(length(stimulus),numberOfConfig);
for ii = 1:numberOfConfig
    CTLE1.ConfigSelect = ii-1;
```

```
    release(CTLE1);
    stepResponse(:,ii) = CTLE1(stimulus);
end

t1 = dt*(0:size(stepResponse,1)-1);

figure,
plot(t1,stepResponse)
xlabel('time [s]'),ylabel('[V]')
title('CTLE Step responses')
legend(cellstr(num2str((0:(numberOfConfig-1))')))
grid on
```
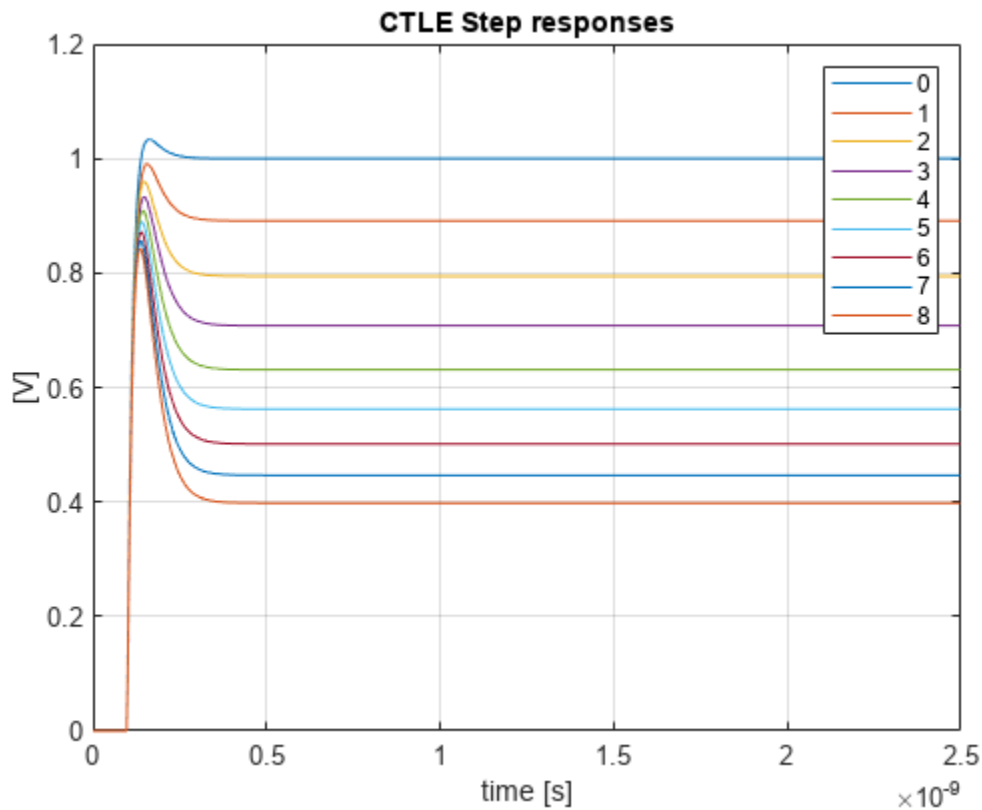

Finally save the matrix of step responses, 'stepResponse', and the sample interval, 'dt', to a .mat file. Observe that each column of the matrix represents a different CTLE configuration. This example uses the filename 'myCTLEdata.mat' for the data. If you change the file name, then you also need to manually update the file references in the stepCTLE.m function and the Simulink Initialize Subsystem references.

If you already have your own CTLE behavior recorded from circuit simulations, you can put the data into the same file format as below, with the fields 'stepResponse' and 'dt' equivalently set. If your own CTLE is characterized by impulse responses, you can use the function impulse2step to first convert them to step responses.

Create the reference data file.

```
save('myCTLEdata.mat','stepResponse','dt')
```

Create SerDes System Model

Use the SerDes Designer App to create a receiver model with a Pass Through and a CTLE block. This setup allows for a straightforward validation process to show that the step response based CTLE has the same behavior as the pole/zero based CTLE.

Open the SerDes Designer app.

>> serdesDesigner

Add a Passthrough block and rename it to 'StepCTLE'

Add a CTLE block.
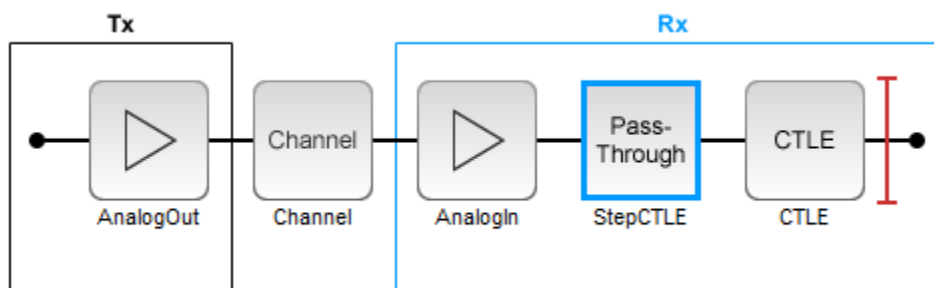
Select the Channel block to open its Block Parameters dialog box. Include crosstalk by selecting Enable Crosstalk parameter. Having crosstalk enabled ensures that the impulse response matrix input to Init will have multiple columns during testing and so any custom Init code will need to correctly process multi-dimensional arrays. Ensuring proper behavior here will avoid later issues when the model is exported to IBIS-AMI.

Export the SerDes system to Simulink.

Setup Simulink Model

Modify the Simulink model to include the MATLAB function block and parameters to control the custom CTLE block. Open the Block Parameters dialog box for the Configuration block, then click on the Open SerDes IBIS-AMI Manager button and select the AMI-Rx tab. Under the Model_Specific parameter, select StepCTLE and click the Add Parameter... button. In the newly opened window, set Parameter name to Mode, Usage to In, Type to Integer, Format to List, List values to [2 0 1], and List_Tip values to ["adapt" "off" "fixed"].

Select StepCTLE under the Model_Specific parameter again and click the Add Parameter... button. In the newly opened window, set Parameter name to ConfigSelect, Usage to InOut, Type to Integer, Format to List, and List values to [0 1 2 3 4 5 6 7 8].

With the inclusion of the above parameters, the Simulink model view shows the StepCTLE subsystem in the StepCTLE window.

Add a MATLAB Function block to the StepCTLE subsystem and open it.

Copy the contents of the file stepCTLE.m in the MATLAB Function block.

5-39

To associate the Model Workspace variable `SampleInterval` to the function input `SampleInterval`, set `SampleInterval` to be a parameter with the Symbols pane and Property Inspector.

1 Open the MATLAB Function Block Editor.

2 In **Modeling** tab, in the **Design** section, click **Symbols** Pane.

3 Open the Property Inspector. Right-click on the `SampleInterval` and click **Inspect**.

4 Change the **Scope** to `Parameter`.

stepCTLE Function

The stepCTLE function has two primary behaviors:

*   It loads the step response data, resamples it according to the simulation sample interval, and differentiates the step response to obtain the impulse response.
*   It filters (or convolves) the incoming waveform with the impulse response.

The first primary behavior is essential so that the stepCTLE has consistent behavior over changes in the simulation time step size.

Persistent Variables

Observe that this function utilizes `persistent` variables. Persistent variables have permanent storage in MATLAB similar to global variables. But unlike global variables, persistent variables are known only to the function that declares them. In a Simulink model, each MATLAB function block contains its own copy of persistent data. If a MATLAB function that contains a persistent variable is called from two different blocks, the model has two persistent variables. Also, each run of the simulation creates a new copy of the persistent data.

Using coder.load to Including Data Files into Model

Observe the use of the `coder.load` (MATLAB Coder) function instead of the 'load' function to access the data in the .mat file. When this code is compiled, the data in the .mat file will be hard coded into the executable and is an excellent way of including large data files into the model.

Connectivity

From Simulink, connect the parameter blocks to the MATLAB function block. Delete the ConfigSelect write block as it will not be used. Also delete the Pass Through System object block.

Add a display block from the Simulink Library Browser to observe the adapted value of ConfigSelect parameter. To observe the adapted ConfigSelect parameter of the SerDes Toolbox CTLE, also add a display block under the CTLE mask. You can then verify that both blocks adapt to the same configuration select.
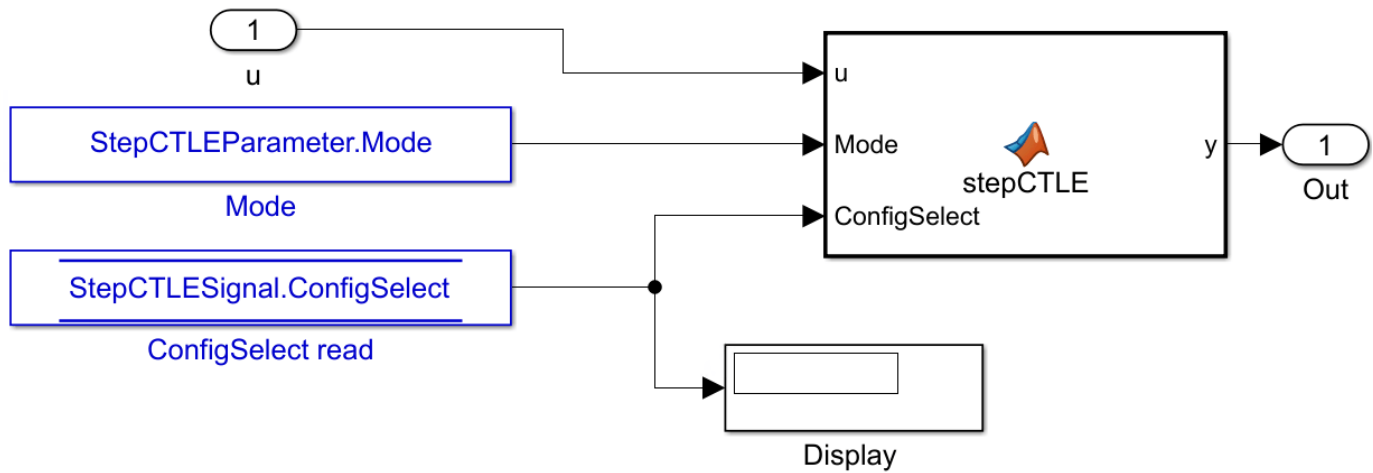
Setup Init

The SerDes Simulink model can perform Init (impulse-based) analysis before the zero simulation time with the Initialize Subsystem block. Open the Init block in the Rx subsystem. Click the Refresh Init button and then the Show Init button to bring up the MATLAB Editor. Disregard any warning messages about refresh Init skipping the MATLAB Function block.

Cut and paste the contents of CustomUserCodeForInit.m in the custom user code area of the Init function.

Observe that like in the stepCTLE function, this code loads the step response data file, then resamples it and converts to impulse responses. This code additionally performs optimization to select which of the many CTLE configurations 'best' equalizes the signal using the SNR metric as the goodness criteria. Once the ConfigSelect has been determined, the CTLE response is applied to the primary impulse and crosstalk impulse responses.

An alternative to using MATLAB function blocks is to use System object™. System objects do not require the use of persistent variables (which are not currently allowed in the Initialize Subsystem block) and allow for better code sharing between the Simulink model version of a block and the Initialize subsystem version of the block like many of the System objects in the SerDes Toolbox. For more information on System objects, see "What Are System Objects?".

Validation

To validate that the step response based CTLE is equivalent to the default CTLE from the SerDes Toolbox, log the output waveform and perform two simulations:

- Enable the step based CTLE and disable the pole/zero based CTLE
- Disable the step based CTLE and enable the pole/zero based CTLE

The top level output waveform of the model is already logged (as shown by the broadcast or wifi symbol) for use by the post-simulation analysis results.
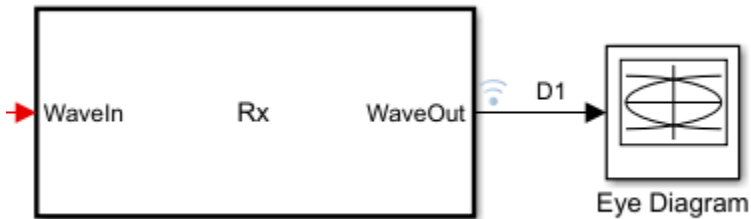
Open the SerDes IBIS-AMI Manager dialog box from Configuration block. Set the Mode of the pole/zero based CTLE's to off. Set the Mode of the step based CTLE to adapt and run the simulation.

Then set the Mode of the pole/zero based CTLE's to adapt. Set the Mode of the step based CTLE to off and rerun the simulation

From the Simulink toolstrip, click the Data Inspector button from the Simulation tab. Change the line color of the most recent simulation results and zoom in on the first few symbols of the simulation. Observe how the only difference between the waveforms is a one symbol delay thus validating the accuracy of the step response based CTLE. The one symbol-time delay is due to the step response characterization data and while this can be removed, it doesn't make a large impact on SerDes simulations or analysis.
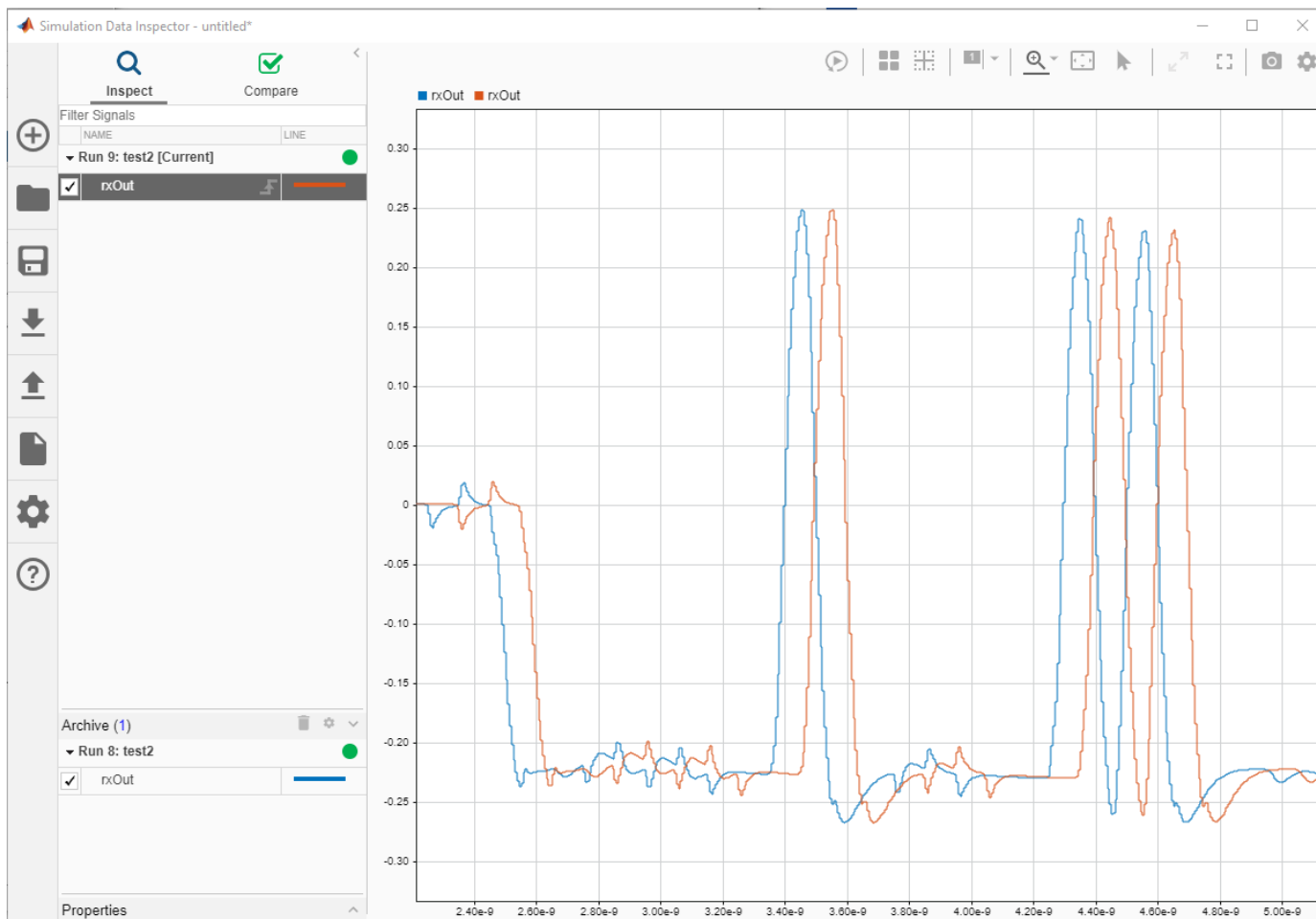
Customize IBIS-AMI Models

Managing AMI Parameters

This example shows how to add, delete, modify, rename and hide AMI parameters for an IBIS-AMI model built with SerDes Toolbox. These AMI parameters are then available to be used with existing datapath blocks, user-created MATLAB function blocks or optimization control loop, and 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')
```


Copyright 2019 The MathWorks, Inc.
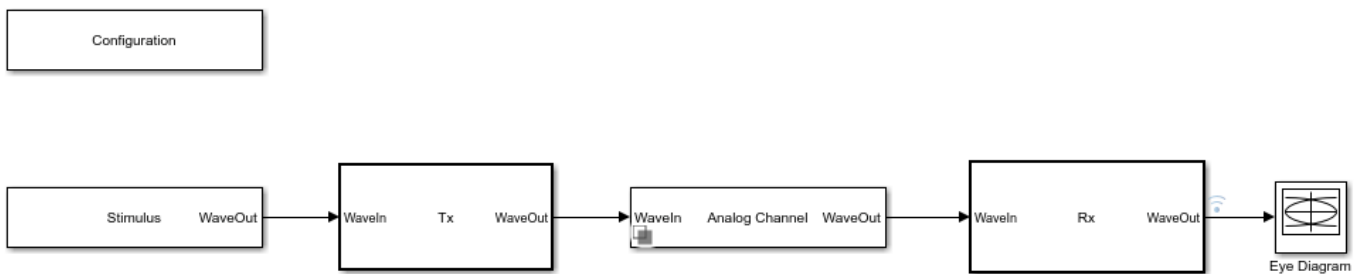
This Simulink SerDes System consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks.

- 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.
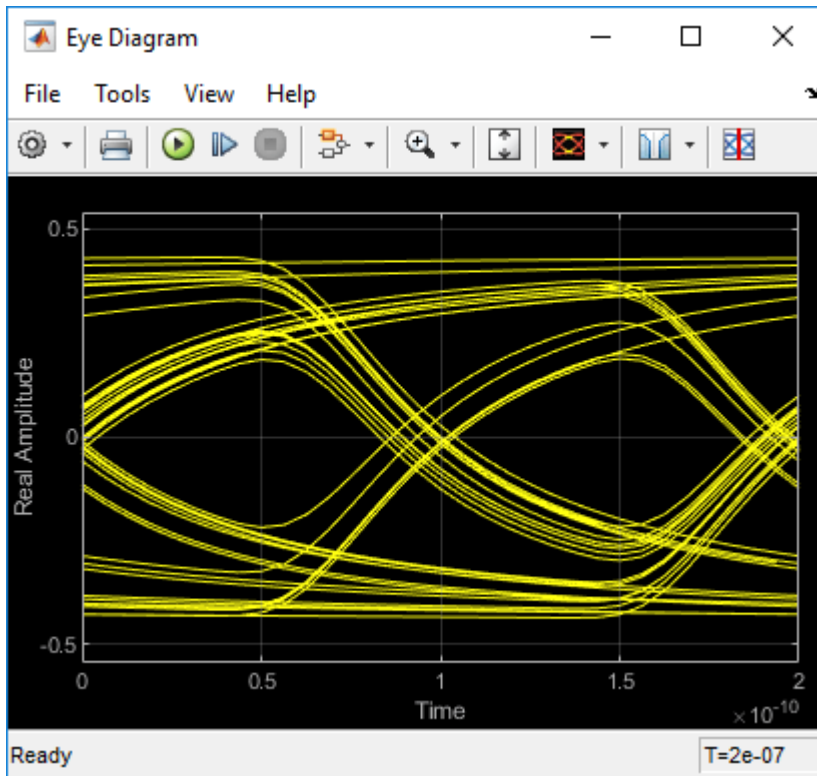
The second plot contains views of the results from statistical (Init) and time domain (GetWave) simulation.
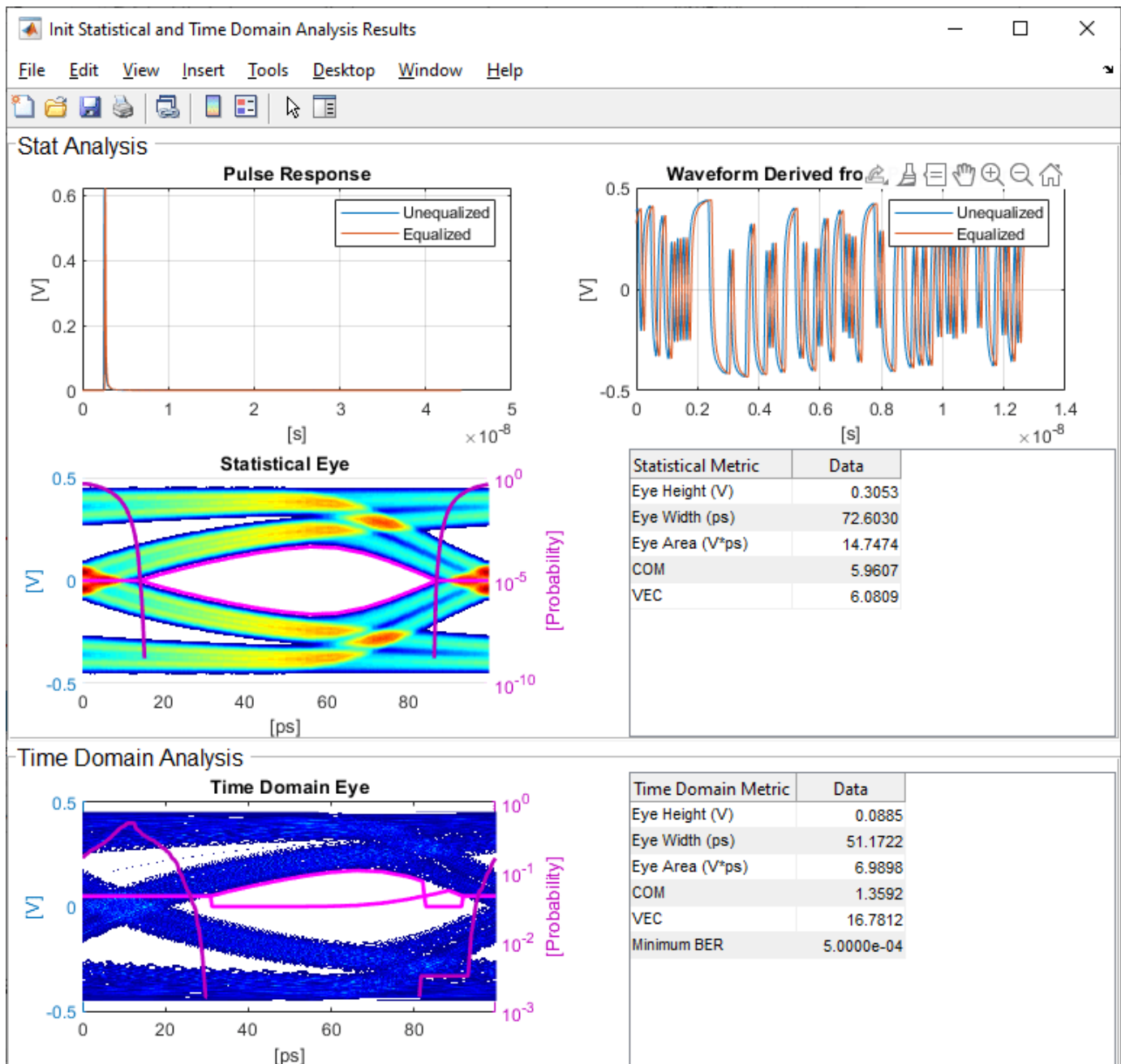
How to Add a new Parameter

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. Click **OK** to create the new parameter, then you will see the new blocks automatically placed on the canvas.

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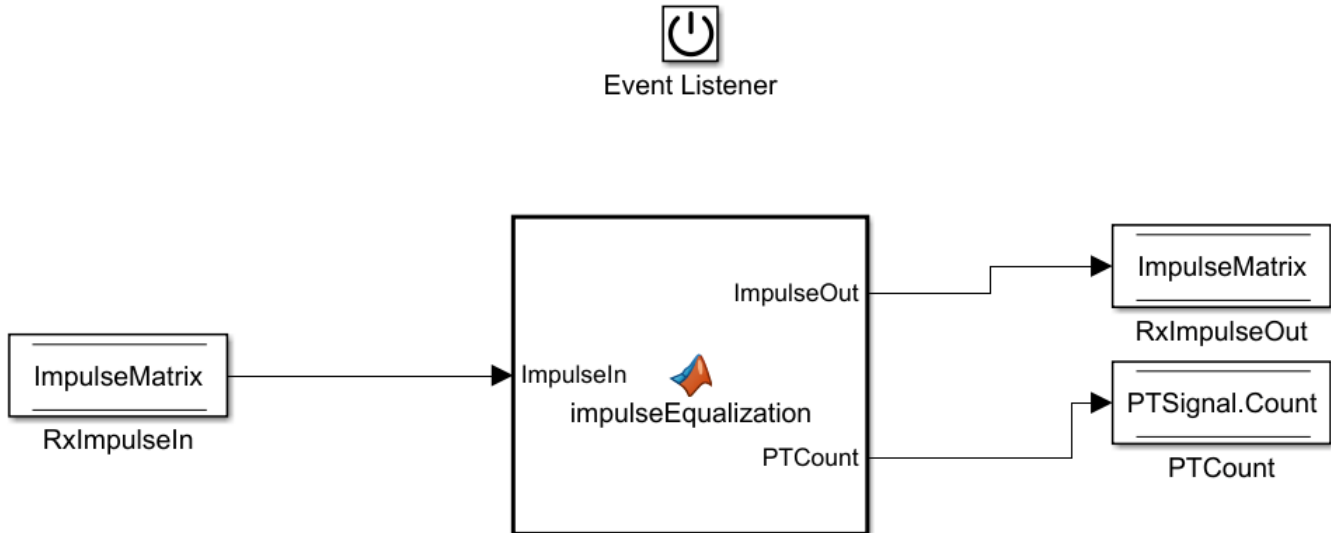
5. Double-click on the **impulseEqualization MATLAB function block** to open the function in MATLAB. The '%% BEGIN:' and '% END:' lines within this function block denote the section where custom user code can be entered. Data in this section will not get over-written when Refresh Init is run:

```
%% 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):

VGAInOutParameter
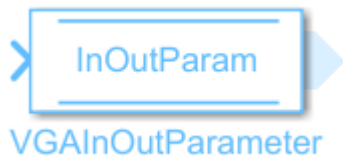
Accessing a new Parameter from the GetWave Function

New parameters are automatically created as blocks of type Constant, Data Store Read or Data Store Write and added to the canvas of 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.
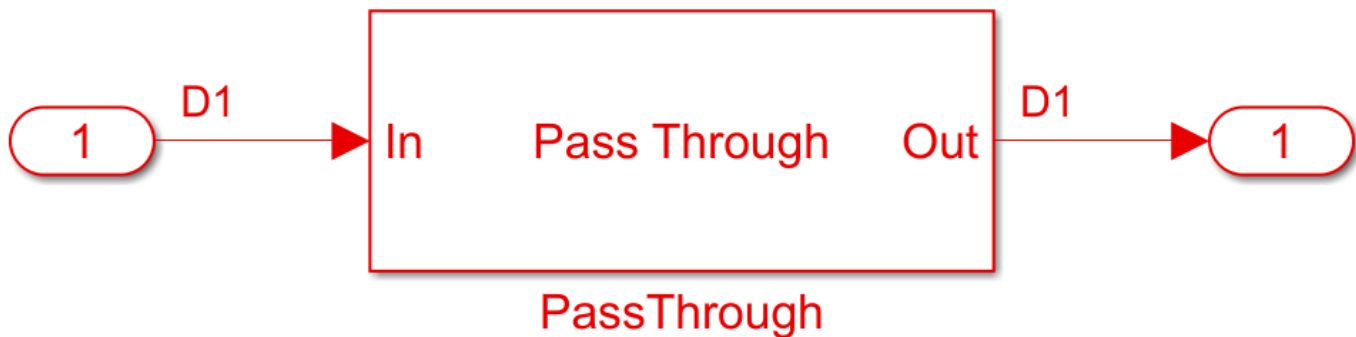
PassThrough

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

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

4. Wire up each of the elements so that the Pass Through block now looks like the following:
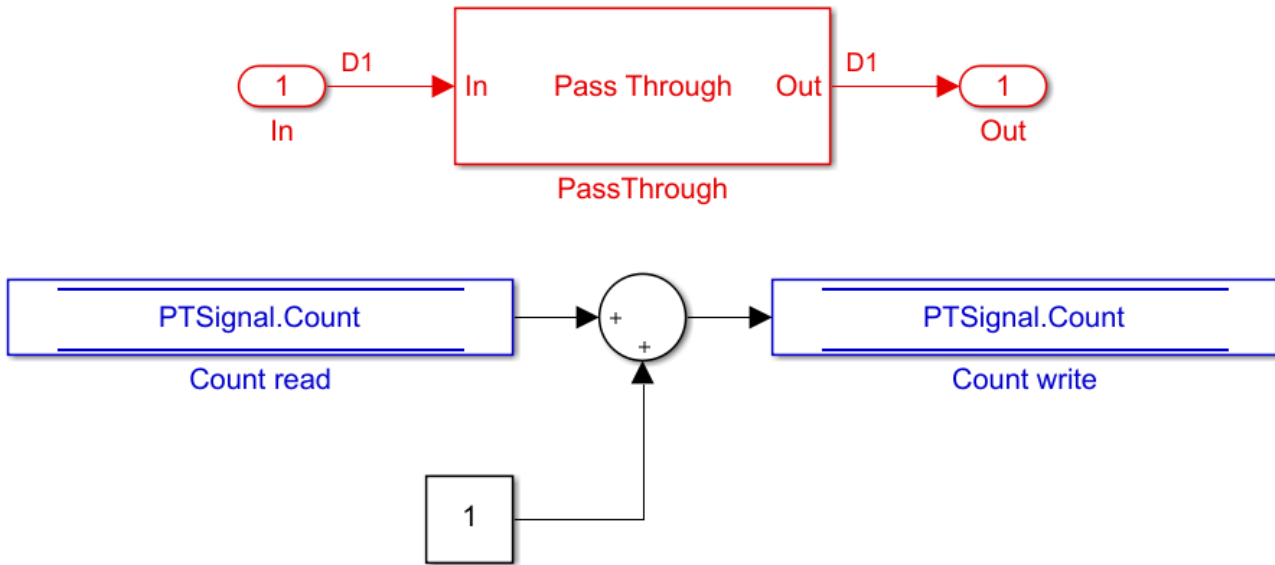
7. Save, then run the Simulink project to test the new code.

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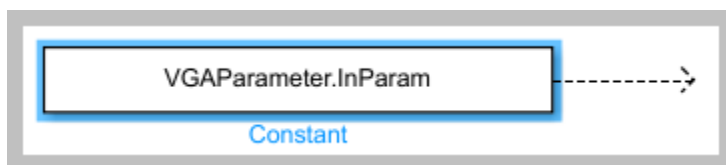
For more information, see "Customizing SerDes Toolbox Datapath Control Signals" on page 5-2.

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:
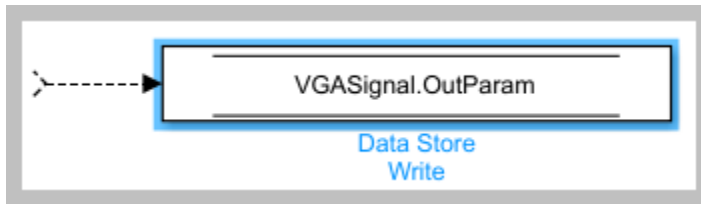
InOut Parameters

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:
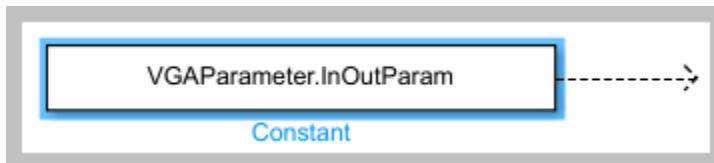
Alternately, the updated Input value can be accessed with a Data Store Read block like this:
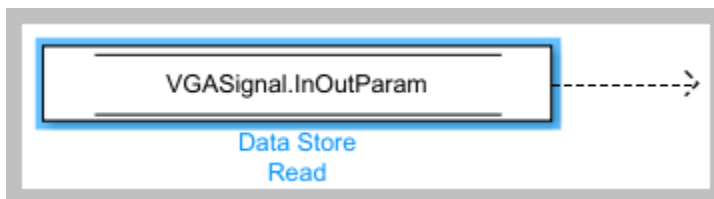
The output value can be written to with a Data Store Write block like this:
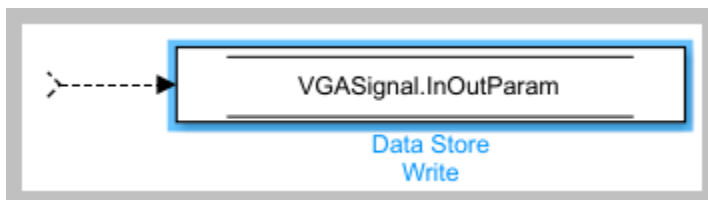
How to Rename a Parameter

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. Click **OK**, then you will see the new parameters automatically renamed on the canvas.

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. You can confirm all instances of the parameter have been renamed.

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. You will see the paramter blocks automatically removed from the canvas.

Update Init

Note: Parameters in the custom user code area are not automatically removed, so you will comment or delete them with the following steps:

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, edit the FFE mode parameter and check the "Hidden" checkbox.

This will prevent this parameter from being present in the .ami file - effectively hardcoding the parameter to its default value. In other words, the FFE 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.

Add Reserved Parameters for Jitter, Analog Buffer Modeling, and Data Management

Reserved AMI parameters include:

- Jitter and noise parameters such as Tx_Rj, Tx_Dj, Tx_DCD, Rx_Rj, Rx_Dj, Rx_DCD, Rx_GausianNoise, and others
- Analog buffer modeling parameters such as Ts4file, TX_V, and RX_R
- Data management using DLL_ID

These 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 on the **AMI-Tx** or **AMI-Rx** tabs.

Note: The reserved parameter **AMI_Version** will automatically change to 7.0 if any IBIS 7.0 reserved parameters are enabled in the IBIS-AMI Manager.

Note: Some of the reserved parameters only effects the exported IBIS-AMI model and is not included in Simulink simulation results. For more information, see "Customize AMI Parameters" on page 1-17.

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 open 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 appear in the Reserved_Parameters section of the .ami file as shown:

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

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

For another example, you can use **Touchstone** files (also known as SnP files) to customize analog buffer modeling of a transmitter or receiver. This option can be enabled using the reserved parameter **Ts4file** in the IBIS AMI Manager.
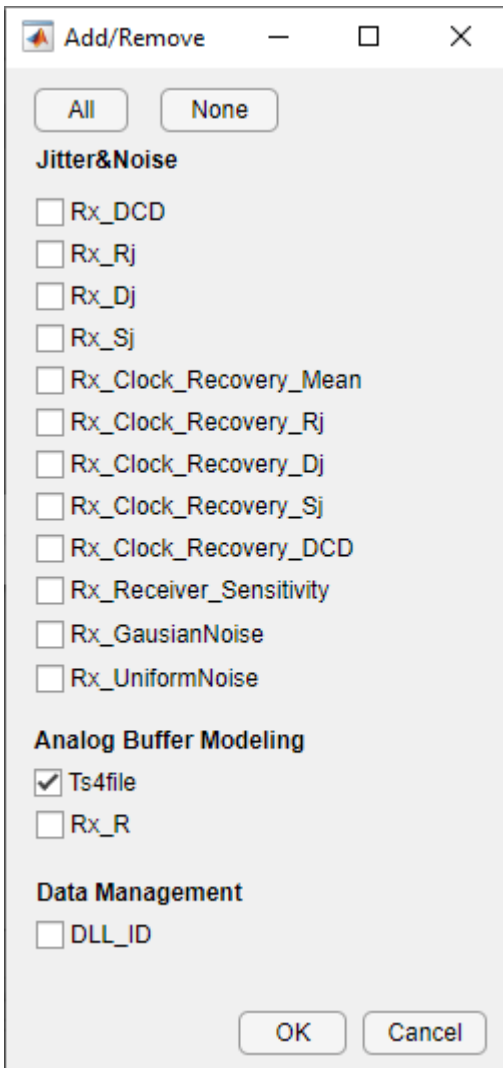
When you click the **Export** button in the IBIS AMI Manager, a dialog will appear where you can select the s-parameter files for each process-corner model to support the reserved parameter **Ts4file**.
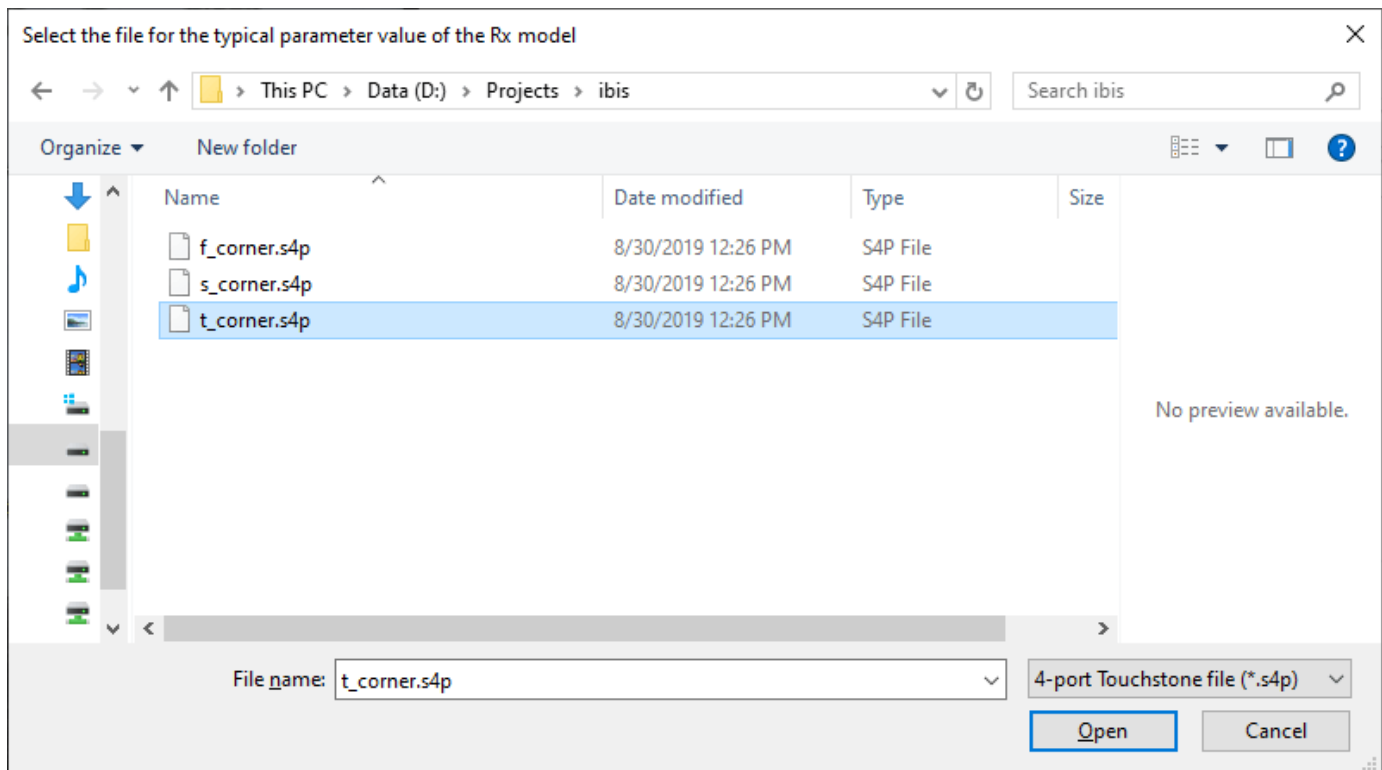
For more information on IBIS reserved parameters see the IBIS specification.

References

IBIS 7.0 Specification

See Also
SerDes Designer | FFE | PassThrough

More About
- "Customizing SerDes Toolbox Datapath Control Signals" on page 5-2

Design IBIS-AMI Models to Support Clock Forwarding

This example shows how to create Rx AMI models that support clock forwarding as defined in the IBIS 7.1 specification by modifying the library blocks in SerDes Toolbox™. This example will use a DDR5 write transfer (Controller to SDRAM) to demonstrate the setup.

Background

The IBIS 7.1 specification adds the ability pass in an external clock signal, either as a waveform or clock-times, to a data IBIS-AMI receiver GetWave model, using the `clock_times` pointer as defined in the IBIS specification. A new AMI Reserved Parameter, **Rx_Use_Clock_Input**, is used to enable this functionality.

The figure below shows a typical DDR5 coupled channel simulation setup using clock-forwarding. The clock times or waveform generated by DQS0 is passed to DQ[7:0] (DQ0, DQ1... DQ7) using the DQ Rx DLL's `clock_times` pointer. The DQ Rx DLL then operates on these clock times as desired (for example triggering DFE taps, modelling the DQS clock delay tree or centering the DQ on the DQS waveform) and then passes out the same or modified `clock_times` as usual. This same process is repeated for DQS1 and DQ[8:15].

Step 1: compute analog channel output according to IBIS 5.1-7.0 (crosstalk taken into account)
Step 2: compute output of all DQS Rx DLLs according to IBIS 5.1-7.0
        Use either DQS Rx clock_times or wave output values as DQ Rx clock_times input values
Step 3: compute output of all DQ Rx DLLs
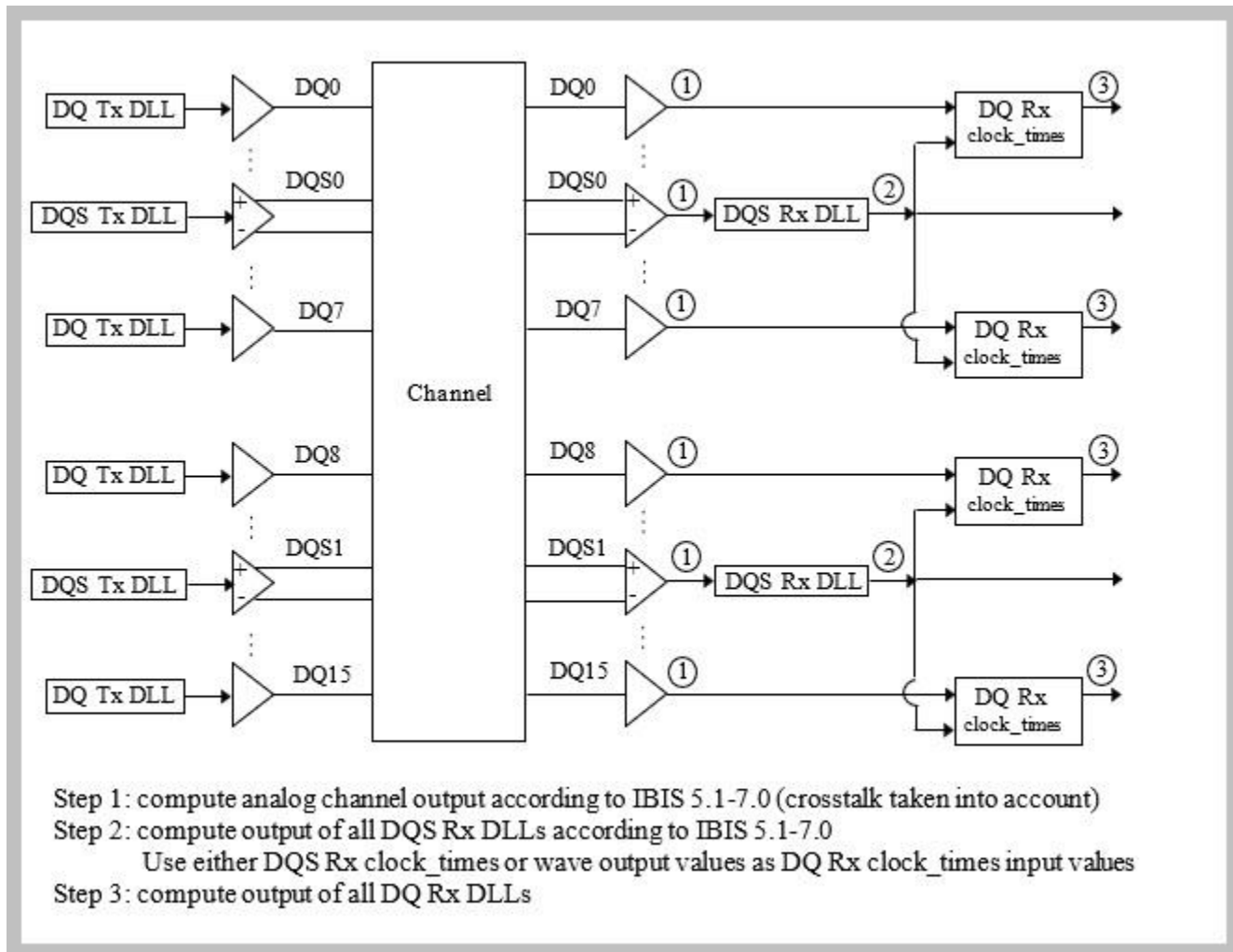
This example provides an introduction to clock-forwarding in SerDes Toolbox and shows how to use various Simulink® tools and MATLAB® functions to generate and test an IBIS-AMI executable that supports clock-forwarding. It does not provide a specific clock-forwarding algorithm.

Rx IBIS-AMI Model Setup in Simulink

To begin, load the clock forwarding Simulink model and review the model setup. Start by typing the following command:

```
>> open_system('dq_clock_forward.slx')
```

This will bring up the following SerDes system:
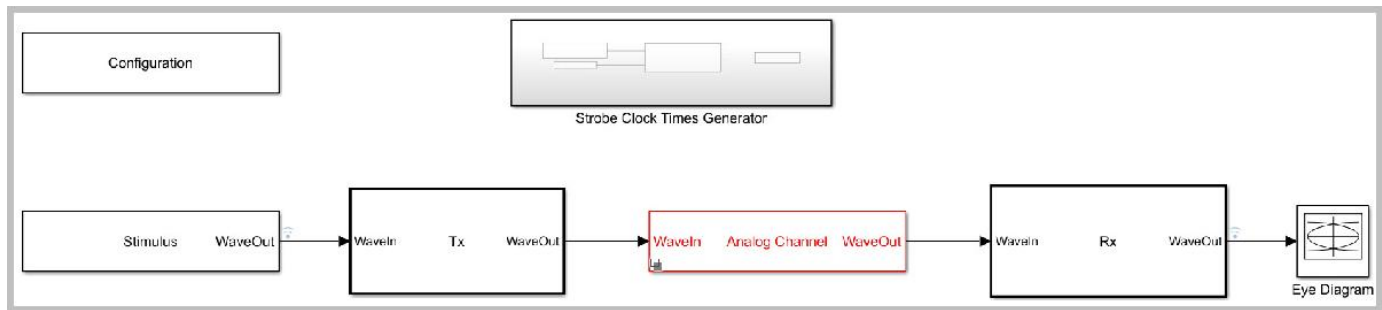
Review Simulink Model Setup

In addition to the normal SerDes Configuration, Stimulus, Tx, Analog Channel and Rx blocks, this Simulink SerDes system adds a new Strobe Clock Times Generator block. The setup of each of these blocks will be reviewed below.

Configuration Block

- **Symbol Time** is set to `200.0` ps (5.0Gbps)
- **Target BER** is set to `1e-16.`
- **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.

Stimulus Block

- The Stimulus block has been modified to point to the custom stimulus pattern `SILinkStimulus_dq.`

Tx Block

The Tx block uses a single FFE with 5 taps. Since this example is focused on the Rx model, the Tx block will be untouched.

Analog Channel Block

- **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 `2.5` GHz, which is the Nyquist frequency for 5.0 GHz
- The Tx Analog model is set up so that **Voltage** is `1.1` V, **Rise time** is `10` ps, **R** (output resistance) is `50` ohms, and **C** (capacitance) is `0.65`pF.
- The Rx Analog model is set up so that **R** (input resistance) is `40` ohms and **C** (capacitance) is `0.65`pF.

Rx Block

The single Rx block is a pass-through block that consists of a DFE System Object, a CDR MATLAB function block and a Clock Times block. The DFE block is set up for four DFE taps. The first tap has the **Initial tap weight** set to -0.1 (so it's visible in simulation), while the remaining **Initial tap weights** are 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.

Per the IBIS 7.1 specification, the clock times received by a clock-forwarding data receiver are used directly, therefore there no clock recovery is required when **Rx_Use_Clock_Input** is set to "Times" or "Wave". In place of the CDR, a MATLAB function block named forwardCDR is used to pass the clock times to the DFE, which signals when to apply the DFE taps. In addition, this block passes the clock times, unchanged, to the IBIS-AMI clock_times block to generate the normal clock times for use by the EDA tool. The MATLAB function block can be copy-pasted from this example into your own Simulink model.

The Clock Times block is a SerDes Toolbox library block which formats the clock-times generated by the DFECDR, CDR or forwardCDR block for output to the EDA tool. This library block is available from the Simulink Library Browser.

The DFE block is a custom SerDes Toolbox system object. It can be added to your own Simulink model by adding a MATLAB System block and then pointing it to the DFE.m file included in this example.

The CDR block is a standard SerDes Toolbox system object. When **Rx_Use_Clock_Input** is set to "None", the CDR outputs clock-times for the Clock Times library block. This is selected by the **Forward Clock Times** Data Store Read block and the four Switch blocks. Note that the CDR block is required in order for the model to output clock-times per the IBIS-AMI specification when clock forwarding is not being used.

Strobe Clock Times Generator Sub-System

The Strobe Clock Times Generator Block either reads a named clock stimulus pattern stored in the Model Workspace or reads in an array of clock times named `clockTicks` which is also stored in the Model Workspace. The mask for this sub-system is used to select which input to use and to set the name of the external clock stimulus pattern.
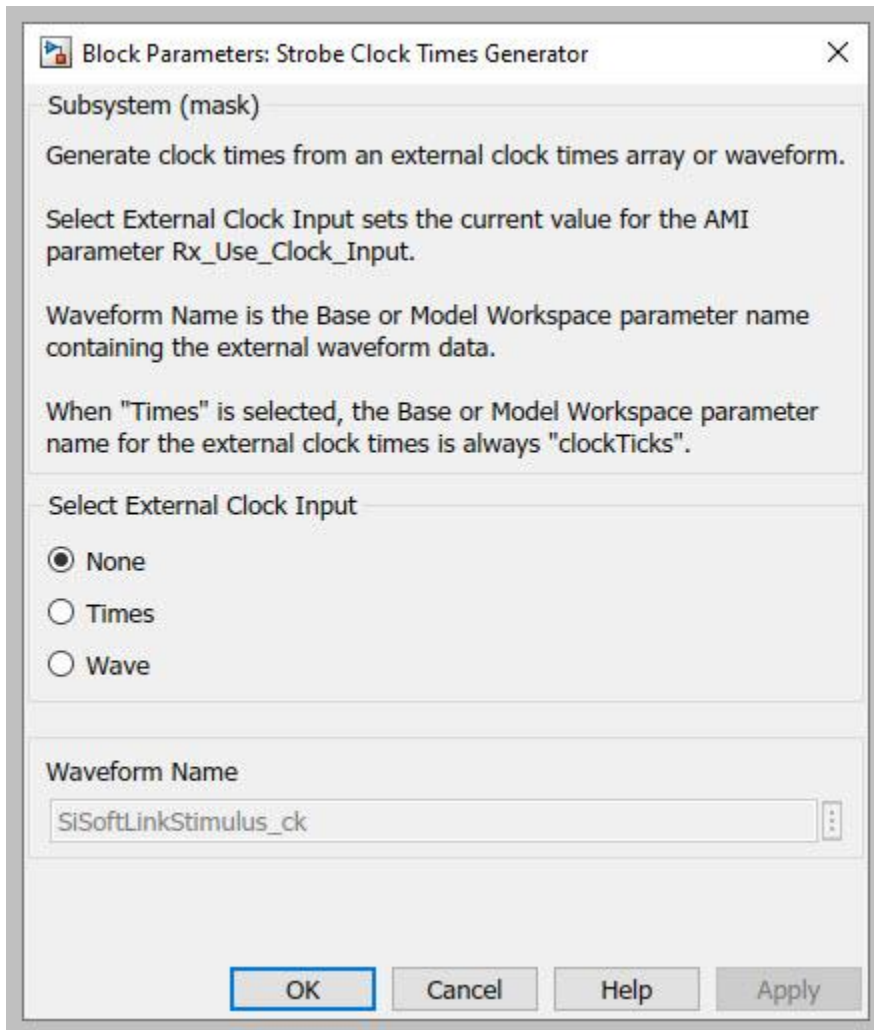
You can add the Strobe Clock Times Generator subsystem to a new Simulink model by copying it from this example. Copying this subsystem into a new SerDes model will also add the required **Rx_Use_Clock_Input** parameters and `ForwardClockOffset` Simulink signal to the Model Workspace.

Run the Simulink Model

The Simulink model is ready to run. In order to make the effects of the clock location more visible, the first DFE tap has been set to `-0.1`V and the DFE mode is set to `Fixed`. Press the run button to launch the simulation.

As the simulation runs, the Time Domain eye diagram gets constantly updated:
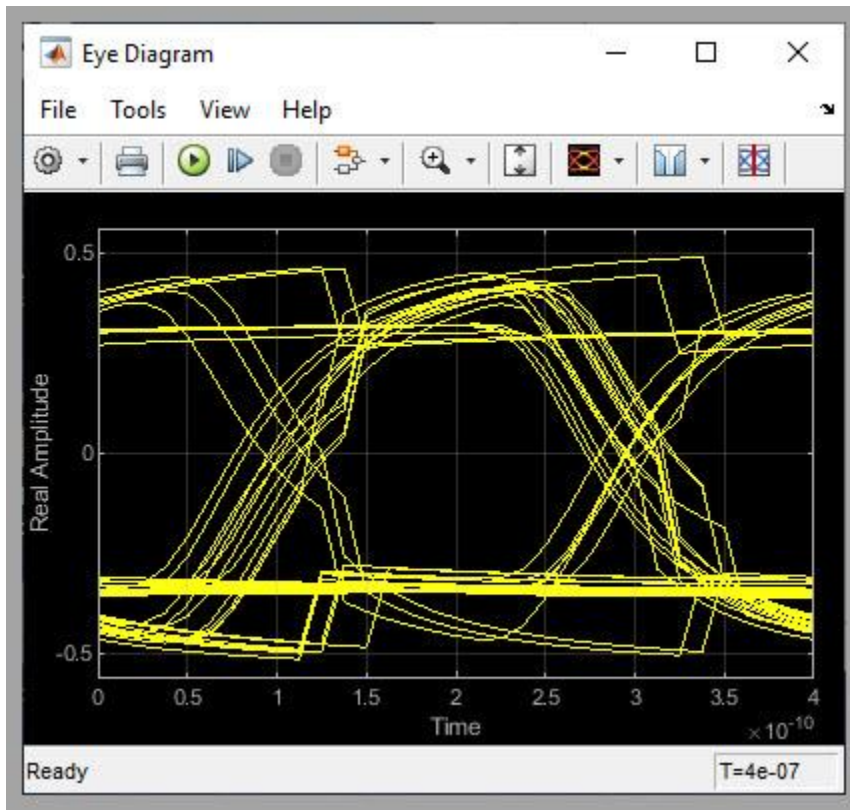
After the simulation is complete, the Init Statistical and Time Domain Analysis Results plot becomes available:
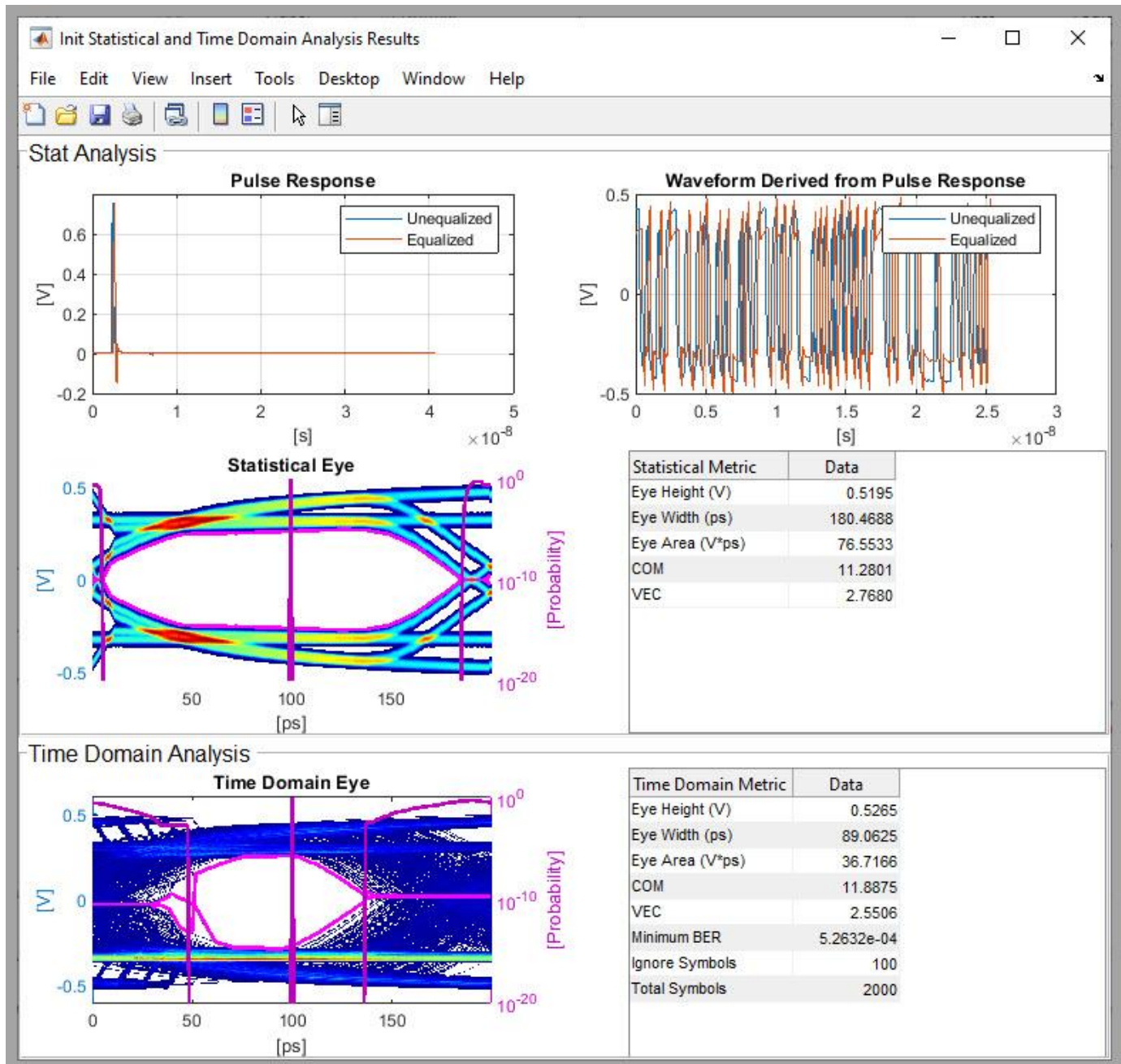
Note that since clock-forwarding only affects the Time Domain results, the Statistical results does not reflect the effects of clock-forwarding.

How to visualize results

To verify proper operation of clock-forwarding, plotting the resulting waveforms and/or clock-ticks can be very helpful. Several signals have data logging turned on to enable the use of the Data Inspector for plotting waveforms. To turn on additional data logging, right-click on any signal and select `Log Selected Signals`.

Plotting clock and data waveforms

After running a simulation open the Data Inspector by clicking on the ⬜-icon in the Simulink Simulation tab. In the Data Inspector check the boxes for Stimulus:1 (the incoming stimulus waveform, Red in the figure below) and for rxOut (the Rx data out waveform, blue in the figure below). You should see that rising and falling edges of the external-clock waveform (Red) correspond with the peaks of the data waveform (Blue). If they do not line up as expected, the offset can be adjusted by using the clock offset (see Setting the clock offset on page 6-29).
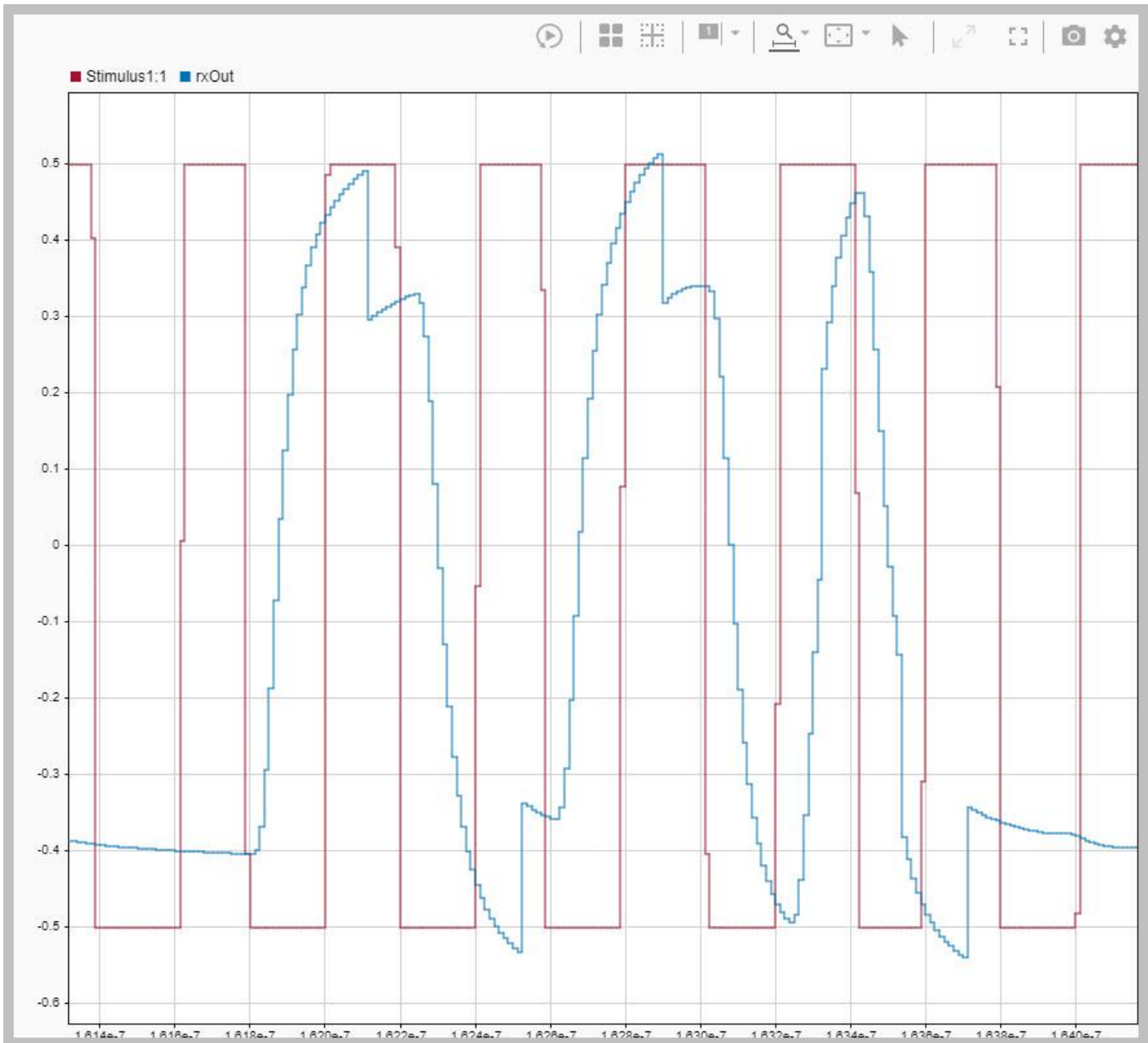
Plotting clock-ticks in and out

After running a simulation, in the Data Inspector check the boxes for Forward Clock Times:1 (the external clock-times from the Clock Times Generator block) and clockTime (the clock-times being passed out of the Rx model). When "Times" is selected as the external clock times, these two signals are expected to be identical.
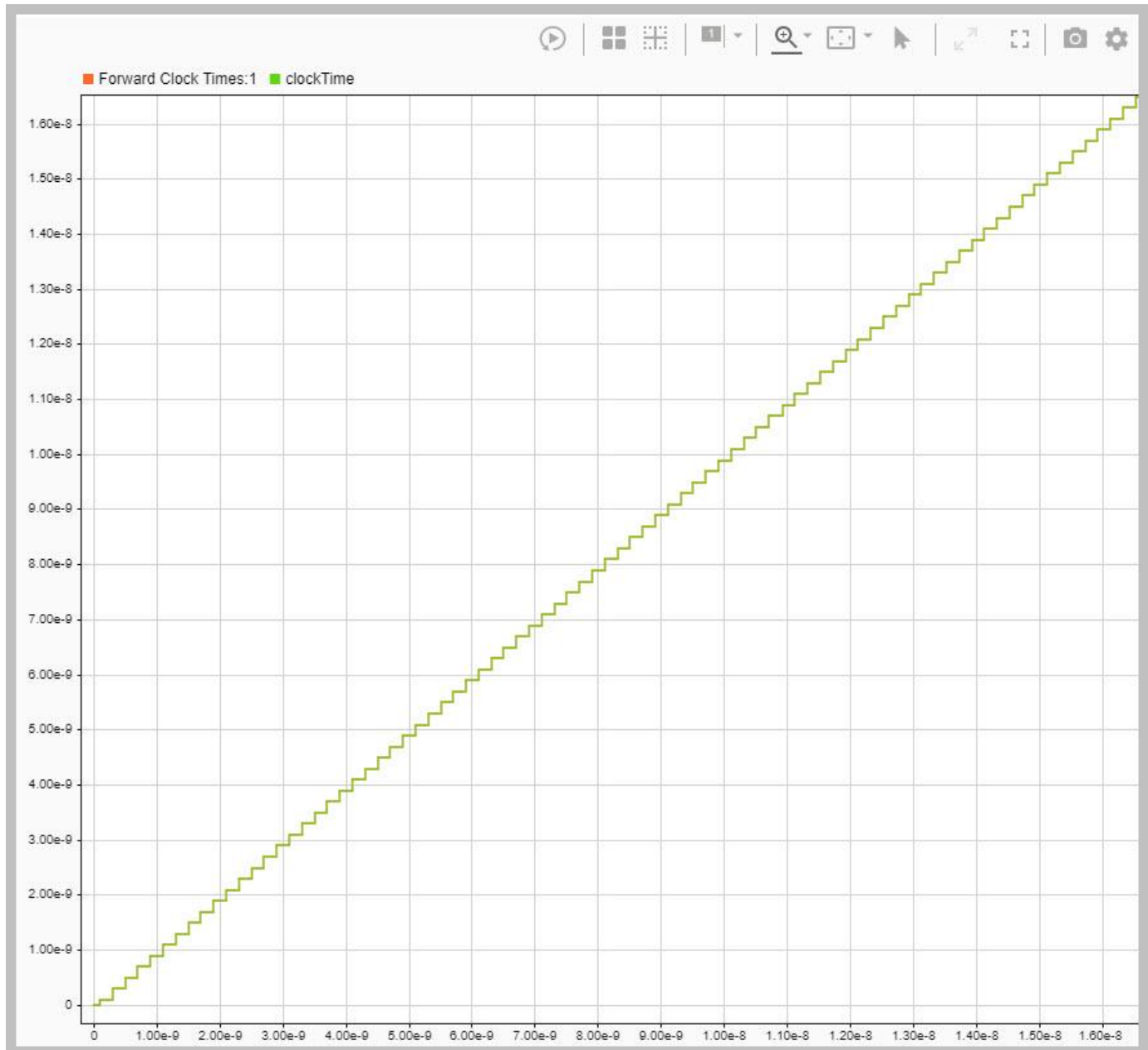
Changing the data pattern

The Rx Data pattern is set using the Stimulus block of the SerDes system as usual. A PRBS pattern can be selected, or a named stimulus pattern that lives in the model workspace can be used. The current stimulus pattern is named SILinkStimulus_dq.

Changing the clock pattern

Two clock patterns are included in this Simulink model:

- `SILinkStimulus_ck`: This is a periodic clock pattern generated by Parallel Link Designer (in Signal Integrity Toolbox)
- `SILinkStimulus_dqs`: This is a DQS pattern with an 8-bit DDR burst followed by a 4-bit static low.

To change this pattern, specify the desired pattern by name in the Strobe Clock Times Generator mask.

To create a new pattern, see Creating a new clock waveform on page 6-27.

Switching between an external waveform and clock ticks

Switching between using an external waveform to generate clock times to using an external clock ticks array directly is accomplished by changing the value of the parameter **Rx_Use_Clock_Input** from the Strobe Clock Times Generator mask. There are 3 options for External Clock Input:

- **None**: The Rx AMI model uses it's built-in CDR to generate clock times.
- **Times**: Use the external clock times given in the Model Workspace parameter `clockTicks`.
- **Wave**: Use the external clock waveform from the `Waveform Name` in the mask.

Creating a new clock waveform

Generating a new clock or strobe waveform for use in the Stimulus block inside the Strobe Clock Times Generator is accomplished using Parallel Link Designer (part of Signal Integrity Toolbox) and Signal Integrity Link. Note that since you are only interested in creating a stimulus pattern, any AMI model can be used for this process. The following steps assume you are using the `dq_clock_forward` model from this example, however the `dqs_clock_forward` model can be used as well.

Here is an overview of the required steps. For additional information on using Signal Integrity Link, see "Signal Integrity Link" on page 3-11. Note that if you have previously run Signal Integrity Link on this Simulink model you can begin with step 3.

1 Start by using the SerDes IBIS-AMI Manager to export the Tx and Rx models. Make sure that the IBIS file, AMI files and DLL files boxes are checked.

2 Use Signal Integrity Link to Create a new Parallel Link Designer project.

3 In the new Parallel Link Designer project, double-click on the Tx designator then press the **IO Stimulus** button in the Designator Element Properties dialog.

4 In the Stimuli dialog, press the **New** button to open the Stimulus Editor and create the desired clock pattern. For example, you can set a continuous-clock pattern that repeats, or a burst-strobe pattern that starts and stops.

5 When you are done creating a new stimulus, make sure the new named stimulus pattern is selected in Designator Element Properties.

6 Use the Simulation Parameters dialog to set the desired Samples Per Bit, Record Start and Record Bits values to capture the desired number of samples. For example, to record 32,000 samples, set **Samples Per Bit** to 32, **Record Start** to 0UI, and **Record Bits** to 10,000UI, making sure that **Time Domain Stop** is >= 10,000UI. Note: Number of samples = Samples Per Bit * Record Bits.

7 Run the Parallel Link Designer simulation to generate the new Stimulus pattern.

8 In Signal Integrity Link, in the **Import parallel link project** section, select the proper simulation. The **Stimulus pattern** box is must be checked. Then click the **Import parallel link project** button.

9 Back in Simulink, the new stimulus pattern will automatically be set in the top-level Stimulus block. Change this pattern back to either PRBS or SILinkStimulus_dq as was previously set.

10 In the Stimulus block inside the Strobe Clock Times Generator, select the newly created stimulus SILinkStimulus.

Note: The SILinkStimulus pattern is over-written each time this process is performed. To save a named stimulus pattern, open the Model Explorer, browse to the Model Workspace and rename SILinkStimulus to a new name. This re-named parameter is saved along with the rest of the Simulink model.

Creating new clock ticks

Generating new clock ticks for use inside the Strobe Clock Times Generator is accomplished using Parallel Link Designer (part of Signal Integrity Toolbox) and Signal Integrity Link. Unlike with the new clock waveform process above, here it is recommended to use the actual Clock or Strobe AMI model. The following steps assume you are using the dqs_clock_forward model included with this example. For more information on the Clock/Strobe model see Strobe Rx IBIS-AMI Model Requirements on page 6-31.

Here is an overview of the required steps. Note that if you have previously run Signal Integrity Link on the dqs_clock_forward Simulink model you can begin with step 3.

1 Start by using the SerDes IBIS-AMI Manager to export the Tx and Rx models. Make sure that the IBIS file, AMI files and DLL files boxes are checked.

2 Use Signal Integrity Link to Create a new Parallel Link Designer project.

3 In the new Parallel Link Designer project open the Simulation Parameters dialog and set the parameter Output Clock Ticks to Yes.

4 <Optional>: While you can run the simulation using the default setup from Signal Integrity Link, it is recommended that you set up a widebus simulation with a realistic topology that includes both the dq_clock_forward and the dqs_clock_forward models.

5 Run the desired Parallel Link Designer simulation.

6 From the MATLAB command line, type the following to import the clock times out of your Parallel Link Designer project and format them for use in Simulink. Note: Make sure to point the Clock/Strobe designator name and not the Data designator name.

```
%% Read the Parallel Link Designer generated clock_ticks from a file
filename = '<path_to_pld_project>/<project_name>/interfaces/<interface_name>/pre_sims/<sheet_name
csv = readmatrix(filename,'Range','A7');

%% Format input
count = csv(:,1);
clock = csv(:,2);

%% Output clock_ticks for Simulink
clockTicks = [count, clock];
```

Note: If you wish to save multiple clockTicks arrays, or switch between arrays, you need to update the **clockTicks** parameter name in the clockTimesGen MATLAB function block inside the Strobe Clock Times Generator sub-system.

Setting the clock offset

The Input AMI parameter **ForwardClockOffset** has been added to the DFEandCDR block. This parameter is of type Integer, with a Default of 0 and a Range of 0 to 512. In the DFEandCDR block, this parameter controls a Delay block which is used to delay the incoming clock times by up to 512 samples. Using the SerDes IBIS-AMI Manager you can use this delay to adjust the location of the external clock with respect to the data waveform as desired.

For example, here is the time domain eye diagram with **Select External Clock Input** set to Wave and **ForwardClockOffset** set to 5:
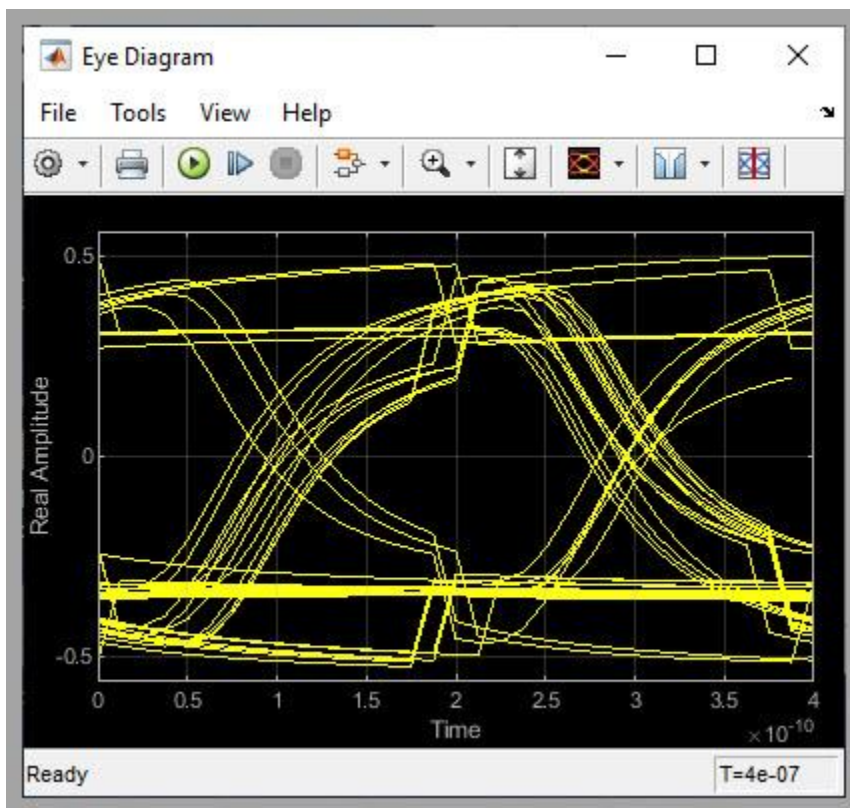
Note how the DFE taps are being applied in the center of the eye instead of at the edges of the eye. With the **ForwardClockOffset** delay set to 14, the DFE taps are being applied at the ideal location at the edge of the eye:
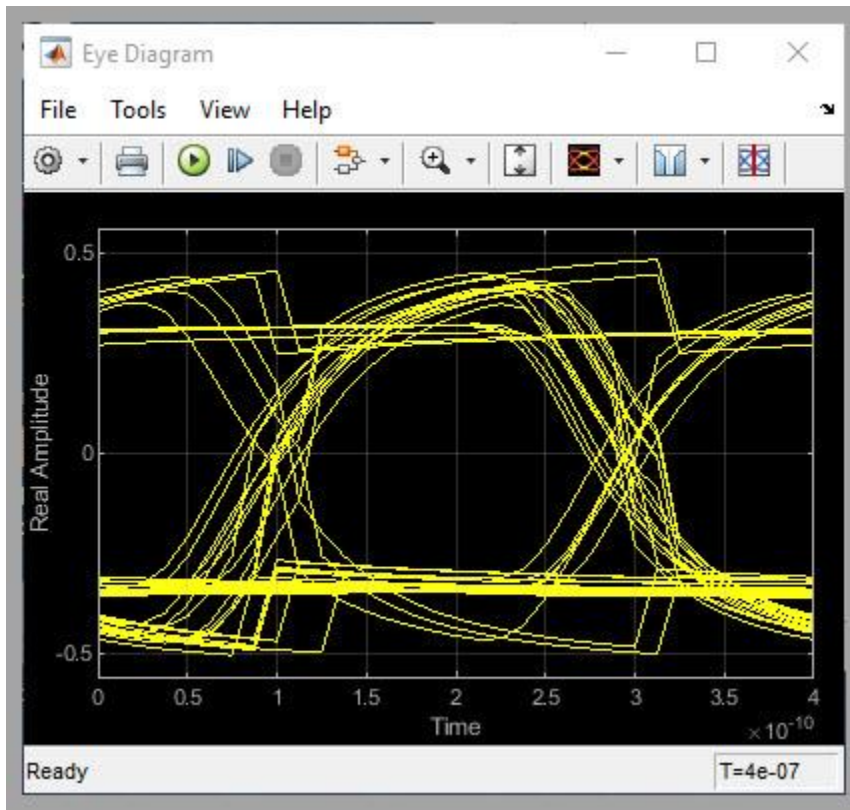
Note: Delay values less that 0 will have no effect on the resulting waveform.

Changing the current value of Rx_Use_Clock_Input

The operation of the clock forwarding is controlled by the reserved AMI parameter **Rx_Use_Clock_Input**. Changing the current value of this parameter is not supported by the SerDes IBIS-AMI Manager, so all updates to the current value are done from the Strobe Clock Times Generator mask using the **Select External Clock Input** radio buttons.

Note: If the IBIS-AMI Manager is already open, you may need to close and re-open for the changes to be visible.

Generate Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model and generates IBIS-AMI compliant model executables, IBIS and AMI files for the clock forwarding receiver.

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

Required Keywords

The IBIS-AMI Reserved input parameter **Rx_Use_Clock_Input** is required for codegen to work properly. If this parameter is not present in your model, while the model may codegen the clock-forwarding properties will not be enabled.

Export Models

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

- Update the **Rx model name** to `clock_forward_dq_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 the Rx AMI Model Settings. This will create a model executable that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Rx model Bits to ignore value** to `10` to allow enough time for the external clock waveform to settle during time domain simulations.
- Set **Models to export** to **Rx only** since we are only generating a Rx model.
- Set the **IBIS file name** to be `clock_forwarding.ibs`
- Press the **Export** button to generate models in the Target directory.

Review AMI file

The resulting Rx AMI file will look like a normal Rx AMI file with two exceptions. First, the AMI_Version is set to 7.1, while the second is the inclusion of the reserved parameter **Rx_Use_Clock_Input**.

Model Limitations

This clock forwarding AMI model requires an EDA tool that supports the new IBIS 7.1 reserved parameter **Rx_Use_Clock_Input.**

Per the IBIS 7.1 specification, it is intended that Data and Strobe AMI models will be delivered as a matched pair. This may place additional requirements on the Strobe AMI model which are discussed in the next section.

Strobe Rx IBIS-AMI Model Requirements

Per the IBIS 7.1 specification, it is intended that Data and Strobe AMI models will be delivered as a matched pair. This means that a Strobe (or Clock) AMI model will also need to be generated for use along with this Data AMI model. The Strobe model should not use a CDR to generate clock times, but instead should only output clock times at the zero-crossings of the output waveform, which can be handled by using a MATLAB Function Block. Additional features for the Strobe AMI model include the handling of strobe pre-amble (only output clock times after the end of the pre-amble) and handling of Single-Data-Rate (SDR) signals such as a Clock that only outputs clock times on the rising edge of the output waveform.

An example Strobe/Clock AMI model is attached to this example and can be opened by typing the following command:

```
>> open_system('dqs_clock_forward.slx')
```

This model contains a pass-through differential Tx and Rx. The Rx AMI model outputs clock times only at zero-crossings and contains two controls:

- **DQS_Preamble**: The number of DQS transitions to skip before outputting clock times during a DQS burst. Currently requires at least 4 DQS UI between bursts. (1tCK = 2 DQS UI). The Default value is "2tCK". When using this Rx as a Clock receiver, this value should be set to "0tCK".

- **Strobe_or_Clock**: Switch between a Strobe or Clock signal. A Strobe signal returns clock times every edge (DDR) while a Clock signal only returns clock times on rising edges (SDR). The Default is "Strobe".

Test Generated IBIS-AMI Models

The clock forwarding receiver IBIS-AMI model is now complete and ready to be tested in any industry standard AMI model simulator that supports IBIS 7.1.

References

1 IBIS-AMI 7.1 Specification

2 IBIS BIRD 204

3 IBIS BIRD 209

Design IBIS-AMI Models to Support DC Offset

This example shows how to create Rx AMI models that support DC Offset as defined in the IBIS Buffer Issue Resolution Document (BIRD) 197.7 by modifying the library blocks in SerDes Toolbox™. This example will use a DDR5 read transfer (SDRAM to Controller) to demonstrate the setup.

Background

Statistical simulations with AMI models use an Impulse Response as the input to the Init function. Since an Impulse Response loses any DC information about a signal, an AMI simulation will always be centered around the new mid-point of the resulting signal swing. Since AMI models are now being used in single-ended NRZ channels (such as DDR5), the signal will have lost the original mid-point of its signal swing. Losing this original DC information means the receiver AMI model cannot accurately model things like saturation.

IBIS BIRD 197.7, adds a new AMI reserved parameter, **DC_Offset**, which allows the EDA tool to compute the mid-point of the signal-swing and pass this value to the receiver AMI model. The input value of DC_Offset is the mean value of the steady state high and low voltages of the analog channel step response at the Rx pad.

This example introduces DC Offset in SerDes Toolbox by showing how to generate and test an IBIS-AMI executable that supports DC_Offset.

Rx IBIS-AMI Model Setup in Simulink

To begin, load the DC Offset Simulink model and review the model setup. Start by typing the following command:

```
>> open_system('dc_offset.slx')
```

This will bring up the following SerDes system:
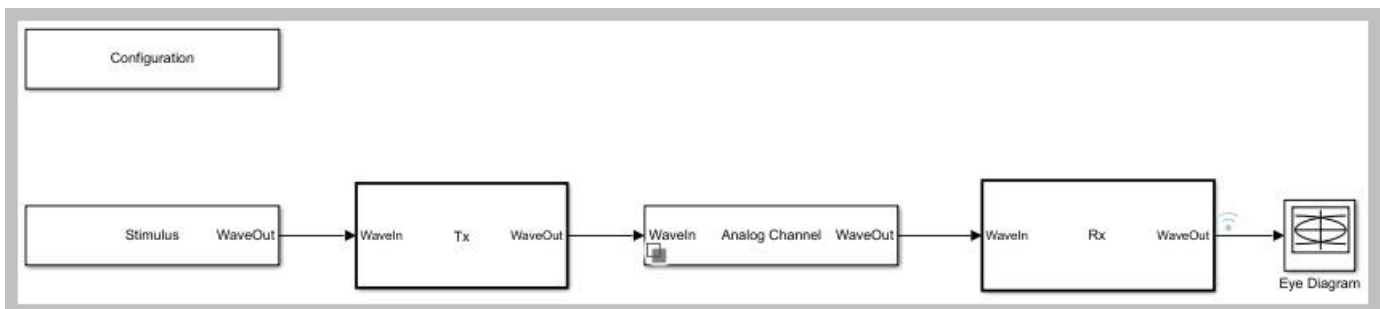
Review Simulink Model Setup

This Simulink SerDes system contains the standard configuration: Stimulus, Tx, Analog Channel and Rx blocks, with a new DC Offset block added to the SatAmp block in the Rx. The setup of each of these blocks will be reviewed below.

Configuration Block

- **Symbol Time** is set to `200.0`ps (5.0Gbps).
- **Target BER** is set to `1e-16`.

- **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.

Stimulus Block

- The Stimulus block is set to default values.

Tx Block

The Tx contains no equalization blocks. Since this example is focused on the Rx model, the Tx block is untouched.

Analog Channel Block

- **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 `2.5` GHz, which is the Nyquist frequency for 5.0 GHz
- The Tx Analog model is set up so that **Voltage** is `1.1` V, **Rise time** is `10` ps, **R** (output resistance) is `50` ohms, and **C** (capacitance) is `0.65`pF.
- The Rx Analog model is set up so that **R** (input resistance) is `40` ohms and **C** (capacitance) is `0.65`pF.

Rx Block

The Rx block contains 3 equalization blocks: a CTLE block with an AC gain of 0dB and 8 DC Gain settings, a DFE block that uses four DFE taps with **Initial tap weights** set to 0, the **Minimum tap values** set to `[-0.2 -0.1 -0.1 -0.1]`V, and the **Maximum tap values** set to `[0.2 0.01 0.1 0.1]`V, and a SatAmp block.

The SatAmp block has the **Limit** set to 1.1V and the **Linear gain** set to 1V/V. It also contains the new DC Offset sub-system.

SatAmp Block

Per IBIS BIRD 197.7, "it is assumed that the waveform input to the Rx AMI_GetWave function is the physical Rx input waveform minus the input value of this DC_Offset. The Rx AMI_GetWave function may choose to reconstruct the physical input waveform by adding the input value of DC_Offset to the input waveform."

Inside the SatAmp block, the value of DC Offset is being added to the incoming waveform before it is applied to the Saturating Amplifier system object. This allows the saturation amplifier to be applied to the waveform with the correct signal swing.

Also per BIRD 197.7, "the Rx AMI_GetWave output waveform returned by the AMI model shall swing around zero volts." Therefore, after the saturation amplifier, the value of DC Offset is subtracted from the incoming waveform to retain the original signal swing.
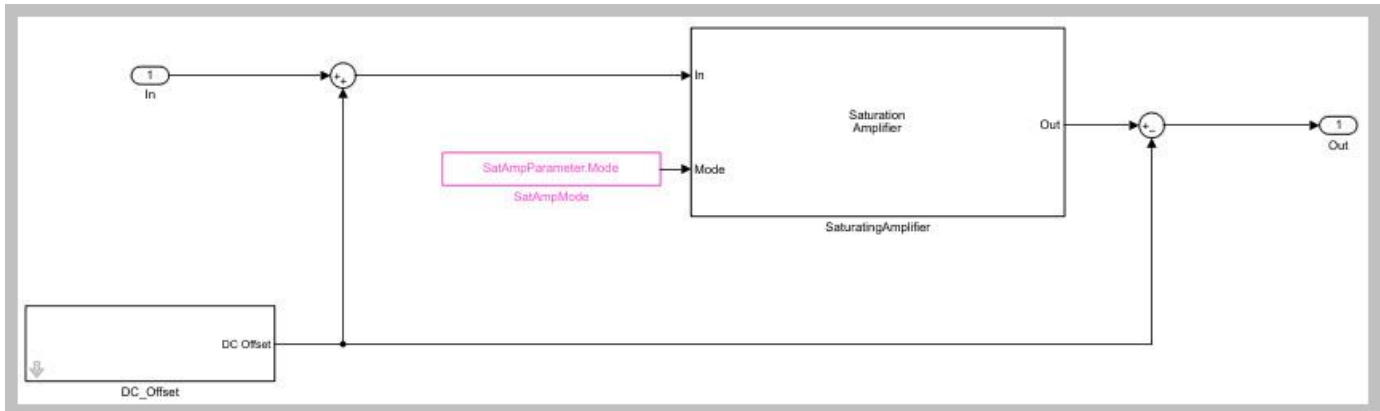
DC Offset Sub-System (New)

The DC Offset sub-system block is used to set the current value of DC_Offset for testing purposes. While the DC_Offset parameter is an input to the AMI model, the EDA tool ignores the value specified in the .ami file and calculates the correct value at simulation time and passes this value to the model instead. Since this value is not calculated by Simulink, the mask for this sub-system provides a method for specifying this value manually.
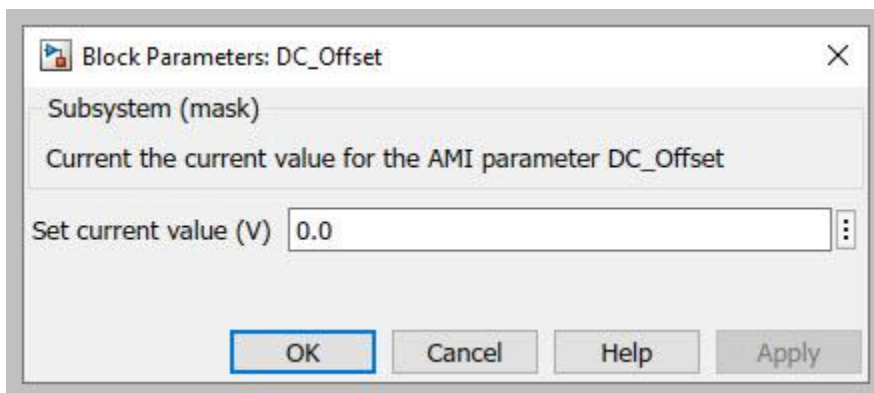
This new DC_Offset block is not yet included in the SerDes Toolbox library. You can add this subsystem to a new Simulink model by copy-pasting it from this example. Pasting this block into a new SerDes Toolbox model will also add the required **DC_Offset** AMI parameter and `DC_Offset` Simulink signal to the Model Workspace.

Run the Simulink Model

The Simulink model is ready to run. Press the run button to launch the simulation.

As the simulation runs, the Time Domain eye diagram gets constantly updated. With DC Offset set to 0.0V in the mask, the eye diagram should look like the following:
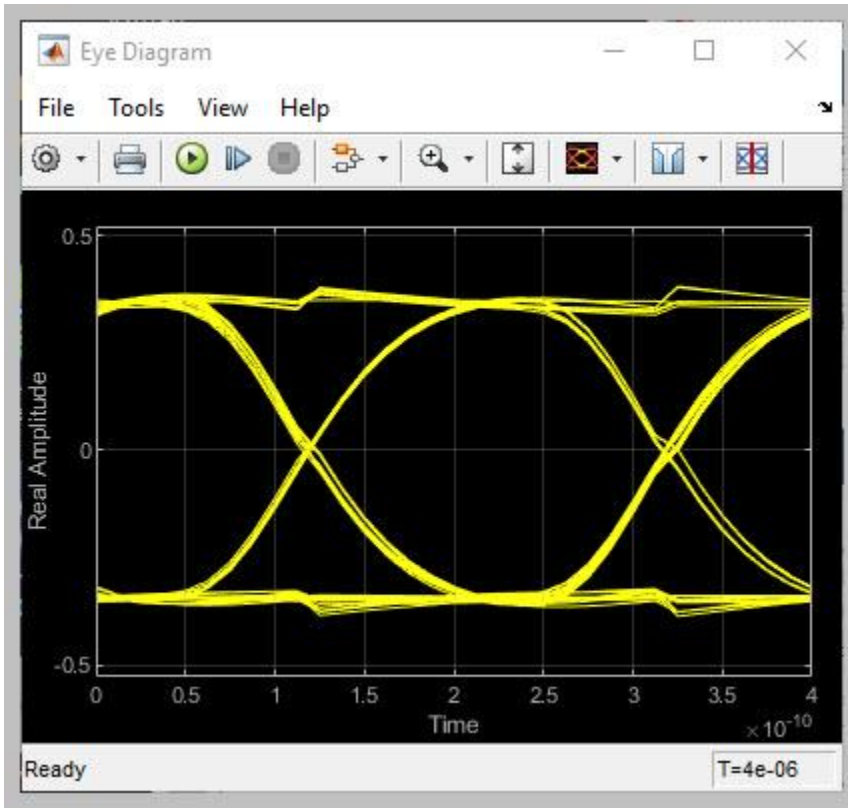
After the simulation is complete, the Init Statistical and Time Domain Analysis Results plot becomes available:
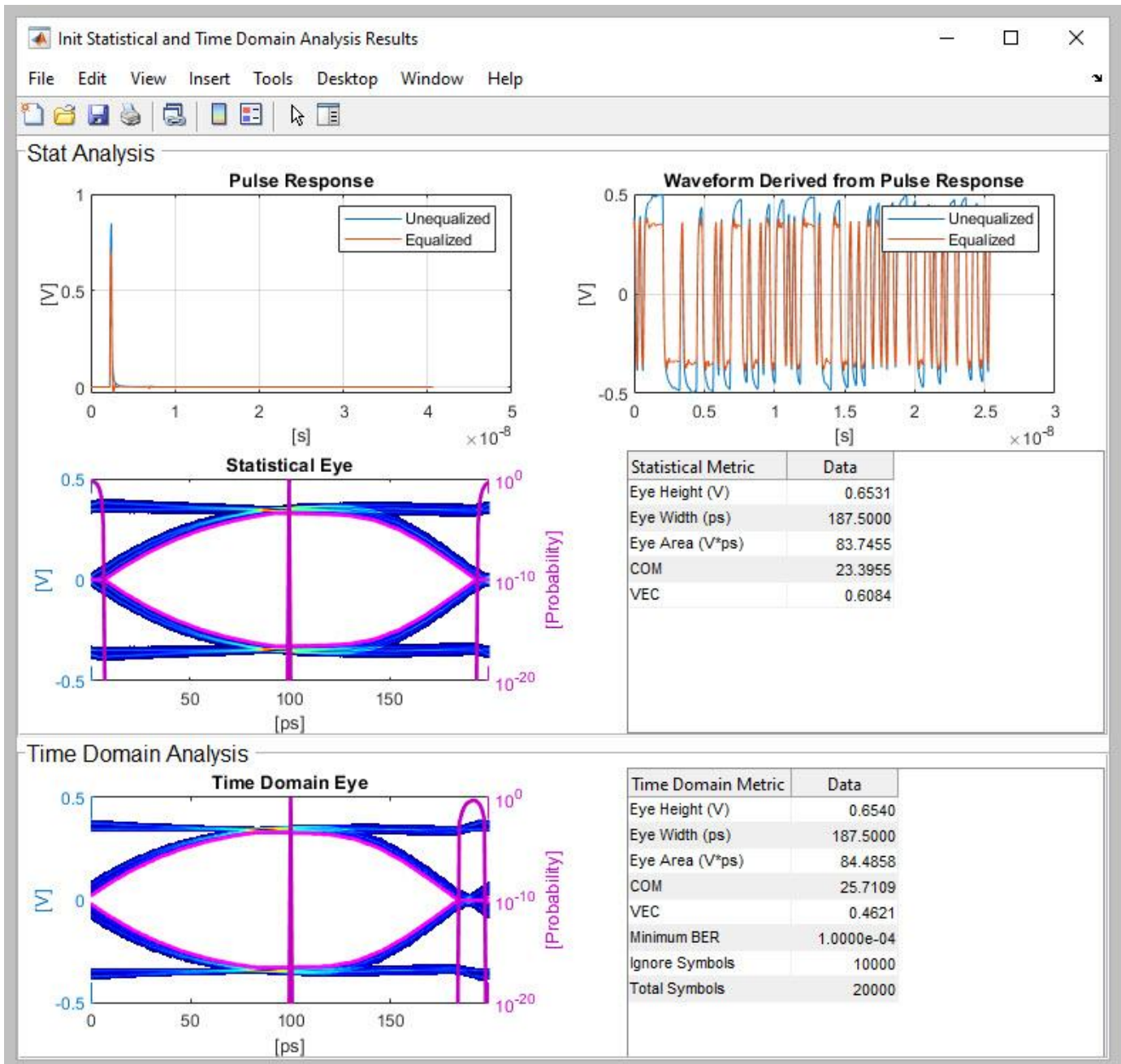
Open the DC Offset mask, change the DC Offset value to 0.755V, then re-run the simulation. Now the Time Domain eye diagram should look like the following. Note the reduced signal swing due to the saturation.
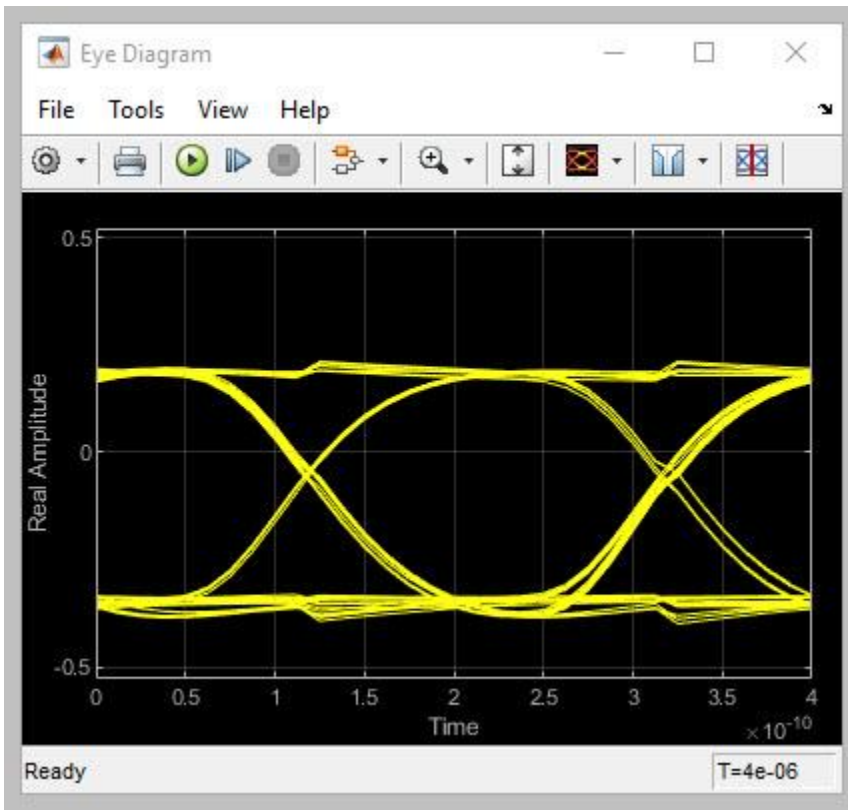
After the simulation is complete, the Init Statistical and Time Domain Analysis Results plot becomes available. Note that the Statistical eye remains unchanged, while the Time Domain eye is showing the non-linear effects of saturation.
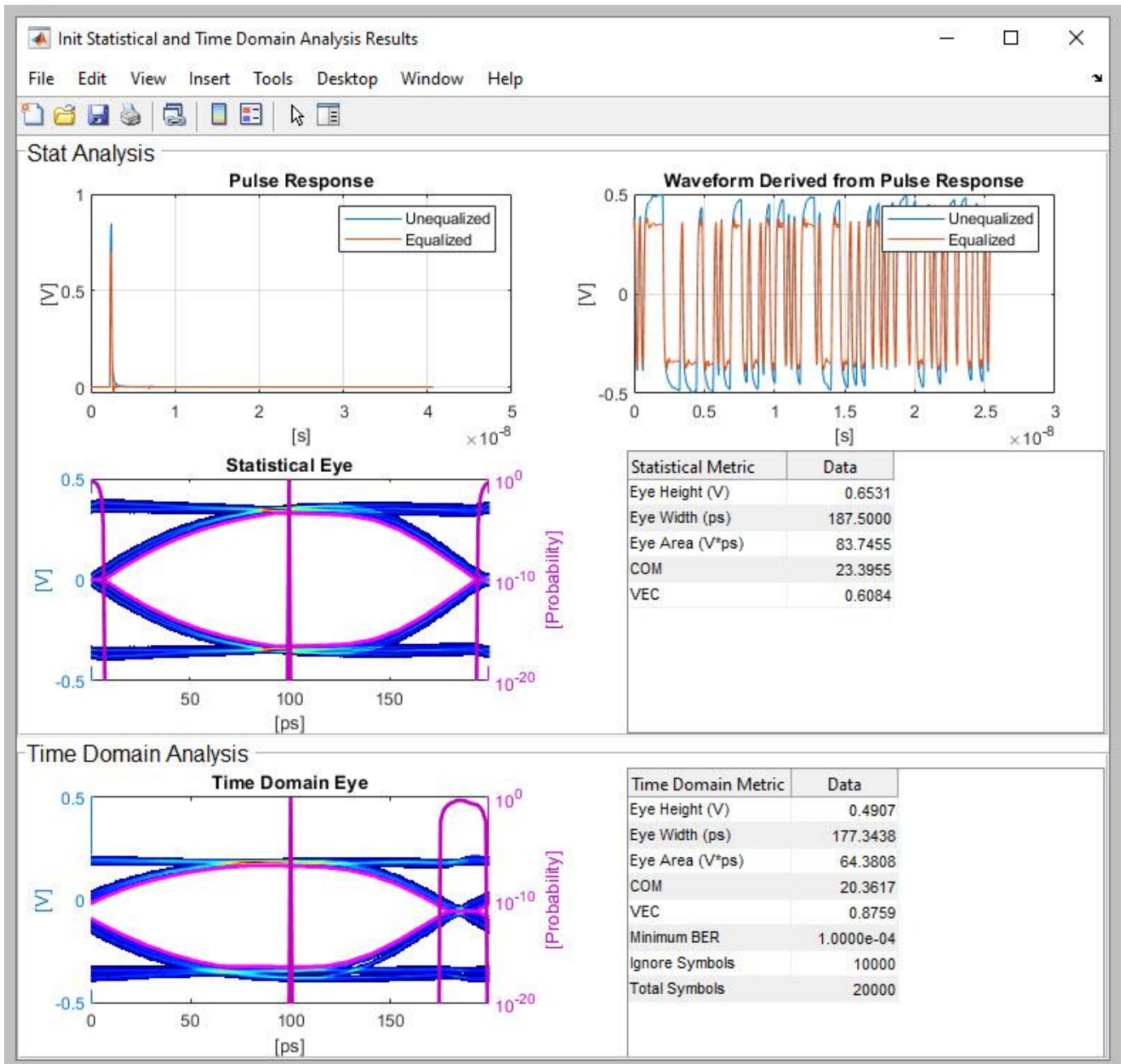
Note that since DC Offset only affects the Time Domain results, the Statistical results do not reflect the effects of DC Offset.

Changing the current value of DC_Offset

The operation of the DC Offset is controlled by the reserved AMI parameter **DC_Offset**. Changing the current value of this parameter is not supported by the SerDes IBIS-AMI Manager, so all updates to the current value are done from the DC Offset sub-system mask.

Note: If the IBIS-AMI Manager is already open, you may need to close and re-open it for the changes to be visible.

Generate Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model and generates IBIS-AMI compliant model executables, IBIS and AMI files for the DC Offset receiver.

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

Required Keywords

The IBIS-AMI Reserved input parameter **DC_Offset** is required for codegen to work properly. If this parameter is not present in your model the model may codegen, however the DC Offset properties will not be enabled.

Export Models

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

- Update the **Rx model name** to dc_offset_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 the Rx AMI Model Settings. This will create a model executable that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Rx model Bits to ignore value** to 10,000 to allow enough time for the external clock waveform to settle during time domain simulations.
- Set **Models to export** to **Rx only** since we are only generating a Rx model.
- Set the **IBIS file name** to be dc_offset.ibs
- Press the **Export** button to generate models in the Target directory.

Review AMI file

The resulting Rx AMI file will look like a normal Rx AMI file with two exceptions. First, the AMI_Version is set to 7.1. The second is the inclusion of the reserved parameter **DC_Offset**. Since both of these changes are from an unreleased version of the IBIS Specification, either one will cause this AMI file to fail the IBIS AMI Checker (which is currently on version 7.0.1). If this causes any problems in your EDA tool you may want to skip the running of the AMI Checker.

Model Limitations

This DC Offset AMI model requires an EDA tool that supports BIRD 197.7.

Test Generated IBIS-AMI Models

The DC Offset receiver IBIS-AMI model is now complete and ready to be tested in any industry standard AMI model simulator that supports BIRD 197.7.

References

IBIS-AMI Specification

IBIS BIRD 197.7

Simulate Crosstalk Cancellation in IBIS AMI Receiver Models

This example shows how to simulate crosstalk cancellation in a SerDes Rx AMI model. You can disable crosstalk cancellation or select a specific aggressor column in the impulse_matrix to cancel. You can also replace the filter used in the example `cancellationFilter.m` with a CTLE that represents the filter implemented in the hardware.

Open the **SerDes Designer** app. Crosstalk cancellation is implemented in a Pass-Through block inserted at the beginning of the Rx model. Any additional equalization (e.g., CTLE, AGC, DFE) is added after this pass-through block.
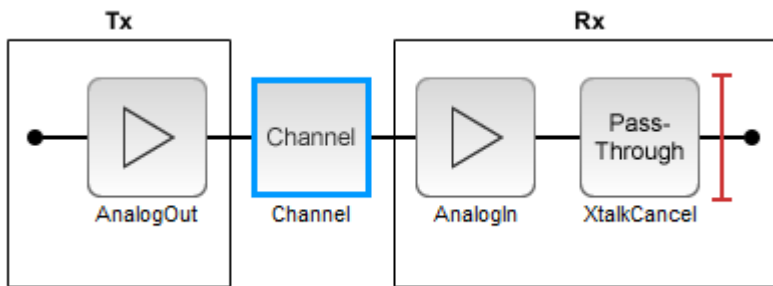
Export the model with default values to Simulink®.
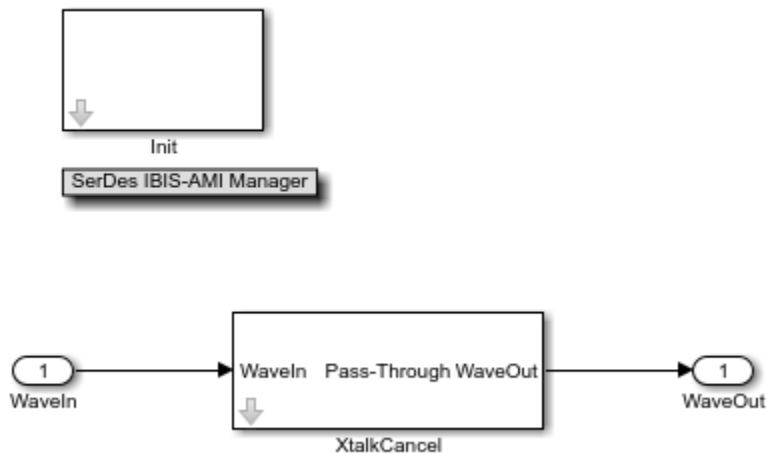
Add `Model_Specific` parameters **Column**, **Gain** and **Delay** to the Rx XtalkCancel pass-through block in the SerDes IBIS-AMI Manager.

Crosstalk Cancellation applies a filter to an aggressor waveform. This filtered aggressor waveform is then amplified by Gain and shifted by Delay to maximize the amount of cancellation of the crosstalk applied to the victim waveform.

Column is the column in the `impulse_matrix` to be cancelled. **Column** is **Usage** In, **Type** Integer and **Format** Range. The first column in the `impulse_matrix` is the victim through impulse response. If Column is <= 1 then no aggressor will be cancelled. If Column = 2, then the first

aggressor in the `impulse_matrix` will be cancelled. The Min value of Range should be zero, the Max value should be set to the value of `Max_init_Aggressors` +1.

Gain is **Usage** `Out`, **Type** `Float` and **Format** `Value`. **Gain** is unitless.

Delay is **Usage** `Out`, **Type** `Float` and **Format** `Value`. The unit of **Delay** is seconds.

Crosstalk Cancellation is implemented in the custom user code of the Initialize Function.

```
28      %% BEGIN: Custom user code area (retained when 'Refresh Init' button is pre
29 -    XtalkCancelDelay=0; % User added AMI parameter from SerDes IBIS-AMI Manage
30 -    XtalkCancelGain=1; % User added AMI parameter from SerDes IBIS-AMI Manager
31 -    XtalkCancelParameter.Column; % User added AMI parameter from SerDes IBIS-A

33 -    iAgr=XtalkCancelParameter.Column;
34 -    if(iAgr > 1)
35 -        if(iAgr < Aggressors+1)
36              % Crosstalk cancelation is enabled if Column>1 and < # columns in
37              % impulse_matrix
38 -            SamplesPerBit=round(SymbolTime/SampleInterval);
39 -            [~,maxIR]=max(abs(LocalImpulse(1:end,1)));
40 -            lengthIR=maxIR+20*SamplesPerBit;
41 -            [canceledSR,Gain,idelay]= ...
42                  crosstalkCancel(LocalImpulse,lengthIR,iAgr, ...
43                  SamplesPerBit,SampleInterval);
44 -             XtalkCancelGain=Gain;
45 -             XtalkCancelDelay=idelay*SampleInterval;

47              % now replace begining of xtalk IR with cancelled xtalk IR
48              % canceliled IR is derivitive of canceled SR

51 -            for k=abs(idelay)+1:lengthIR-abs(idelay)-1
52 -                LocalImpulse(k,iAgr)=(canceledSR(k+1)-canceledSR(k))/SampleInte
53 -            end
54 -        end
55 -    end

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

Lines 34:35 limits cancelation to a single aggressor column in the impulse matrix

Lines 39:40 limits the cancellation matrix to 20 UI past the cursor location of the through channel (peak value of the through impulse response).

Lines 41:43 calls `crosstalkCancel` to calculate the cancelled step response of the column `iAgr` of the `LocalImpulse` matrix.

Lines 51:53 converts the returned cancel step response to a cancelled impulse response and replaces that section of the iAgr column in the impulse matrix. Function `crosstalkCancel` applies the `cancellationFilter` to the aggressor through step response. In this example `cancellationFilter` takes the derivative of the victim impulse response. The best **Gain** and **Delay** is determined by doing a coarse and fine grid search. The first/coarsest grid has Gain=.001, 4,8,16.

Subsequent searches reduce the grid size by a factor of 4. For each Gain the function `fitDelay` determines the best delay by sweeping the delay from -½ to +½ UI in increments of `sampleInterval`. The cancelled crosstalk step response is the aggressor step response minus the filtered through step response with the Gain and Delay applied. The objective function is the sum of the squares of the cancelled crosstalk step response from (1:1UI) past the peak magnitude of the aggressor step response.

The actual shape and magnitude of cancellationFilteris implementation specific. `thruSRfiltered` is the `cancellationFilter` applied to victim through step response. FEXT (far-end crosstalk) assumes that the aggressor through step response is the same as the victim through step response.

```matlab
1  function thruSRfiltered=cancellationFilter(thruSR,sampleInterval)
2      % this function applies the hardware cancellation filter to the
3      % aggressor waveform. This filter just takes the derivitive of the
4      % aggressor waveform and scales it to be a reasonable amplitude
5      lngth=length(thruSR);
6      thruSRfiltered=zeros(lngth,1);
7      for i=2:length(thruSR)
8          thruSRfiltered(i)=(thruSR(i)-thruSR(i-1))/sampleInterval;
9      end
10     thruSRfiltered=-thruSRfiltered*sampleInterval;
11 end
```

Lines 7:9 take the derivative of the through step response. This represents the shape of the crosstalk step response. Multiplying this derivative filter by `sampleInterval` makes the magnitude in the range of what might be expected in a real filter.

The crosstalk cancellation example is tested with a simple loss channel and idealize FEXT crosstalk.
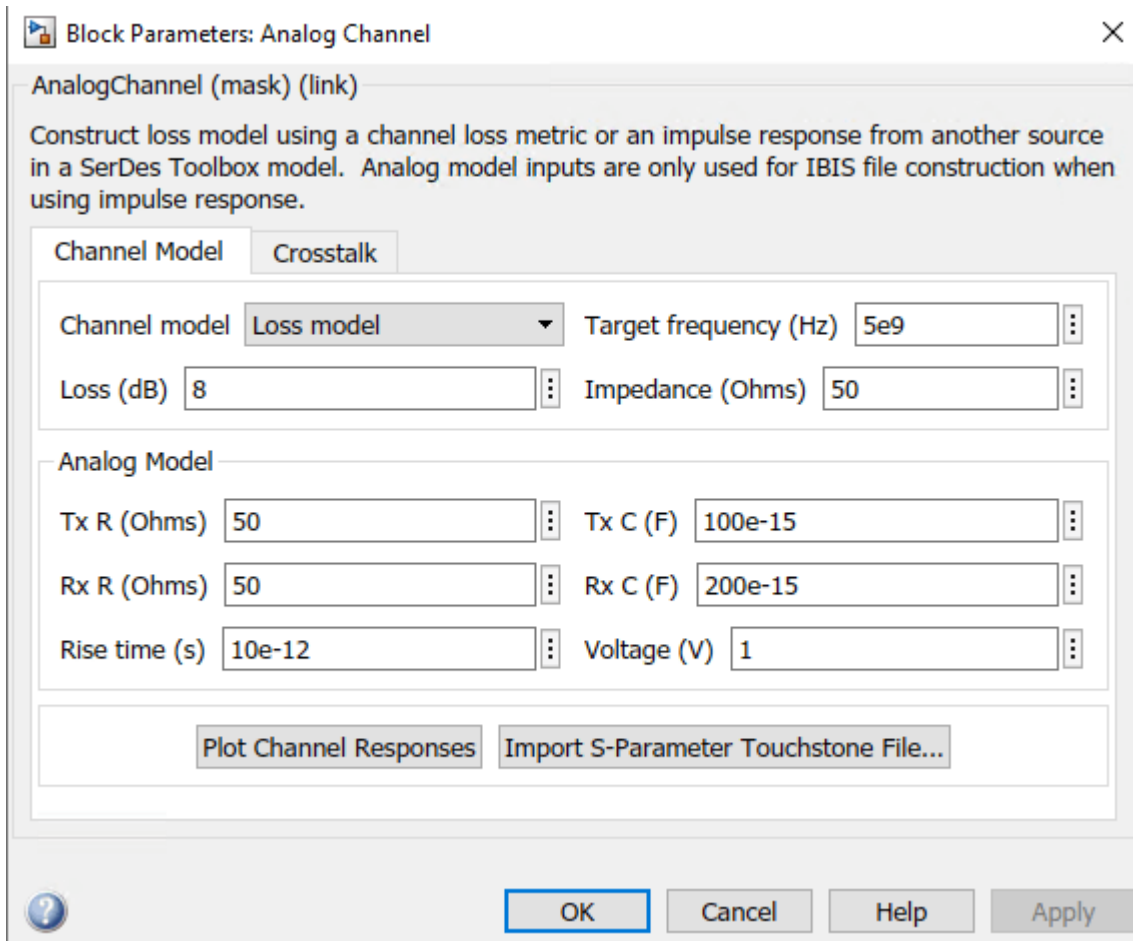
The near end crosstalk (NEXT) to `0`, and the far end crosstalk (FEXT) is set to `0.1`.
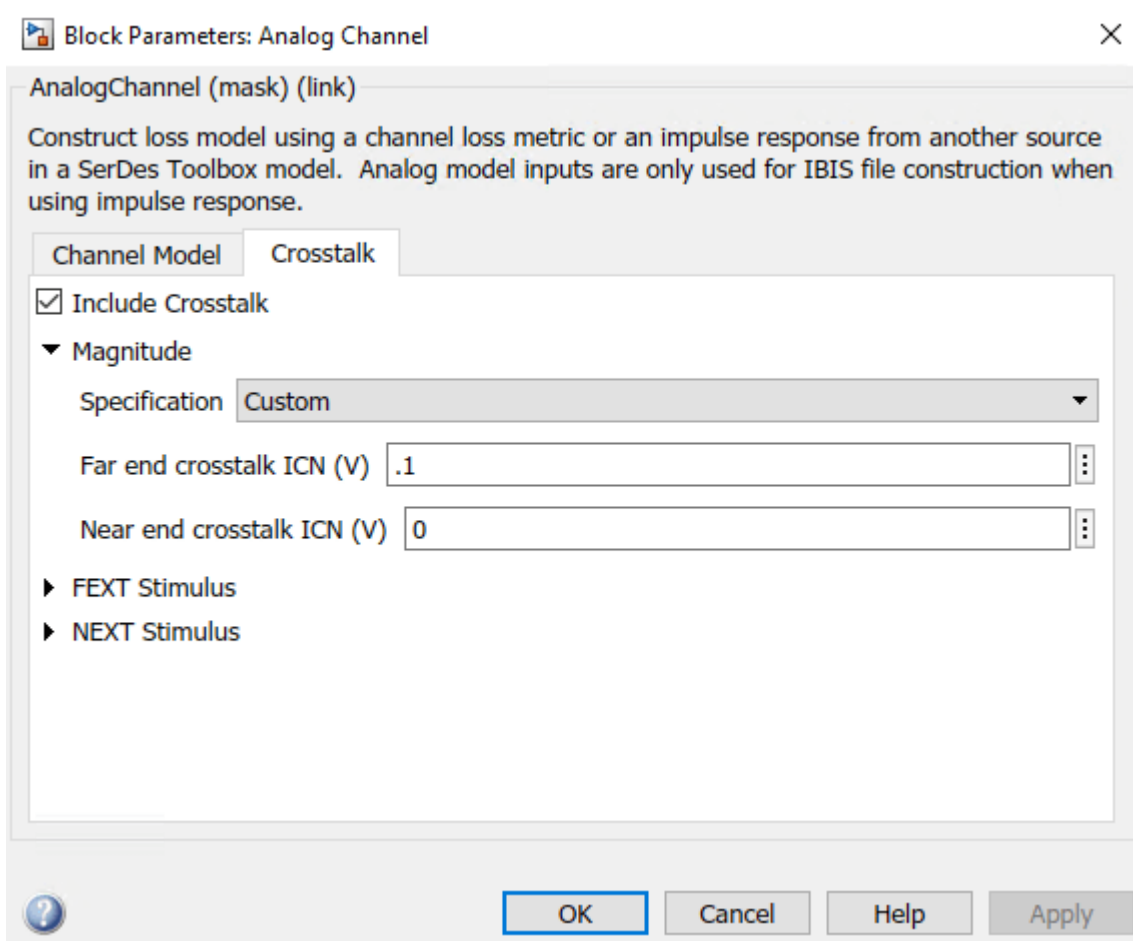
The `impulse_matrix` has three columns in this example. The first column contains the victim through impulse response. The second column contains the FEXT aggressor crosstalk impulse response. In the `XtalkCancel` Column parameter is set to 2, to cancel the second column aggressor crosstalk impulse response.

Run the simulation to view the results.

The `agr1` Pulse Response is totally cancelled. Compare the Unequalized `agr1` Pulse Response with the Equalized `agr1` Pulse Response. The channel crosstalk model is ideal, it assumes the crosstalk applied to the victim is the derivative of the aggressor waveform. The filter applied to the victim through step response is also ideal. Therefore, the crosstalk cancelation is totally effective in this example. Real crosstalk channel channels and real crosstalk cancellation filters will be less effective.

IBIS AMI supports crosstalk cancelation in the statistical flow because the input to the `AMI_Init` function contains both the victim through and aggressor impulse responses. Crosstalk cancellation cannot be supported in IBIS time domain simulations because `AMI_GetWave` only has the victim waveform as input. The IBIS standard needs to be enhanced to add aggressor waveforms to the `AMI_GetWave` function.
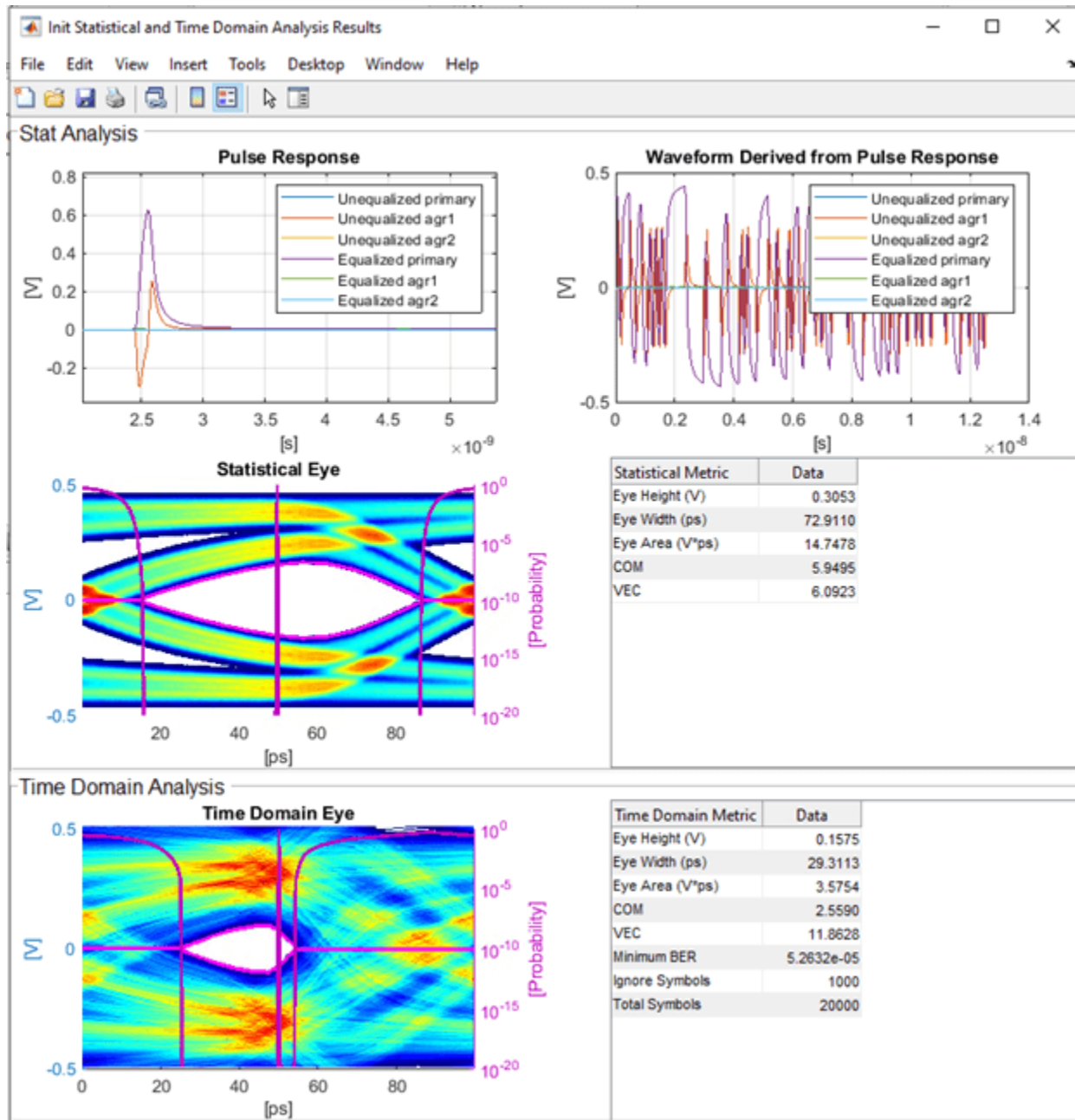
Design IBIS-AMI Models to Support Rx Decision Time

This example shows how to create Rx AMI models that support Rx Decision Time as defined in the IBIS (I/O Buffer Information Specification) version 7.1, by modifying the library blocks in SerDes Toolbox™. This example will use a typical SerDes Rx AMI model to demonstrate the setup.

Background

In the AMI Init flow the IBIS-AMI model delivers to the EDA tool an impulse response, but no information about any clock that might be recovered from the signal. To apply statistical analysis to this information, the EDA tool must determine the time at which it expects the signal to be sampled at the data decision point. The method used to determine this time is not defined in the specification so depending on the algorithm used, each EDA tool may come up with a slightly different clock location, which in turn can result in inconsistent statistical results across EDA tools.

For more consistent results, the EDA tool should use a time determined by the receiver IBIS-AMI model instead of an internal algorithm. IBIS 7.1 introduced a new AMI reserved parameter, **Rx_Decision_Time**, which is passed from the Rx AMI model to the EDA tool to specify the exact time to sample the impulse response.

This example introduces Rx Decision Time in SerDes Toolbox by showing how to generate and test an IBIS-AMI executable that supports Rx_Decision_Time.

Rx IBIS-AMI Model Setup in Simulink

To begin, load the Rx Decision Time Simulink model and review the model setup. Start by typing the following command:

```
>> open_system('rx_decision_time.slx')
```

This will bring up the following SerDes system:
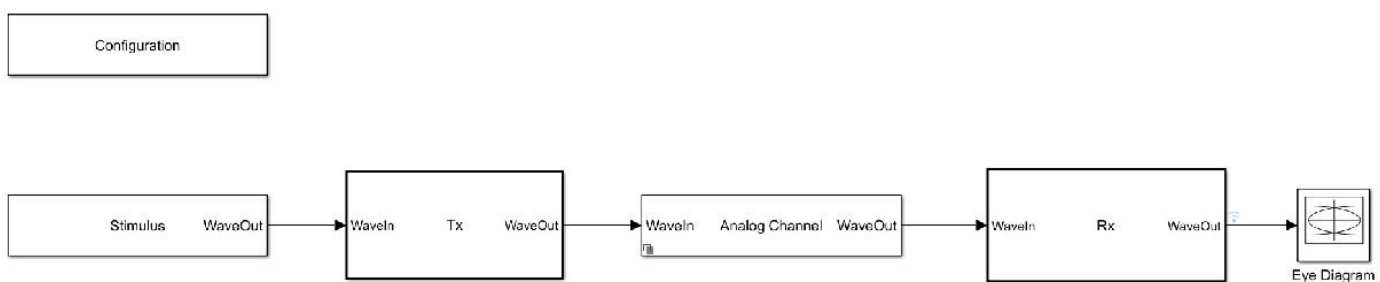
Review Simulink Model Setup

This Simulink SerDes system contains the standard configuration: Stimulus, Tx, Analog Channel and Rx blocks. The setup of each of these blocks is in their default configuration and will be reviewed below.

Configuration Block

• **Symbol Time** is set to `100.0`ps (10.0Gbps).
• **Target BER** is set to `1e-06.`

- **Signaling** is set to `Differential`.
- **Samples per Symbol** and **Modulation** are kept at default values, which are `16` and `NRZ` (nonreturn to zero), respectively.

Stimulus Block

- The Stimulus block is set to default values.

Tx Block

The Tx contains no equalization blocks. Since this example is focused on the Rx model, the Tx block is untouched.

Analog Channel Block

- **Channel loss** is set to 8 dB.
- **Impedance** is set to `100` ohms.
- **Target Frequency** is set to `5.0` GHz, which is the Nyquist frequency for 10.0 GHz
- The Tx Analog model is set up so that **Voltage** is `1.0` V, **Rise time** is `10` ps, **R** (output resistance) is `50` ohms, and **C** (capacitance) is `0.1`pF.
- The Rx Analog model is set up so that **R** (input resistance) is `50` ohms and **C** (capacitance) is `0.2`pF.

Rx Block

The Rx block contains 3 equalization blocks: a CTLE block with 9 DC Gain and Peaking Gain settings, a DFE block that uses four DFE taps with **Initial tap weights** set to 0, and a SatAmp block with **Limit** set to 1.2V and the **Linear gain** set to 1V/V

Enable Rx Decision Time

To enable Rx Decision Time, open the SerDes IBIS-AMI Manager, switch to the **AMI – Rx Tab**, and click on **Reserved Parameters**. Under **General**, check the box for `Rx_Decision_Time`, then click OK. This will add the Rx_Decision_Time parameter to the Reserved Parameters sections of the Rx AMI model.
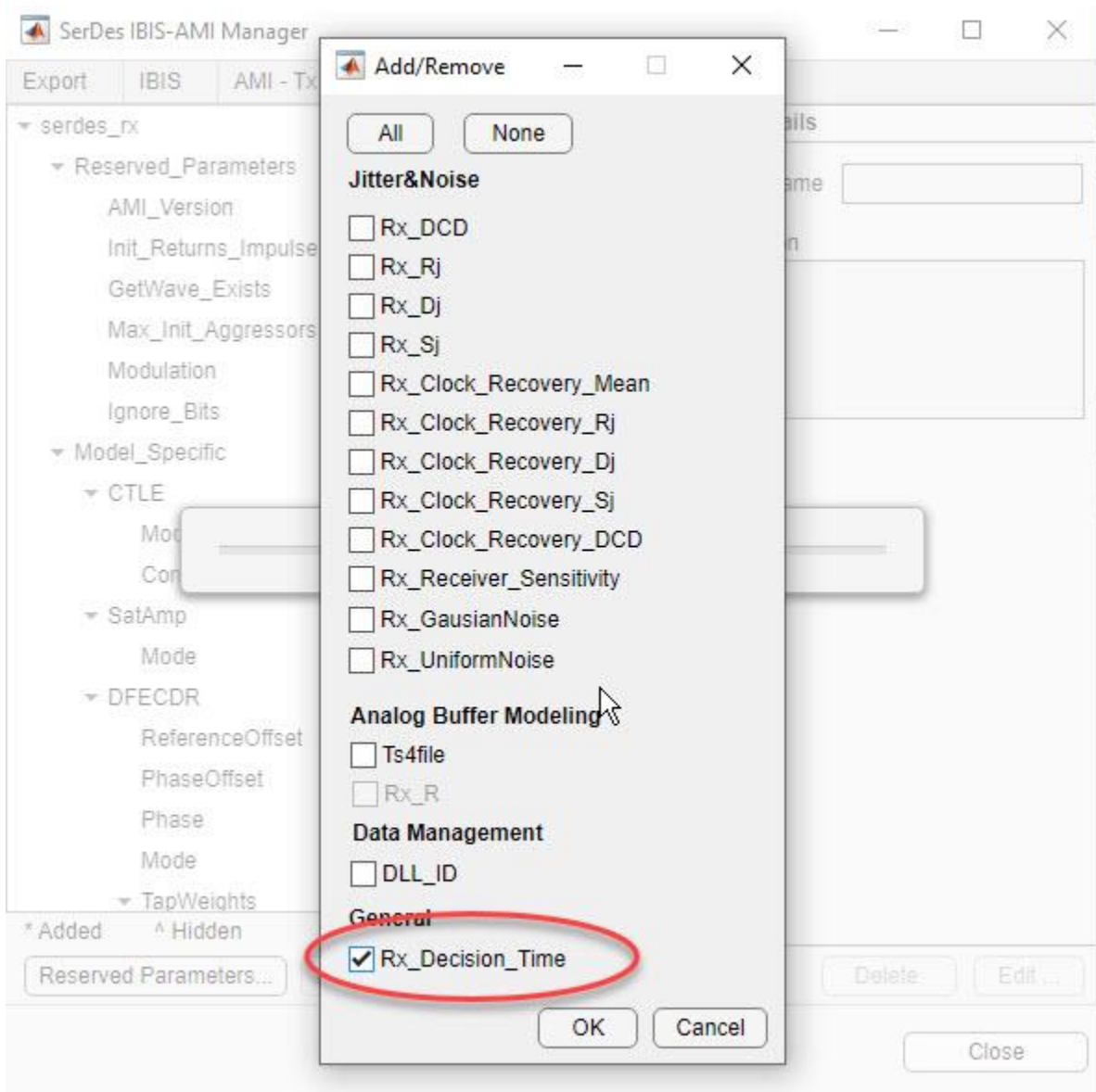
Note that the default Type for Rx_Decision_Time is **UI**, but this can be changed to **Float** by clicking on the Edit button. Keep this Type in mind if you decide to customize the calculation of Rx_Decision_Time.

After pressing **OK**, the dialog closes and you will get the following pop-up:
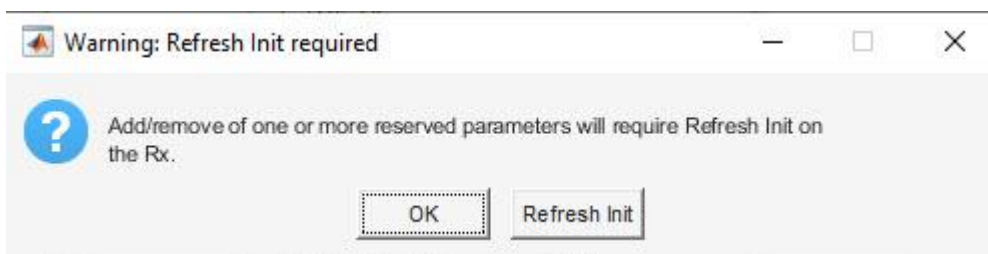
If you wish to run Refresh Init later, clicking on **OK** will do nothing and close the dialog.

Clicking on **Refresh Init** will add the Rx_Decision_Time parameter to the Init code. This parameter is automatically associated with the parameter DFECDRPhase, which is the clock phase output of the DFECDR block.

```
71
72    %% Impulse response processing via system objects
73    % Return impulse response and any Out or InOut AMI parameters for serdes.CTLE instance
74    [LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
75    % Direct Impulse response processing by serdes.SaturatingAmplifier is not recommended.
76    % LocalImpulse = SatAmpInit(LocalImpulse);
77    % Return impulse response and any Out or InOut AMI parameters for serdes.DFECDR instance
78    [LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
79    Rx_Decision_Time = DFECDRPhase;
80    %% Impulse response reformating
81    % Reshape LocalImpulse matrix into a vector using RowSize and Aggressors
82    ImpulseOut(1:RowSize*(Aggressors+1)) = LocalImpulse;
83    end
```

If your goal for Rx_Decision_Time was only to achieve clock location consistency across EDA tools, you are now done with the implementation. The next steps are to Run the Simulink Model on page 6-50 and then Generate IBIS-AMI Models on page 6-60.

If you would like to customize the calculation of Rx_Decision_Time, see Customizing the Value of Rx Decision Time on page 6-52.

Run the Simulink Model

The Simulink model is ready to run. Press the run button to launch the simulation.

As the simulation runs, the Time Domain eye diagram gets constantly updated. The eye diagram should look like the following:
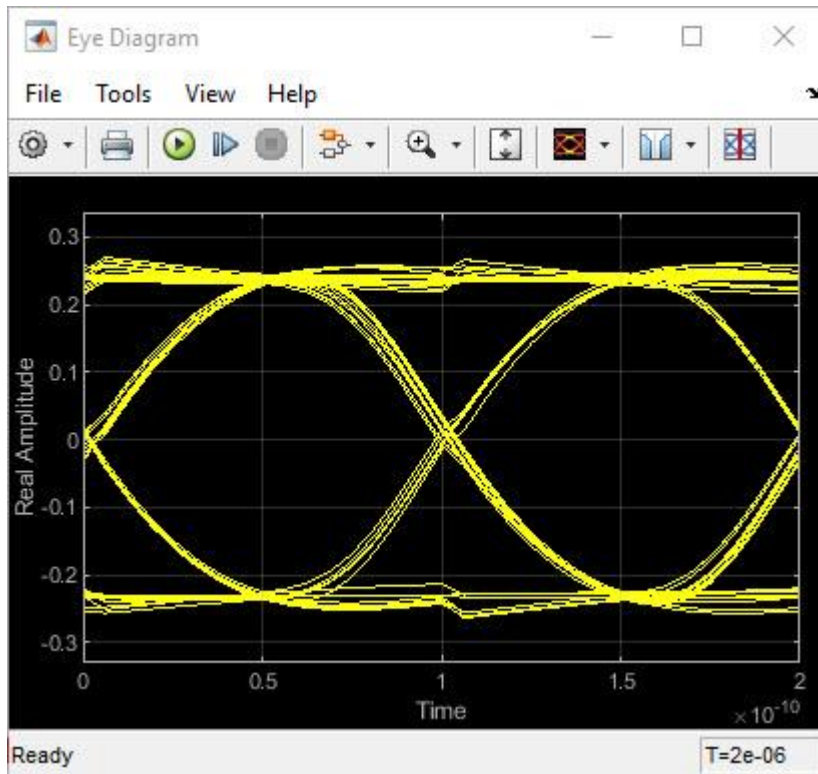
After the simulation is complete, the Init Statistical and Time Domain Analysis Results plot becomes available. The clock PDF shown in the Statistical Eye has now been placed using the value of Rx_Decision_Time:
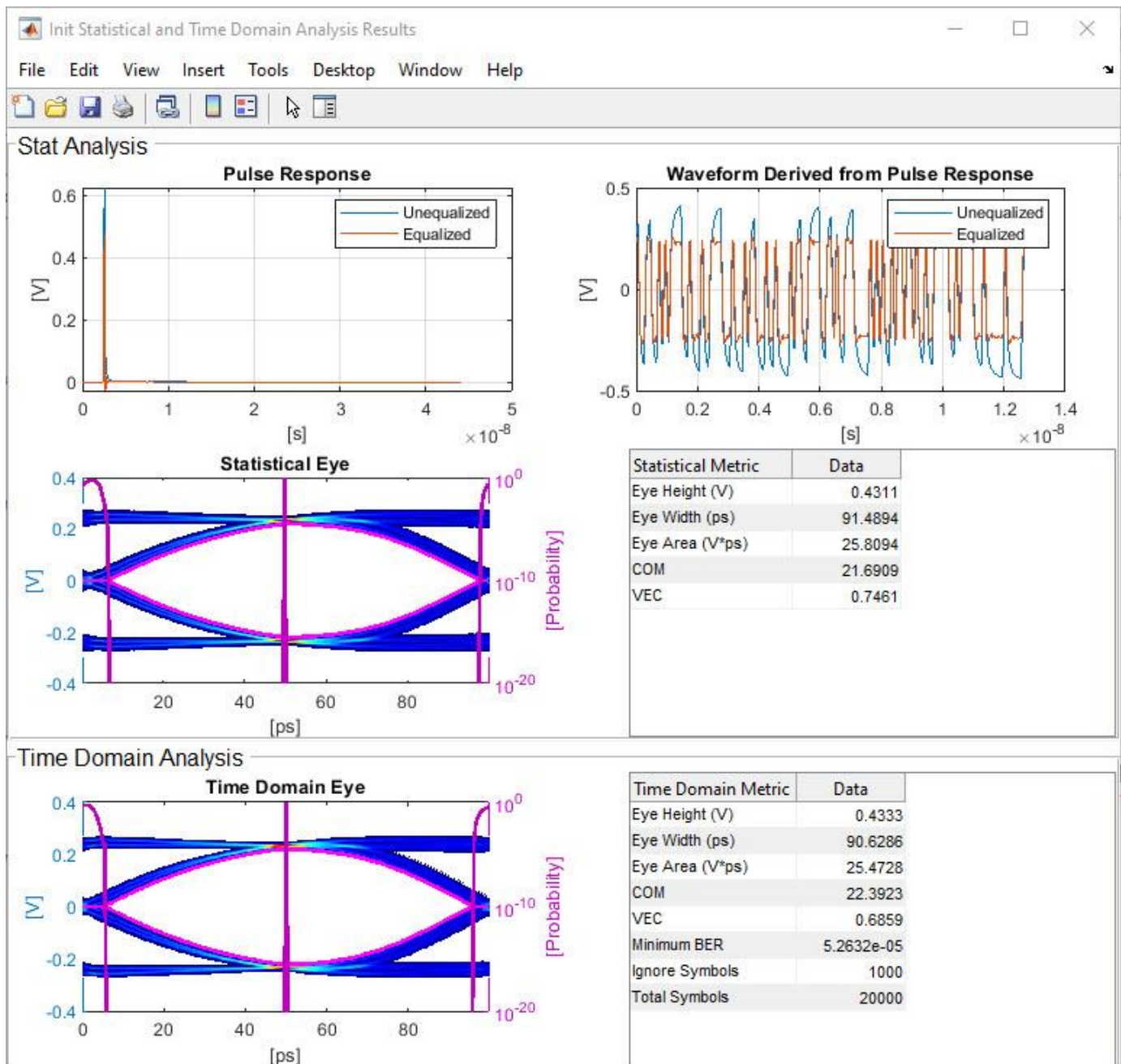
Customizing the Value of Rx Decision Time

If you would like to use a value for Rx_Decision_Time other than the Phase output of the CDR or DFECDR, this is accomplished by customizing the "Custom user code area" in the Initialize MATLAB function block.

If you have not already done so, you must first run Refresh Init from the Init block mask by pressing the **Refresh Init** button:
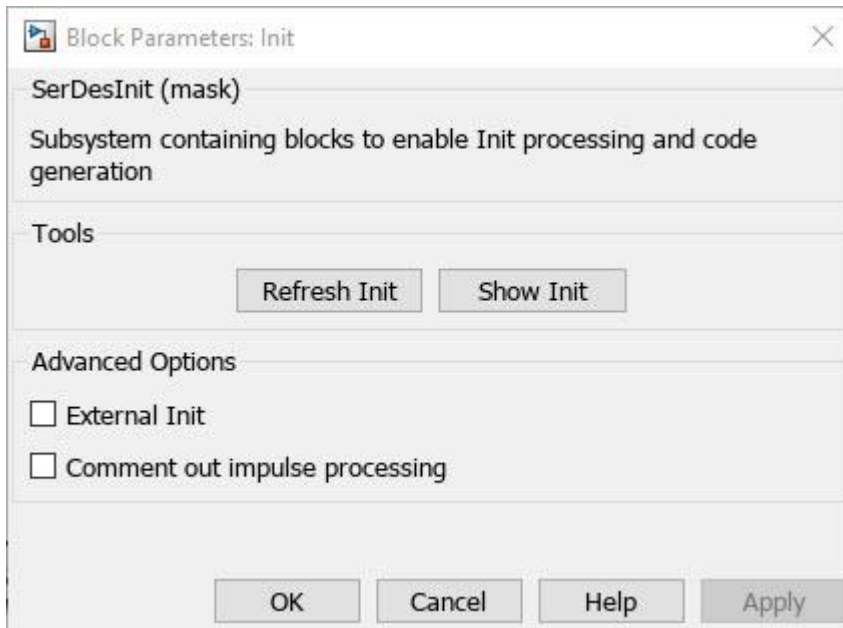
You can then view the Init code by clicking on the **Show Init** button.

If you review the Init code, you will see that the `Rx_Decision_Time` parameter has been automatically associated with the parameter `DFECDRPhase`, which is the clock phase output of the DFECDR block as noted previously. However, you will also see that this association was made <u>after</u> the Custom user code area, which means that any code entered in the Custom user code area above will be completely ignored as `Rx_Decision_Time` gets redefined in the Impulse response processing section. You should also note that since the association was made <u>outside</u> of the Custom user code area, if you were to manually remove this association it will be added back in the next time Refresh Init is run.

Comment out impulse processing

These limitations are resolved by using the option to comment out the impulse response section of the Init code. Check the box for **Comment out impulse processing**, then run **Refresh Init**. Looking at the bottom of the Init code, you will see that the entire Impulse response processing via system objects section of the Init code has been commented out:

```
71
72      %% Impulse response processing via system objects
73   ⊟  % Return impulse response and any Out or InOut AMI parameters for serdes.CTLE instance
74      % [LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
75      % Direct Impulse response processing by serdes.SaturatingAmplifier is not recommended.
76      % LocalImpulse = SatAmpInit(LocalImpulse);
77      % Return impulse response and any Out or InOut AMI parameters for serdes.DFECDR instance
78      % [LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
79      % Rx_Decision_Time = DFECDRPhase;
80      %% Impulse response reformating
81      % Reshape LocalImpulse matrix into a vector using RowSize and Aggressors
82      ImpulseOut(1:RowSize*(Aggressors+1)) = LocalImpulse;
83   └  end
```

By moving the rows for Init processing (lines 74, 78 and 79 above) to the Custom user code area the impulse response process will still occur but can now be customized without being overwritten the next time **Refresh Init** is run.

```
67
68      %% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
69      [LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
70      [LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
71      Rx_Decision_Time = DFECDRPhase;|
72      % END: Custom user code area (retained when 'Refresh Init' button is pressed)
```

The value of `Rx_Decision_Time` can now be customized without having to worry about Refresh Init overwriting the value. Keep in mind that if additional SerDes blocks are added to the Rx data path, they will also need to be manually copied to the Custom user code area.

To revert to the original configuration, simply uncheck the box for **Comment out impulse processing,** then **Refresh Init** again.

Comment out impulse processing can also be used in conjunction with External Init on page 6-54 if desired.

Optional: External Init

By checking the box for **External Init**, after running **Refresh Init** the contents of each of the Init MATLAB function blocks are copied to `txInit.m` and `rxInit.m` files, and links to these functions are made back to the Simulink model. It also creates a `runExternalInit.m` MATLAB script to run these external Init files in MATLAB.

External initialization functions allow you to only run the Init portion of the algorithmic models, directly from the MATLAB command line, without having to wait for Simulink to compile the complete model. This allows you to quickly debug or iterate on a difficult implementation. Once debugging is complete, you can leave the Tx and Rx Init functions external, or move them back inside the Simulink model by unchecking the box for **External Init** and running **Refresh Init**.

You run the external Init functions by running the MATLAB script `runExternalInit` which was automatically generated along with `txInit.m` and `rxInit.m`. When running runExternalInit from the MATLAB command line, you may encounter warnings that are not seen when running from Simulink. These warnings are due to unused parameters from some of the SerDes blocks. These warnings can be ignored. When runExternalInit is complete, it will pop up the Init Statistical Analysis

Results window. The results should be identical to those from the Init Statistical and Time Domain Analysis Results plot above:
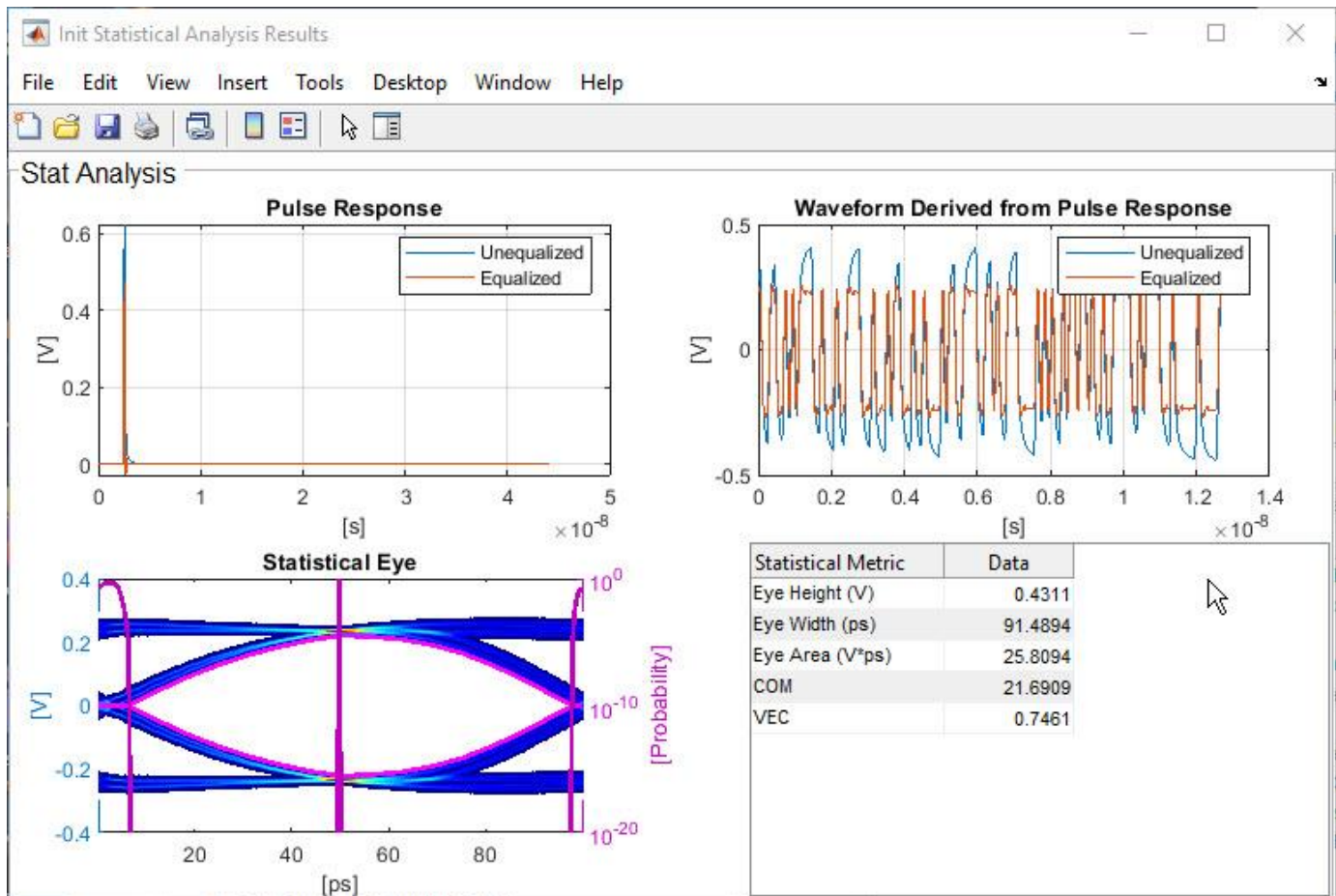
Example Customization of Rx_Decision_Time

As an example of how to customize the calculation of Rx_Decision_Time, we will use the option to comment out impulse response processing on page 6-53. After selecting **Comment out Impulse Processing**, running **Refresh Init**, then copying the 3 lines to the Custom user code section, you are ready to start. For this simple example we will still use the `Phase` output of the DFECDR block, but we will add a new AMI parameter that allows us to adjust the resulting clock location.

Start by using the SerDes IBIS-AMI Manager to add a new parameter to the DFECDR block as follows:

1) In the AMI-Rx tab, highlight the DFECDR and click on **Add Parameter...**

2) In the **Add/Edit AMI Parameter** dialog, enter the following:

- **Parameter name**: StatClockOffset
- **Current value**: 0
- **Description**: Offset in UI for the clock location used in Statistical processing.
- **Usage**: In

- **Type**: UI
- **Format**: Range
- **Typ**: 0
- **Min**: -1.0
- **Max**: 1.0

3) Click **OK** to save the new parameter.

4) By default, SerDes toolbox automatically adds new parameters to the corresponding data path block.

- Since the datapath blocks represent the Time Domain functionality of the AMI model, and Rx_Decision_Time is a Statistical only feature, you can delete the `StatClockOffset` constant block from the DFECDR block.
- Delete the `StatClockOffset` constant block now.

Advanced implementation: To get similar results in both Statistical and Time Domain you could use the `StatClockOffset` constant block to modify the value of Phase generated by the DFECDR.

- In this case, do not delete the `StatClockOffset` constant block.
- Use a `Sum` block to connect the `Phase` output and the `StatClockOffset` output.
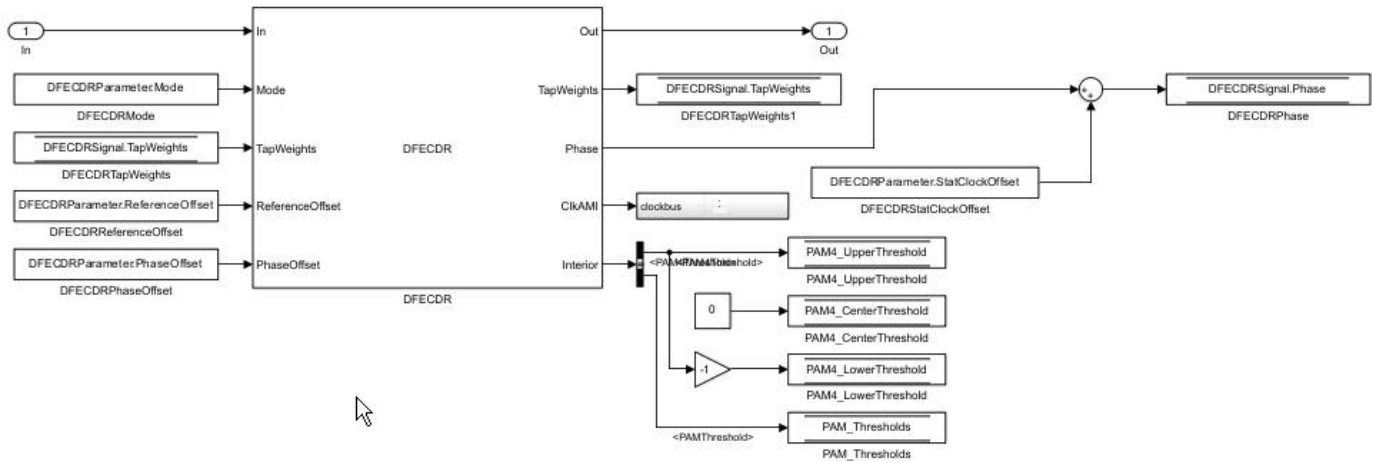
5) Run **Refresh Init** on the Rx block to add the new `StatClockOffset` parameter to the Initialize Function.

- You should see the following line added to top of the custom user code section:
- `DFECDRParameter.StatClockOffset; % User added AMI parameter from SerDes IBIS-AMI Manager`

6) Modify the Rx_Decision_Time calculation to include the value of StatClockOffset:

- `Rx_Decision_Time = DFECDRPhase + DFECDRParameter.StatClockOffset;`

7) When completed, the Custom user code section should look something like this. Note that the impulse response processing at the bottom of the function is still commented out:

```
67
68      %% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
69      DFECDRParameter.StatClockOffset; % User added AMI parameter from SerDes IBIS-AMI Manager
70      % Return impulse response and any Out or InOut AMI parameters for serdes.CTLE instance
71      [LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
72      % Direct Impulse response processing by serdes.SaturatingAmplifier is not recommended.
73      % LocalImpulse = SatAmpInit(LocalImpulse);
74      % Return impulse response and any Out or InOut AMI parameters for serdes.DFECDR instance
75      [LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
76      |
77      Rx_Decision_Time = DFECDRPhase + DFECDRParameter.StatClockOffset;
78
79      % END: Custom user code area (retained when 'Refresh Init' button is pressed)
80
81      %% Impulse response processing via system objects
82      % Return impulse response and any Out or InOut AMI parameters for serdes.CTLE instance
83      % [LocalImpulse, CTLEConfigSelect] = CTLEInit(LocalImpulse);
84      % Direct Impulse response processing by serdes.SaturatingAmplifier is not recommended.
85      % LocalImpulse = SatAmpInit(LocalImpulse);
86      % Return impulse response and any Out or InOut AMI parameters for serdes.DFECDR instance
87      % [LocalImpulse, DFECDRTapWeights, DFECDRPhase, ~, ~] = DFECDRInit(LocalImpulse);
88      % Rx_Decision_Time = DFECDRPhase;
```

8) Test the model by running with different values for `StatClockOffset`.

Init Statistical and Time Domain Analysis Results plot with StatClockOffset = 0.2UI:

Init Statistical and Time Domain Analysis Results — □ ✕

File   Edit   View   Insert   Tools   Desktop   Window   Help

Stat Analysis

Pulse Response

Unequalized
Equalized

[V]

[s] ×10⁻⁸

Waveform Derived from Pulse Response

Unequalized
Equalized

[V]

[s] ×10⁻⁸

Statistical Eye

[V]   [ps]   [Probability]

Statistical Metric	Data
Eye Height (V)	0.3678
Eye Width (ps)	91.4894
Eye Area (V*ps)	25.8094
COM	14.1047
VEC	1.9072

Time Domain Analysis

Time Domain Eye

[V]   [ps]   [Probability]

Time Domain Metric	Data
Eye Height (V)	0.4333
Eye Width (ps)	90.6286
Eye Area (V*ps)	25.4728
COM	22.3923
VEC	0.6859
Minimum BER	5.2632e-05
Ignore Symbols	1000
Total Symbols	20000

Init Statistical and Time Domain Analysis Results plot with StatClockOffset = -0.2UI:

Note that unless you decided to do the advanced implementation above on page 6-56, the Time Domain results should not change.

Generate IBIS-AMI Models

The final part of this example takes the customized Simulink model and generates IBIS-AMI compliant model executables, IBIS and AMI files for the Rx Decision Time enabled receiver.

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

Export Models

On the AMI-Rx tab, verify that the value for `StatClockOffset` is set to 0.

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

- Update the **Rx model name** to `rx_decision_time_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 the Rx AMI Model Settings. This will create a model executable that support both statistical (Init) and time domain (GetWave) analysis.
- Set the **Rx model Bits to ignore value** to `1000` to allow enough time for the external clock waveform to settle during time domain simulations.
- Verify that **Models to export** is set to **Both Tx and Rx.**
- Set the **IBIS file name** to be `rx_decision_time.ibs`
- Press the **Export** button to generate models in the Target directory.

Test Generated IBIS-AMI Models

The Rx Decision Time receiver IBIS-AMI model is now complete and ready to be tested in any industry standard AMI model simulator that supports the Rx_Decision_Time parameter per the IBIS 7.1 specification.

You can use Signal Integrity Link to automatically generate a Signal Integrity Toolbox Parallel Link Designer or Serial Link Designer project with the newly created AMI models already instantiated. This is an excellent way to test the new AMI model executables and verify that all features are working correctly.

References

IBIS-AMI Specification

See Also

More About
- "Clock and Data Recovery in SerDes System" on page 1-3
- "Statistical Analysis in SerDes Systems" on page 1-19
- "Signal Integrity Link" on page 3-11

7

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.

For a visual walkthrough of this example, watch the video.
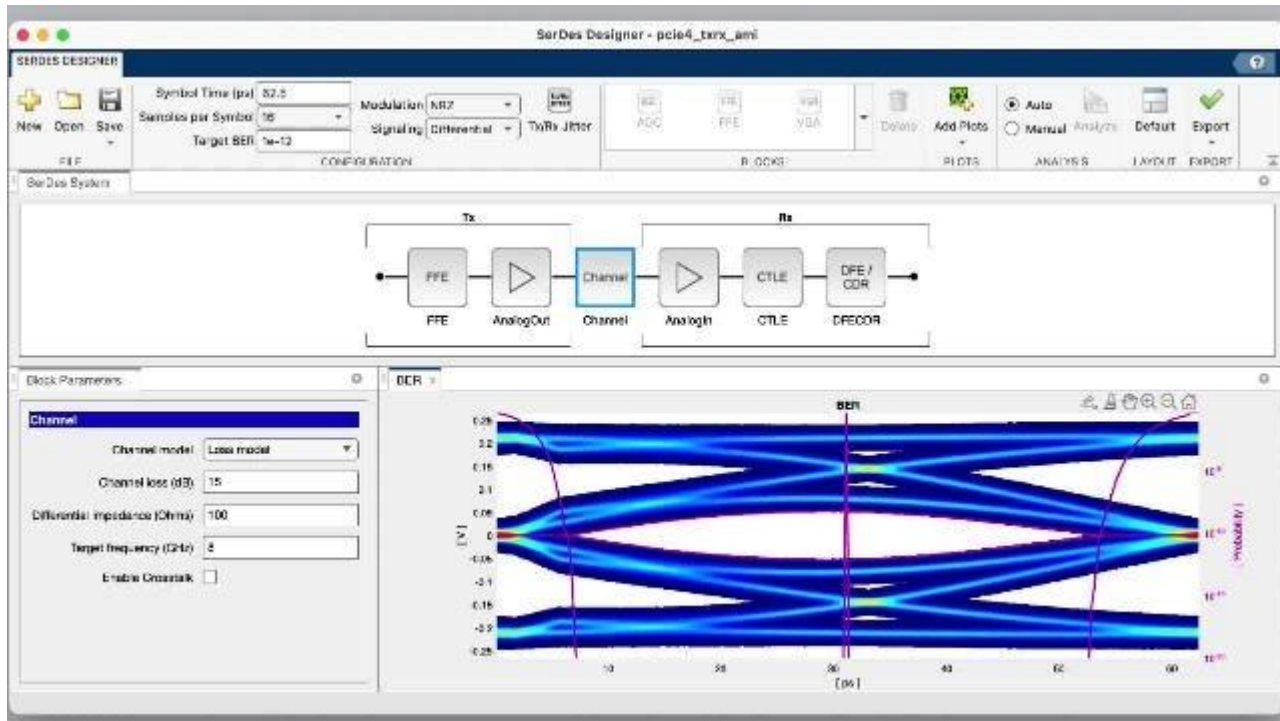
PCIe4 Tx/Rx IBIS-AMI Model Setup in SerDes Designer App

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')
```
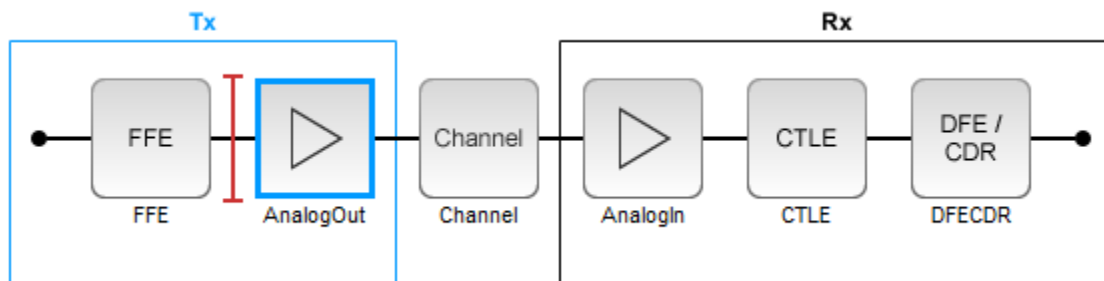
A PCIe4 compliant transmitter uses a 3-tap feed forward equalizer (FFE) with one pre-tap and one post-tap, and ten presets. The receiver model uses a continuous time linear equalizer (CTLE) with seven pre-defined settings, and a 2-tap decision feedback equalizer (DFE). To support this configuration the 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.
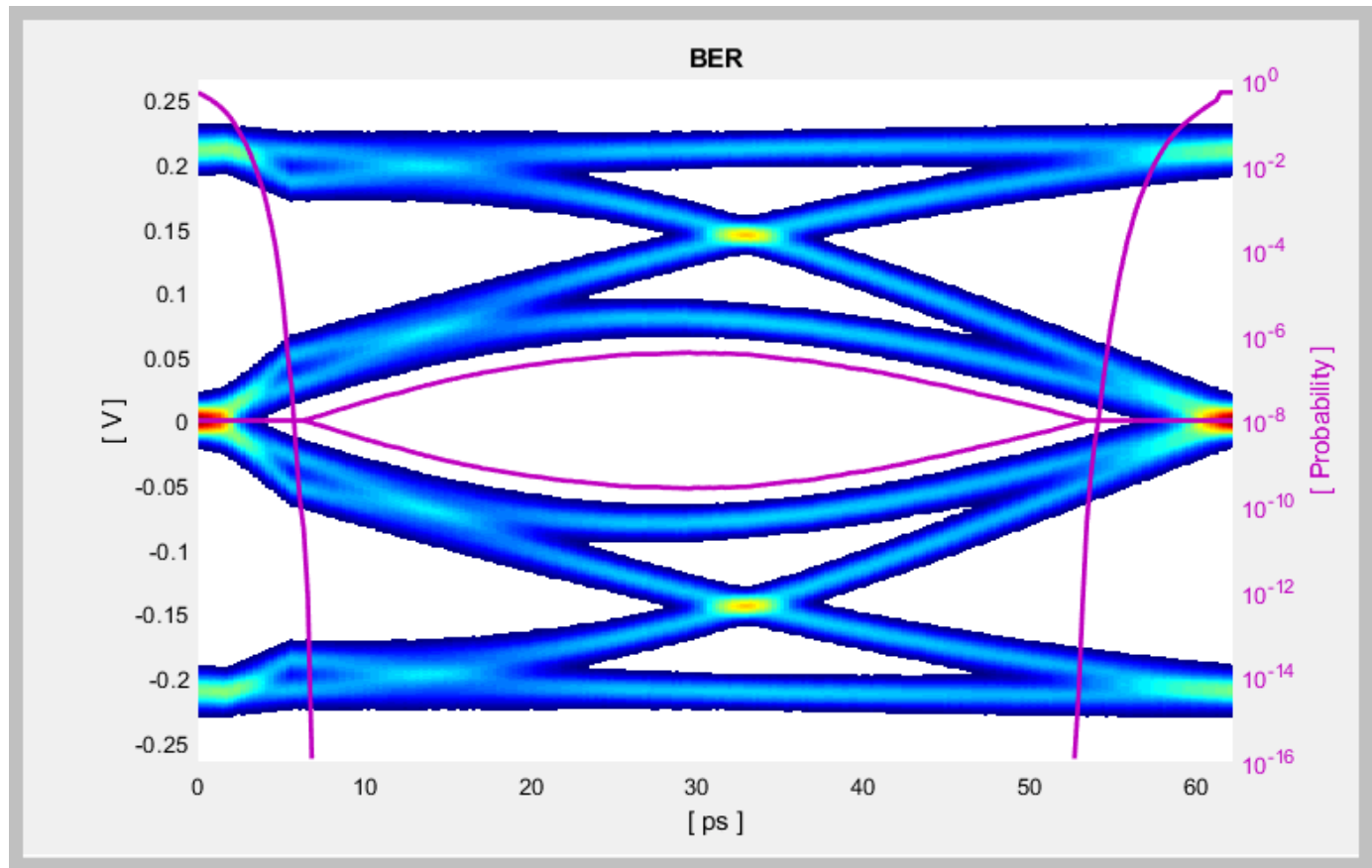
Change the Rx CTLE **Configuration select** parameter value from 0 to 6 and observe how this changes the data eye.
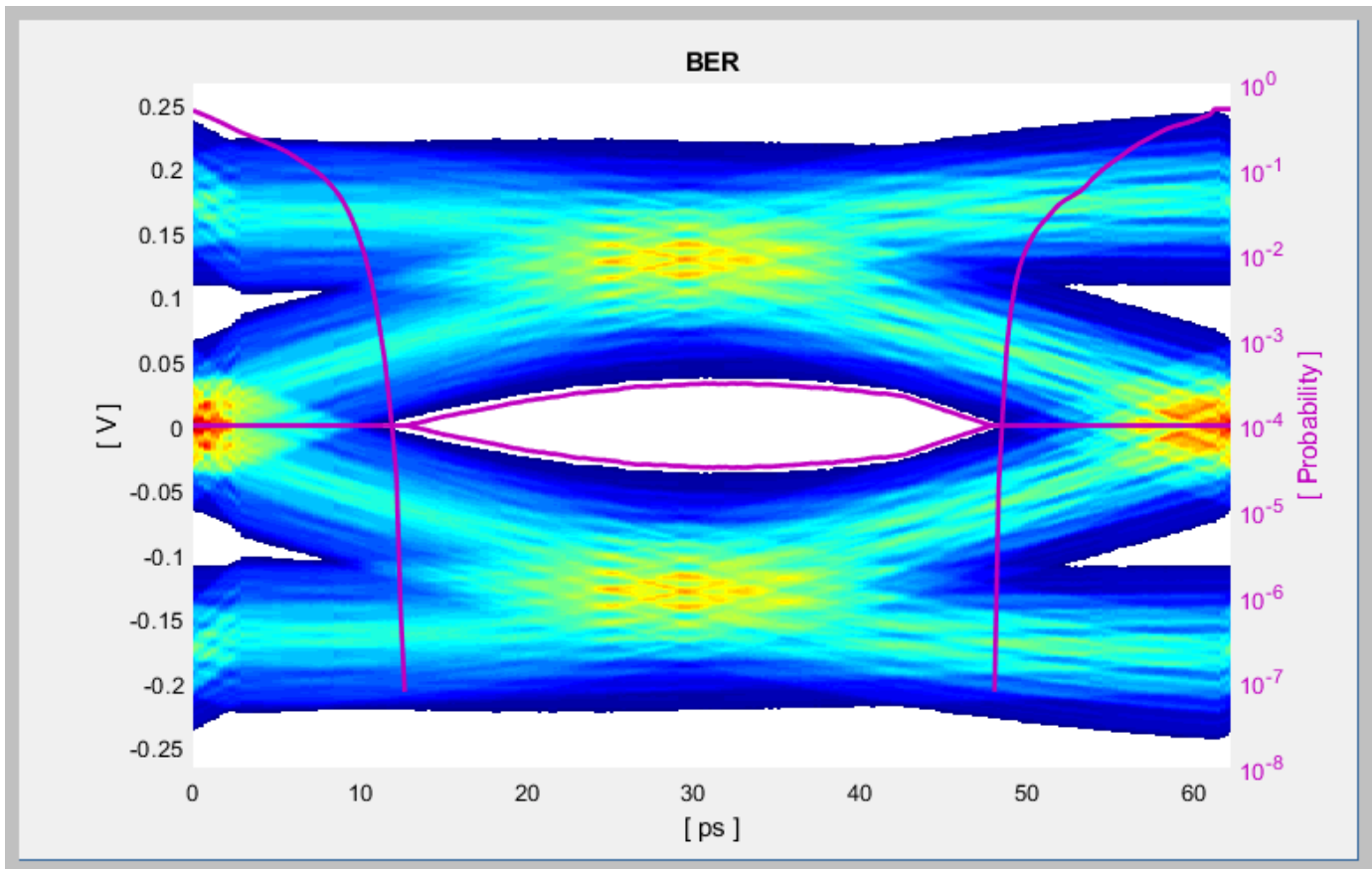
Change the value of the Tx FFE **Tap weights** from [0 1 0] to [-0.125 0.750 -0.125] and observe the results.
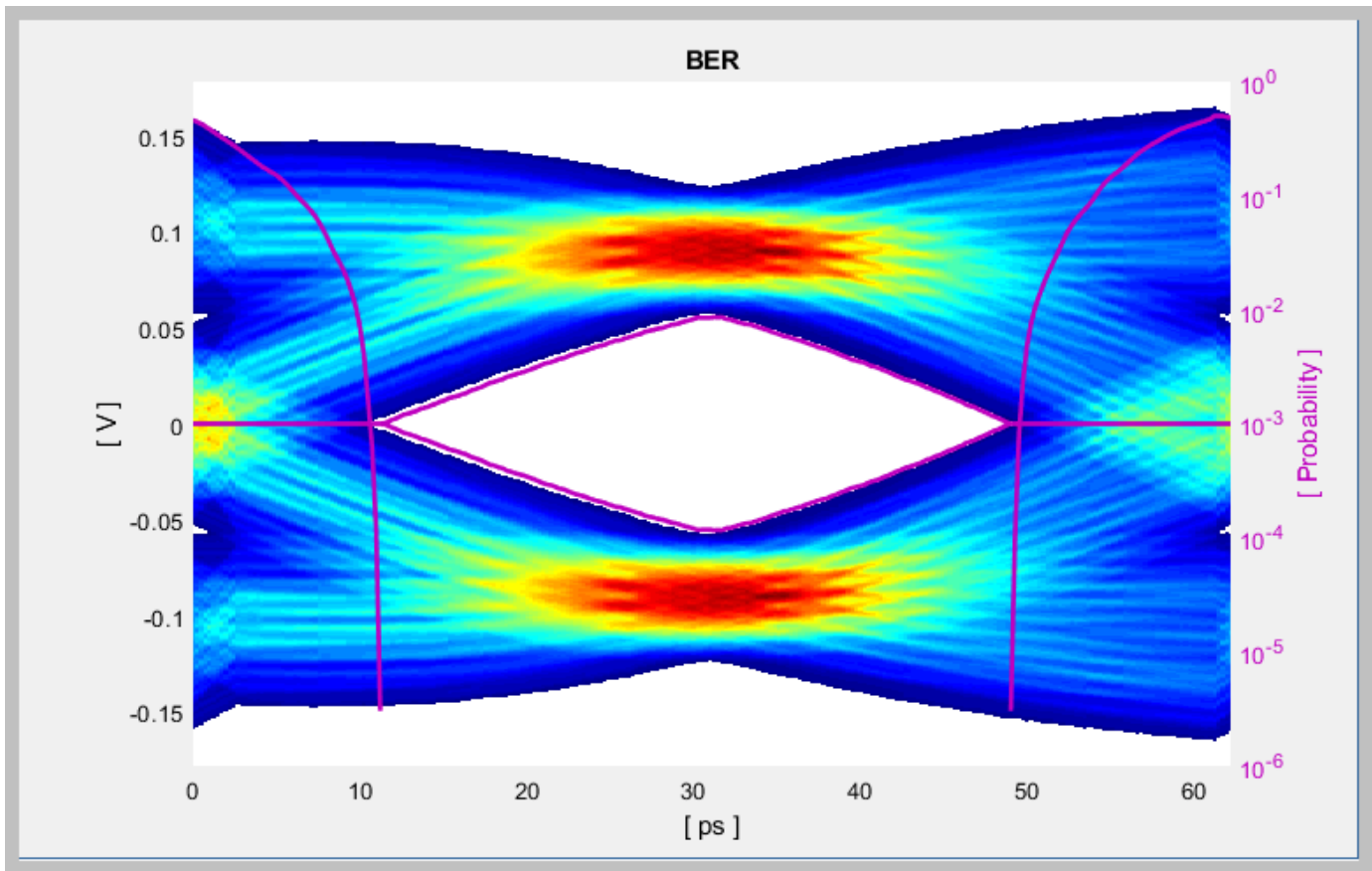
Change the Rx CTLE **Mode** to `Adapt` and observe the results. In this mode all CTLE values are swept to find the optimal setting.
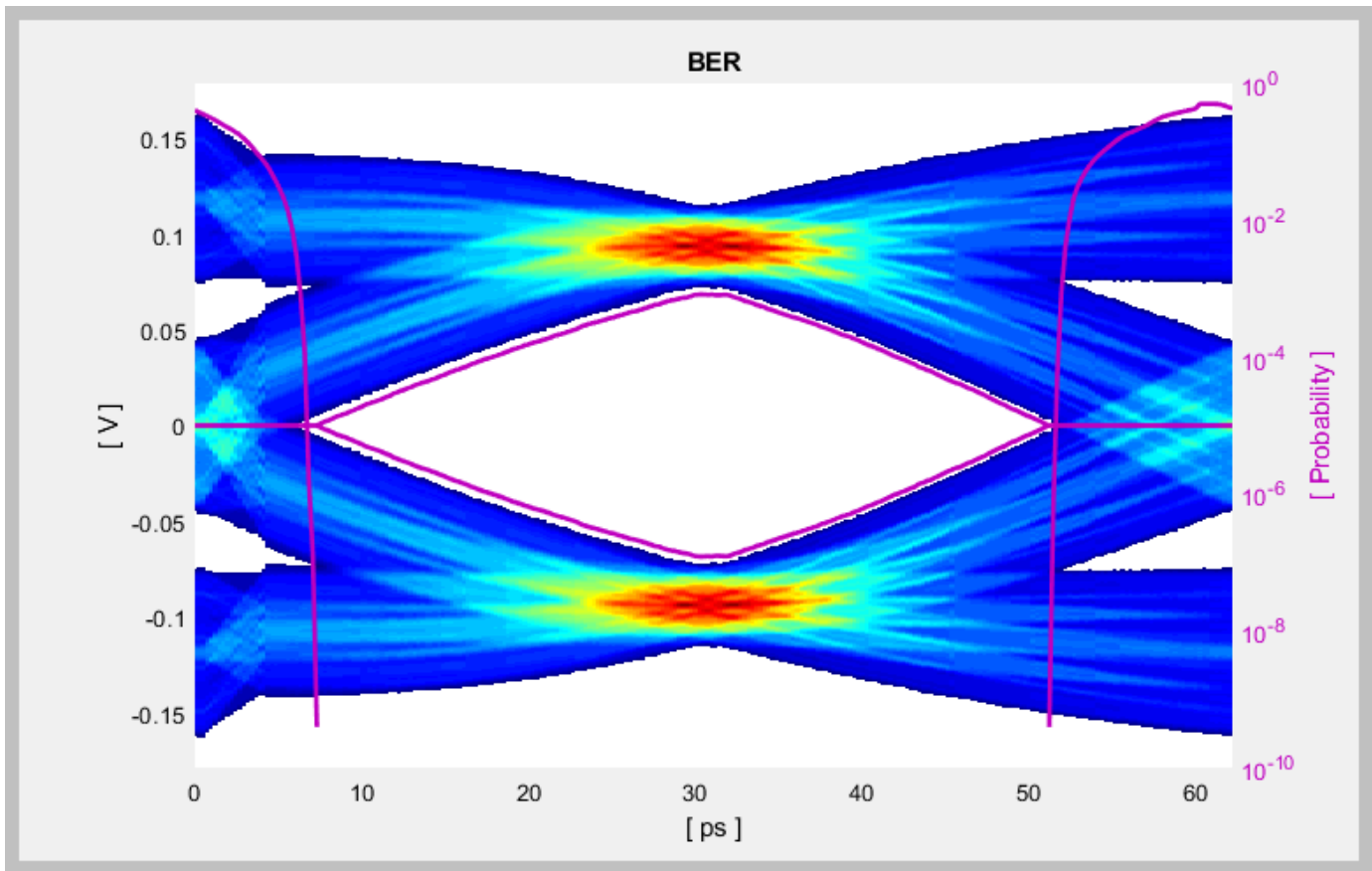
Before continuing, reset the value of the Tx FFE **TapWeights** back to `[0 1 0]` and Rx CTLE **ConfigSelect** back to `0`. Leave the Rx **CTLE** Mode at `Adapt`. Resetting these values here will avoid the need to set them again after the model has been exported to Simulink. These values will become 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.
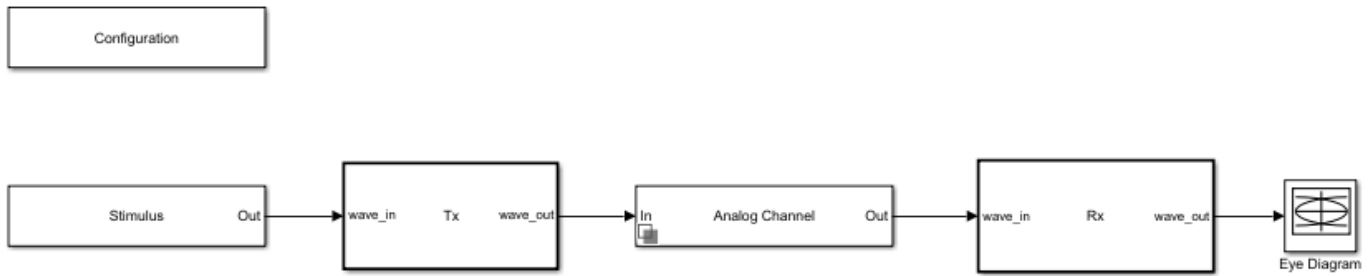
- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** is carried over from the SerDes Designer app.
- Double click the Stimulus block to open the Block Parameters 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 views of the statistical (Init) and time domain (GetWave) 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.

See Also

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

More About

- "PCIe-4 Compliance Kit" (Signal Integrity Toolbox)
- "PCIe5 Transmitter/Receiver IBIS-AMI Model" on page 7-16
- "Managing AMI Parameters" on page 6-2
- "Customizing SerDes Toolbox Datapath Control Signals" on page 5-2

PCIe5 Transmitter/Receiver IBIS-AMI Model

This example shows how to create IBIS-AMI models for a PCI-Express Generation 5 (PCIe Gen5) transmitter and receiver using the library blocks in SerDes Toolbox. The IBIS-AMI models generated by this example can be used in Serial Link Designer from Signal Integrity Toolbox and conform to the PCIe Gen5 Base-Specification published by the PCI-SIG.

PCIe Gen5 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 PCIe Gen5 in the SerDes Designer app. The model is then exported to Simulink® for further customization.

This example uses the SerDes Designer model `pcie5_ibis_txrx`. Type the following command in the MATLAB® command window to open the model:

```
>> serdesDesigner('pcie5_ibis_txrx');
```
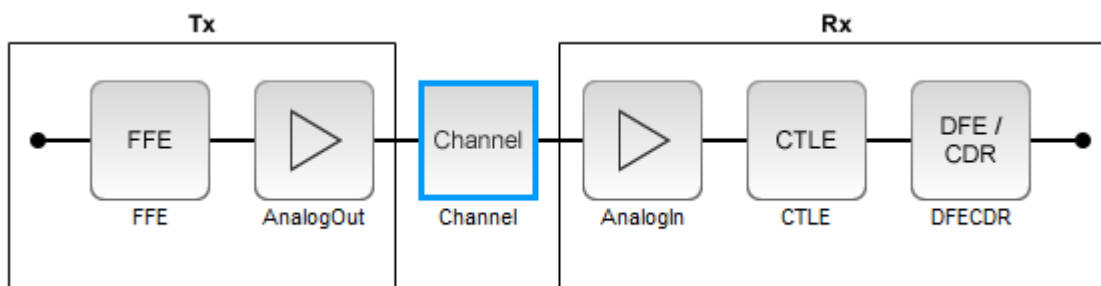

Configuration Setup

- **Symbol Time** is set to `31.25` ps, since the maximum allowable PCIe Gen5 data rate is 32 GT/s with a Nyquist frequency of 16GHz.
- **Target BER** is set to `1e-12` (BER as defined by the PCIe Gen5 BASE specification).
- **Samples per Symbol** is set to 16.
- **Modulation** is set to NRZ (non-return to zero).
- **Signaling** is set to `Differential`.

Transmitter Model Setup

- The Tx FFE block is set up for one pre-tap, one main tap, and one post-tap by including three tap weights [0 0.75 -0.25]. Specific tap presets can be configured later in the example when the model is exported to Simulink.
- The Tx AnalogOut model is set up so that **Voltage** is 1V, **Rise time** is 12 ps, **R** (output resistance) is 50 Ohms (Table 8-10 note 3) , and **C** (capacitance) is 0.5 pF according to the PCIe Gen5 specification.

Channel Model Setup

- **Channel loss** is set to 24 dB (37 dB is maximum loss for Base channel plus CEM card).

- **Differential impedance** is set to 85 Ohms (see PCIe Gen5 Base Spec, section 8.4.1.2, Figure 8-28 and 8-29).
- **Target Frequency** is set to the Nyquist frequency for 32GT/s data rate, which is 16 GHz.

Receiver Model Setup: Find the CTLE GPZ Matrix from the PCIe Gen5 BASE Specification

You can see from the loaded model, that the Rx CTLE block is setup for 11 configurations (0 through 10) to accommodate the Gain range of -5 to -15dB for this peaking filter. The code below shows you how a method to find a GPZ Matrix for the CTLE block using the transfer equation provided by the PCIe Gen5 Base Specification (Equation 8-7):

```
% Based on the PCIe Gen5 Base Specification, Equation 8-7:
DCgain = 5:15; %Adc defined as a range from -5 to -15dB.
ADC = 10.^(-DCgain/20);
z1 = 450e6; %Hz
wz1 = 2*pi*z1;
p1 = 1.65*z1; %Hz
wp1 = 2*pi*p1;
p2 = 9.5e9; %Hz
wp2 = 2*pi*p2;
p3 = 28e9; %Hz
wp3 = 2*pi*p3;
p4 = 28e9; %Hz
wp4 = 2*pi*p4;

f = linspace(0,100e9,1001);
w = 2*pi*f;
s = 1j*w;

H = zeros(length(ADC),length(f));

for ii = 1:length(ADC)
    z2(ii) = abs(ADC(ii))*p2;
    wz2(ii) = 2*pi*z2(ii);
    H(ii,:) = ((wp1*wp3*wp4)/(wz1)).*(((s+wz1).*(s+wp2*ADC(ii)))./((s+wp1).*(s+wp2).*(s+wp3).*(s-
end

figure(1)
ax1(1) = subplot(211);
semilogx(f,db(H))
grid on
xlabel('Hz')
ylabel('dB')
title('Reference CTLE from PCIe 5.0 BASE Spec, Equation 8-7')
ax1(2) = subplot(212);
semilogx(f,unwrap(angle(H)))
grid on
xlabel('Hz')
ylabel('Radians')
linkaxes(ax1,'x')
```

Reference CTLE from PCIe 5.0 BASE Spec, Equation 8-7
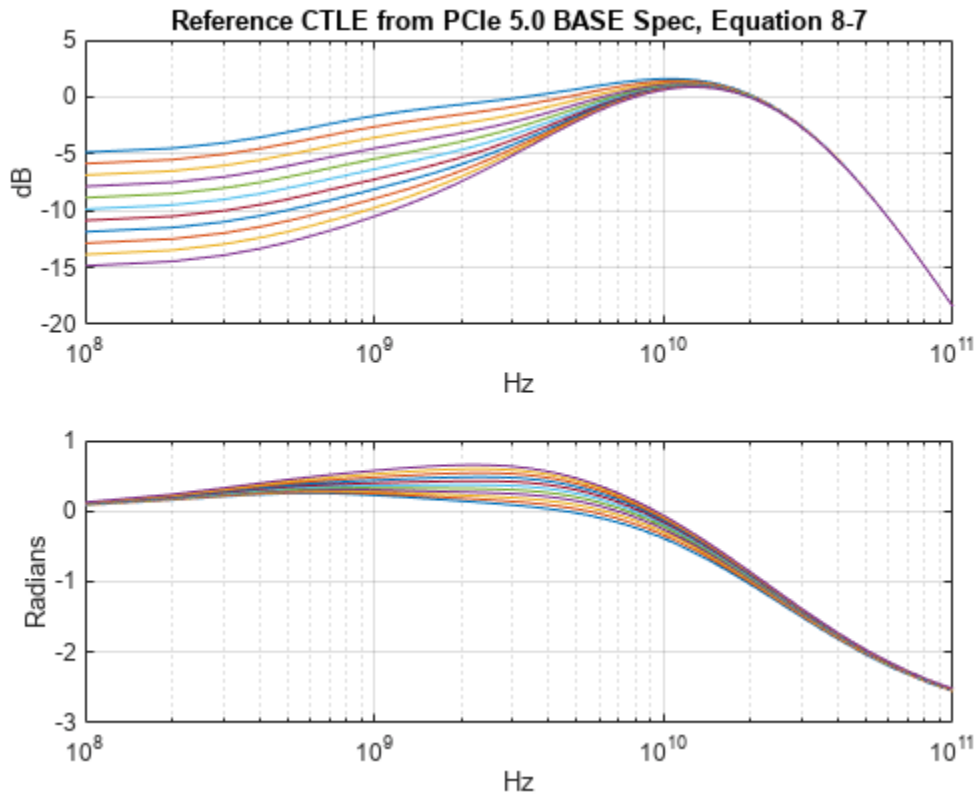
```
%Define gpz matrix:
gpz = zeros(length(ADC),8);
gpz(:,1) = -DCgain;
gpz(:,2) = -p1;
gpz(:,3) = -z1; %z1
gpz(:,4) = -p2;
gpz(:,5) = -z2;
gpz(:,6) = -p3;
%Note: column 7 is already set to all zeros.
gpz(:,8) = -p4;
```

Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (input resistance) is 50 Ohms (Table 8-10 note 3), and **C** (capacitance) is 0.5 pF according to the PCIe Gen5 specification.

- For the Rx CTLE block, set the **FilterMethod** to Cascaded to accommodate repeated poles as detailed in the PCIe Gen5 Base Specification (Equation 8-7). The Rx CTLE set up for 11 configurations (0 to 10) and the associated GPZ Matrix matches the Poles and Zeros given in the PCIe Gen5 Base Specification.

- The Rx DFE/CDR block is set up for three DFE taps. The limits for each tap have been individually defined according to the PCIe Gen5 specification to +/-80 mV for tap 1, +/-20 mV for tap 2, and +/-20 mV for tap 3.

Plot Statistical Eye Diagram and BER

You can use various plot types in SerDes Designer to visualize the output and performance of the PCIe Gen5 system. You can test the Rx CTLE functionality by setting its **Mode** to `fixed` and **Configuration** to `10.` Select the **BER** plot from the "ADD Plots" menu in the toolstrip and observe the Statistical Eye Diagram is displayed along with BER.

Note: The Statistical Eye Diagram and BER shown here represents a configuration with all **Jitter Parameters** disabled.
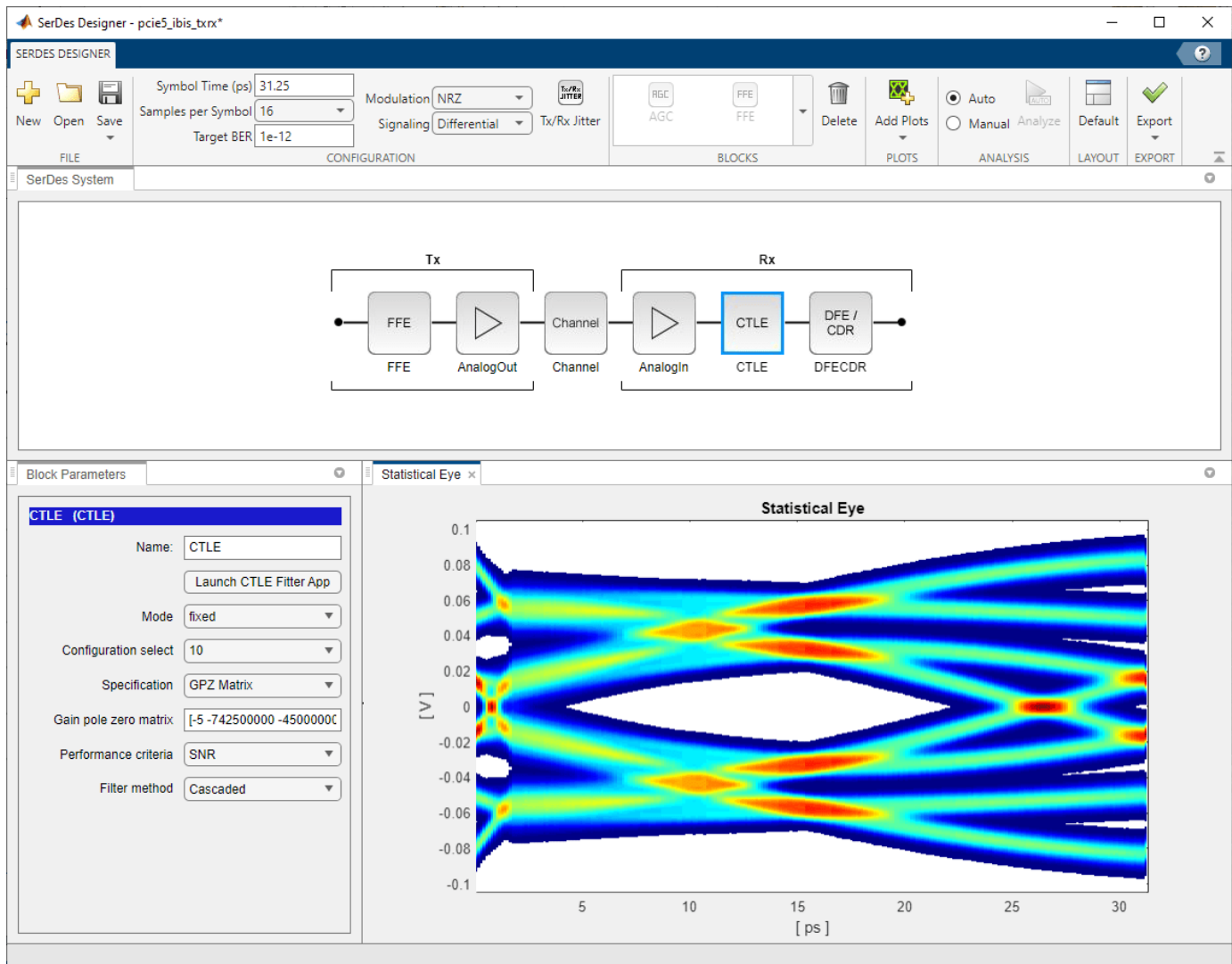
Change the Rx CTLE **ConfigSelect** to 3 and observe the plot changes:
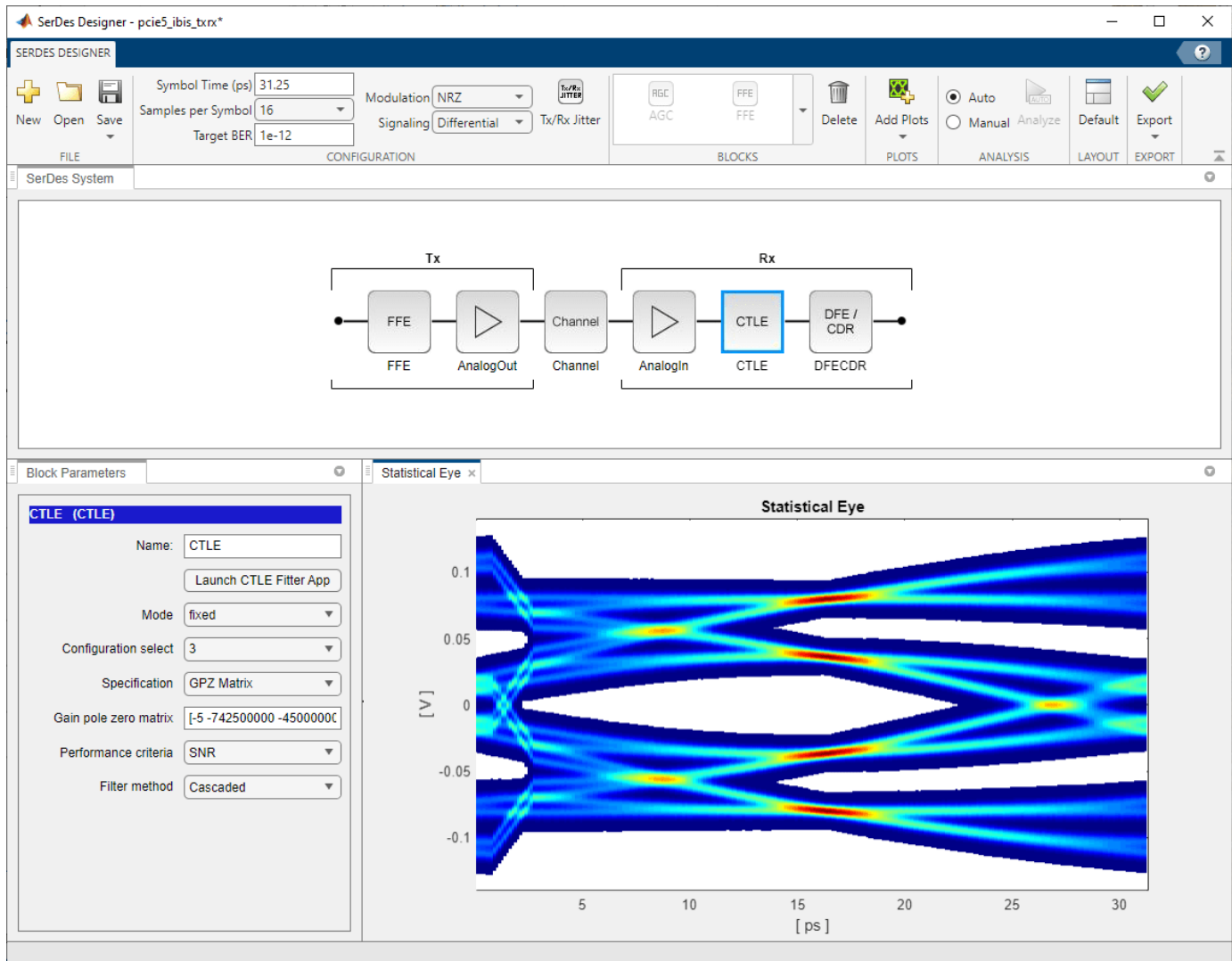
Before continuing, change the Rx CTLE **Mode** back to `Adapt`. Resetting this value for the RX CTLE will avoid the need to set it again after the model has been exported to Simulink.

Jitter Setup for Transmitter and Receiver

You can click on the Tx/Rx Jitter button on the toolstrip to view the **Jitter Parameters** tab. At this point you can also add the **Report** tab from the "Add Plots" menu in the toolstrip. The **Report** tab will enable you to observe how much the Statistical Eye Diagram and BER plot change as you enable or disable the various jitter types in the **Jitter Parameters** tab.

You can enable which jitter parameters are exported to Simulink by enabling the check boxes for **Tx_DCD**, **Tx_Rj**, **Tx_Dj, x_DCD**, **Rx_Rj**, and **Rx_Dj**. If they are disabled, they will not be exported, however you can also add jitter parameters in Simulink using the IBIS-AMI manager.

Tx Jitter Parameters

You can verify the Jitter parameters are set correctly by referencing the PCIe Gen5 Base specification, table 8-6, "*Data Rate Dependent Transmitter Parameters.*"

Note: These parameters will export as type "Float" with format "Value." After exporting to Simulink, you can change these to format "Range" using the IBIS-AMI Manager.

Tx DCD Jitter Value (Ttx-upw-tj from Table 8-6 in the PCIe Gen5 Base Spec)

- Confirm value is `6.25e-12` (Note: this is the maximum allowed per specification).
- Confirm units is set to seconds.
- Enable **Tx_DCD.**

Tx Rj Jitter Value (Ttx-rj from table 8-6)

- Confirm value is `0.45e-12` (Note: this is the maximum allowed per specification).
- Confirm units is set to seconds.
- Enable **Tx_Rj.**

Tx Dj Jitter Value (Ttx-upw-djdd from Table 8-6)

- Confirm value is `2.5e-12` (Note: this is the maximum allowed per specification).
- Confirm units is set to seconds.
- Enable **Tx_Dj**.

Rx Jitter Parameters

Rx DCD Jitter Values

- Confirm value is `0.`
- Confirm units is set to seconds.
- Enable **Rx_DCD.**

Rx Rj Jitter Values (Trx-st-rj from Table 8-9)

- Confirm value is `0.5e-12` to seconds (Note: this is the maximum allowed per specification).
- Confirm units is set to seconds.
- Enable **Rx_DCD.**

Rx Dj Jitter Values

- Confirm value is `0.`
- Confirm units is set to seconds.
- Enable **Rx_Dj.**

You may observe that the Statistical Eye and BER plot is nearly closed, but this is expected. The jitter parameters from the specification are maximum allowed values and there is no expectation they

would each be maximum at the same time in a real world system. The reason you are enabling these jitter parameters is so they are automatically included in this SerDes System when exporting to Simulink. Later you will see how to change these to ranges and set each one to 0 as a default value.

Note: After exporting to Simulink, you can also edit their Type, Usage, Format, and Value using the IBIS-AMI manager.

Export SerDes System to Simulink

Click on the **Export** button in the toolstrip to export the above configuration to Simulink for further customization and generation of the IBIS-AMI model files.

PCIe Gen5 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 PCIe Gen5 in Simulink. You can also open the support file attached to this example `pcie5_ibis_txrx.slx` and explore the system in Simulink. The following steps show how to configure the details of the system to prepare for exporting IBIS-AMI models.

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.
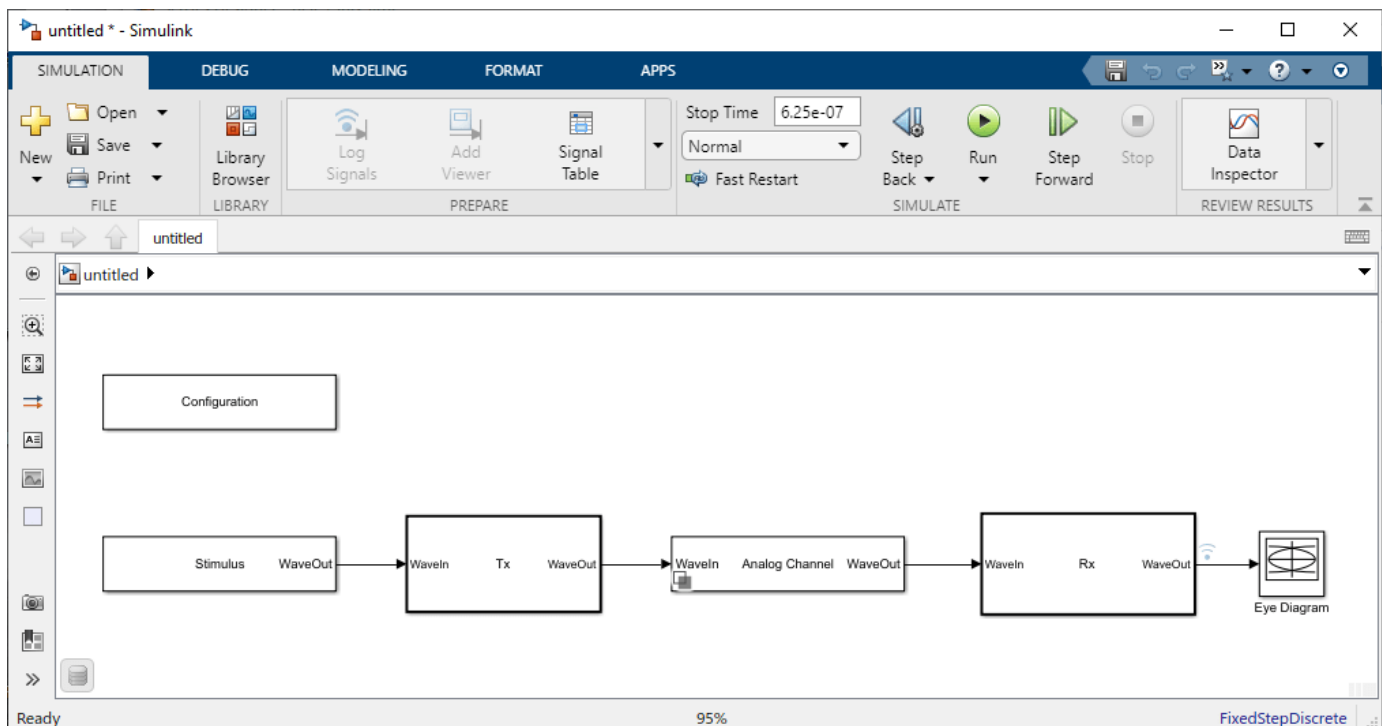
- You can confirm settings are carried over from the SerDes Designer app by double clicking the Configuration block and the Analog Channel block. Then open the Block Parameters dialog box and check their values.

- Double click the Stimulus block to open the Block Parameters dialog box. You can customize the Stimulus pattern here by configuring pattern and number of number of symbols to simulate. This block is not carried over from the SerDes Designer app.
- You can double click the Tx block and the Rx block to look inside each of their subsystems which are inherited from the SerDes Designer app.

Set Ignore Bits

Before running the simulation, open the **IBIS-AMI Manager**. You can set the "bits to ignore" for the Tx to 3, because the FFE has 3 taps. Also, you can see that "bits to ignore" for the Rx is 1000 (default), which should be a sufficient value for the **DFECDR** to converge during time domain simulation.

Update Tx Jitter Parameters

You can change the Format to "Range" for the Jitter Parameters by clicking on the AMI - Tx tab, select **Tx_DCD** and press the Edit button.
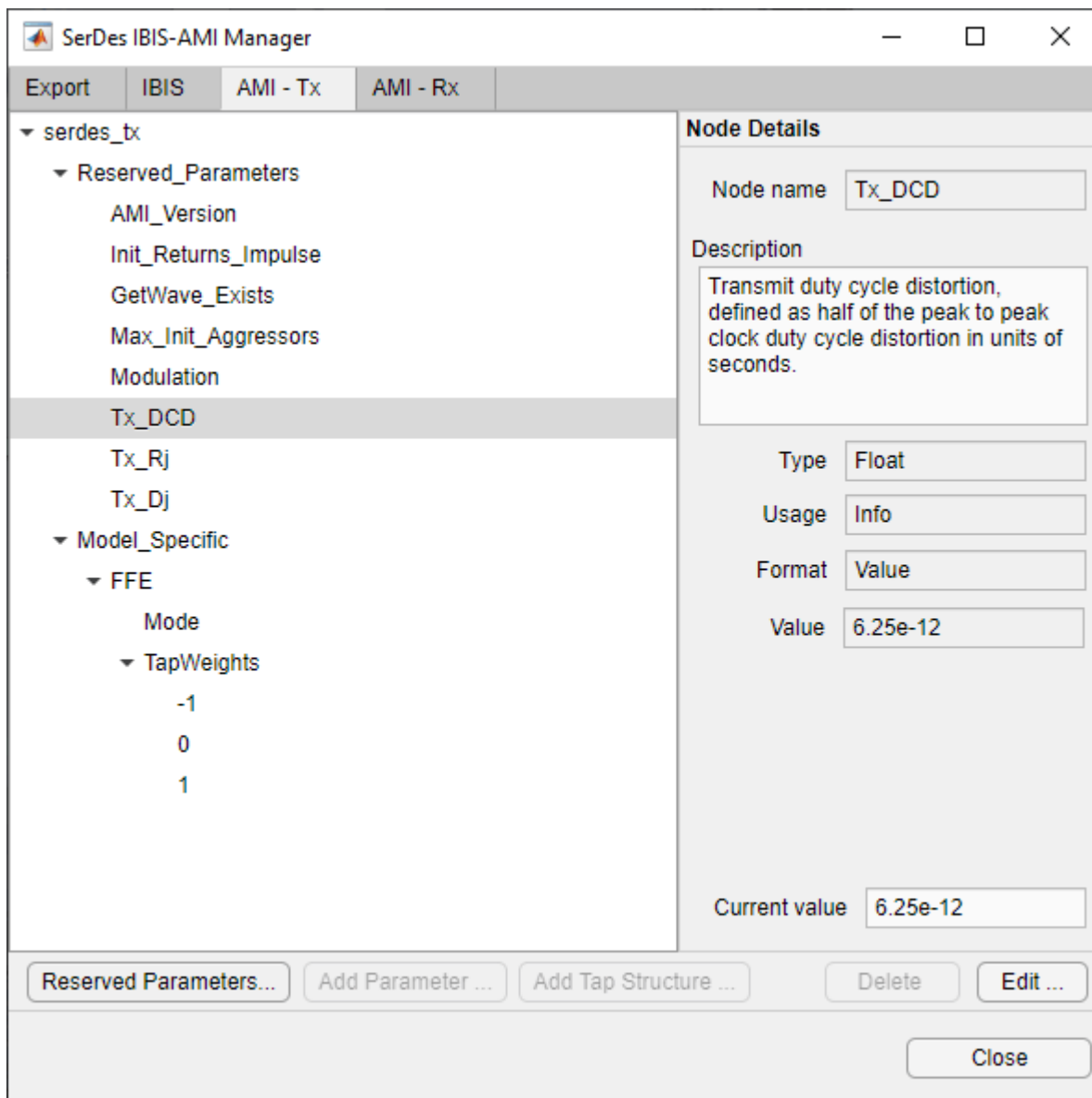
The following ranges allow you to fine-tune the jitter values to meet PCIe Gen5 jitter mask requirements. For example, see table 8-6, "*Data Rate Dependent Transmitter Parameters*" in the PCIe Gen5 Base specification.

Set Tx DCD Jitter Values (Ttx-upw-tj from Table 8-6)

- Select **Tx_DCD**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Verify the **Type** to `Float`.
- Change the **Format** to `Range`.

- Set the **Current Value** to `0`.
- Set the **Typ** value to `0`.
- Set the **Min** value to `0`.
- Set the **Max** value to `6.25e-12`.
- Click **OK** to save the changes.

Set Tx Rj Jitter Values (Ttx-rj from table 8-6)

- Select **Tx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Follow the steps for **Tx_DCD**, above.
- Set the **Max** value to `0.45e-12`.
- Click **OK** to save the changes.

Set Tx Dj Jitter Values (Ttx-upw-djdd from Table 8-6)

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

- Follow the steps for **Tx_DCD**, above.
- Set the **Max** value to `2.5e-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.

Update Rx Jitter Parameters

Select the **Rx_DCD**, **Rx_Dj** and **Rx_Rj** and follow the steps above from **Tx_DCD**. The following ranges allow you to fine-tune the jitter values for your own system.

Set Rx DCD Jitter Values

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

Set Rx Rj Jitter Values (Trx-st-rj from Table 8-9)

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Follow the steps for **Rx_DCD**.
- Set the **Max** value to `0.5e-12.`
- Click **OK** to save the changes.

Set Rx Dj Jitter Values

- Select **Rx_Dj**, then click the **Edit...** button to bring up the **Add/Edit AMI Parameter** dialog.
- Follow the steps for **Rx_DCD**.
- Click **OK** to save the changes.

Note: you can close the **IBIS AMI Manager** for the next section, you can revisit this dialog to export IBIS-AMI models later.

Review Tx FFE Block

- Inside the Tx subsystem, double click the FFE block to open the FFE Block Parameters dialog box.
- Verify that the current value of **Mode** is set to `Fixed`.

Review Rx CTLE Block

- Inside the Rx subsystem, double click the CTLE block to open the Block Parameters dialog box.
- **Gain pole zero** data is carried over from the SerDes Designer app. This gain pole zero data applies the transfer function of the behavioral CTLE given by the PCIe Gen5 Base Specification.

- CTLE **Mode** is set to `Adapt`, which means an optimization algorithm built into the CTLE system object selects the optimal configuration at run time.

Review 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.
- The DFE tap value(s) are carried over from the SerDes Designer app.

Generate PCIe Gen5 Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model, modifies the AMI parameters for PCIe Gen5, then generates IBIS-AMI compliant model 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.

You how to set the TX FFE tap values by creating new AMI parameters and implementing an algorithm in the Init customer specific code section to select PCIe Gen5 Preset values P0 through P10.

- Click on the **FFE** parameter listed under the **Model_Specific** parameter, then click on the button "Add Parameter." Then configure the options in the dialog as follows:
- Parent Node: FFE (default)
- Parameter name: ConfigSelect
- Current value: User Defined
- Description: PCIe Gen5 Tx tap weights configuration
- Usage: In
- Type: Integer
- Format: List
- Default: -1 (-1 is used to select "user defined" taps, it is not a preset from the specification)
- List values: [-1 0 1 2 3 4 5 6 7 8 9]
- List_Tip values: ["User Defined" "P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9"]

Note: 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 eleven defined presets will be allowed, and all three taps must be set to the same preset to get the correct values.

Modify Init to Select Presets for Preshoot Tap, Main Tap, and De-emphasis Tap

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. Then click the **Show Init** button to open the function in an editor window.

Note: You can also type **Ctrl-U** to look under the mask for the Init block, then double-click on the initialize block to open the Initialize Function.
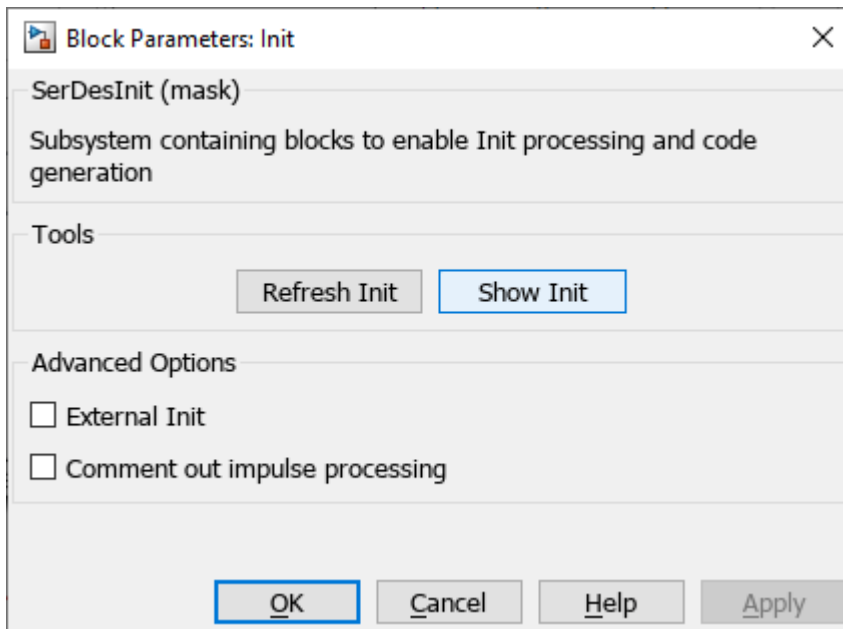
Double-click on the impulseEqualization MATLAB function block to open the function in MATLAB. This is an automatically generated function which provides the impulse response processing of the SerDes system block (IBIS AMI-Init). The `%% BEGIN:` and `% END:` lines denote the section where custom user code can be entered. Data in this section will not get over-written when Refresh Init is run:

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.

Modify GetWave to Select Presets for Preshoot Tap, Main Tap, and De-emphasis Tap

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.

You can see that a new constant block has been added called **FFEParameter.ConfigSelect.** This is created automatically by the IBIS-AMI Manager when a new Reserved Parameter is added. Next, you can follow these steps to re-configure the selection of tap weight presets for time domain (GetWave) simulation:

- Add a MATLAB Function block to the canvas from the Simulink/User-Defined library.
- Rename the MATLAB Function block to `PCIe5FFEconfig`.
- Double-click the MATLAB Function block and replace the template code with the following:

```
% PCIe5 tap configuration selector

% Selects pre-defined Tx FFE tap weights based on PCIe5 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 = PCIe5FFEconfig(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.

Run the SerDes System Model in Simulink

You can now run the model to simulate the SerDes System. Many plots are generated, including a live time-domain (GetWave) eye diagram that is updated as the model is running.
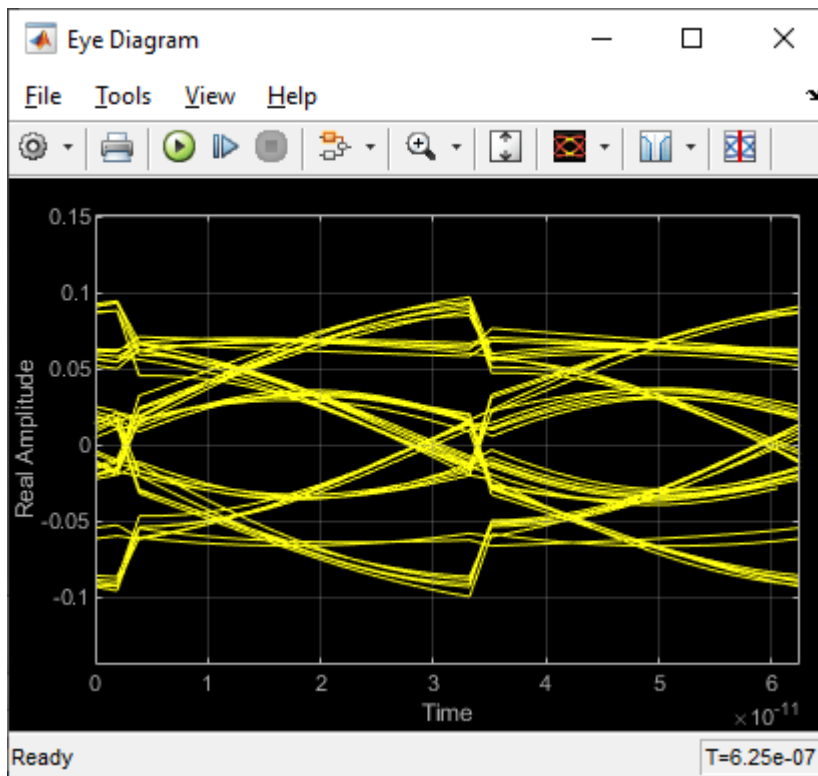
The second plot contains views of the statistical (Init) results and persistent time domain (GetWave) results, similar to what is available in the SerDes Designer App.
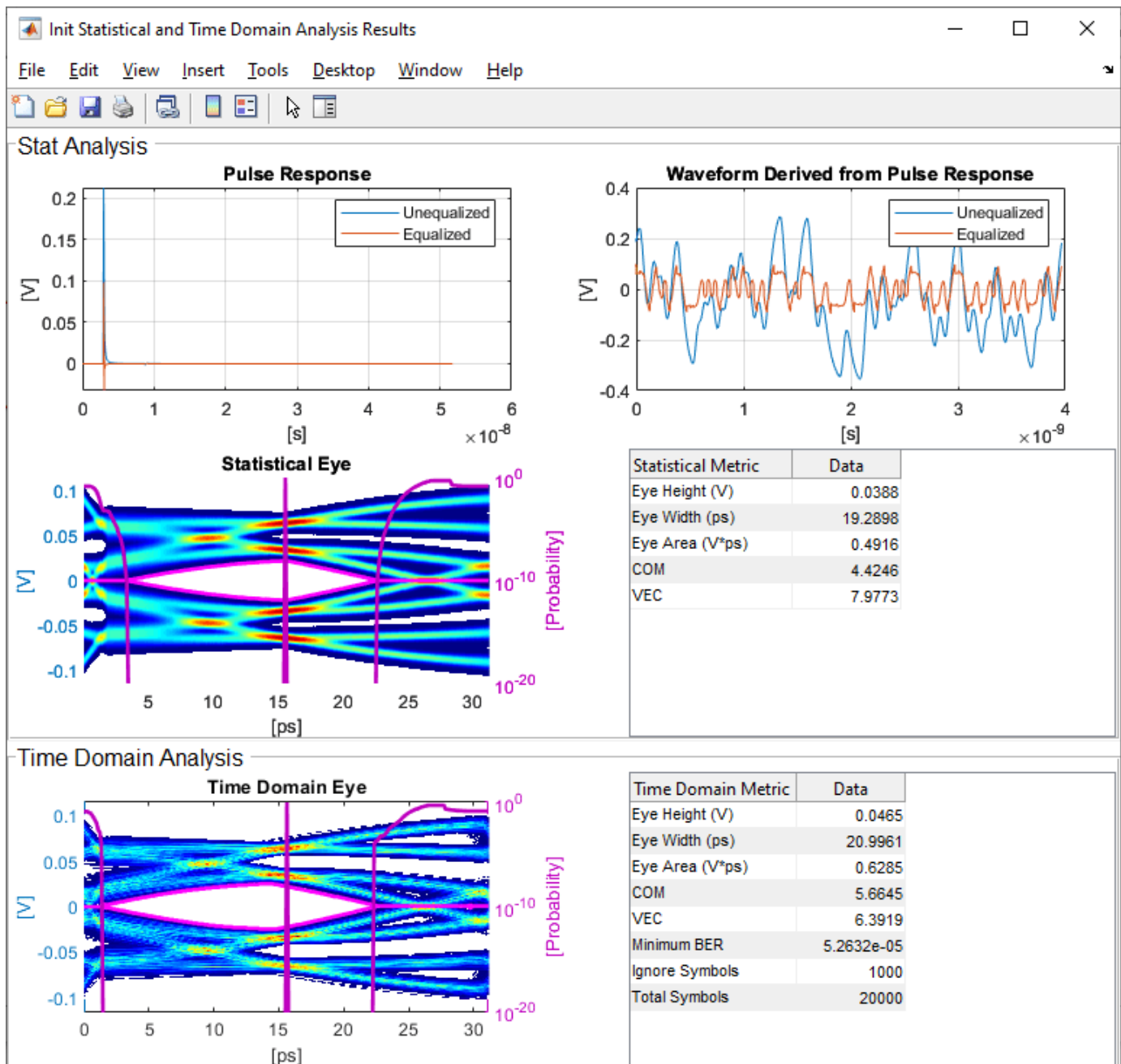
You may notice that the eye height can differ between Statistical and Time Domain. There are many possible reasons for larger eye diagram openings in Time Domain: one example could be use of encoding in the Stimulus to control ISI, another example could be active behavior of a DFE, etc. You can explore the system to visualize its performance while changing various characteristics such as setting CTLE to fixed, etc.

Export IBIS-AMI Models

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

- Verify the **Tx model name** is pcie_g5_tx.
- Verify the **Rx model name** is pcie_g5_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 1000 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 pcie5ami.
- Press the **Export** button to generate models in the Target directory.

Test Generated IBIS-AMI Models

The PCIe Gen5 transmitter and receiver IBIS-AMI models are now complete and ready to be utilized in any industry standard EDA tool capable of IBIS-AMI simulation.

References

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

See Also
FFE | CTLE | DFECDR | **SerDes Designer**

More About
- "PCIe-5 Compliance Kit" (Signal Integrity Toolbox)
- "PCIe4 Transmitter/Receiver IBIS-AMI Model" on page 7-2
- "Managing AMI Parameters" on page 6-2
- "Customizing SerDes Toolbox Datapath Control Signals" on page 5-2

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™ and have been Verified by Intel®. 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')
```
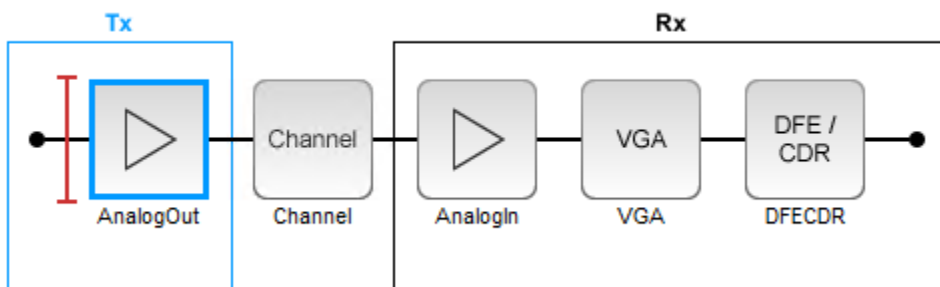

The SDRAM has a DDR5 transmitter (Tx) using no equalization. The SDRAM also has a DDR5 receiver (Rx) using a variable gain amplifier (VGA) with 7 pre-defined settings and a 4-tap decision feedback equalizer (DFE) with built-in clock data recovery.

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 0.65pF. 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.
- **Note:** the DFECDR offers an option for "2X Taps." Check this option to have pulse response values match convention used by JEDEC. Uncheck this option to use pulse response values directly from the plot.

-

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.
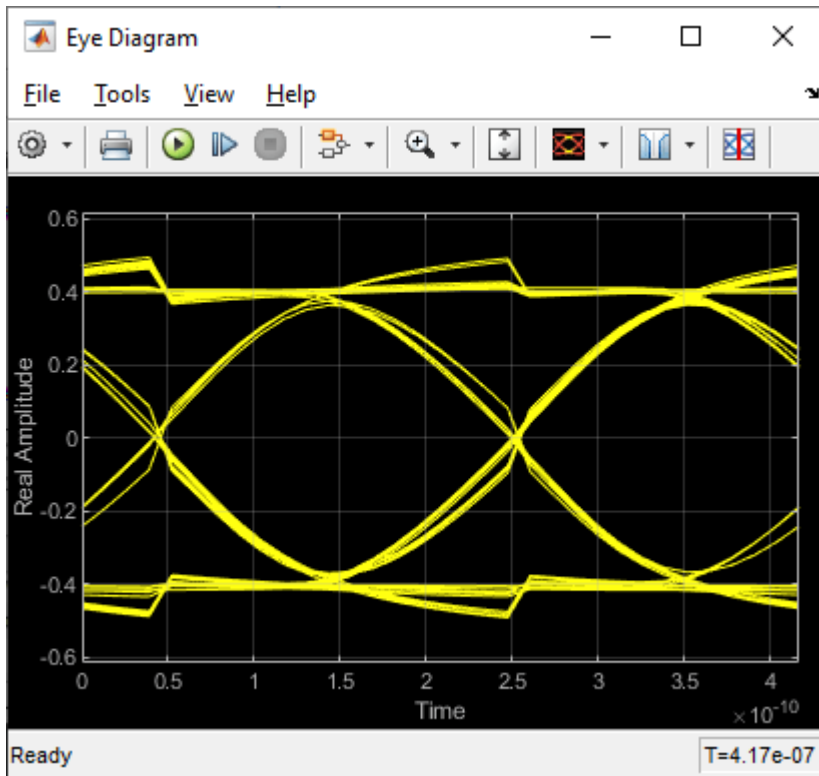
After the simulation has completed the second plot contains views of the Statistical (Init) and Time Domain (GetWave) results, along with reported Eye metrics for each.

Init Statistical and Time Domain Analysis Results

Stat Analysis

Pulse Response
Unequalized
Equalized

Waveform Derived from Pulse Response
Unequalized
Equalized

Statistical Eye

Statistical Metric	Data
Eye Height (V)	0.6777
Eye Width (ps)	189.5855
Eye Area (V*ps)	82.8512
COM	17.7636
VEC	1.2033

Time Domain Analysis

Time Domain Eye

Time Domain Metric	Data
Eye Height (V)	0.7029
Eye Width (ps)	196.0949
Eye Area (V*ps)	87.0596
COM	20.2283
VEC	0.8901
Minimum BER	5.2632e-04

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 available for DDR5-4800.

Set Tx Deterministic Jitter Value

- Select **Tx_Dj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.1000
- 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.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.0050
- 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 available for DDR5-4800.

Set Rx Random Jitter Value

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

- Change the **Format** to `Value`.
- Set the **Current Value** to `0.01750`
- Click **OK** to save the changes.

Set Rx Receiver Sensitivity Value

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

See Also
VGA | DFECDR | **SerDes Designer**

More About
- "DDR5 Controller Transmitter/Receiver IBIS-AMI Model" on page 7-50
- "Design DDR5 IBIS-AMI Models to Support Back-Channel Link Training" on page 7-79
- "DDR5 Implementation Kit" (Signal Integrity Toolbox)

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™ and have been Verified by Intel®. 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')
```
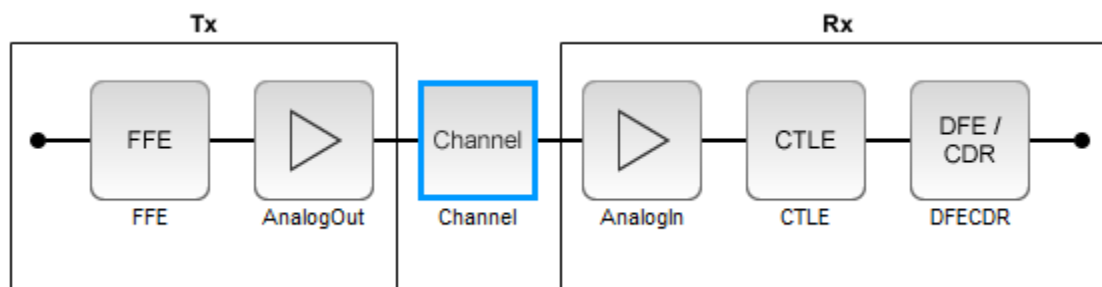

The controller has a DDR5 transmitter (Tx) using 4-tap feed forward equalization (FFE). The controller also has a DDR5 receiver (Rx) using a continuous time linear equalizer (CTLE) with 8 pre-defined settings and a 4-tap decision feedback equalizer (DFE) with built-in clock data recovery.

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 two post-taps by including four tap weights. This is done with the array [0 1 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

- **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
- **Channel loss** is set to 5 dB at Nyquist, which is typical of DDR channels.

Receiver Model Setup

- The Rx AnalogIn model is set up so that **R** (input resistance) is 40 Ohms and **C** (capacitance) is 0.65pF. 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 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 -0.1 -0.1 -0.1] V and the **Maximum tap value** is set to [0.2 0.1 0.1 0.1] V.
- **Note:** the DFECDR offers an option for "2X Taps." Check this option to have pulse response values match convention used by JEDEC. Uncheck this option to use pulse response values directly from the plot.
-

DFECDR (DFECDR)	
Name:	DFECDR
Mode	adapt
Initial tap weights (V)	[0 0 0 0]
Minimum tap value (V)	[-0.2 -0.1 -0.1 -0.1]
Maximum tap value (V)	[0.2 0.1 0.1 0.1]
2x tap weights	☑

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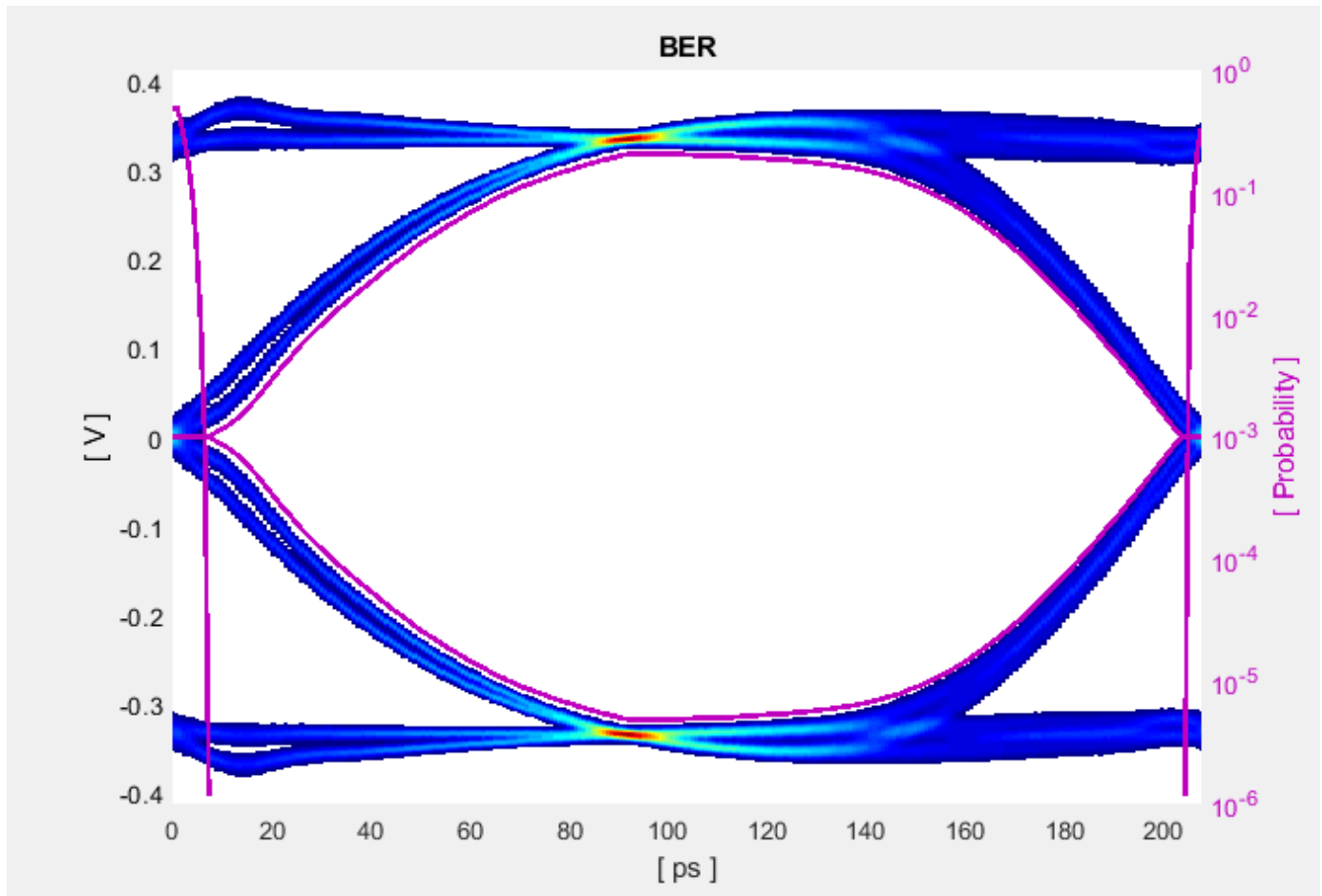
Add the Pulse Response plot from **Add Plots** and zoom into the pulse area to observe the results.
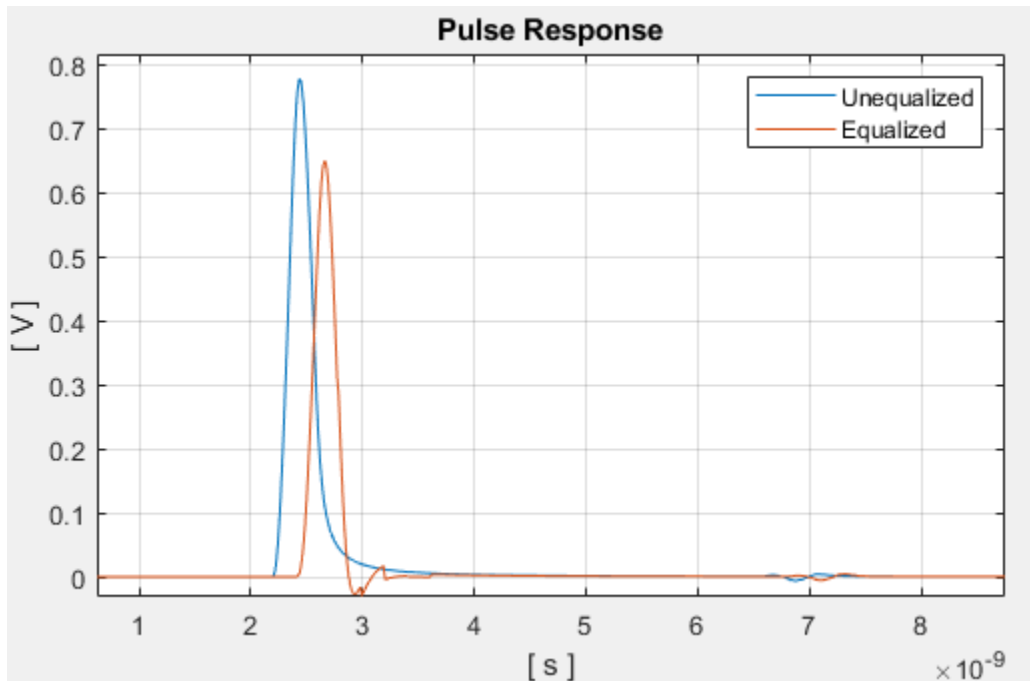
Export SerDes System to Simulink

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 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 views of the Statistical (Init) and Time Domain (GetWave) results, along with Eye metrics reported for each.
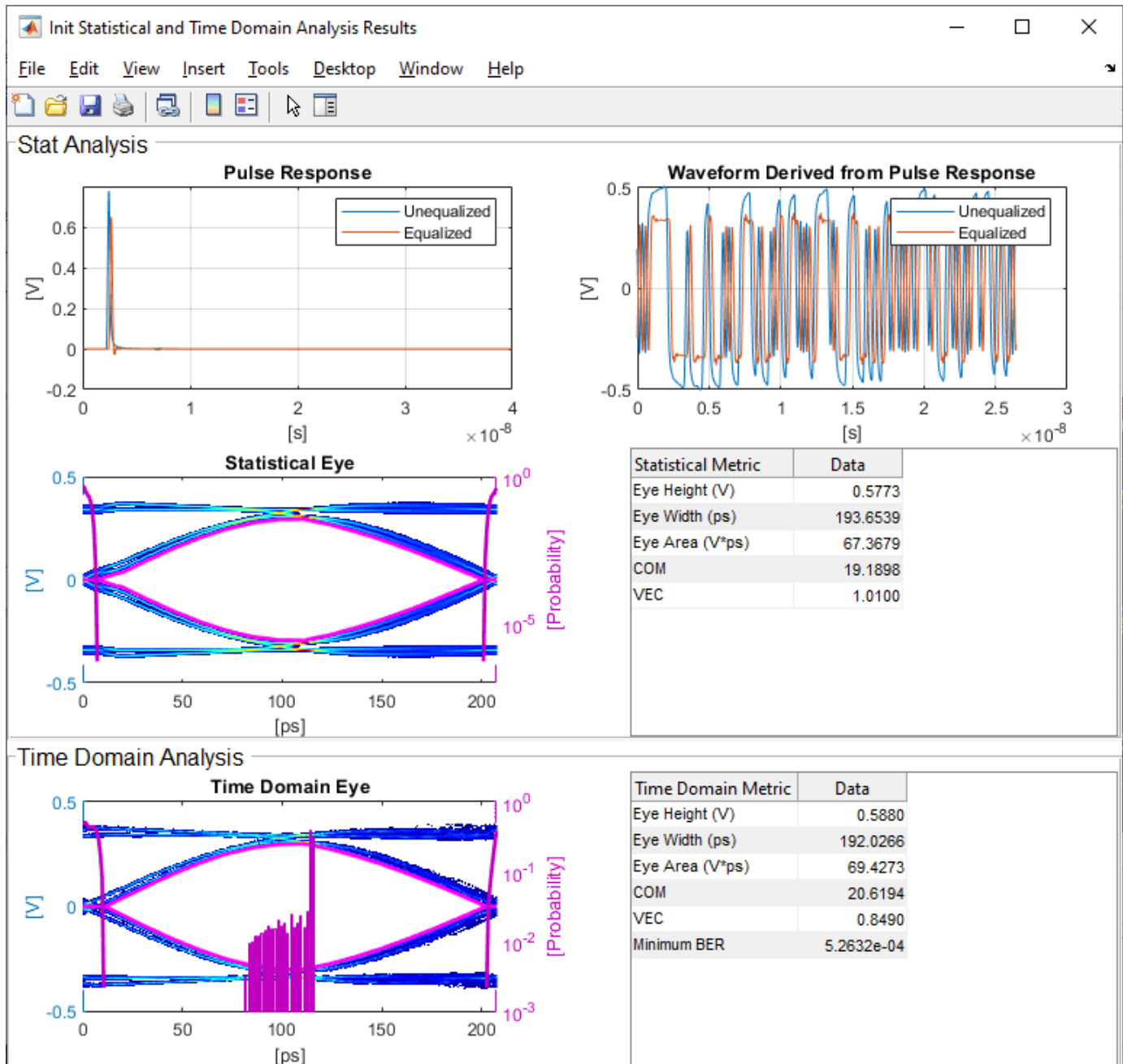
Review Tx FFE Block

- 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.

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**.

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.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.0500
- 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.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.0025
- 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 Random Jitter Value

- Select **Rx_Rj**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.00375
- 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.
- Change the **Type** to UI.
- Change the **Format** to Value.
- Set the **Current Value** to 0.0125
- Click **OK** to save the changes.

Set Rx Receiver Sensitivity Value

- Select **Rx_Receiver_Sensitivity**, then click the **Edit...** button to bring up the Add/Edit AMI Parameter dialog.
- Change the **Format** to Value.
- Set the **Current Value** to 0.040
- 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.
- Change the **Format** to `Value`.
- Set the **Current Value** to `0.0015`
- 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.
- Change the **Format** to `Value`.
- Set the **Current Value** to `0.0025`
- 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 4 since there are four taps in the Tx FFE.
- Set the Rx model **Bits to ignore** to `250000(is 10000 enough?-GK)` 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` directory so that the example file `ddr5_controller.ibs` is not overwritten.
- 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 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.

Note: It is always recommended to verify the values for vinl, vinh, c_comp and other variables in the .ibs file match your device datasheet values.

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.

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

More About
- "DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model" on page 7-38
- "Design DDR5 IBIS-AMI Models to Support Back-Channel Link Training" on page 7-79
- "DDR5 Implementation Kit" (Signal Integrity Toolbox)

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')
```
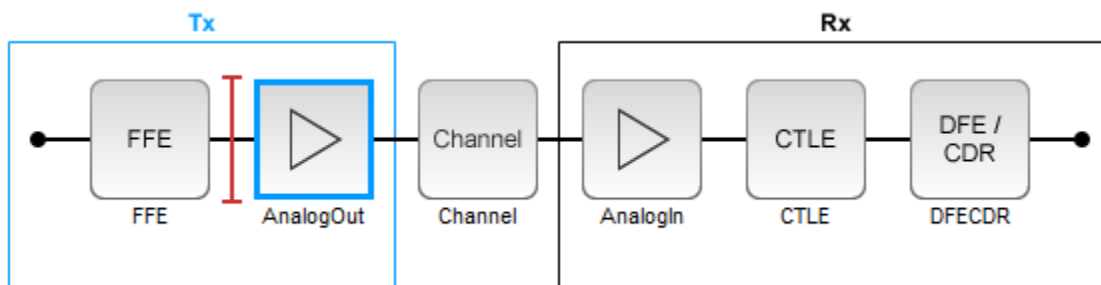

A CEI-56G-LR compliant transmitter uses a 4-tap feed forward equalizer (FFE) with two pre-taps and one post-tap. The receiver model uses a continuous time linear equalizer (CTLE) with 17 pre-defined settings, and a 12 to 18 tap decision feedback equalizer (DFE). To support this configuration the SerDes 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.
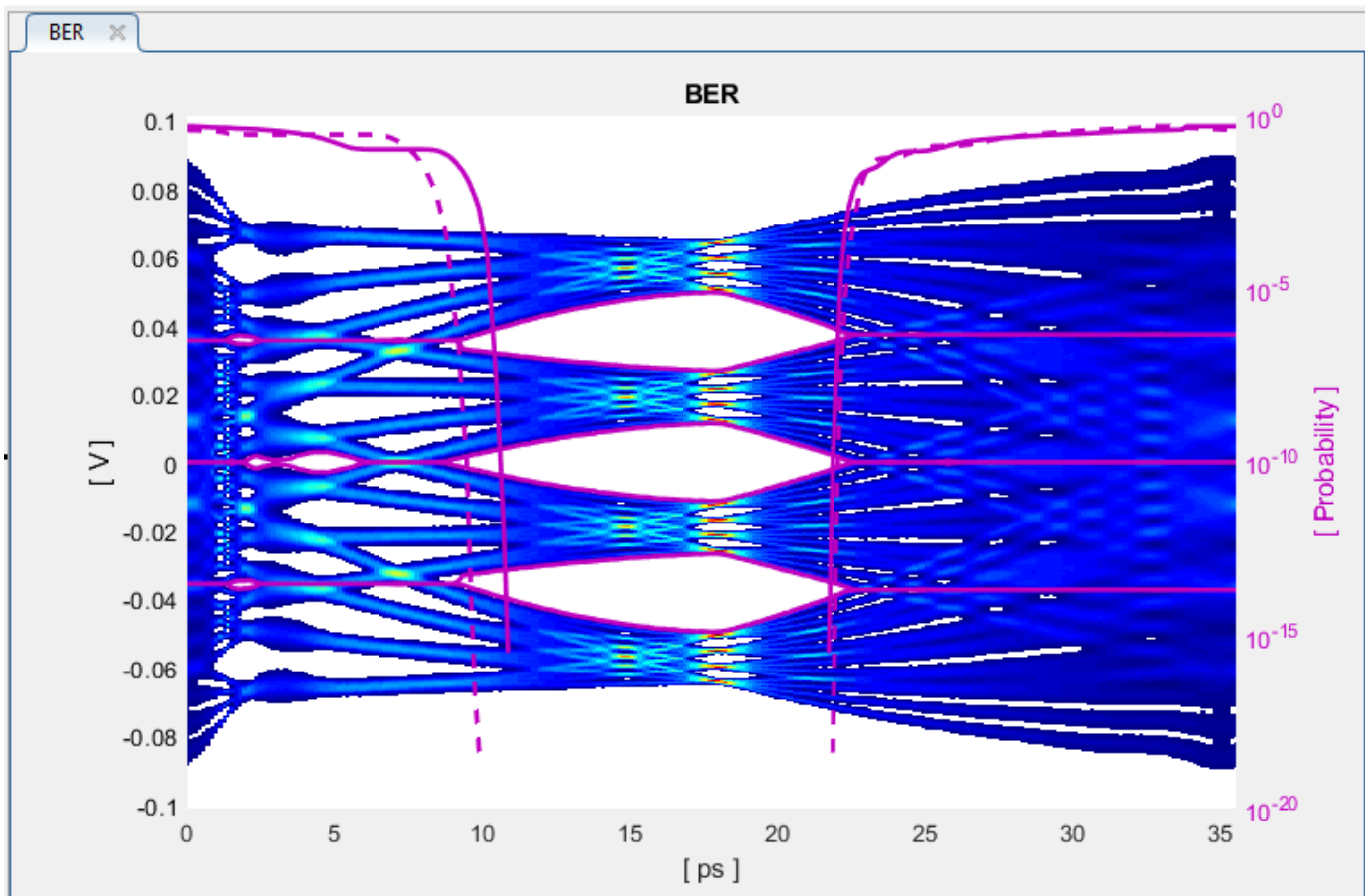
Add the report from Add Plots and observe that the CTLE Config is 129.

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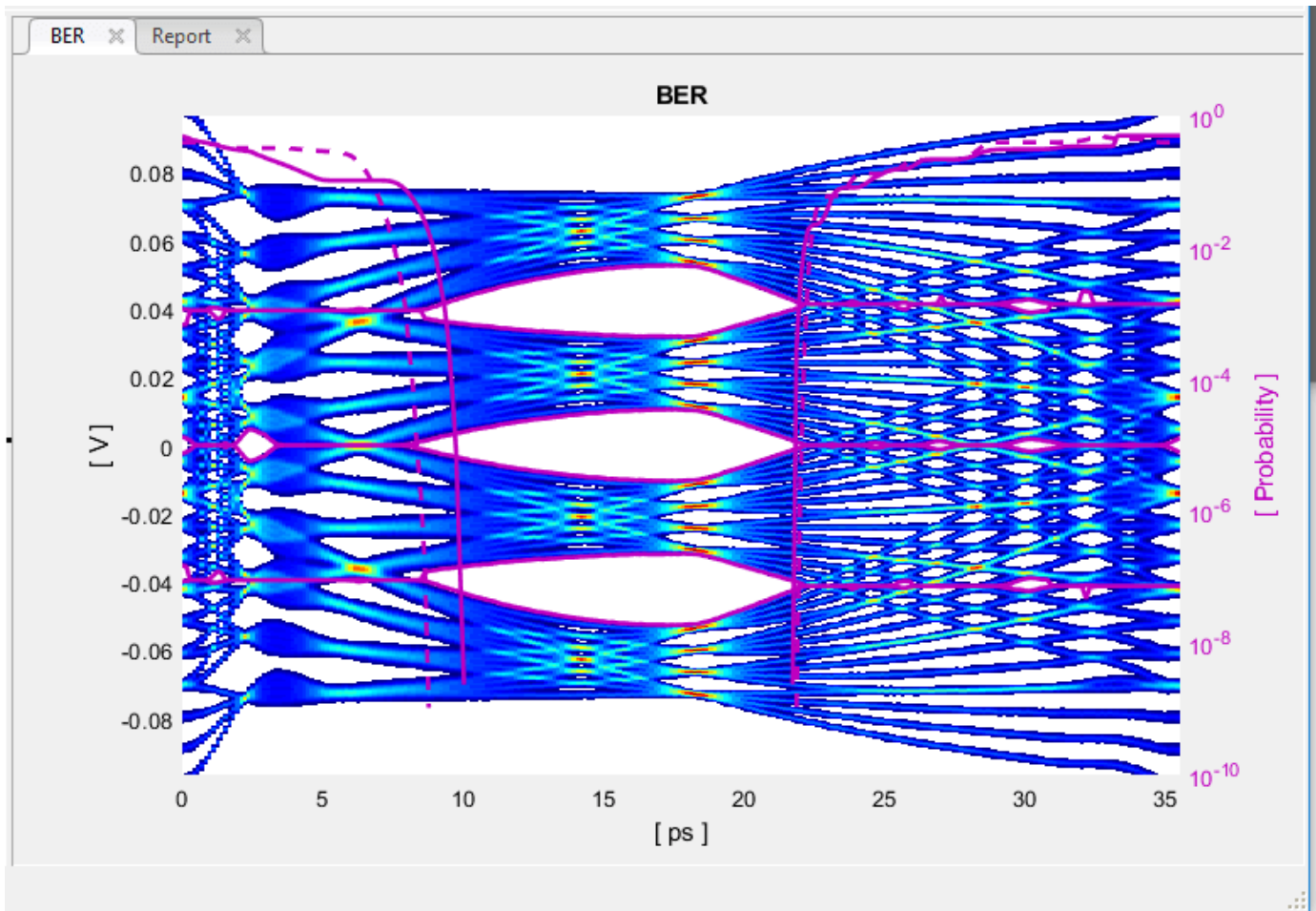
Before continuing, reset the value of Rx CTLE **Mode** back to `adapt`. Resetting here will avoid the need to set it again after the model has been exported to Simulink.

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 views of the statistical (Init) and time domain (GetWave) 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.

See Also
FFE | CTLE | DFECDR | **SerDes Designer**

More About
- "Managing AMI Parameters" on page 6-2
- "CEI 56G-VSR Compliance Kit" (Signal Integrity Toolbox)
- "CEI 56G-LR Compliance Kit" (Signal Integrity Toolbox)

USB 3.1 Transmitter/Receiver IBIS-AMI Model

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

USB 3.1 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 USB 3.1 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')
```
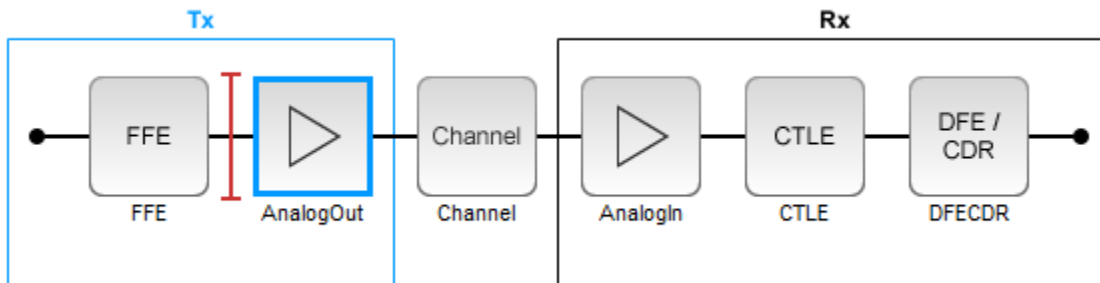

A USB 3.1 compliant transmitter uses a 3-tap feed forward equalizer (FFE) with one pre-tap and one post-tap. The receiver model uses a continuous time linear equalizer (CTLE) with seven pre-defined settings, and a 1-tap decision feedback equalizer (DFE). To support this configuration the SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to 100 ps, since the maximum allowable USB 3.1 operating frequency is 10 GHz.
- **Target BER** is set to 1e-12 as specified in the USB 3.1 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 USB 3.1 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.
- In the Rx CTLE, set the **Mode** to `fixed` and **Configuration Select** for "6." 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 USB 3.1 Behavioral CTLE specification.
- In the Rx DFE/CDR, set the **Mode** to `adapt`. The Rx DFE/CDR block is set up for one DFE tap. The limits for the tap are as defined by the USB 3.1 specification: `+/-50 mV`.

Plot Statistical BER Results

Use the SerDes Designer plots to visualize the results of the USB 3.1 setup.

Add the BER plot from **ADD Plots** and observe the results.
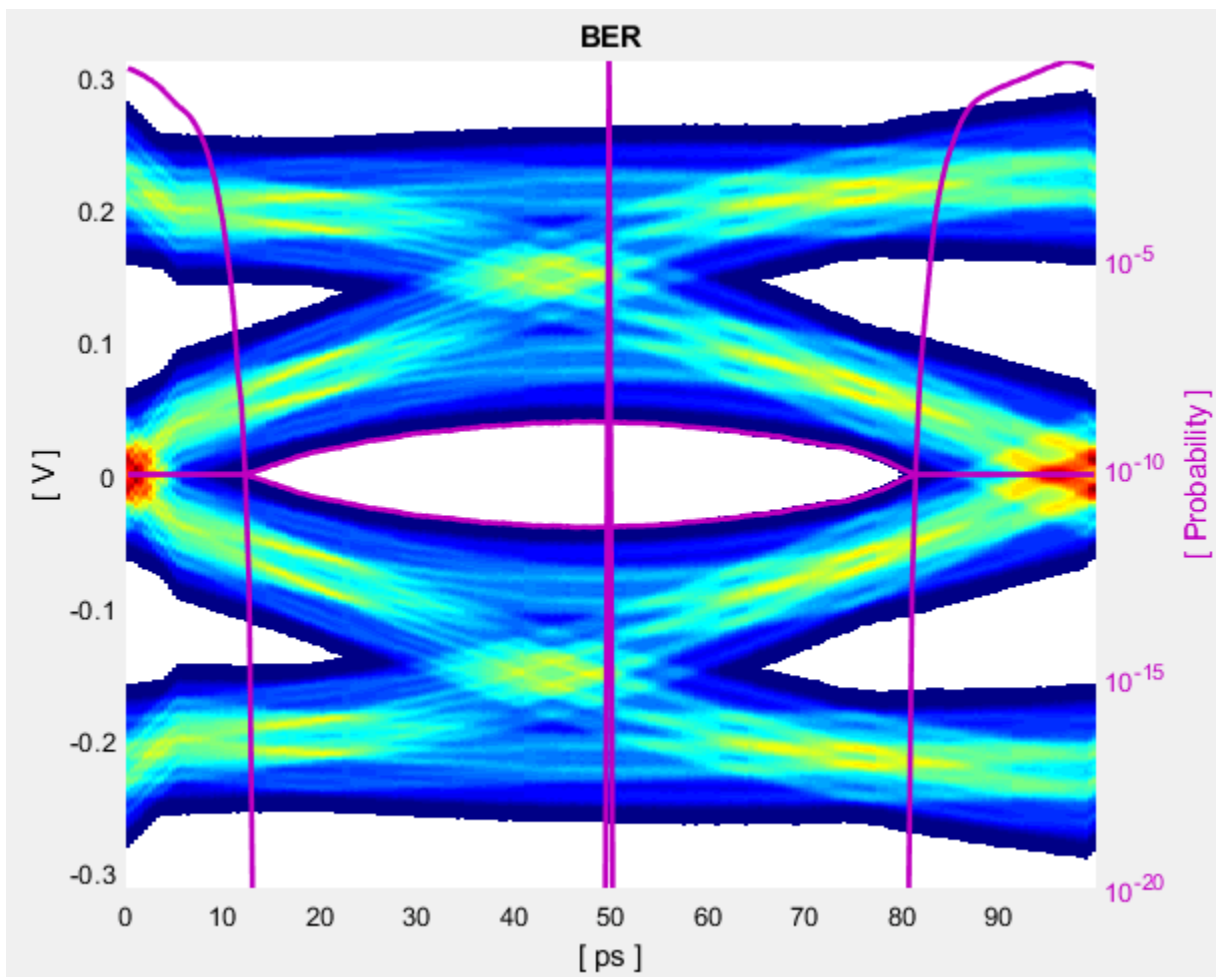
Change the Rx CTLE **Mode** parameter from `adapt` to `fixed` and change the **ConfigSelect** parameter value from 6 to 2 and observe how this changes the data eye.
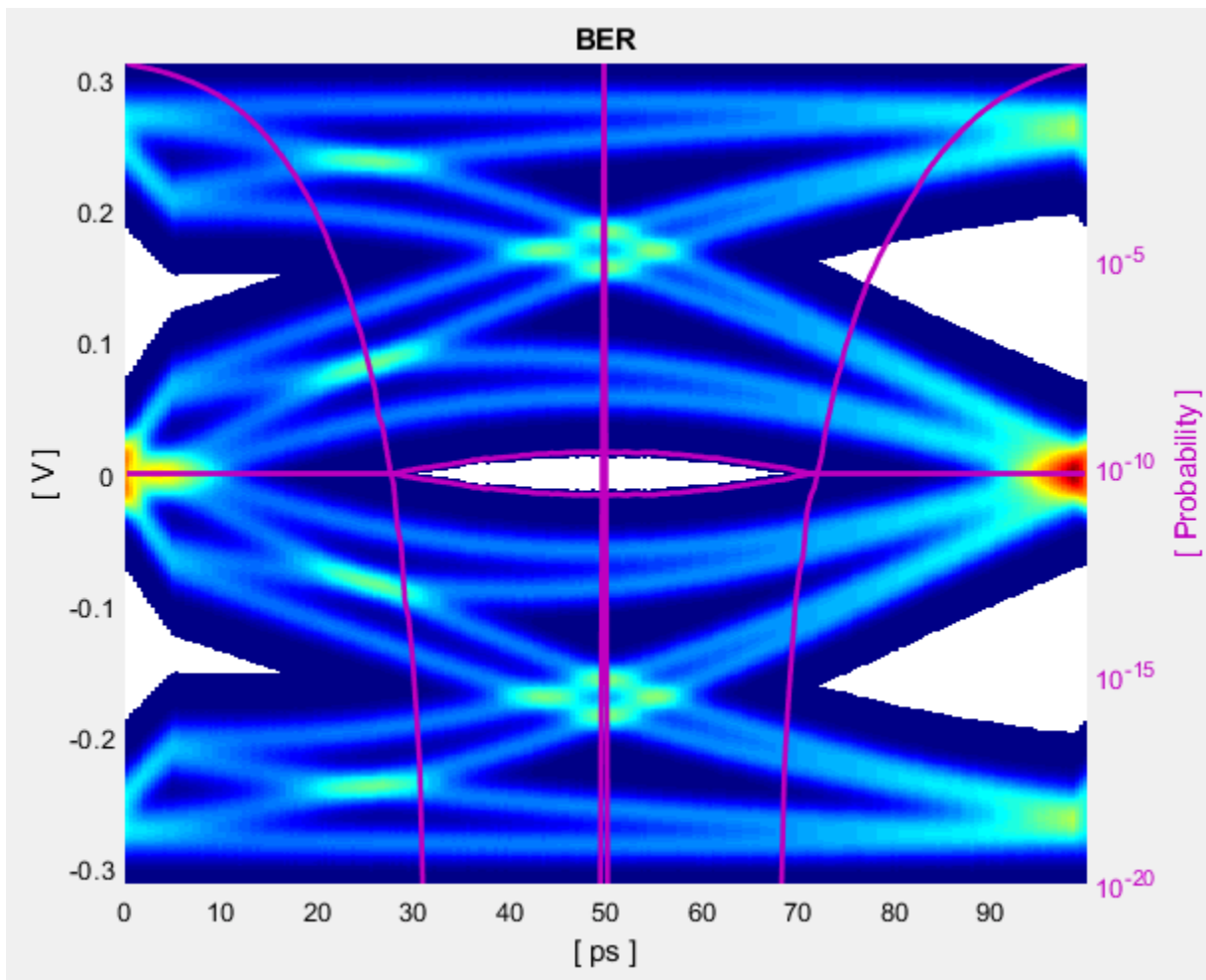
Before continuing, change the value of Rx CTLE **Mode** back to `adapt`. Resetting the value here will avoid the need to set it again after the model has been exported to Simulink.
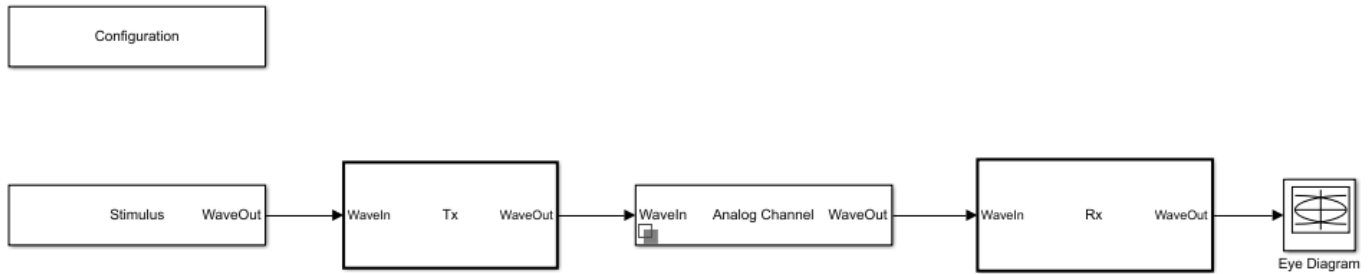
Export SerDes System to Simulink

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

USB 3.1 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 USB 3.1 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.
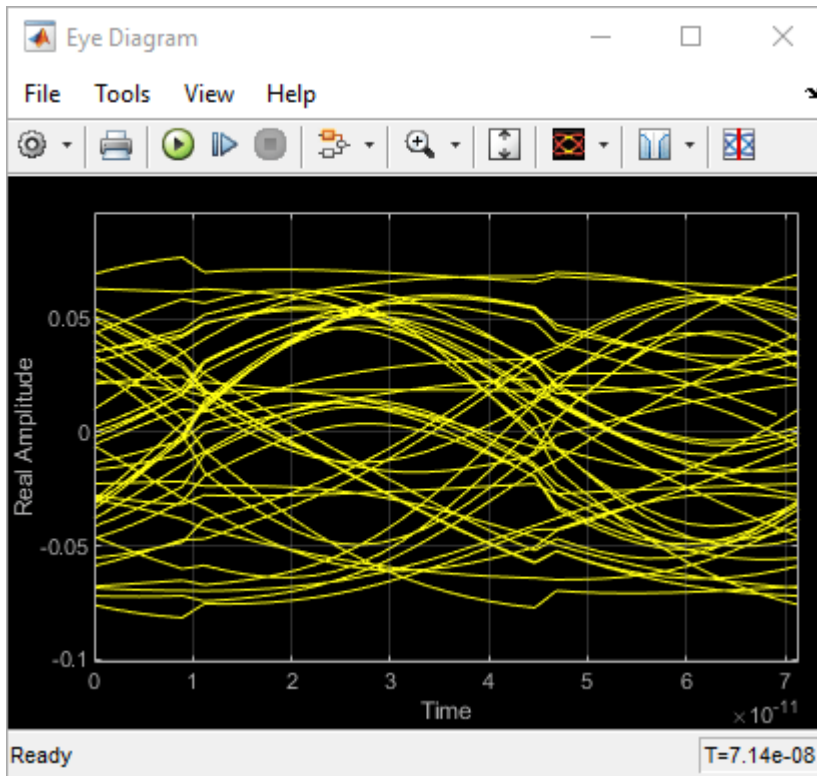
- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** are carried over from the SerDes Designer app.

- 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.
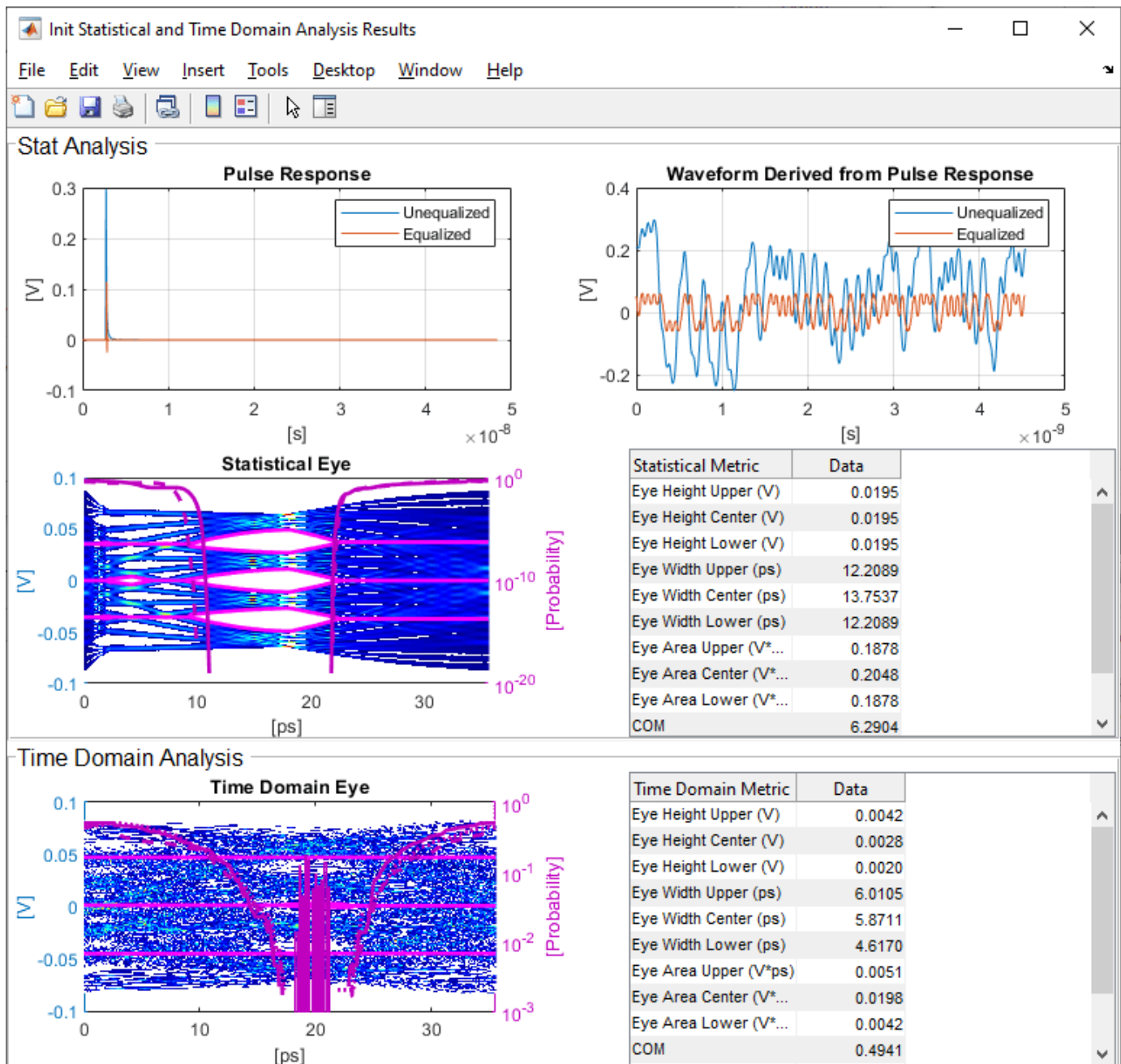
Run the Model

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 (note: this may not look exactly the same due to real-time adaptation).

After the simulation has completed the second plot contains views of the statistical (Init) and time domain (GetWave) 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 USB 3.1 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 USB 3.1 Tx/Rx IBIS-AMI Model

The final part of this example takes the customized Simulink model, modifies the AMI parameters for USB 3.1, then generates IBIS-AMI compliant USB 3.1 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 USB 3.1 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 USB 3.1 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 USB 3.1 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.

See Also
FFE | CTLE | DFECDR | **SerDes Designer**

More About
- "Managing AMI Parameters" on page 6-2
- "USB 3.1 Compliance Kit" (Signal Integrity Toolbox)

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) using a similar method as the one 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.

Introduction

IBIS 7.0 introduced the ability for models to perform link training, or auto-negotiation, by providing a mechanism for the Tx and Rx AMI executable models to communicate during GetWave operation. A link training algorithm can either emulate what the silicon is doing, or it can use channel analysis methods to determine the optimal Tx and Rx equalization settings, then lock in those settings for the remainder of the simulation.

Communications between the Tx and Rx executable models are in messages that both the Tx and Rx executable models understand, and the EDA tool does not need to understand. These agreed upon messages are called a Back-Channel Interface Protocol. The IBIS specification does not describe the details of the Back-Channel Interface Protocol but only a method to make the communication work. In this example we will be generating a new protocol named DDRx_Write.

Currently, SerDes Toolbox does not support the IBIS-AMI back-channel interface reserved parameters directly. Instead, it supports model specific parameters, which have "_ST" appended to their name, that perform a similar function. Since these model specific parameters do not use the same name as the reserved parameters from the IBIS specification, they must be used either as a "matched set" or with other back-channel models developed by SerDes Toolbox that support the same protocol. These models should work well in any industry standard AMI model simulator.

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 7-50 and the SDRAM receiver AMI model from "DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model" on page 7-38. We've added a few additional pass-through blocks to support the back-channel communication and you will then 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')
```
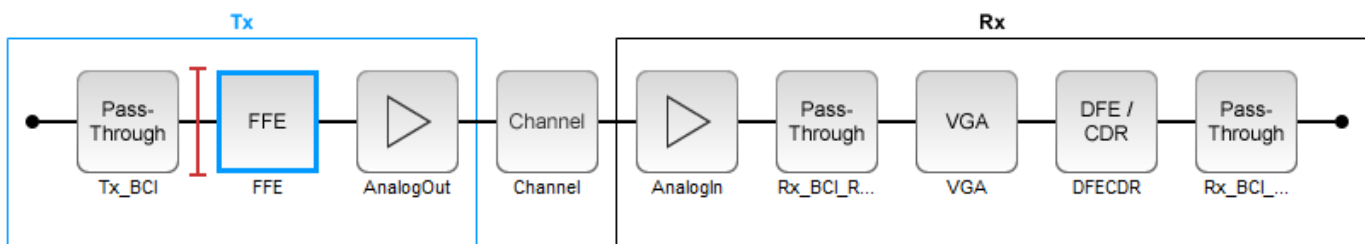

For a write transaction, the transmitter (Tx) is a DDR5 controller using 3-tap feed forward equalization (FFE), while the receiver (Rx) is using a variable gain amplifier (VGA) with 7 pre-defined

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 Rx AnalogIn model is set up so that **R** (input resistance) is 40 ohms and **C** (capacitance) is 0.65pF. The actual analog models used in the final model will be generated later in this example.
- 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 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 DFE has been configured to use 2x tap weights in order to be consistent with the JEDEC DFE tap definition.
- 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 and note that this automatically creates Data Stores in the Tx_BCI PassThrough block.

- **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_ST:** This parameter is only used to generate a parameter named "BCI_Protocol_ST" in the .ami file for partial 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_ST`, **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 the reserved parameter BCI_Protocol as an input to the model.` Save the changes.

- **BCI_ID_ST:** This parameter is only used to generate a parameter named "BCI_ID_ST" in the .ami file for partial 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_ST, **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 the AMI reserved parameter BCI_ID as an input to the model.` Save the changes.

- **BCI_State_ST:** 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_ST, **Usage** to InOut, **Type** to `Integer`, and **Format** to `List`. Set the **Description** as: `Back channel training status. NOTE: This model does not currently support the AMI reserved parameter BCI_State as an input to the model.` 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 change and note that this automatically creates Data Stores in the Rx_BCI_Write PassThrough block.

- **BCI_Protocol_ST:** This parameter only generates a parameter named "BCI_Protocol_ST" in the .ami file for partial 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_ST, **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 the AMI reserved parameter BCI_Protocol as an input to the model. Save the changes.

- **BCI_ID_ST:** This parameter only generates a parameter named "BCI_ID_ST" in the .ami file for partial 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_ST, **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 the reserved parameter BCI_ID as an input to the model. Save the changes.

- **BCI_State_ST:** 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_ST, **Usage** to InOut, **Type** to Integer, and **Format** to List. Set the **Description** as: Back channel training status. NOTE: This model does not currently support the AMI reserved parameter BCI_State as an input to the model. 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_ST:** This parameter only generates a parameter named "BCI_Message_Interval_UI" in the .ami file for partial compliance to the IBIS-AMI specification. This parameter is not used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to BCI_Message_Interval_UI_ST, **Current Value** to 64, **Usage** to Info, **Type** to Integer, and **Format** to Value. Set the **Description** as: Thie BCI model requires 1024 Samples Per Bit for proper operation. Save the changes.

- **BCI_Training_UI_ST:** This parameter only generates a parameter named "BCI_Training_U_STI" in the .ami file for partial compliance to the IBIS-AMI specification. This parameter is not used by this model. Click the **Add Parameter...** button. Set **Parameter Name** to BCI_Training_UI_ST, **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. NOTE: This model does not currently support the AMI reserved parameter BCI_Training_UI as an input to the model. Save the changes.

Run Refresh Init

To propagate all 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 plus two views from the Time Domain (GetWave) results.

Note: You can ignore any warnings for the unconnected blocks. These are due to the automatically generated data store blocks that will be addressed later.

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 the following 9 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:
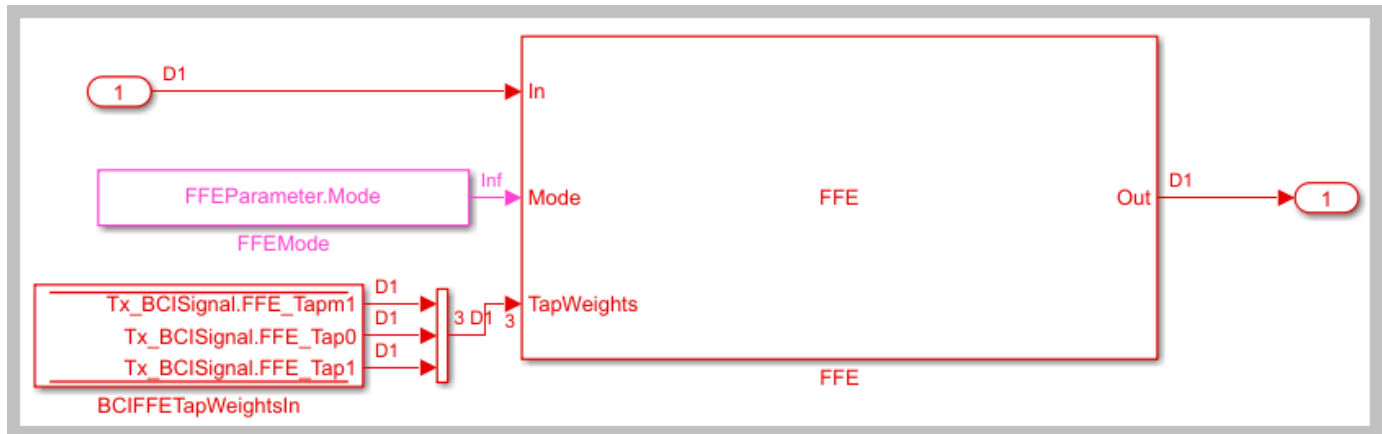
Type **Ctrl-D** to compile the model and check for errors. You can ignore any warnings for the unconnected blocks. These are due to the automatically generated data store blocks that will be addressed later

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 Elements in the bus.
2   Hit **Select>>** to move it to the list of Selected elements.
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` on the Parameters tab.
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:
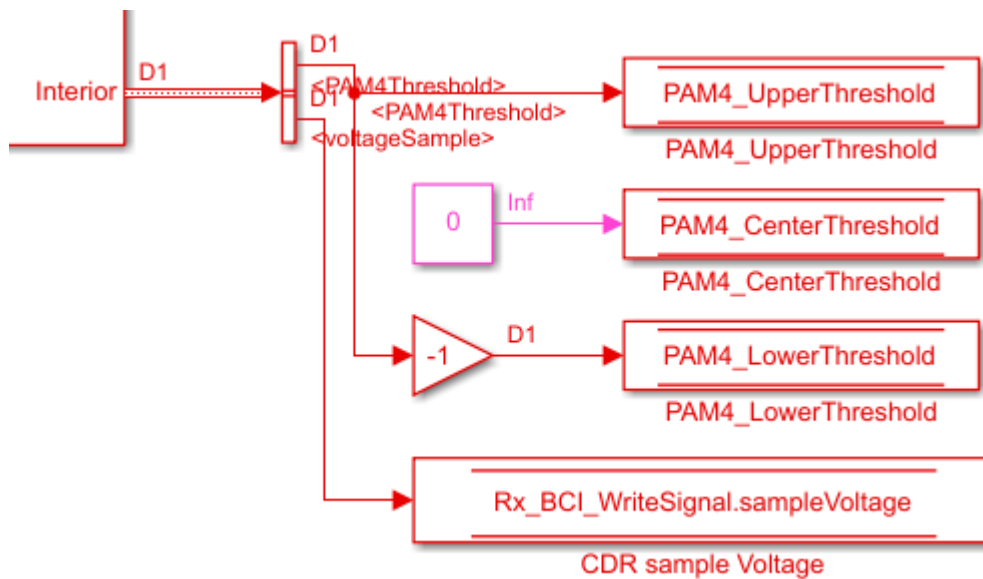
Type **Ctrl-D** to compile the model and check for errors. You can ignore any warnings for the unconnected blocks. These are due to the automatically generated data store blocks that will be addressed later

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_ST 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_ST 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_ST**.

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:
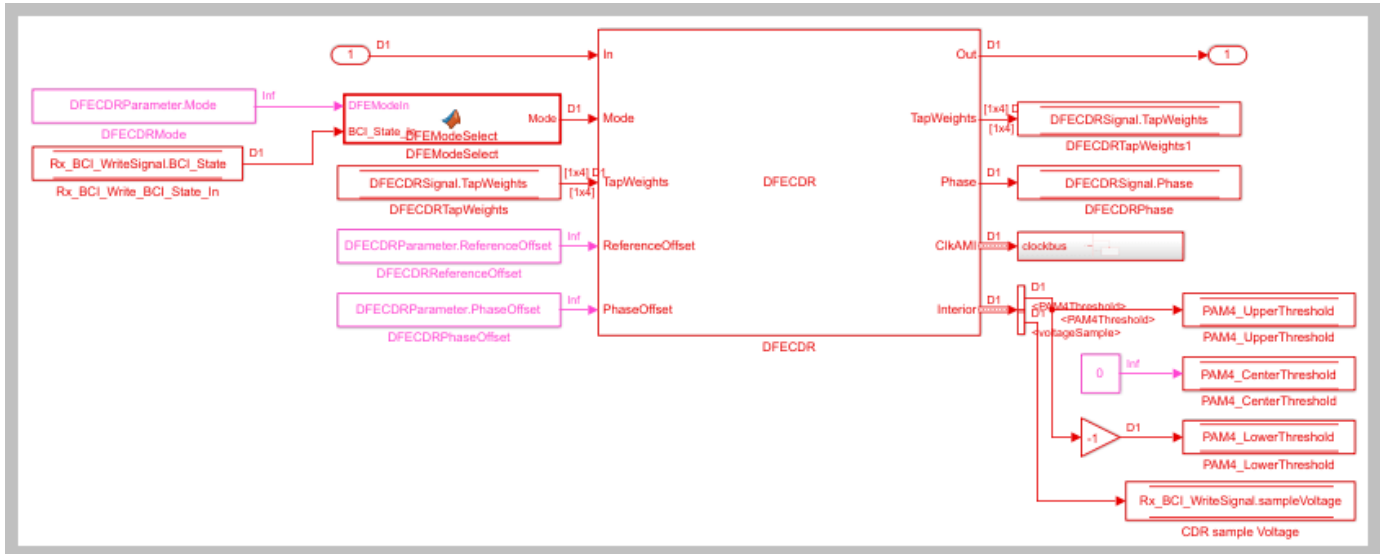
Type **Ctrl-D** to compile the model and check for errors. You can ignore any warnings for the unconnected blocks. These are due to the automatically generated data store blocks that will be addressed later

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, double-click on the Init block, then click on **Show Init** to open the Initialize Function in MATLAB.

The Initialize Function 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_ST = Tx_BCIParameter.BCI_State_ST; % User added AMI parameter from SerDes IBIS-AM
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_ST = Tx_BCIParameter.BCI_State_ST; % User added AMI parameter from SerDes IBIS-AM
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 Man

%% Set up for GetWave back-channel operation
if Tx_BCIBCI_State_ST == 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_ST, numDFEtaps, numFFEtaps, DF

end
```

To test that the new user code is working correctly, save and run the model, then verify that the new back-channel communication (BCI_comm.csv) and log (BCI_comm_log.csv) files have been created in the model directory 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 double click on the Init block, then click on **Show Init** to open the Initialize Function in MATLAB.

The Initialize Function 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_ST = Rx_BCI_WriteParameter.BCI_State_ST; % User added AMI parameter from Se

% 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_ST = Rx_BCI_WriteParameter.BCI_State_ST; % User added AMI parameter from Se

%% Set up for GetWave back-channel operation
if Rx_BCI_WriteBCI_State_ST == 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_ST, numDFEtaps, numFFEtap

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

end
```

To test that the new user code is working correctly, save and run the model, then 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. In the BCI_comm_log.csv file you should see that the first RX call has been added to the log file (Sequence #2).

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. You should see 8 automatically generated DataStore read/write blocks.

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

Add a **Mux** block and set the number of inputs to 3. This will be used to multiplex the three tapWeightsIn DataStoreRead signals into a single vector.

Add a **Demux** block and set the number of outputs to 3. This will be used to demultiplex the tapWeightsOut vector into three separate DataStoreWrite 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 persistent 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 In the MATLAB function signature highlight the parameter **SymbolTime**.
3 Right-click on the parameter and select `Data Scope for "SymbolTime"`.
4 Change the Data Scope from `Signal` to `Parameter`.
5 Repeat this process for **SampleInterval**.
6 When you save the MATLAB function you should see that these two input parameters have been removed from the function block on the Simulink canvas.

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 Simulink 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 will

be 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 it in the MATLAB editor.
2   Delete all the default contents.
3   Insert the following function signature:

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

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   In the MATLAB function signature highlight the parameter **SymbolTime**.
3   Right-click on the parameter and select `Data Scope for "SymbolTime"`.
4   Change the Data Scope from `Signal` to `Parameter`.
5   Repeat this process for **SampleInterval**.
6   When you save the MATLAB function you should see that these two input parameters have been removed from the function block on the Simulink canvas.

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_ST`.

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_ST`.

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 will be 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.

2   In the MATLAB function signature highlight the parameter **SymbolTime**.

3   Right-click on the parameter and select `Data Scope for "SymbolTime"`.

4   Change the Data Scope from `Signal` to `Parameter`.

5   Repeat this process for **SampleInterval**.

6   When you save the MATLAB function you should see that these two input parameters have been removed from the function block on the Simulink canvas.

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. You can see four automatically generated DataStore read/write blocks.

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

Delete the **DataStoreWrite** block labeled `sampleVoltage` write. It will not be used.

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.

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 will be covered later in this example. However, to complete the Rx_BCI_Write block connections you must display all the correct nodes. To enable this:
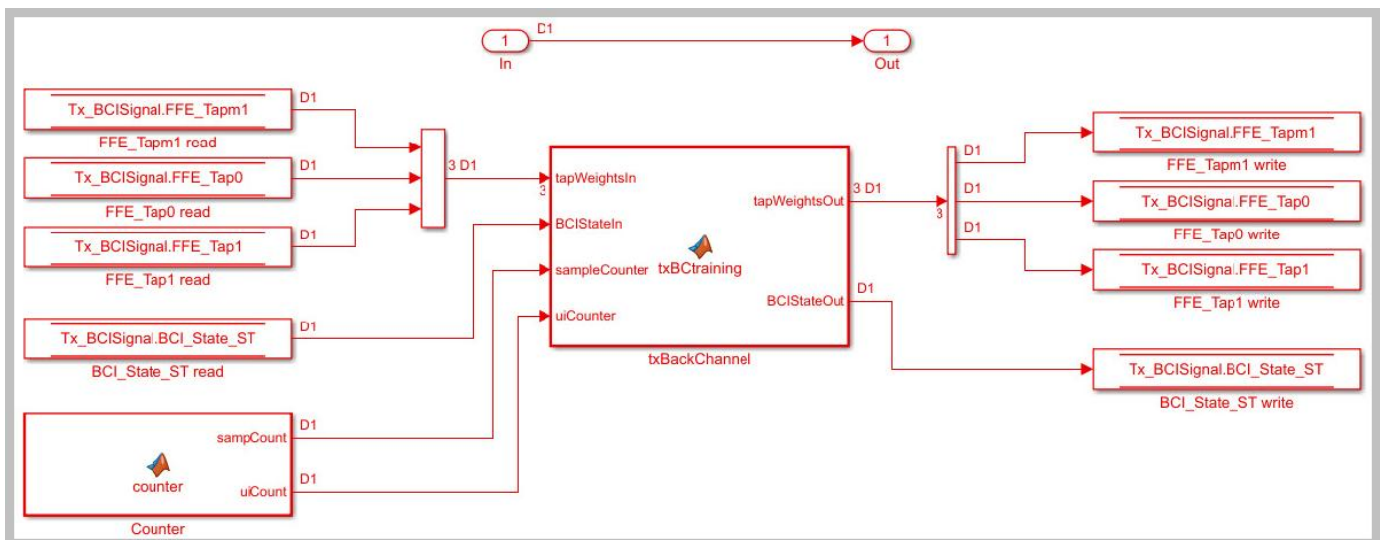
1 Double click the rxBackChannelWrite MATLAB function block to open in the MATLAB editor.

2 Delete all the default contents.

3 Insert the following function signature:

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

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 In the MATLAB function signature highlight the parameter **SymbolTime**.

3 Right-click on the parameter and select `Data Scope for "SymbolTime"`.

4 Change the Data Scope from `Signal` to `Parameter`.

5 Repeat this process for **SampleInterval**.

6 When you save the MATLAB function you should see that these two input parameters have been removed from the function block on the Simulink canvas.

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 you will 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 the 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 was earlier 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 was earlier 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 added earlier 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 PF

%% 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(F
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(FI
        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_ST 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(:);
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 earlier 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 PI

% 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 -
    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_ST 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 earlier 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 PI
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, FF
    %
    % 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_ST 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 enought time to complete.

Test proper operation 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 something like:

```
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,0.610993,
```

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 **SerDes IBIS-AMI Manager.**

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 on the AMI-Tx and AMI-Rx tabs.
- 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.

The BCI_State_ST parameter has 5 states required for complete back-channel training, however to make these models more user-friendly the end user only really needs 2 states: "Off" and "Training". To make this change, update the BCI_State_ST 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:

- It is intended that these models will be run as a "matched set" or with other AMI models that have been generated using SerDes Toolbox.
- These models will not work with AMI models generated outside of SerDes Toolbox. Specifically, any model that uses the IBIS standard BCI_* keywords.
- 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.
- Back-channel training must be enabled on both models for training to be enabled. This is done by setting the BCI_State_ST parameters to "Training".
- These models must be run with a block size of 1024 for proper operation.
- 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
FFE | PassThrough | VGA | DFECDR | **SerDes Designer**

More About

- "DDR5 Controller Transmitter/Receiver IBIS-AMI Model" on page 7-50
- "DDR5 SDRAM Transmitter/Receiver IBIS-AMI Model" on page 7-38
- "Managing AMI Parameters" on page 6-2
- "DDR5 IBIS-AMI with Clock Forwarding" (Signal Integrity Toolbox)

ADC IBIS-AMI Model Based on COM

This example shows how to create IEEE 802.3ck specification ADC-based transmitter and receiver IBIS-AMI models using library blocks in the SerDes Toolbox™ library and custom blocks to model a time-agnostic ADC. The generated models conform to the IBIS-AMI standard. The virtual sampling node, which exists in slicer-based SerDes systems, but does not exist in ADC-based SerDes systems, is emulated to allow for virtual eye diagram generation in the Simulink® and IBIS-AMI simulators for evaluating the channel.

SerDes IBIS-AMI Model Setup Using MATLAB Script

This example uses a MATLAB® script to first construct a SerDes System representing the transmitter and receiver of an ADC architecture and then export to a SerDes Simulink model. Type this command in the MATLAB command window to run the script:

```
buildSerDesADC
```

A SerDes System is configured with the following attributes before being exported to Simulink. Note that custom blocks will function as pass-throughs until the Simulink customizations discussed later in the example are applied.

Configuration Setup

* **Symbol Time** is set to 18.8235ps, since the maximum allowable 802.3ck operating data-rate is 106.25Gb/s.
* **Target BER** is set to 1e-4.
* **Samples per Symbol** is set to 32.
* **Modulation** is set to PAM4.
* **Signaling** is set to Differential.

Transmitter Model Setup

* The Tx FFE block is set up for 3 pre-tap and 1 post-tap by including 5 tap weights.
* The Tx VGA block is used to control the launch amplitude.
* The Tx AnalogOut model is set up so that **Voltage** is 1V, **Rise time** is 6.161ps, **R** (output resistance) is 50 Ohms, and **C** (capacitance) is 5fF according to the 802.3ck specification.

Channel Model Setup

* **Channel loss** is set to 15dB.
* **Target Frequency** is set to the Nyquist frequency.
* **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 5 fF according to the 802.3ck specification.
* The Noise custom block injects Gaussian noise to time domain waveform.
* A cascade of 3 Rx CTLE blocks is set up for 7, 21, and 1 configurations respectively. The **GPZ** (Gain Pole Zero) matrix data for each is derived from the transfer function given in the 802.3ck behavioral CTLE specification.

- The Rx VGA custom block applies adapted gain.
- The Saturating Amplifier block applies memoryless non-linearity.
- The ADC custom block quantizes the time domain signal.
- The Rx FFE custom has 21 taps (3-pre and 17-postcursor taps) whose weights will be automatically computed during the Rx global adaptation.
- The Rx DFECDR block is set up for one DFE taps. The DFE tap is limited to be +/- 0.5V amplitude.

ADC-Based SerDes Tx/Rx IBIS-AMI Model Setup in Simulink

The second part of this example takes the SerDes system exported by the script and customizes it as required for an ADC-based SerDes in Simulink.

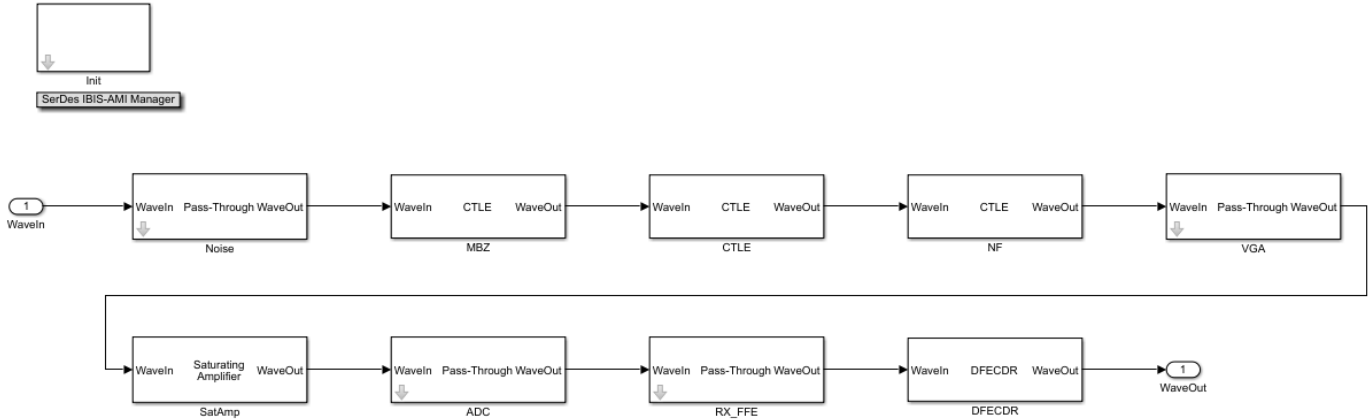**Review Simulink Model Setup**

The SerDes System exported into Simulink consists of Configuration, Stimulus, Tx, Analog Channel and Rx blocks.
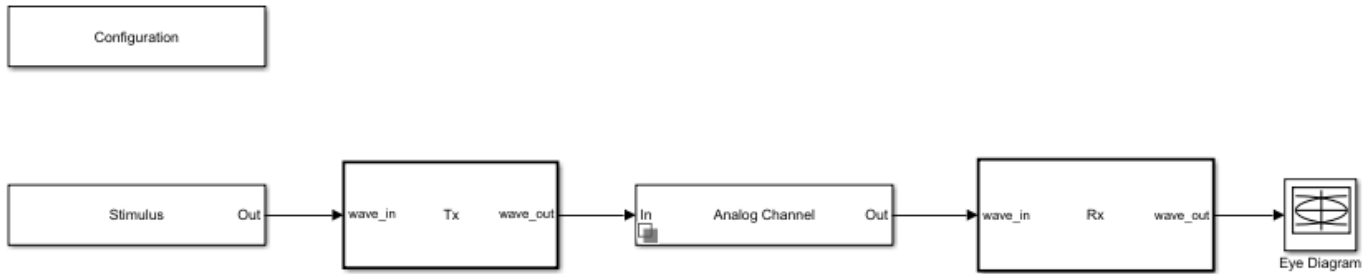
Push inside the Tx subsystem.
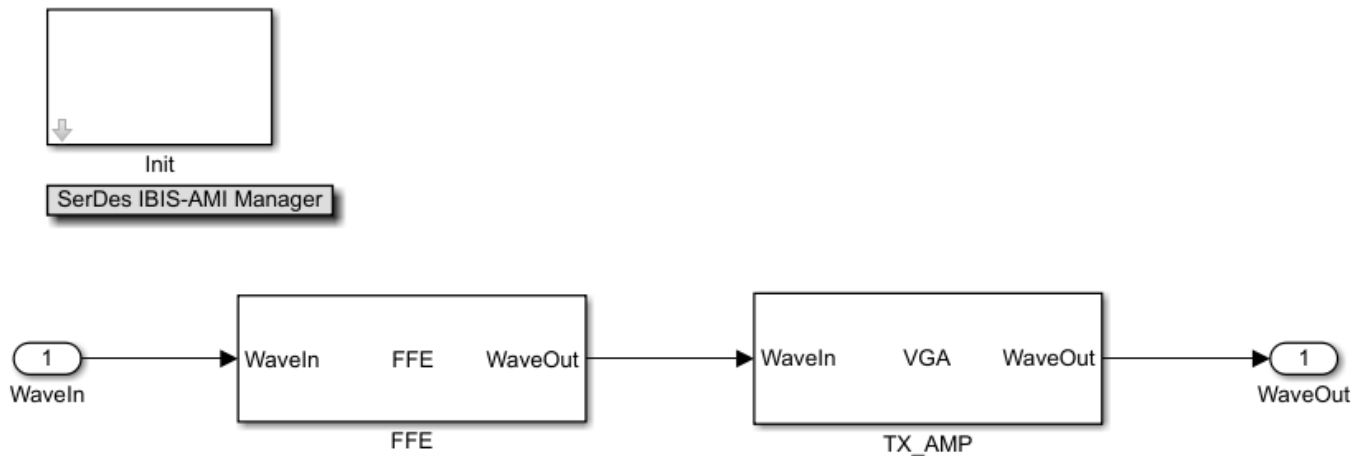

Push inside the Rx subsystem.

Customize the Model for ADC-Based SerDes

The model exported from the SerDes App needs to be first customized to represent an ADC-based SerDes Rx by customizing additional Rx blocks and modifying the Rx Init block code.

Configure Input Referred Rx Noise Block

Noise in the Rx subsystem can be modelled at the output, or at the input. An input referred noise source is shaped by the subsequent equalization stages (CTLE & FFE), and hence better reflects the how noise is shaped by the real system. On the other hand, output referred noise is not shaped, and does not capture how changing the settings on the CTLE and FFE impact noise.

- Descend into the Pass-Through block named Noise by clicking on the down arrow on block.
- Point the existing system object to the Noise.m system object in the example directory. See "Implement Custom CTLE in SerDes Toolbox PassThrough Block" on page 5-28.
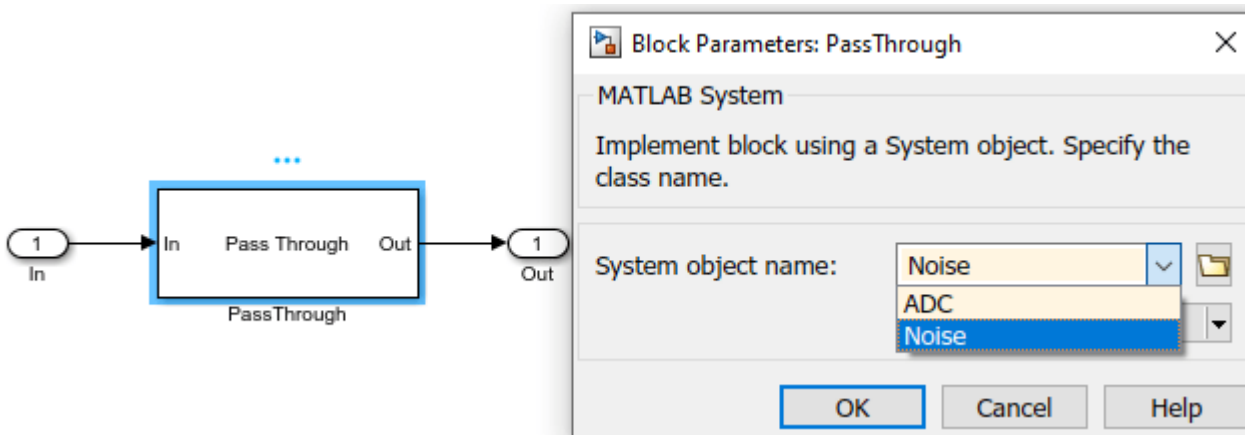
- In the system object mask, configure Symbol Time, Sample Interval, and Modulation with the system variables.
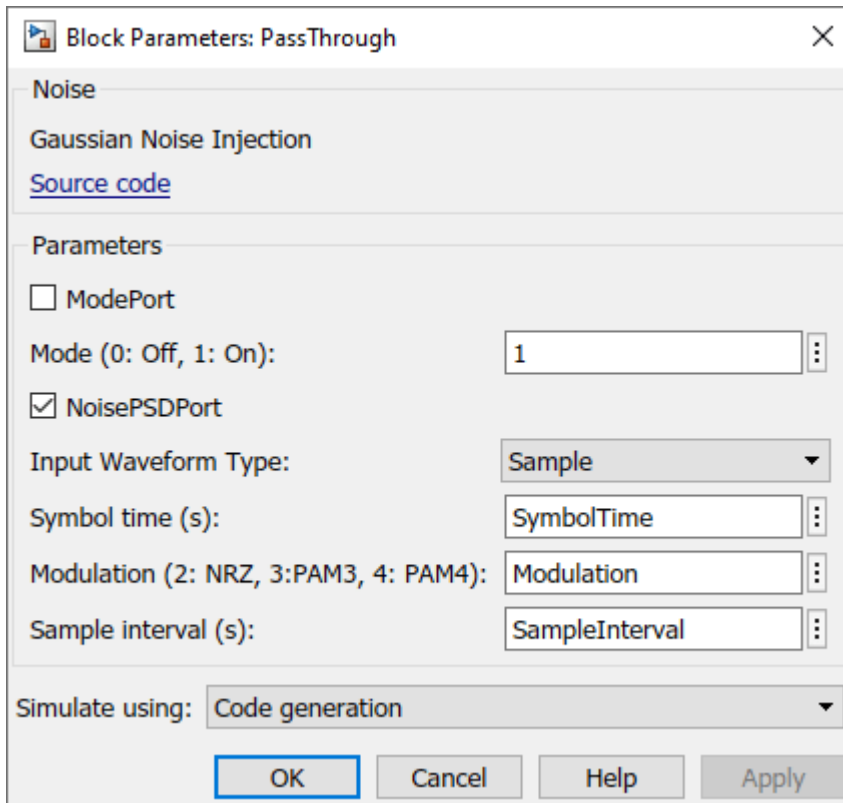
- Create an IBIS-AMI parameter in the IBIS-AMI Manager for the Noise block named **NoisePSD** using the pictured attributes. The value 8.2e-9 comes from the COM standard. See "Managing AMI Parameters" on page 6-2.
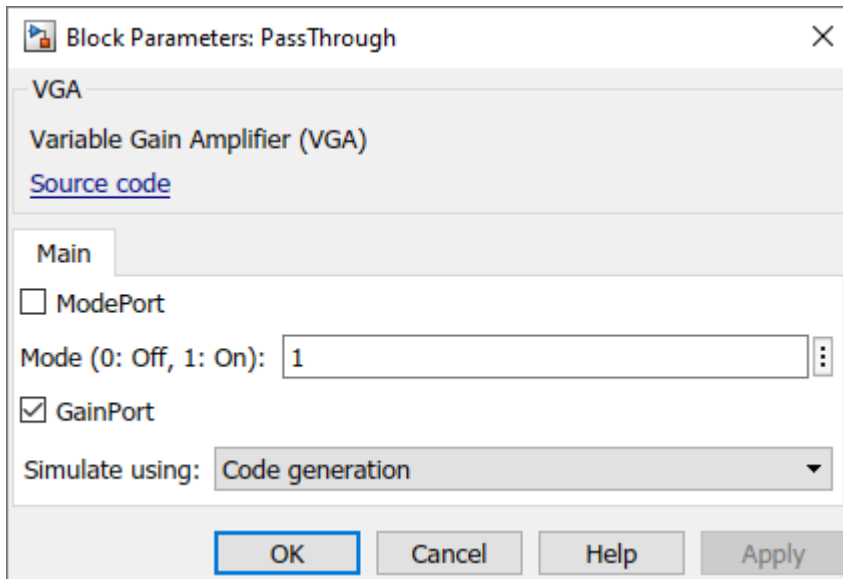
- Connect the generated constant block to Noise input port.
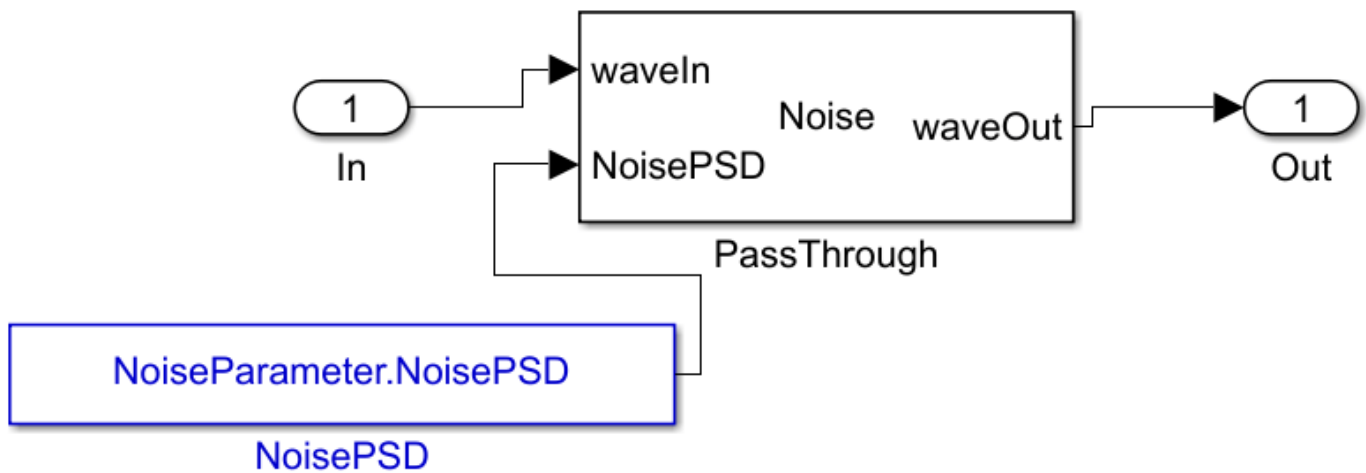
Configure VGA Block

- Descend into the Pass-Through block named VGA.
- Point the existing system object to the serdes.VGA system object included in SerDes Toolbox.
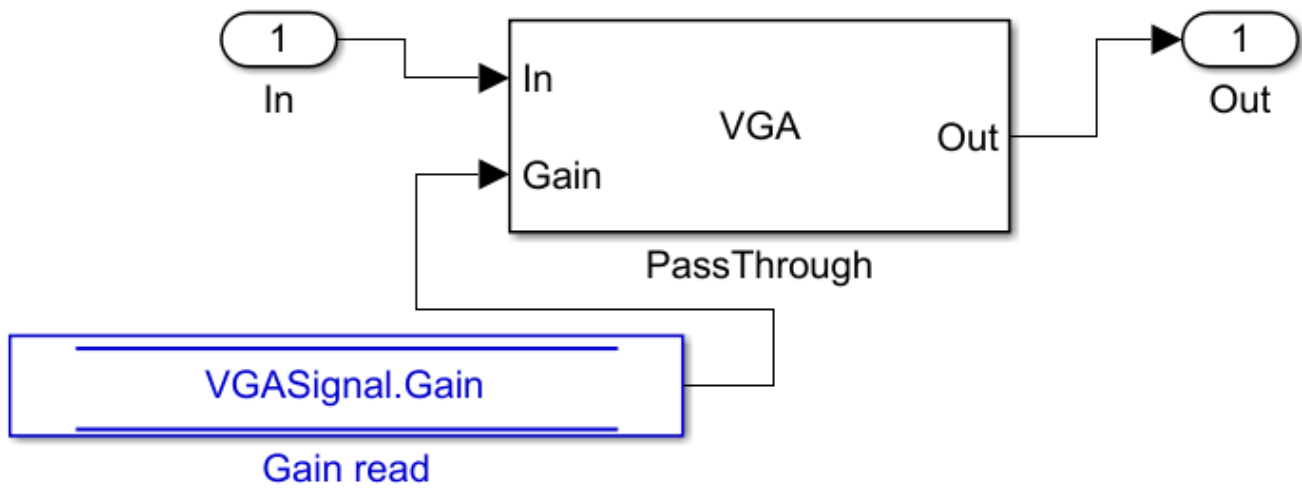- In the system object mask, turn off the Mode Port to force the block to be on.

- Create an IBIS-AMI parameter in the IBIS-AMI Manager for the VGA block named **Gain** using the pictured attributes.

- Connect generated data store read to Gain input port. Delete data store write as it will be unused because the value is only updated in Init and not time domain.

VGA Adaptation

VGA adaptation is straightforward, the required gain is calculated in Init as the ratio of a target pulse amplitude versus the maximum peak value of the input pulse response. Yet, the required VGA gain may be different for different CTLE settings, hence the VGA gain will need to be evaluated at each iteration of the general algorithm described previously.

Configure ADC Block

The ADC model used is a time-agnostic ADC, meaning that each point in the simulation is quantized, rather than just at the sampling instant. However, the DFE and clock recovery will still only use ADC samples at the sampling instant. A time-agnostic ADC allows for the generation of an equivalent waveform as seen at the DFE summing node: allowing for the construction of a signal eye diagram with a representative height and width.

- Descend into the Pass-Through block named ADC
- Point the existing system object to the ADC.m system object in the example directory.
- In the system object mask, configure Symbol Time, Sample Interval, and Modulation with the system variables.
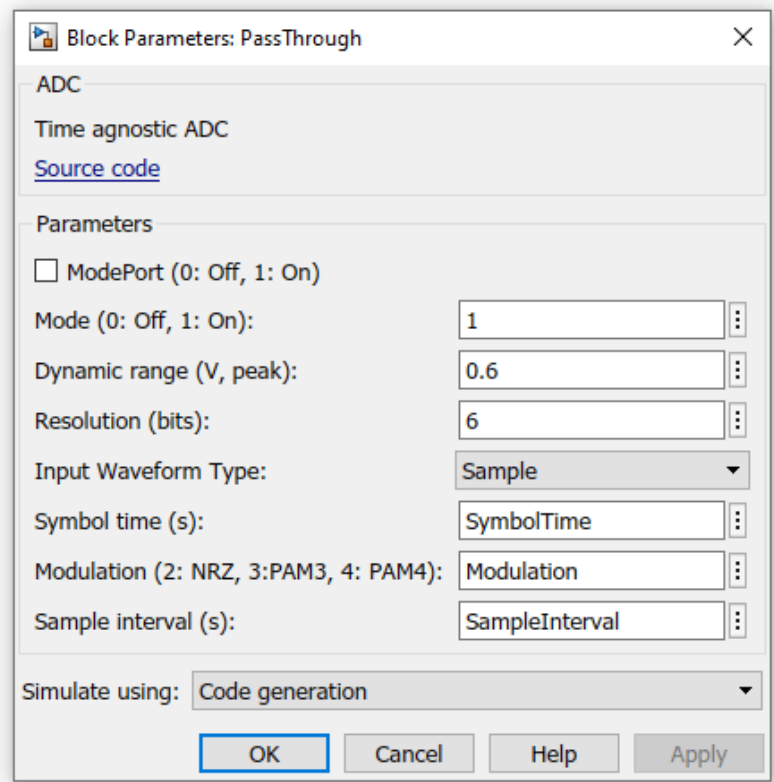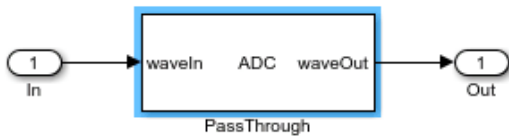
Configure Rx FFE

- Descend into the Pass-Through block named Rx_FFE
- Point the existing system object to the serdes.FFE system object included in SerDes Toolbox.
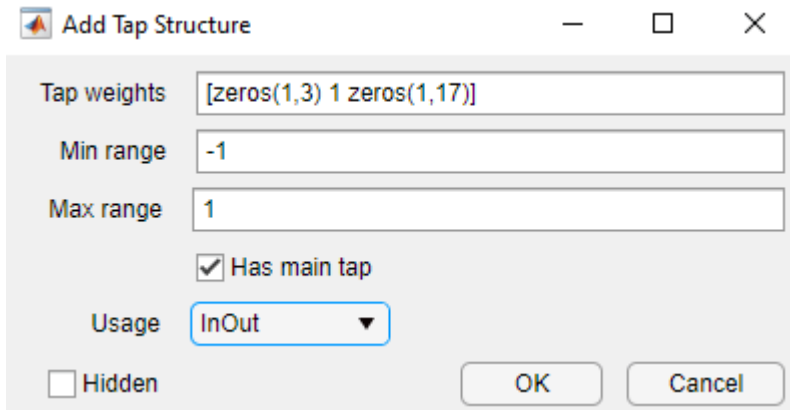- In the main tab of the system object mask, turn off the Mode Port and turn off Normalize Taps. In the advanced tab, configure Symbol Time and Sample Interval with the system variables.
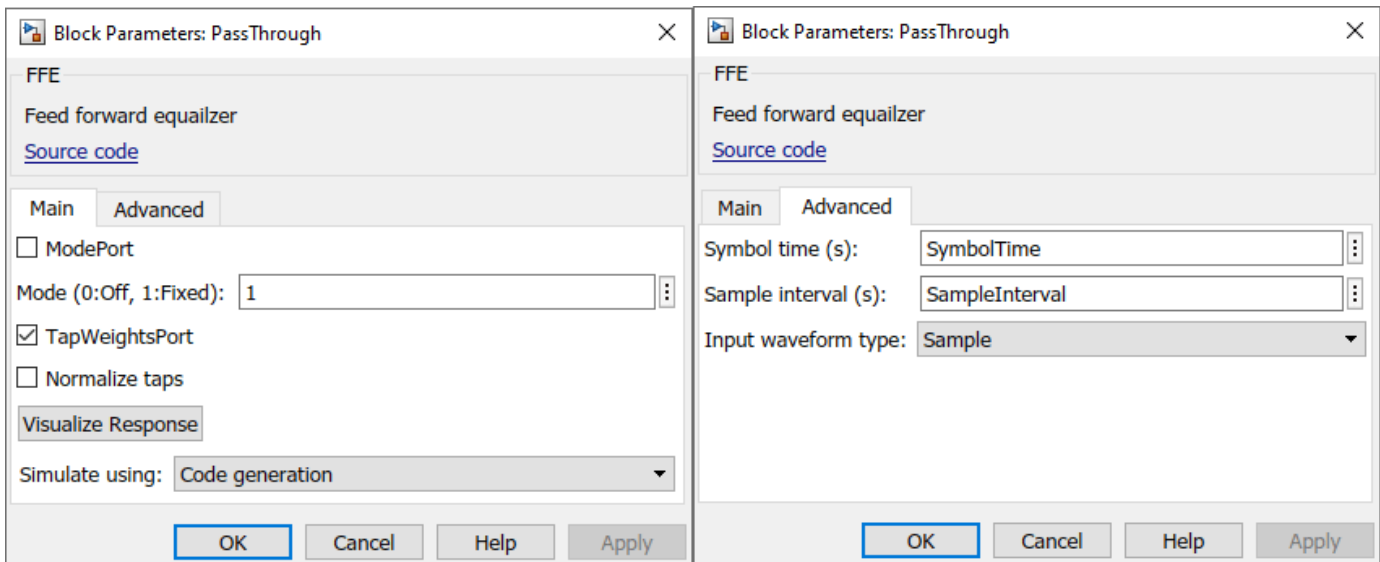
- Create a tap structure in the IBIS-AMI Manager for the Rx_FFE block with 3 pre-cursor taps, 17 post cursor taps, and the pictured attributes.

```
[zeros(1,3) 1 zeros(1,17)]
```


- Connect generated data store read to Tap Weights input port. Delete data store write as it will be unused.
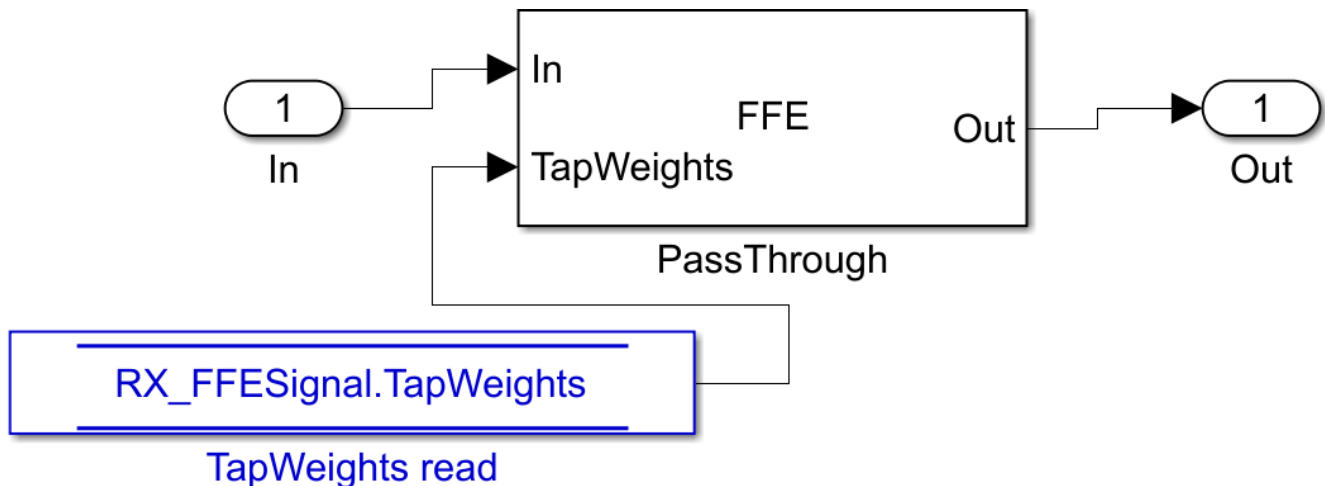
FFE Adaptation

The Rx FFE operates on ADC sampled data, rather than on a continuous waveform. However, during statistical adaptation, it is assumed that all of the waveform points, even in between data samples, are available. The Rx FFE is only adapted in the custom user Init code; adaptation is assisted by the adaptFFE function provided. The Rx FFE adaptation goal is to drive the output pulse response, given an input pulse response, such that the pre and post cursor data samples are driven to zero. This does not mean that the pulse response will be zero other than at the cursor point. Rather, much like a sync waveform, the ISI is only driven to zero at the data sample points.

As the Rx FFE operates on sampled data, the first step in the adaptation process, in adaptFFE, is to assume a data sampling phase for the input pulse response. The approach used is greedy to assume that we can force sampling so that the cursor lands on the peak of the incoming pulse response.

As the Rx FFE, in the Rx subsystem, is followed by a 1-tap DFE, the Rx FFE does not need to zero force the 1st post cursor. Rather, the Rx FFE needs to ensure that the 1st post-cursor falls within the

equalization range of the 1-tap DFE. Note, that if a post Rx DFE is not used, then the goal would be to zero-force all pre- and post-cursor ISI.

Given the now sampled input pulse response, the goal is to find a filter response that drives the pre- and post-cursor data samples to zero, or in the case of the 1st post cursor sample into the range of the DFE. This optimization problem is very closely related to solving a set of linear equations, where we need to find a matrix inverse. This matrix that needs to be inverted is a matrix made up of the circularly shifted input sampled pulse response. This inverted matrix then multiplied by the desired output target pulse response: [0, 0, 0, 1, bmax, 0, 0...] for the case of a 3-tap precursor Rx FFE, where the 1 denotes the cursor position and bmax denotes the maximum range of the DFE. The required Rx FFE FIR filter coefficients are the product of the inverted, circularly shifted input pulse-response matrix and the desired output pulse response.

DFECDR Adaptation

DFECDR adaptation follows Rx FFE adaptation. The DFECDR is the standard block in the SerDes toolbox, please refer to the online documentation for the DFECDR block.

This example uses an Alexander (bang-bang) phase detector, rather than a baud-rate phase detector that is typically used in ADC-based SerDes systems. This modelling choice simplifies the example, as a baud-rate phase detector would interact with the adaptation convergence. The ADC-based SerDes systems need to contend with the interaction between CDR lock point and Rx FFE & DFE adaptation.

Customize Rx Subsystem Init Code Block

In this example, the Rx subsystem adaptation is performed in the statistical domain: involving the co-adaptation of the CTLE, FFE, and DFE to achieve the best possible BER given the channel and Tx FFE settings used. The optimized settings for CTLE and FFE will remain fixed during time-domain simulations, while the DFE and CDR continue to adapt during the time-domain simulation.
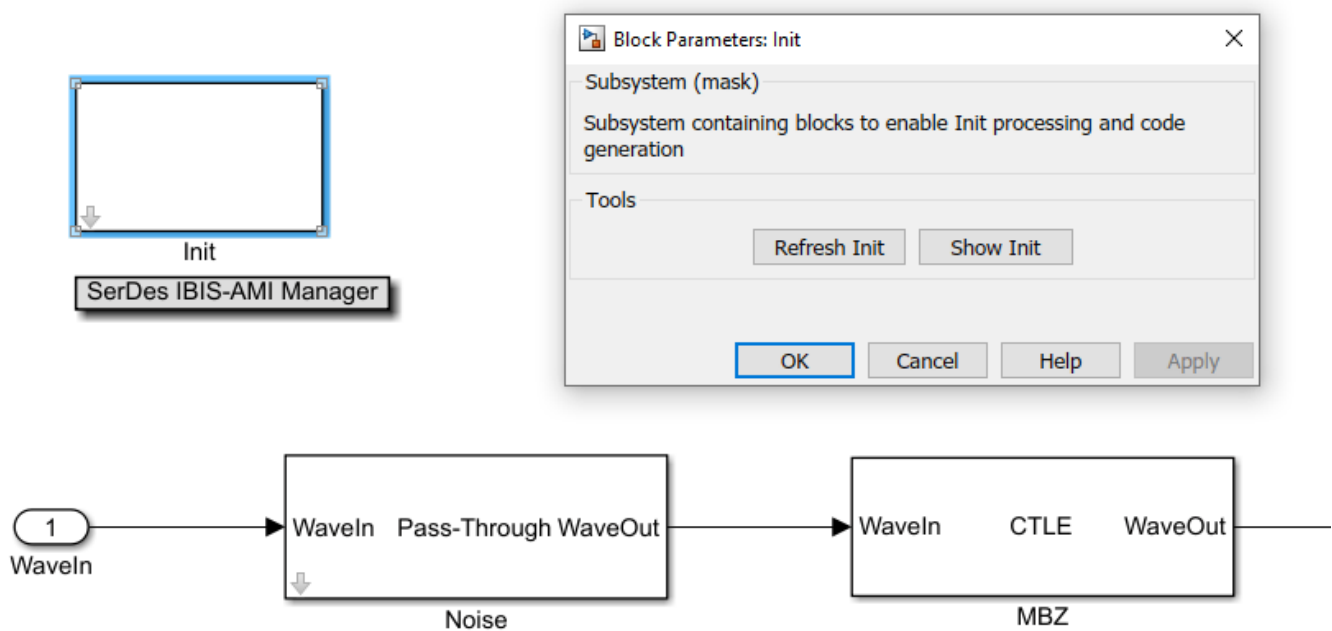
Modify the custom user code area of Init with the code provided with the example. See "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance" on page 4-27.

- Click Refresh Init on the Init mask dialog to update code based on previous steps.
- Click Show Init on the Init mask dialog to open the Init code.
- Copy the code in `adcInitCustomUserCode.m` within the example directory.

```
edit adcInitCustomUserCode.m
```

- Paste the copied code just before the end of the custom user code area. Ensure that the AMI parameters at the top of the custom user area are retained. Do not modify code beyond the end of the custom user area.

```
124 -    DFECDRInit.Count = 16;
125 -    DFECDRInit.ClockStep = 0.0078;
126 -    DFECDRInit.Sensitivity = 0;
127      %% BEGIN: Custom user code area (retained when 'Refresh Init' button is pressed)
128 -    NoiseInit.NoisePSD = NoiseParameter.NoisePSD; % User added AMI parameter from SerDes IBIS-AMI Manager
129 -    RX_FFEInit.TapWeights = RX_FFEParameter.TapWeights; % User added AMI parameter from SerDes IBIS-AMI Manager
130 -    RX_FFETapWeights = RX_FFEParameter.TapWeights; % User added AMI parameter from SerDes IBIS-AMI Manager
131 -    VGAGain = VGAParameter.Gain; % User added AMI parameter from SerDes IBIS-AMI Manager
132 -    VGAInit.Gain = VGAParameter.Gain; % User added AMI parameter from SerDes IBIS-AMI Manager
133      % SerDes ADC Init Custom User Code
134
135      %   Copyright 2020 The MathWorks, Inc.
136
137      % Prepare required parameters
138 -    SamplesPerSymbol = round(SymbolTime / SampleInterval);
139 -    ISILimit = 0.001; % ISI ignore limit, fraction of cursor
140 -    NoiseBW  = 100e9; % Noise integration BW, Hz
141
142      % Set mid-band zero (MBZ) adaptation bounds
143 -    iiMBZ = 1; %Index
144 -    if MBZInit.Mode == 2
145          %If adapt mode
```

Statistical Adaptation Algorithm

The statistical adaptation algorithm processes the impulse response though each of the Rx subsystem blocks, and measures the resulting impulse response figure of merit. As this is an ADC-based system, the figure of merit used is signal-to-noise (SNR), where the noise term also includes residual pre- and post-cursor ISI.

In general, statistical Rx adaptation will proceed as follows:

- An initial CTLE setting is selected
- A VGA setting is chosen such that the pulse amplitude falls within target bounds
- The Rx FFE is automatically adjusted so that ISI at data sample points is minimized.
- The DFE is adapted to remove post-cursor ISI.
- SNR at data sample points is evaluated.
- Steps above are repeated for each possible CTLE setting, keeping track of SNR values for each setting. The setting with the highest SNR is chosen as the global adaptation point.

Run The Simulink Model

- Visit the Stimulus block mask dialog and change number of symbols to 4000.
- Visit the export tab of the IBIS-AMI Manager and update the Rx ignore bits to 2000. This and the previous modification will ensure that the time domain adaptation has ample time to converge. A larger number of symbols and ignore time will yield more realistic results.
- Run the model to simulate the ADC-based SerDes system.
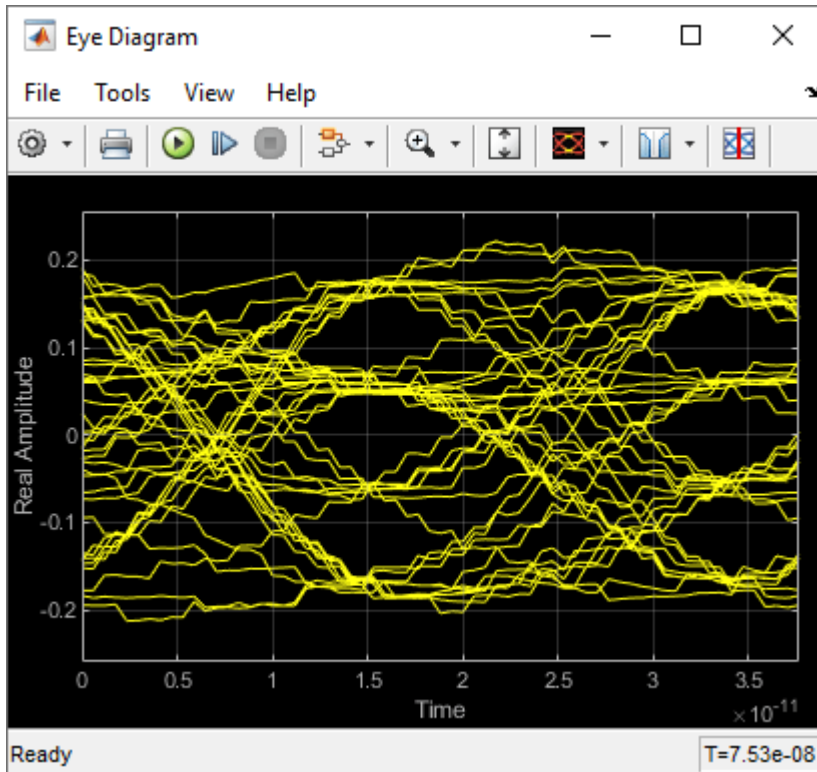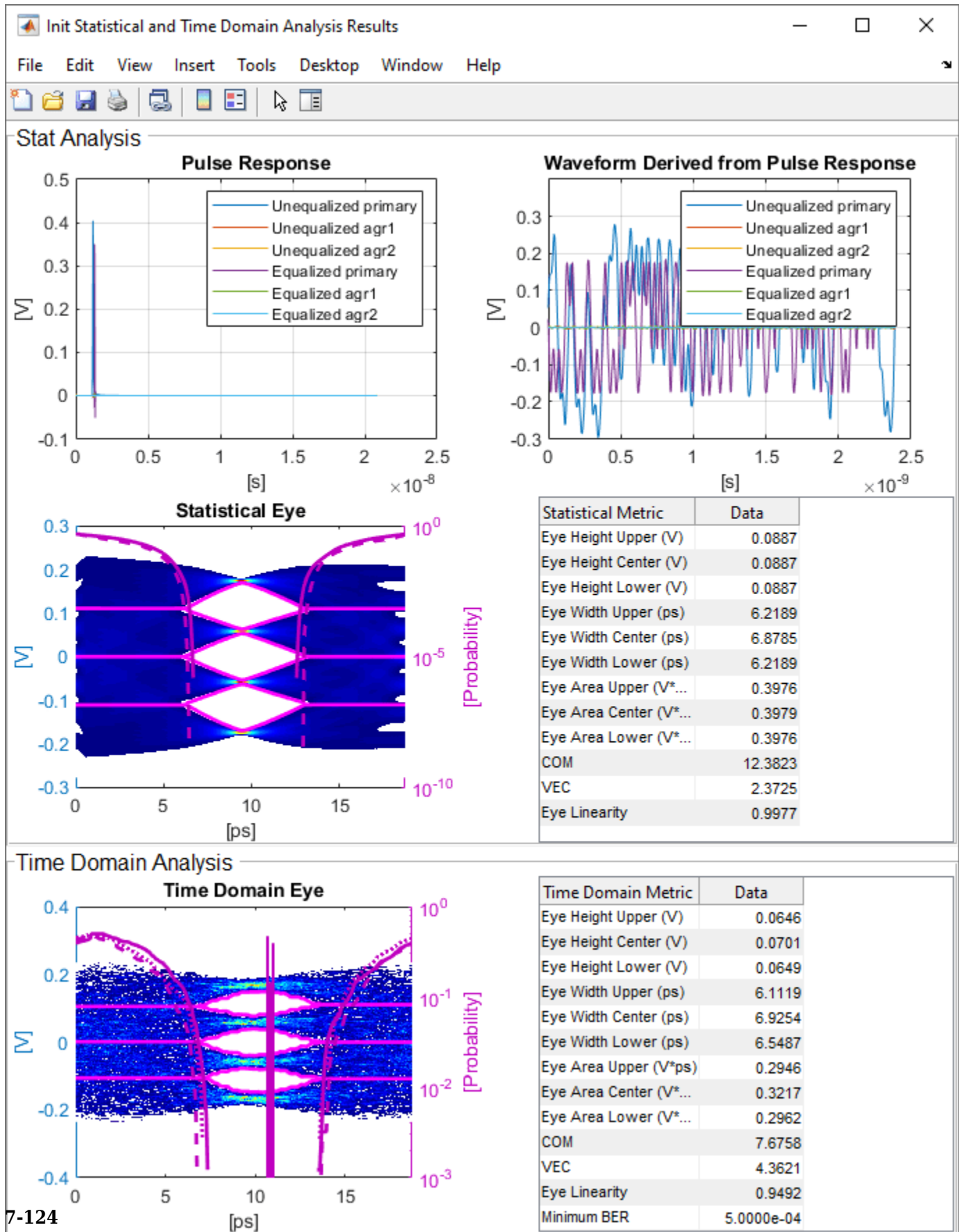
Update the ADC Quantization

In the example the ADC quantization is set to 6b, by default. Try changing the ADC quantization to a lower amount, observe how the time-domain eye shape is affected by reduced ADC precision.

Generate ADC-Based SerDes IBIS-AMI Model

The final part of this example takes the customized ADC-based SerDes Simulink model and then generates an IBIS-AMI compliant model: including model executables, IBIS and AMI files.

The current IBIS AMI standard does not have native support for ADC-based SerDes. The current standard is written for slicer-based SerDes, which contain a signal node wherein the equalized signal waveform is observed. In a slicer-based SerDes this node exists inside the DFE, right before the decision sampler. A continuous analog waveform is observable at that node, which includes the effect of all the upstream equalizers (such as CTLE) and the equalization due to DFE, as tap weighted and fed back prior decisions. Such a summing node does not exist in an ADC-based SerDes, due to the ADC in the system. In a real ADC-based SerDes system the ADC proves a vertical slice though the eye at the sampling instant. To emulate a virtual node, a time-agnostic ADC is used. This ADC quantizes each point in the incoming analog waveform at the simulation time-step rate: i.e. 1/fB/SPS, where SPS is the number of samples per symbol, and fB is the baud-rate. The Rx FFE also processes the input signal as a continuous waveform, rather than as samples. However, the Rx FFE applies a single tap values for SPS-simulation time-steps. The DFE is the stock DFE from the SerDes Toolbox and is written for slicer based SerDes. This signal chain allows for the signal integrity simulator to be able to observe a virtual eye in an ADC-based system.

Export IBIS-AMI Models

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

- 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 5 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).

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

See Also

FFE | CTLE | DFECDR | VGA | SaturatingAmplifier

More About

- "Globally Adapt Receiver Components Using Pulse Response Metrics to Improve SerDes Performance" on page 4-27

- "Implement Custom CTLE in SerDes Toolbox PassThrough Block" on page 5-28

- "Managing AMI Parameters" on page 6-2

- Best Practices for Modeling PAM4 SerDes Systems and Improving IBIS-AMI Correlation

Architectural 112G PAM4 ADC-Based SerDes Model

This example shows how to use a IEEE 802.3ck specification transmitter and receiver architectural model using library blocks in the SerDes Toolbox™ library and custom blocks to model a 112G PAM4 time-interleaved ADC-Based SerDes. The performance impact of the timing mismatch between the time-interleaved ADCs is explored as an example of a design trade-off study. The aspects of the model which are compatible with the IBIS-AMI 7.0 standard are used to create an IBIS-AMI model.

Overview

There are many trade-offs to be explored when designing an ADC-based SerDes. Some of these design trade-offs and resulting questions are summarized below:

ADC Exploration

- What type of ADC will be used in this design? Flash, binary/multibit search, or SAR (successive approximation register)?
- What ADC time-interleave factor should be used? What is the impact on system performance caused by mismatches of the timing, gain, voltage offset and/or bandwidth between parallel ADCs and explore mismatch calibration algorithms?
- What full-scale range should be used?
- What resolution/quantization/number of bits is required?
- Should the quantization be uniform or non-uniform?
- What is the performance impact of quantization noise?

Digital Equalization Exploration

- What number of FFE taps is required to achieve needed performance?
- How many DFE taps can be implemented?
- What DSP resolution is required?
- What impact does frame size or parallelism of digital processing have on system latency. What demultiplexer width should be used?

Clock Recovery

- What bandwidth of CDR is required? How should the loop filter be specified?
- Which Mueller-muller cost function is best for the application?

Analog Front End

- How many CTLE stages are needed?
- How does the CTLE amplified noise impact system performance?
- How to scale signals in order to take advantage of the linear range of the ADC?

This example focuses on determining the impact of the time-interleaved timing mismatch between parallel ADCs. The model can be the basis of exploring many other design trade-offs. The system SNR (signal-to-noise ratio) is compared between cases with and without a 4% symbol time timing offset and shows that this impairment reduces the system performance by about 2.5 dB.

Rx Model Description

The receiver model is composed of an analog front-end (AFE) with CTLE and amplifier blocks. The time-interleaved ADC is further parallelized by a demultiplexer before DSP processing by the FFE and DFE. The baud-rate CDR controls the VCO which drives the ADC. The system performance is quantified by a SNR metric as well as an output waveform. This model is summarized in the following diagram where the time-interleave depth (or the number of ADC's) is four and the demux size is 64.

Open the Simulink® model `ArchitecturalADCBasesSerDes.slx` attached with this example.
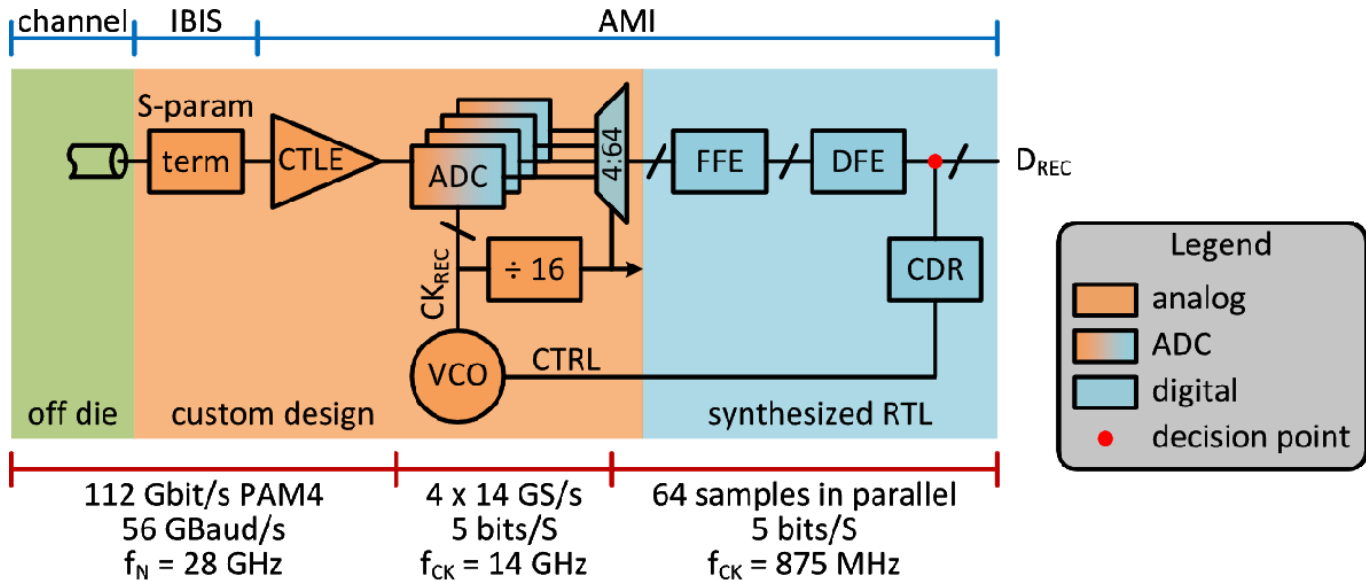
The receiver analog front-end partially equalizes the waveform and is very similar to the "ADC IBIS-AMI Model Based on COM" on page 7-111. Here the first block injects the input referred noise, followed by the mid-band zero CTLE, main CTLE, and noise filter CTLE blocks as specified by channel operating margin (COM) of IEEE 802.3ck. The VGA scales the signal to match the full scale range of the ADC and the Saturating Amplifier block enforces a memoryless-nonlinearity.
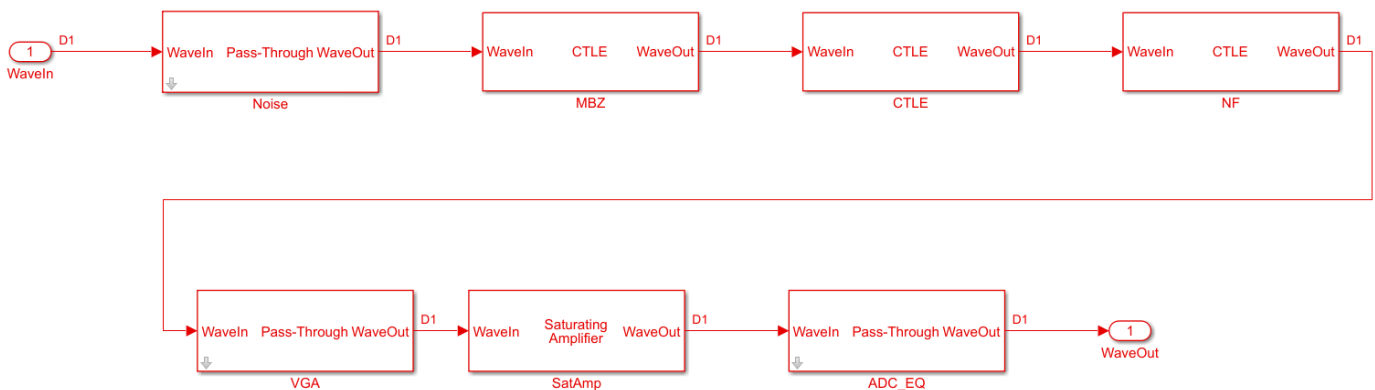
ADC Subsystem

The ADC subsystem is composed of custom ADC, Demux, RxFFE, DFE, Phase Detector, Loop Filter, and VCO System object™ blocks. Additionally, the IBIS-Bridge and IBIS-AMI clock_times blocks facilitate the conversion of the model to an IBIS-AMI model.
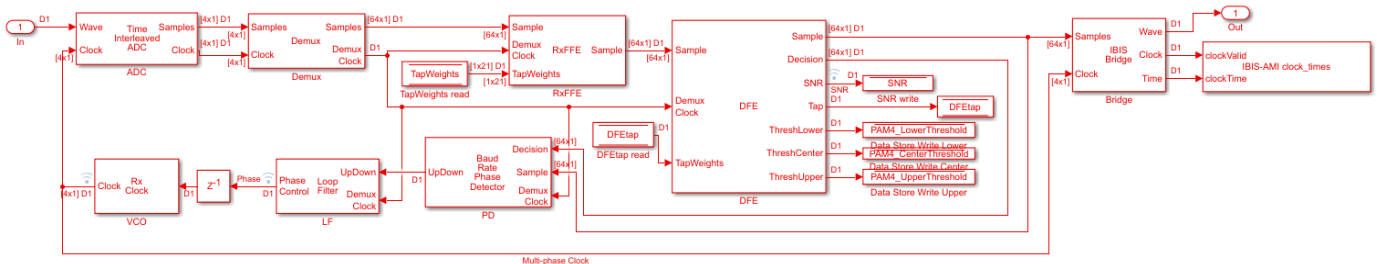
A time-interleaved ADC is utilized to reduce the maximum speed and latency requirements of a full-rate ADC. The diagram below shows how four time-interleaved ADCs can take turns sampling the data signal.
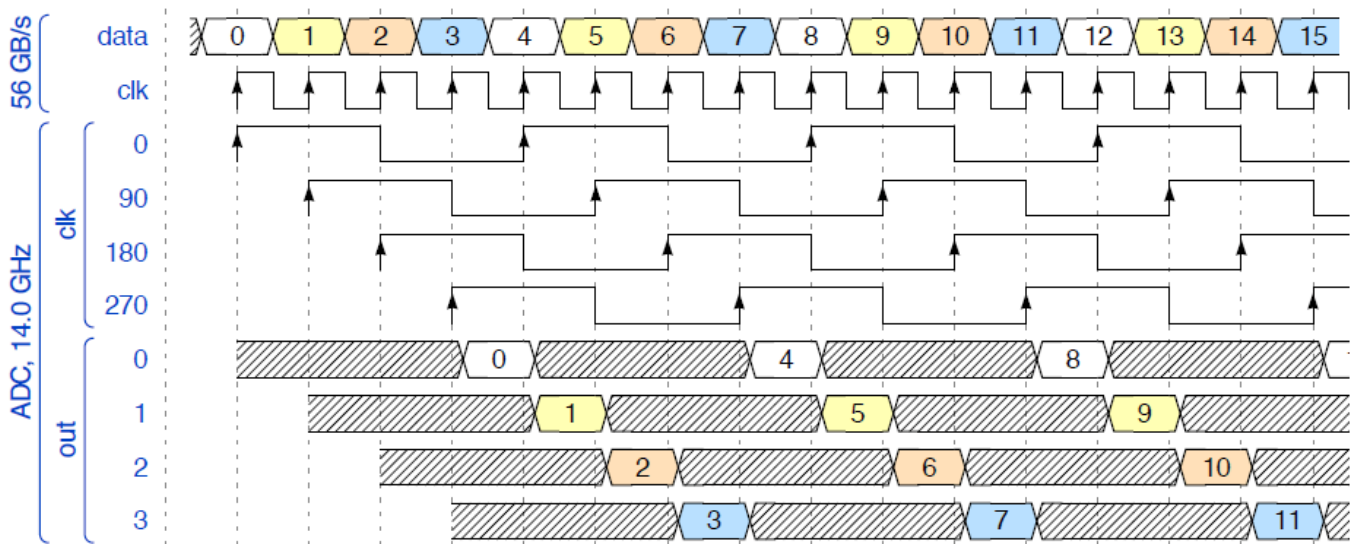
The sampled signals are demuxed or framed to reduce the signaling rate before processing with the DSP equalization. The diagram below illustrates how a demux width of 8 frames slows down and parallelizes the data. The model itself is parameterized to use a demux of 64 but 8 are shown below for illustration purposes only.
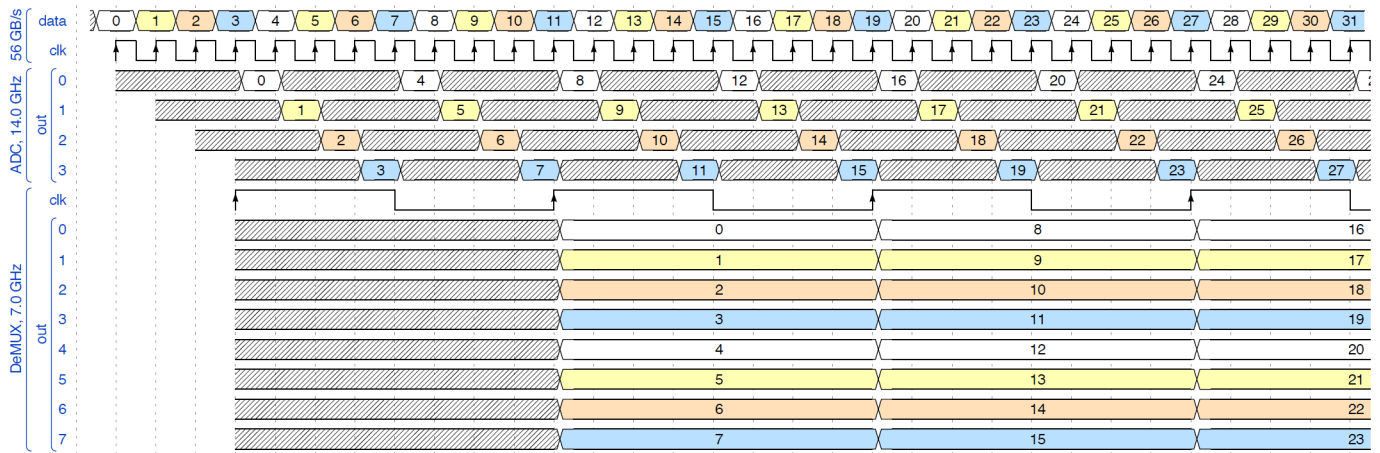
Next a 21 tap FFE is applied to the parallelized signal and is then followed by the single tap DFE and data decision. The baud-rate phase detector utilizes a type A Mueller-Muller phase detector that aims to balance the ISI of the first pre- and post-cursor samples as illustrated below:
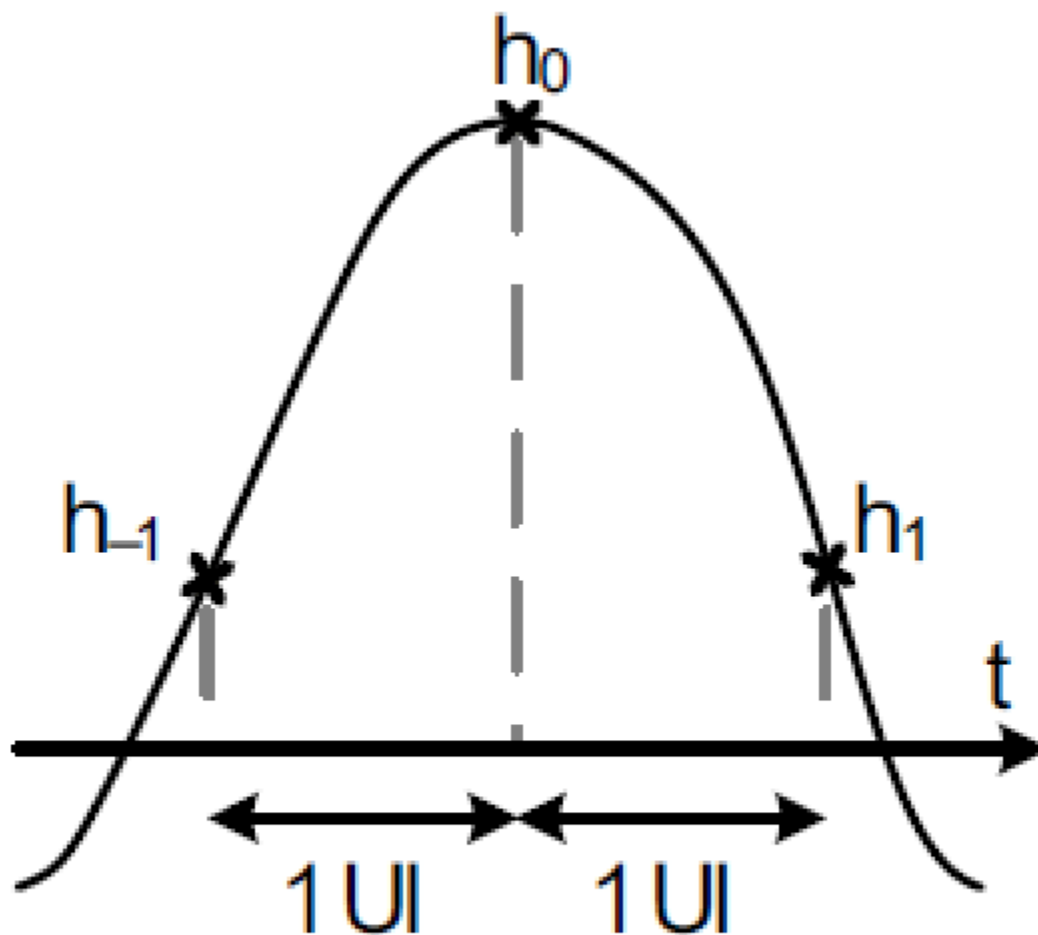
Type A Mueller-Müller PD

The phase detector output is processed with the loop filter which in turn drives the voltage controlled oscillator (VCO) or the heart-beat of the system. This VCO drives the other blocks and closes the system loop.

SNR Calculation

For ADC-based serdes the eye diagram is not as information rich as it is for analog SerDes. Instead signal-to-noise (SNR) calculations and vertical eye slices are more useful insight into the system performance.

The DFE block calculates the SNR as follows:

$$SNR = 10 \log_{10}\left(\frac{\mu^2}{\sigma^2}\right)$$

where μ is the signal strength and σ is the noise strength. If $y[i]$ represents the discrete equalized sampled voltages, then for NRZ the signal and noise strengths are defined as,

$$\mu = \frac{1}{N} \sum_{i=1}^{N} |y[i]|$$

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (|y[i]| - \mu)^2$$

For PAM4, the signal and noise are defined in terms of the middle and outer sampled symbol voltages.

$$\mu_1 = \frac{1}{N_1} \sum_{i=1}^{N_1} |y[i]_{\text{middle}}|$$

$$\mu_2 = \frac{1}{N_2} \sum_{i=1}^{N_2} |y[i]_{\text{outer}}|$$

$$\mu^2 = \frac{(\mu_1^2 + \mu_2^2)}{2}$$

$$\eta_1[i] = y[i]_{\text{middle}} - \mu_1$$

$$\eta_2[i] = y[i]_{\text{outer}} - \mu_2$$

$$\sigma^2 = \frac{1}{N_1 + N_2} \sum_{i} \left[(\eta_1[i])^2 + (\eta_2[i])^2 \right]$$

The vertical eye slice shows the clustering of the four PAM4 symbols.
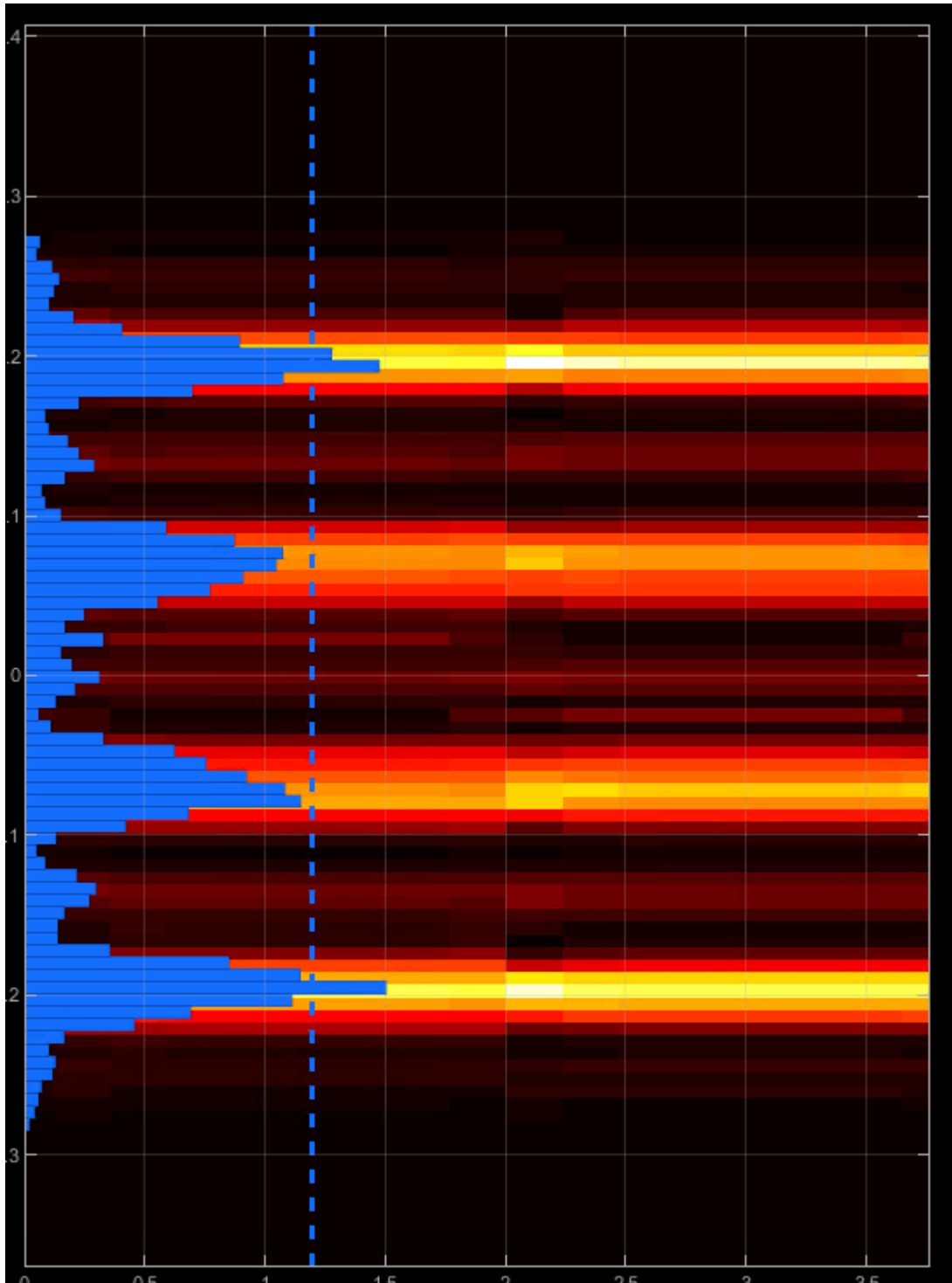
FFE and DFE Equalization Adaptation

Very similar to the process used in the "ADC IBIS-AMI Model Based on COM" on page 7-111 example, the FFE and DFE tap adaptation is performed in AMI Init at time 0 with impulse-response based

analysis in the initialize subsystem. These optimized tap values are then passed to the Simulink equalization blocks and are utilized throughout the simulation.

Impact of Timing Mismatch

If the phases of the several clocks that drive the time-interleaved ADCs are not equally spaced from each other, then system performance degradation occurs. While much of this timing mismatch can be calibrated out, it is important to understand the performance impact of this impairment. The RxClock or the VCO block has a parameter called the **Max Timing Mismatch (UI)**. This parameter injects a phase offset between the first 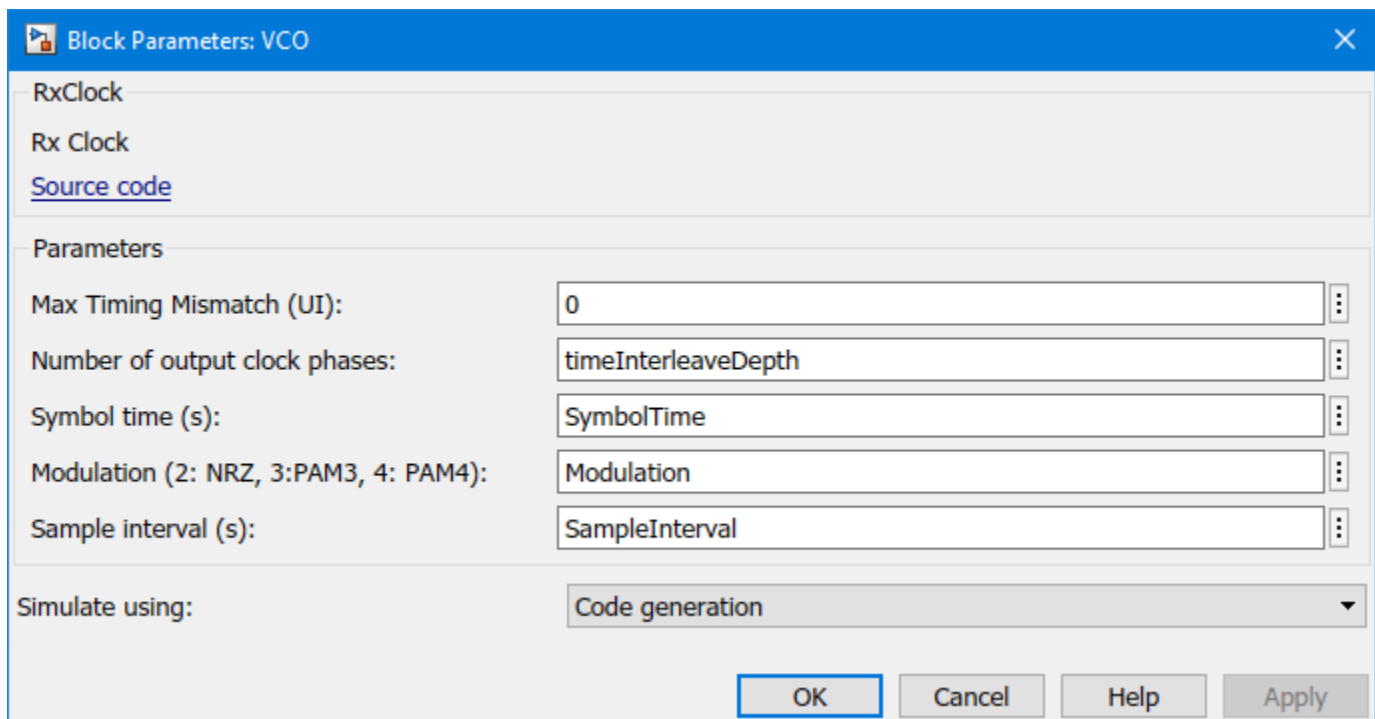and second clocks. While this is a simplistic model of actual system behavior, it is sufficient to illustrate the impact.

If you have not done so yet, open the Simulink model `ArchitecturalADCBasesSerDes.slx` attached with this example.

Run the model and observe the baseline behavior. When the simulation completes, the Simulation Data Inspector automatically loads up the logged signals.

Double click on the Rx Clock/VCO block and change the timing mismatch to be 0.04 and re-run the model. The resulting SNR for the two runs can be compared as follows. This illustrates how a small 4% timing mismatch can reduce the system performance by about 2.5 dB.

Generate ADC-Based SerDes IBIS-AMI Model

The final part of this example takes the customized ADC-based SerDes Simulink model and then generates an IBIS-AMI compliant model including model executables, IBIS, and AMI files.

The current IBIS AMI standard does not have native support for ADC-based SerDes. The current standard is written for slicer-based SerDes, which contain a signal node wherein the equalized signal waveform is observed. In a slicer-based SerDes this node exists inside the DFE, right before the decision sampler. A continuous analog waveform is observable at that node, which includes the effect of all the upstream equalizers (such as CTLE) and the equalization due to DFE, as tap weighted and fed back prior decisions. Such a summing node does not exist in an ADC-based SerDes, due to the ADC in the system. In a real ADC-based SerDes system the ADC proves a vertical slice though the eye at the sampling instant. To emulate a virtual node, the IBIS-AMI Bridge block reassembles the discrete equalized samples according to the time interleave factor and the demux size. A single equalized sample is held constant for the entire IBIS-AMI waveform symbol time.

Export IBIS-AMI Models

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

*   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 5 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).
- Press the **Export** button to generate models in the Target directory.

Explore Further

The number of ADCs and the Demux width in the model is parameterized by the MATLAB workspace variables `timeInterleaveDepth` and `DemuxSize`. They are set in the model `PreLoadFcn` callback and can be changed to other positive integers as part of further exploration. The system objects in the ADC subsystem can be modified to explore many of the design tradeoff questions identified at the first of the example.

References

[1] S. Kiran, S. Cai, Y. Zhu, S. Hoyos and S. Palermo, "Digital Equalization With ADC-Based Receivers: Two Important Roles Played by Digital Signal Processingin Designing Analog-to-Digital-Converter-Based Wireline Communication Receivers," in IEEE Microwave Magazine, vol. 20, no. 5, pp. 62-79, May 2019, doi: 10.1109/MMM.2019.2898025.

# Architectural 100G Dual-Summing-Node-DFE PAM4 SerDes Receiver Model

This example demonstrates the use of an architecturally representative 100G dual-summing-node-DFE PAM4 SerDes receiver model using the library blocks from the SerDes Toolbox™ library and custom blocks. The more representative data path allows for a more accurate modeling of the adaptation of the clock phase, DFE tap weights and PAM4 threshold voltages. This example shows the necessity of managing the adaptive loop update factors to ensure that the numerous adaptive loops don't fight against each other.

Example Overview

Historically, SerDes have been designed using a single summing node for DFE tap feedback. Due to decreasing design margins, the increasing difficulty in meeting timing requirements, and difficulty of routing high-speed clocks, this architectural topology has fundamental limits. An alternative topology uses two summing nodes, rather than one, each to process every other incoming symbol to ease timing margins and to reduce clock speeds at the expense of circuit complexity and area [1]. This example of architecturally representative dual-summing-node-DFE PAM4 receiver model provides access to realize differences between summing node offsets, bandwidths and timing. The use of modular blocks for the DFE, adaptation engine, phase detector, loop filter and VCO allows for the collaboration between multiple teams and with the advantages of utilizing block interfaces for correlation and verification activities. The representative data path allows for a representative adaptation engine model and the exploration between interactions of the adaptation loops for the clock phase, DFE tap weights and PAM4 threshold voltages.

Model Overview

Open the model by opening `DualSummingNodeSerdes.slx` file.
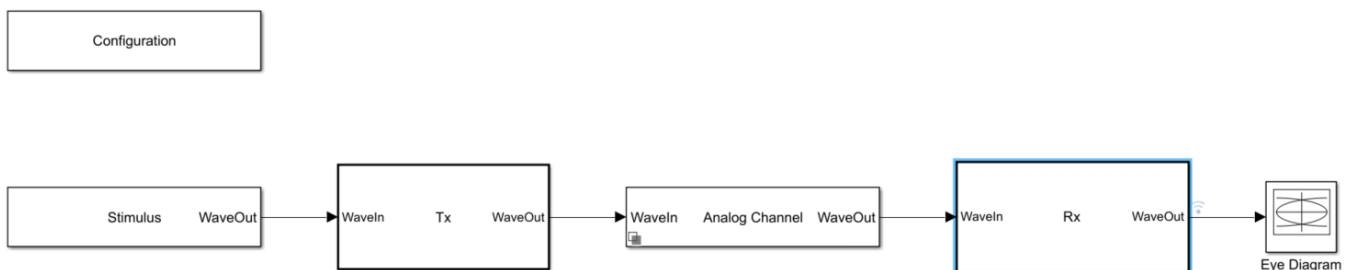
The receiver (Rx) model contains the same analog front end, CTLE, VGA and memory-less non-linearity blocks, as the COM reference receiver detailed in the "ADC IBIS-AMI Model Based on COM" on page 7-111 example and "Architectural 112G PAM4 ADC-Based SerDes Model" on page 7-126 example.
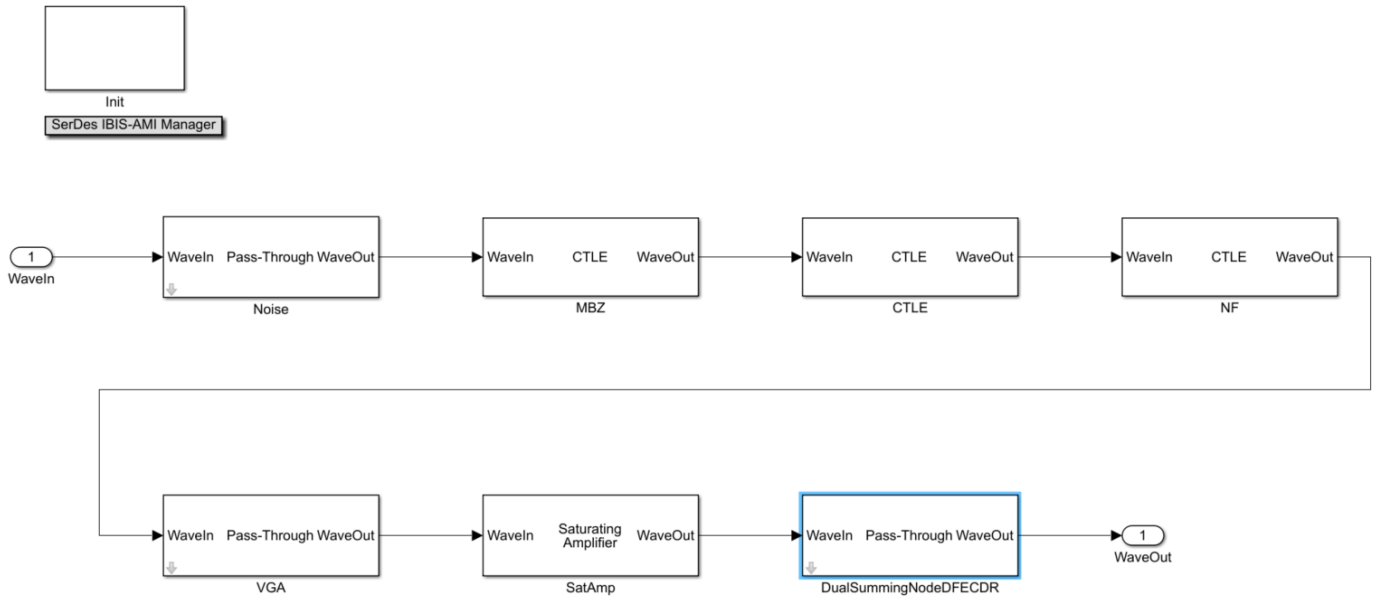
The last block in the receiver contains the dual-summing-node DFE and the clock recovery units.
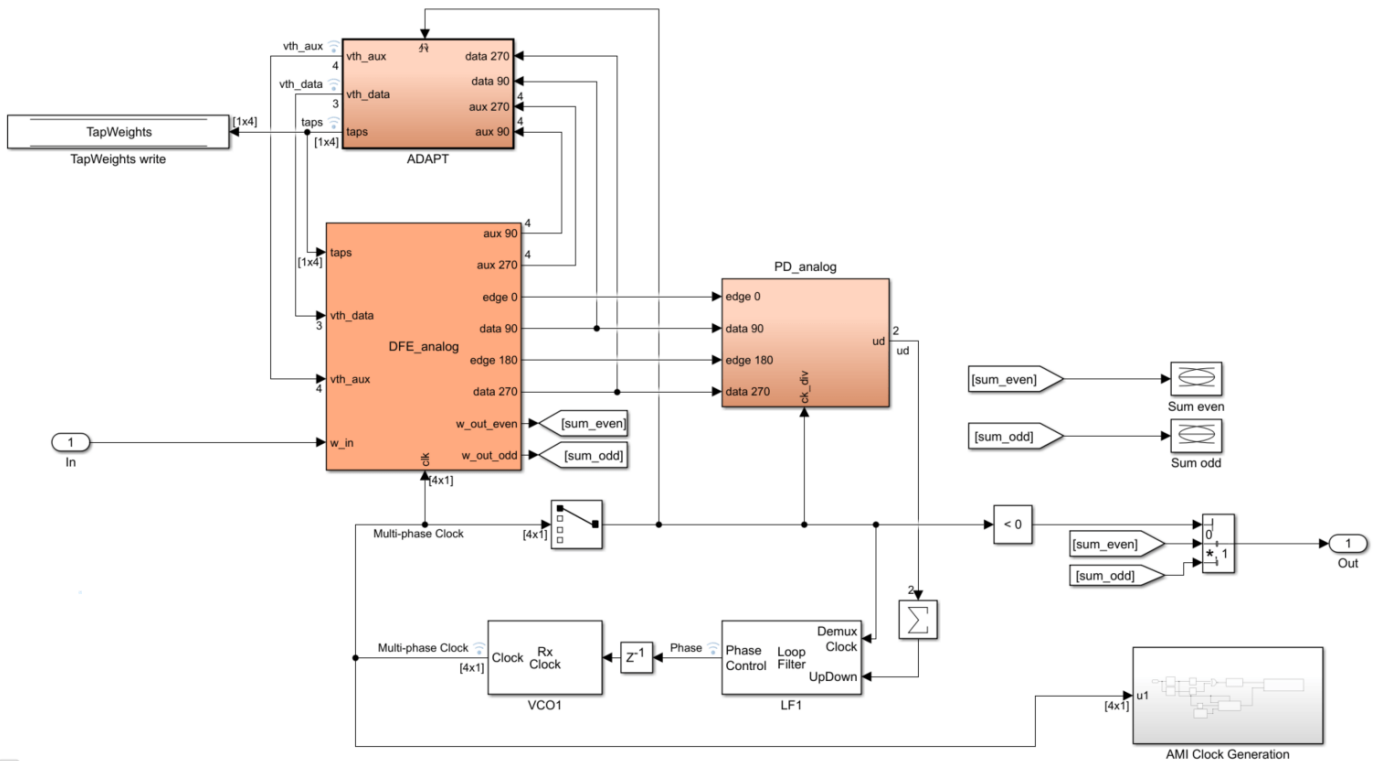
Dual Summing Node Description

The dual summing node DFE architecture relaxes the timing requirements by breaking the data path into two branches ("odd" and "even" in this example) that sample and apply digital-feedback-equalization (DFE) to every other data symbol. The partially equalized input signal, w_in, is fed to the

even and odd summing nodes. The previous data decisions are appropriately weighted by the DFE tap weights and subtracted from the input signal to generate the even and odd summing node outputs, "w_out_even" and "w_out_odd", to further equalize the signal.

α₊₄

α₊₂

edge 0

0°

w_out_even

data 90

90°

aux 90

90°

/4

α₊₁

α₊₃

w_in

α₊₃

α₊₁

edge 180

180°

w_out_odd

data 270

270°

aux 270

270°

/4

α₊₂

α₊₄
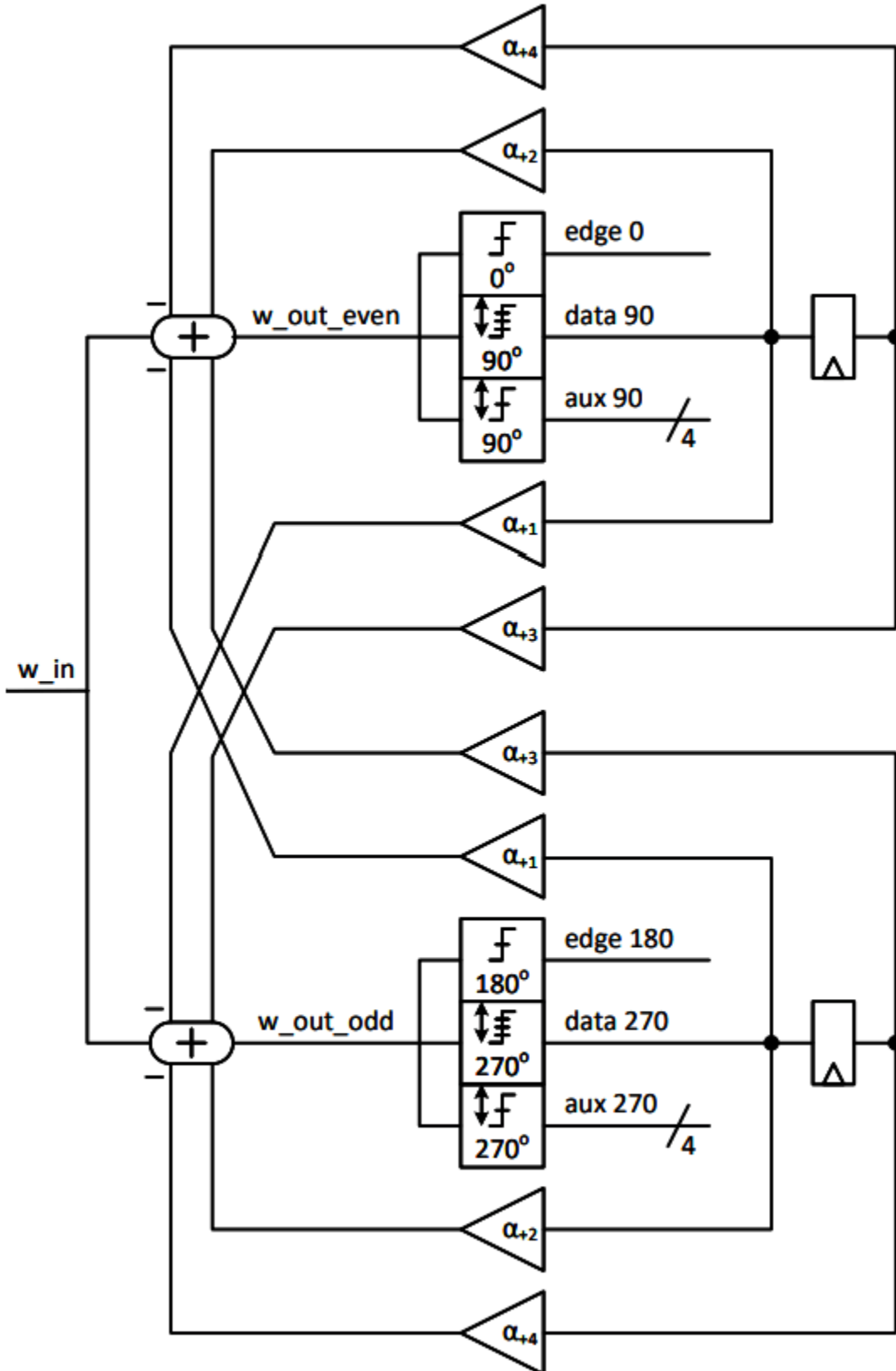
The odd and even branches have their own set of PAM4 data samplers, auxiliary samplers (used to determine the PAM4 data sampler threshold voltages and DFE tap adaptation) and edge samplers (used in the Bang-Bang phase detection and clock recovery). These binary samplers can determine whether the input signal is above or below the sampler's threshold voltage. The auxiliary samplers levels (AUX +1, AUX +1/3, AUX -1/3 and AUX -1) or signal constellation levels will adapt to the observed signal constellation voltages depending on the observed samples. The ideal odd (yellow and red X's) and even (green and black X's) sampler positions (in time and voltage) of a PAM4 eye diagram are:
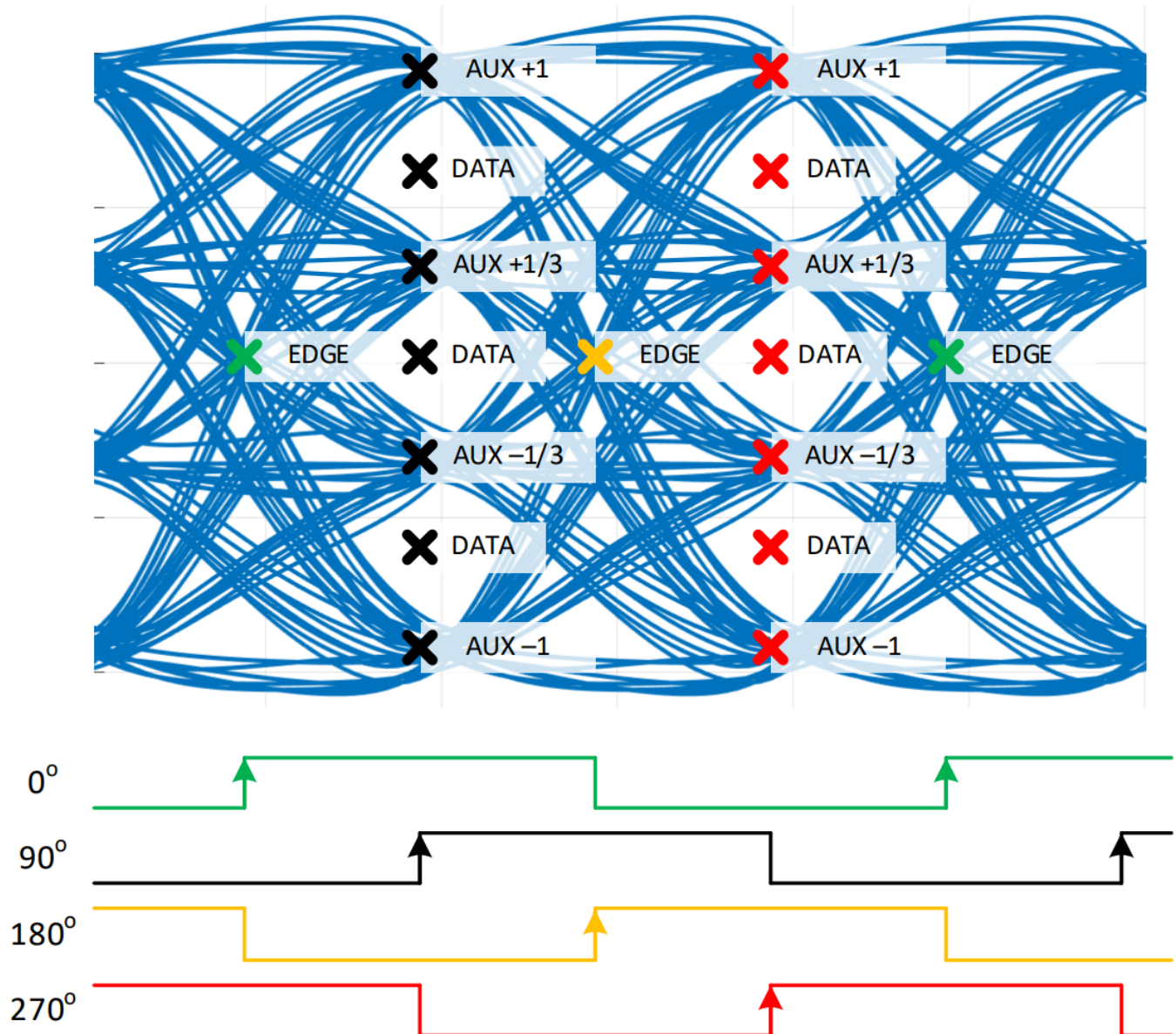
Observe the two eyes in each eye diagram of the odd and even summing node waveform outputs. Note that one eye is sharp and one is blurry illustrating that every other eye in each branch is a "don't care" state and allows for the relaxed timing constraint. The IBIS-AMI standard assumes a

single-summing node model so to make the dual-summing node model IBIS-AMI compliant, the odd and even waveforms are mux'ed together to create a virtual single-node output waveform.

Even eye

Odd eye

Mux'ed even and odd eye for IBIS-AMI
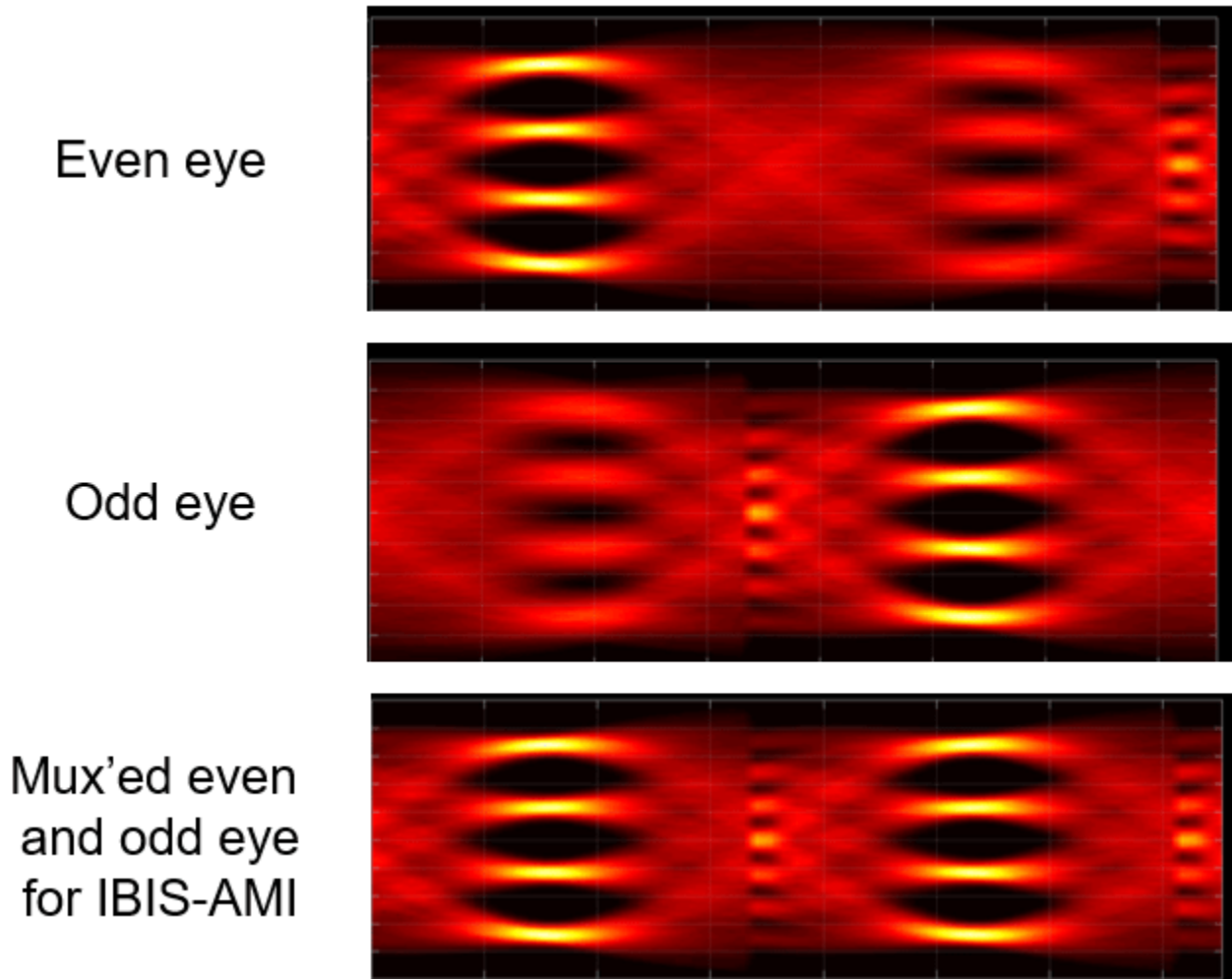
Adaptive Behavior

With the data path accurately represented, the adaptive behavior of the system can be modeled in detail and utilized for in-depth studies of loop dynamics. The model contains three adaptive loops, 1) clock phase recovery, 2) voltage reference threshold estimation for the samplers, and 3) DFE tap weights. It is critical that the loops converge in the order of clock phase, then voltage references, then DFE taps for optimal performance as shown. The common bang-bang clock phase detector is utilized to determine if the edge samples are early or late. This signal is accumulated and smoothed by the loop filter which in turn drives the voltage-controlled oscillator (VCO), providing the half-rate clocks throughout the model.
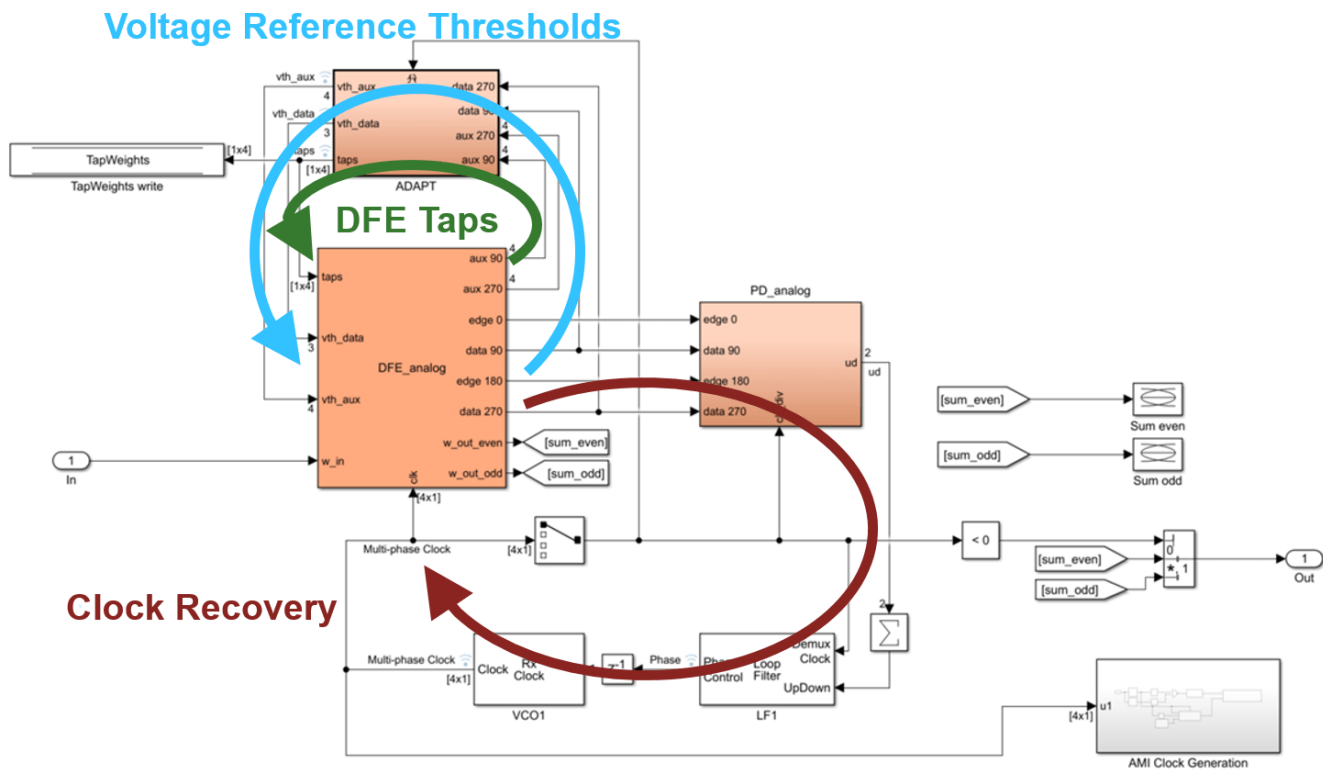
Both the DFE tap weight and reference voltage thresholds are estimated with the sign-sign least-mean-square (SS-LMS) algorithm. An LMS algorithm works by observing an input signal and an error signal. By assuming that the system noise is a combination of both random noise and systematic noise, the LMS algorithm searches for this correlated systematic noise between the input and error signal. For the DFE tap estimation, the systematic noise is the channel's residual inter-symbol-interference (ISI) due to loss and reflections and for the reference voltage threshold estimation, the systematic noise is the offset between data and auxiliary samplers. These adaptive filters continually update their weight estimates proportionally to the detected correlation. The sign-sign LMS algorithm is utilized in this model since the output of the signal samplers is binary.

As the DFE tap weight adaptation is simplest to explain it is addressed first, followed by the voltage threshold adaptation. For clock recovery adaptation, see "Model Clock Recovery Loops in SerDes Toolbox" on page 4-52.

DFE Tap Weight Adaptation

The 4 tap DFE estimates and compensates for the residual channel ISI. This architectural model is unique in that the LMS algorithm is implemented in both the analog domain, when the DFE taps are applied to the signal and the digital domain when the DFE taps are updated with the LMS Update block. The governing equation for the tap weight update is given below.

$$W_k[n] = W_k[n-1] + \mu \left(D[n-k] \cdot E_m[n] \cdot D[n] = = m \right)$$

Where

-   $W_k[n]$ is the tap weight at time sample index n for the kth tap weight (1, 2, 3 or 4).

- μ is the adaptation step-size.
- $D[n]$ is the data symbol value, having the values of -1, -1/3, 1/3 or 1 for PAM4.
- $E_m[n]$ is the sign of the error, -1 or 1.
- m is the signal constellation level, having the values of -1, -1/3, 1/3 or 1 for PAM4.

This portion of the model is shown below followed by the view of the 4 tap weights adapting over the course of a simulation.
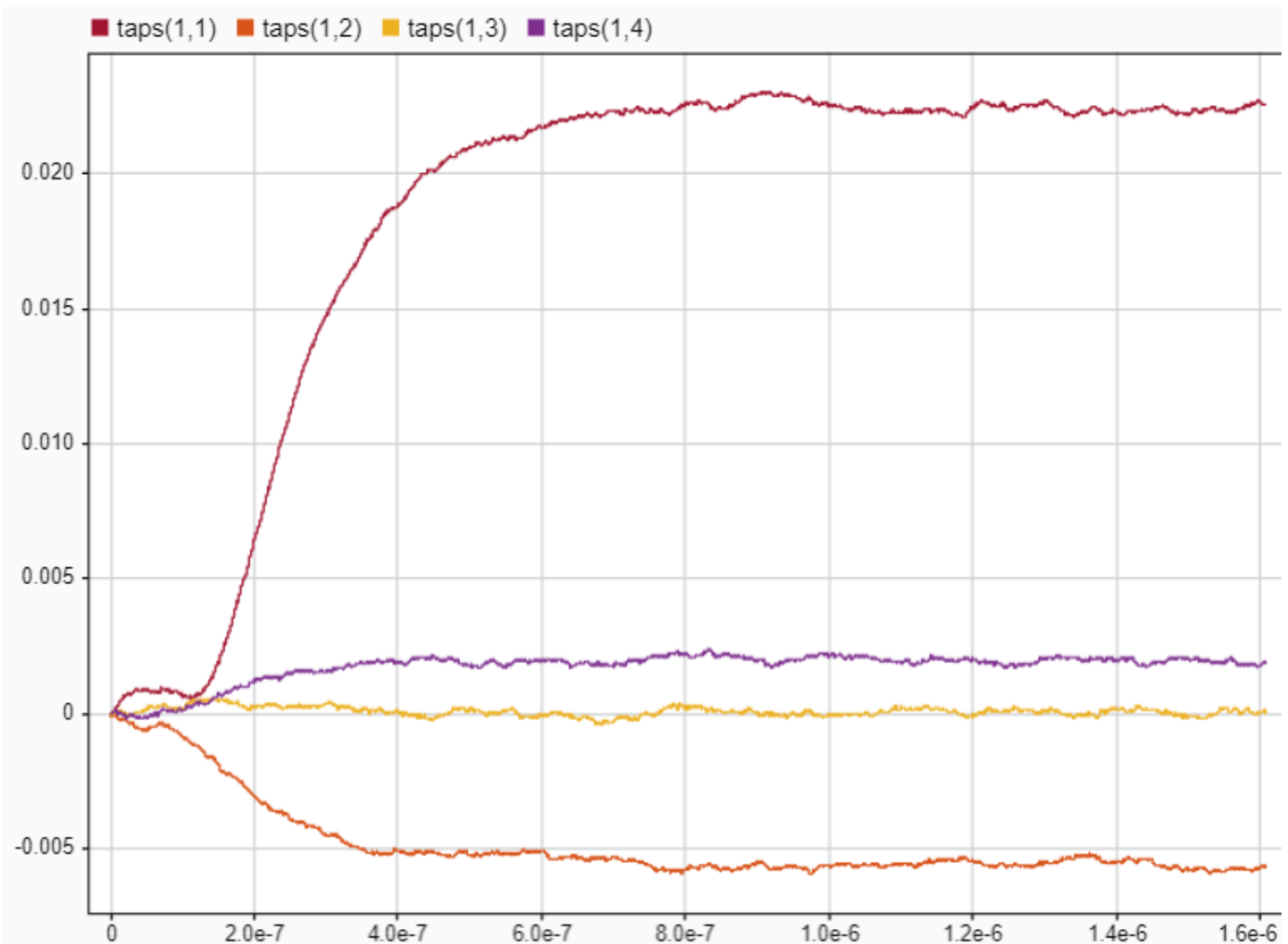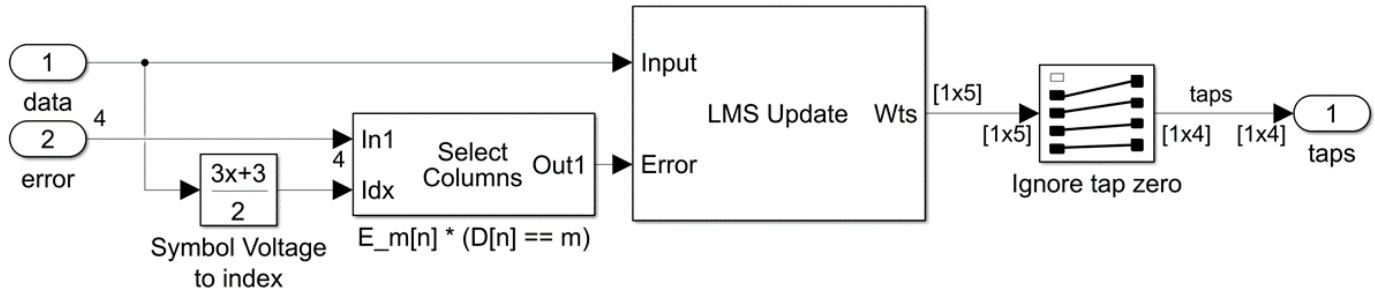
Voltage Threshold Adaptation

The signal constellation reference voltages for the auxiliary samplers are estimated with a sign-sign LMS algorithm and used to derive the PAM4 data sampler voltage thresholds. Conceptually, when the reference voltage for the auxiliary samplers are converged to the signal constellation values, the auxiliary samplers will on average return an equal number of logic high and low outputs. If the auxiliary reference voltage differs significantly from the signal constellation value, then the average of the output of the sampler will be biased. This correlated bias is used by the LMS algorithm to push the auxiliary reference voltage to converge to the signal constellation voltage. The average value of the adjacent signal constellation voltage estimates is used to derive the data sampler thresholds. The governing equation for the voltage threshold adaption is given below:

$$V_{\mathrm{AUX}_m}[n] = V_{\mathrm{AUX}_m}[n-1] + \mu \cdot (1) \cdot E_m[n] \cdot (D[n] = \, = m)$$

Where,

- $V_{\mathrm{AUX}_m}[n]$ is the auxiliary sample reference voltage for symbol m.
- $E_m[n]$ is the sign of the error (-1 or 1)
- μ is the adaption step size
- n is the sample index
- m is the constellation level

This portion of the model is shown below followed by an example of the startup adaptation of the voltage threshold of the auxiliary samplers ("vth_aux") and the voltage threshold of the data samplers ("vth_data").
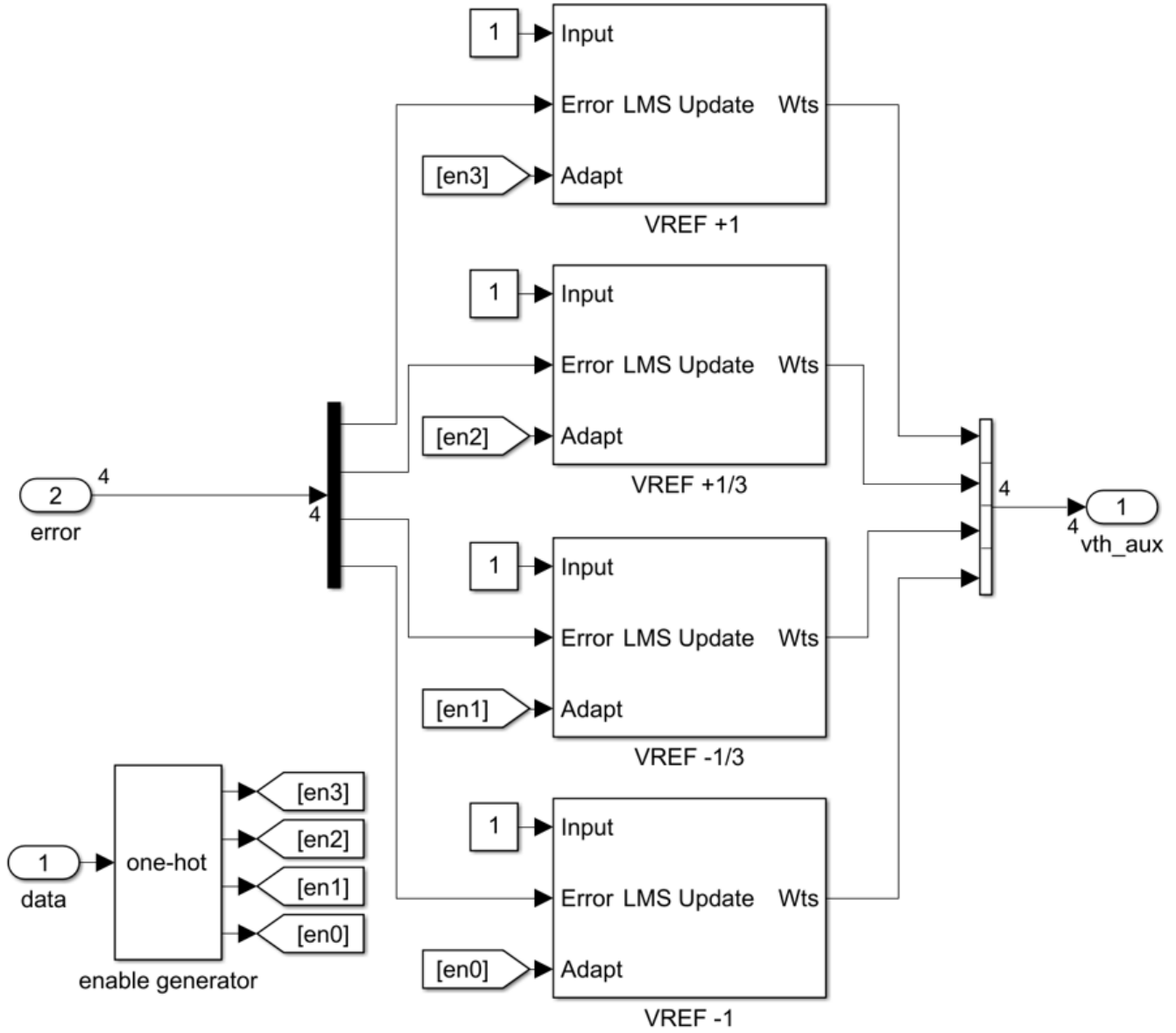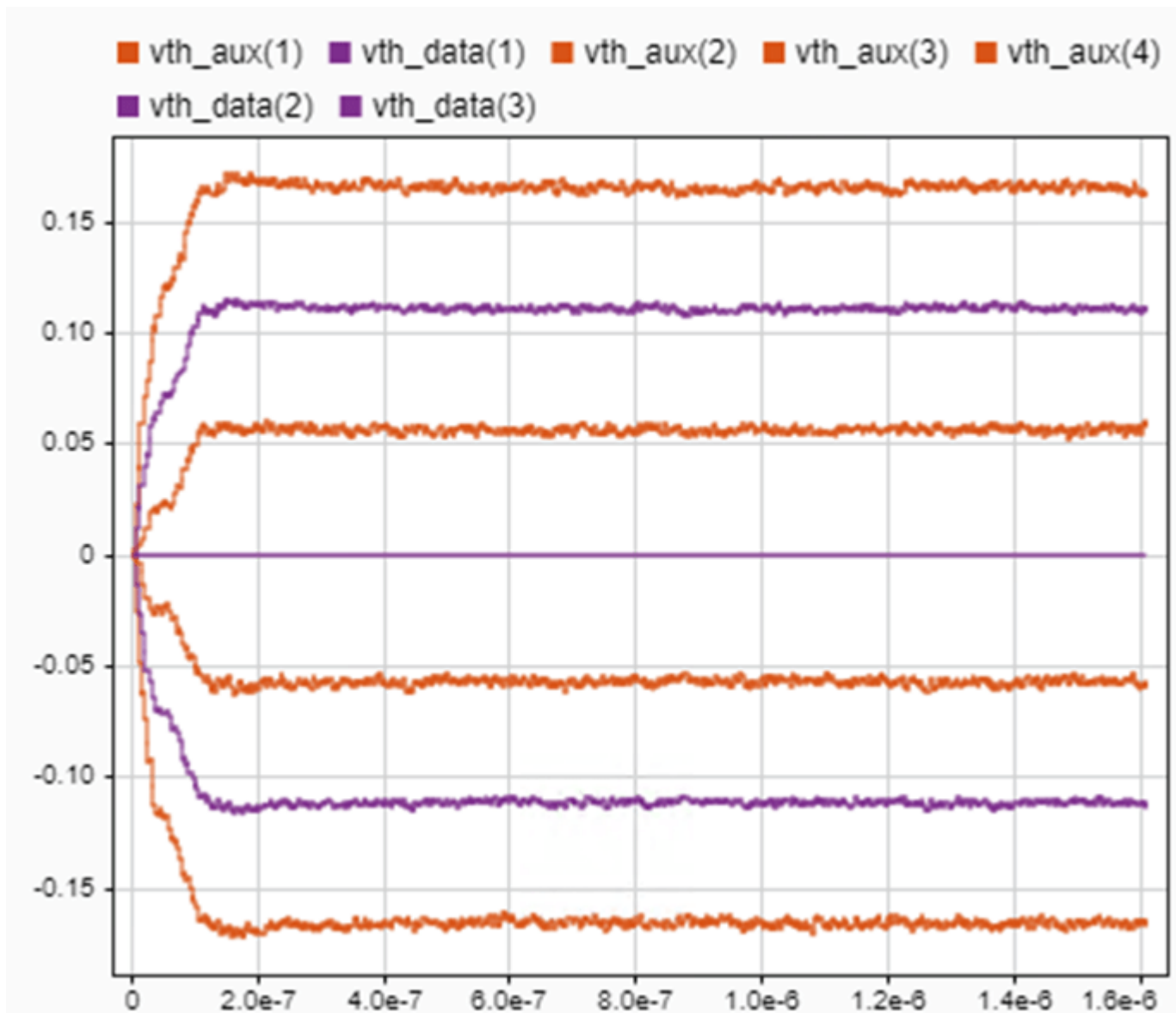
Experiment with Adaptation Loop Convergence

For optimal system performance, it is critical that the adaptive loops converge in the reliable sequence of clock phase first, voltage reference second and DFE tap weights third. The clock phase convergence time is controlled by the loop filter block parameters, up/down counter limit and step size for phase increment/decrement. The DFE tap weight adaptation is controlled by the MATLAB workspace variable, "`muTaps`", which is referenced as the adaptation update step size in the LMS Update block. The voltage threshold adaptation is controlled by the MATLAB workspace varible, "`muThresholds`" which is also referenced as the adaptation update step size in its LMS Update block. These μ terms control how much weight to give the adaptation update value versus the existing value. Large values of μ will result in faster convergence time but larger steady-state variation and smaller values of μ will take longer to converge but have smaller steady-state variation. The determination of what values of μ to use for each adaptation loop at the various stages of system operation (startup vs maintenance) is a topic of serious architectural investigation.

The base line behavior of the system with `muTaps` $= 2^{-18}$ and `muThresholds` $= 2^{-12}$ is:
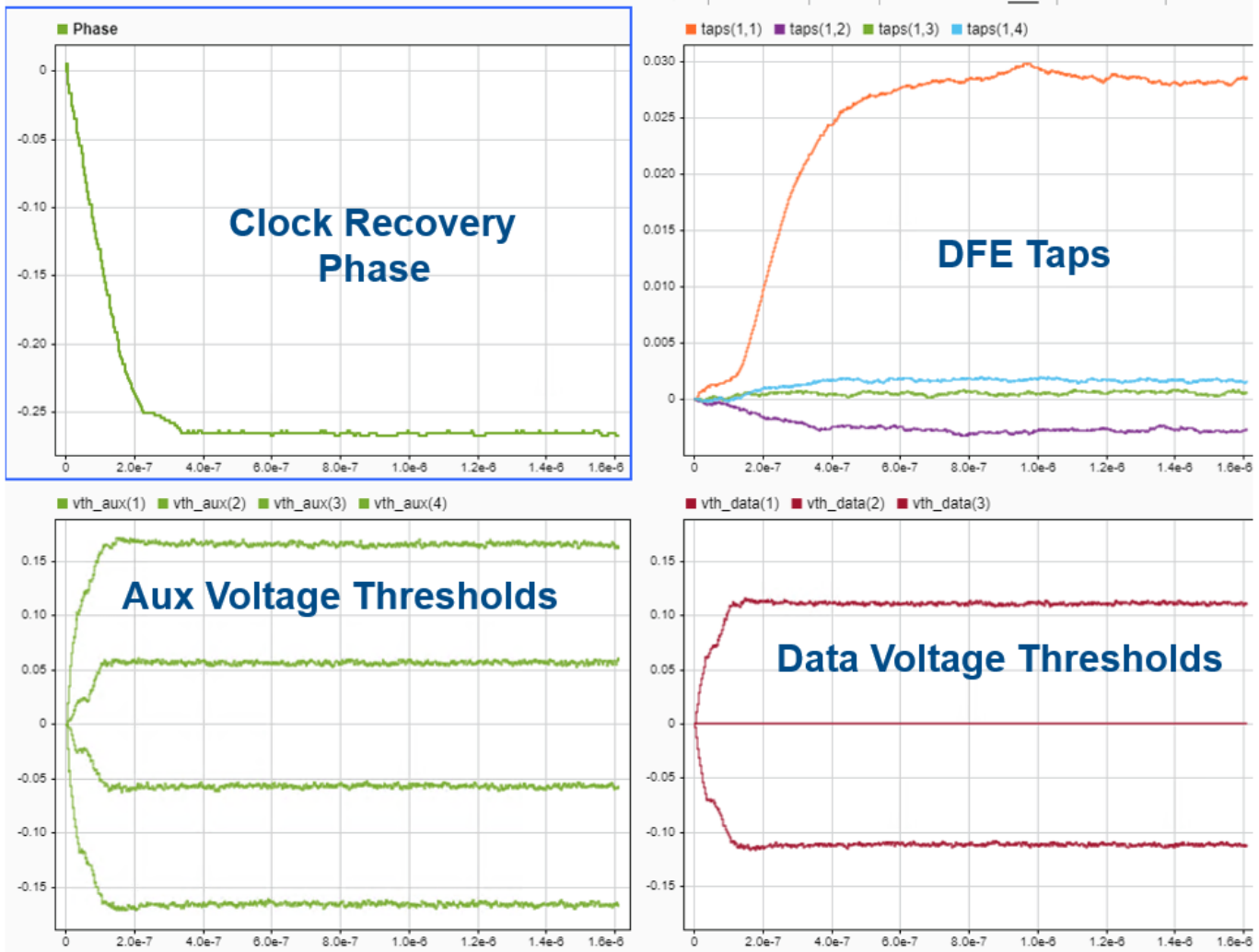
If the adaptive loops update is reversed where the DFE tap weights are allowed to converge before the voltage references then unstable behavior is observed. Below is the behavior of the system with `muTaps` $= 2^{-10}$ and `muThresholds` $= 2^{-16}$. The updated responses as shown in light blue. Observe how the DFE tap weights vary widely until the voltage thresholds converge. This underscores the need to correctly model and control the numerous adaptive loops in a system.
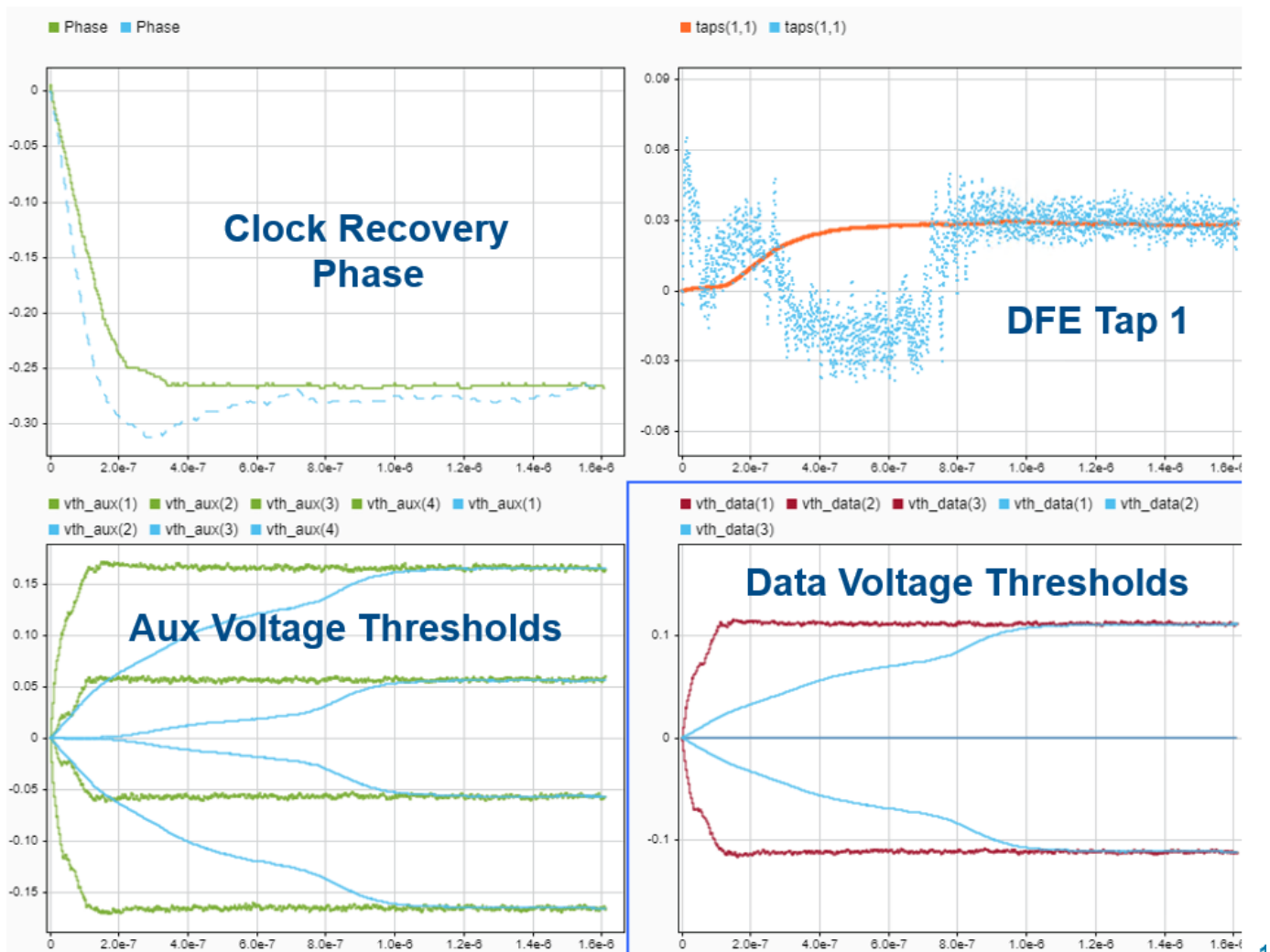
Export to IBIS-AMI and Serial Link Designer

As this model was created within the SerDes Toolbox IBIS-AMI workflow, it is ready to export to AMI and simulated in the **Serial Link Designer** app.

Conclusion

Due to decreasing design margins, the increasing difficulty in meeting timing requirements, and difficulty of routing high-speed clocks, a single summing node architectural topology has fundamental limits. This example shows an architecturally representative dual-summing-node-DFE PAM4 receiver model which provides access to realize differences between summing node offsets, bandwidths and timing. The use of modular blocks for the DFE, adaptation engine, phase detector, loop filter and VCO allows for the collaboration between multiple teams and with the advantages of utilizing block interfaces for correlation and verification activities. The representative data path allows for a representative adaptation engine model and the exploration between interactions of the adaptation loops for the clock phase, DFE tap weights and PAM4 threshold voltages.

References

[1] S. Ibrahim and B. Razavi, "Low-Power CMOS Equalizer Design for 20-Gb/s Systems," in *IEEE Journal of Solid-State Circuits*, vol. 46, no. 6, pp. 1321-1336, June 2011, doi: 10.1109/JSSC.2011.2134450.

CEI-112G-VSR Transmitter/Receiver IBIS-AMI Model

This example shows how to create IBIS-AMI models for an Optical Internetworking Forum (OIF) CEI-112G-VSR (PAM4) transmitter and receiver for Chip-to-Module applications. The generated models conform to the IBIS-AMI and OIF-CEI specifications.

The example is divided into three sections:

- Set up the target transmitter and receiver IBIS-AMI model architecture for CEI-112G-VSR in the **SerDes Designer** app
- Export the SerDes System to Simulink® for further customization
- Generate CEI-112G-VSR Tx/Rx IBIS-AMI Models

Part 1: CEI-112G-VSR Tx/Rx IBIS-AMI Model Architecture Setup

To setup and configure the `cei_112g_vsr` model in the **SerDes Designer** app, type the following in the MATLAB® command window :

```
serdesDesigner('cei_112g_vsr');
```
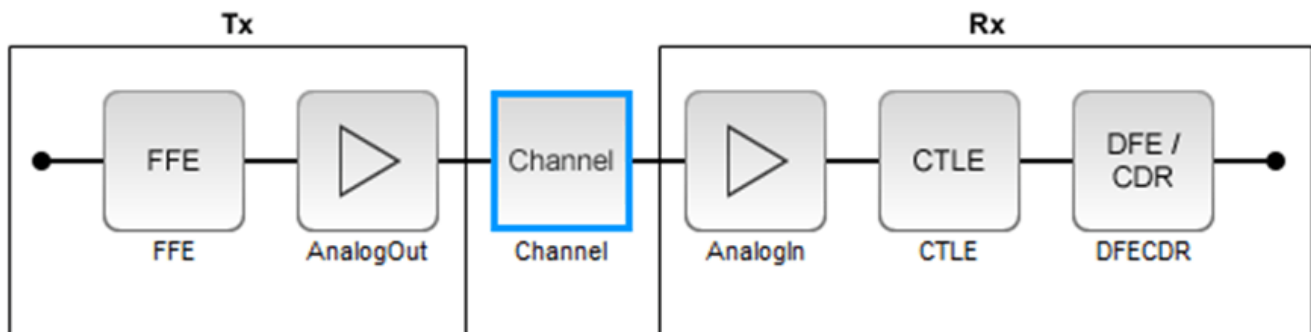

The app will open the model. The CEI-112G-VSR compliant transmitter uses a 4-tap feed forward equalizer (FFE) with 2 pre-taps and 1 post-tap. The receiver uses a continuous time linear equalizer (CTLE) with 14 pre-defined configurations, and decision feedback equalizer (DFE) with 4 taps. You can implement these as defined by the OIF specification for CEI-112G-VSR. Many of the values for this system are defined by, or determined from Table 24-14, "COM Parameter Values." The SerDes System is set up as follows:

Configuration Setup

- **Symbol Time** is set to `17.241 ps`, representing a symbol rate of 58 Giga-baud and bit rate of 112Gbps due to PAM4 modulation
- **Target BER** is set to `1e-6`
- **Modulation** is set to PAM4
- **Samples per Symbol** is set to `16`
- **Signaling** mode is set to `Differential`

Transmitter Model Setup

- The Tx FFE block is set up for 2 pre-taps, 1 main tap and 1 post-tap and is implemented using a row vector `[0 -0.2 0.7 -0.1]` where the main tap is specified by the largest value
- The Tx AnalogOut model is set up so that **Voltage** is `1.2 V`, **Rise time** is `6.869 ps`, **R** (single-ended output resistance) is 50 Ohms, and **C** (capacitance) is `0.217 pF`

Channel Model Setup

- **Channel loss** is set to `16 dB` (Appendix A, "Recommended Electrical Channel")
- **Differential impedance** is kept at the default of `100 Ohms`
- **Target Frequency** is set to the Nyquist frequency or `fb/2 = 58GHz/2 = 29GHz`

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 pF` (Table 24-14)
- The Rx CTLE block is set up for 14 configurations using the **Specification** setting `DC Gain and Peaking Gain` and **Mode** set to `adapt`
- The Rx DFE/CDR block is set up for 4 DFE taps. The **minimum values** are defined as a row vector `[0 -0.1 -0.1 -0.05]` and the **maximum values** are defined as a row vector `[0.4 0.15 0.15 0.1]` with mode set to `adapt`

Plot Statistical Results

You can use plots in SerDes Designer to visualize the results of the CEI-112G-VSR system.

Add the BER plot from **Add Plots** drop-down list in the toolstrip and observe the results.
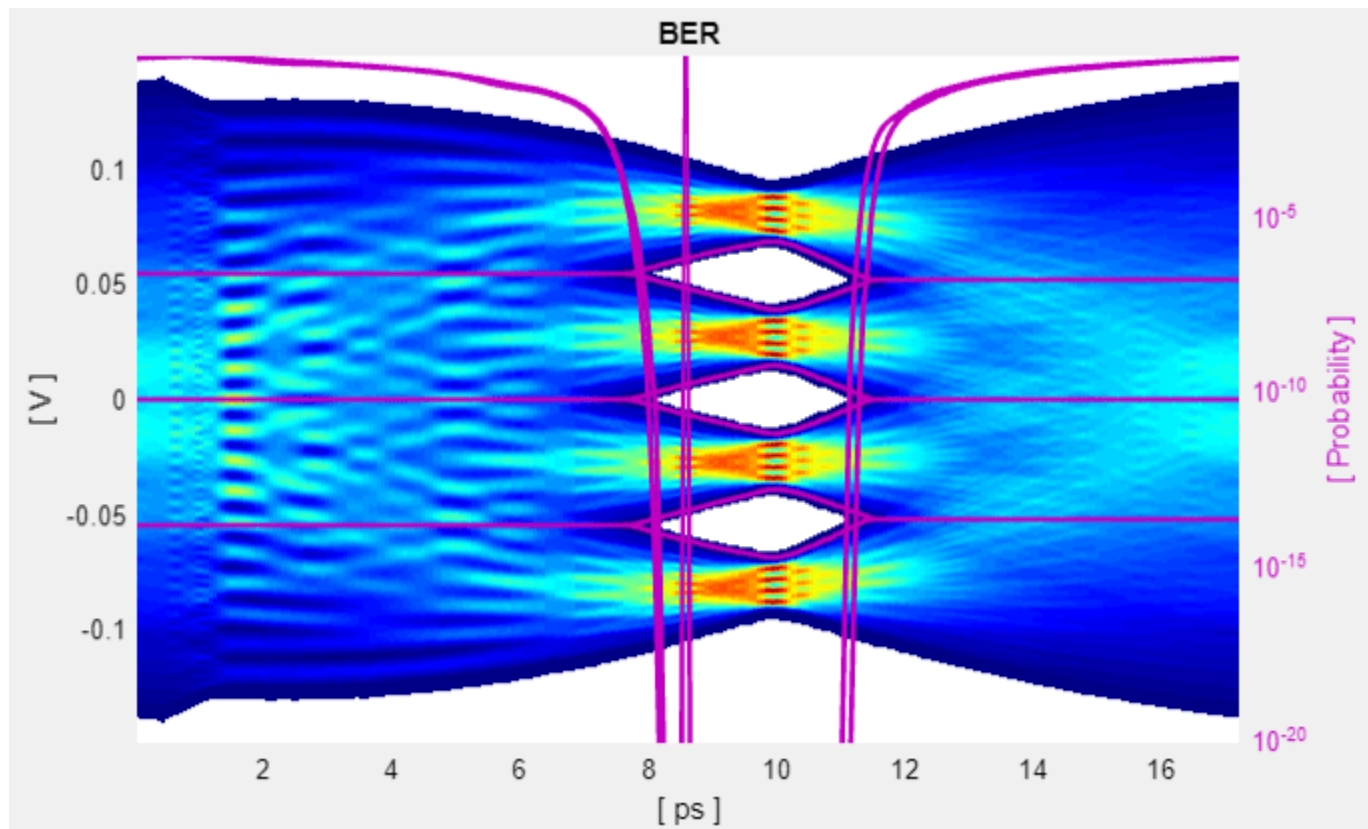
Add a Report from **Add Plots** drop-down list. The report provides data on Eye Heights and Widths, COM, VEC, and post-adaptation tap values for the CTLE and DFE.

	Name	Data
1	Eye Height Upper (V)	0.02079
2	Eye Height Center (V)	0.020812
3	Eye Height Lower (V)	0.02079
4	Eye Width Upper (ps)	3.4718
5	Eye Width Center (ps)	3.7423
6	Eye Width Lower (ps)	3.4718
7	Eye Area Upper (V*ps)	0.06117
8	Eye Area Center (V*ps)	0.061232
9	Eye Area Lower (V*ps)	0.06117
10	COM	4.2024
11	VEC	8.3137
12	Eye Linearity	1
13		
14	CTLE:ConfigSelect	10
15		
16	DFECDR:TapWeights (V)	[0 0.0169 0.0030 -0.0005]
17	DFECDR:Interior:PAM4Threshold (V)	0.0546048

In the above report you can see the CTLE adapted to value 10. Try changing the CTLE **Mode** to `fixed` and the **Configuration Select** to 9 and observe that the BER plot changes.
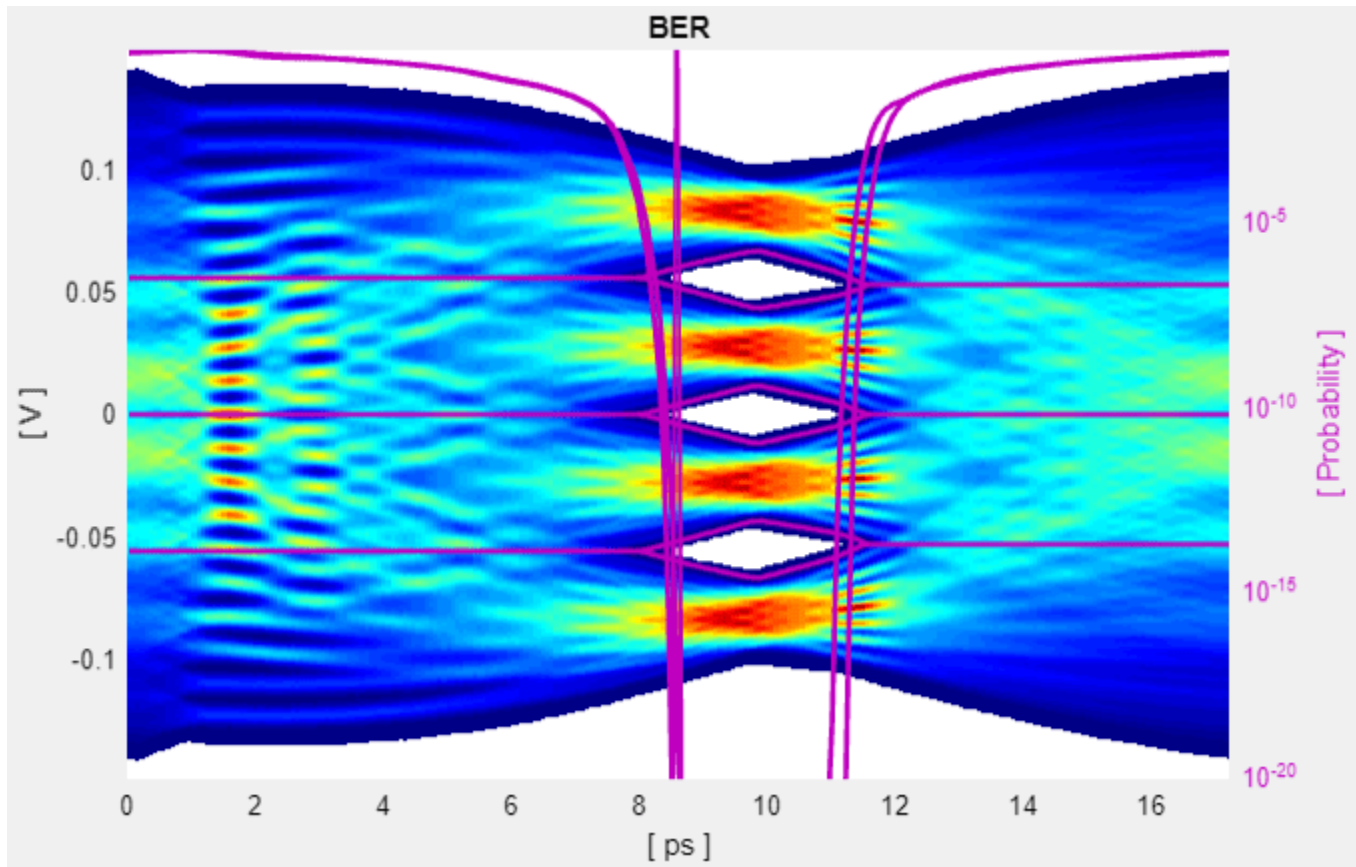
You can also see the values for Eye Heights and Widths change in the Report:

	Name	Data
1	Eye Height Upper (V)	0.015992
2	Eye Height Center (V)	0.015973
3	Eye Height Lower (V)	0.015992
4	Eye Width Upper (ps)	3.2059
5	Eye Width Center (ps)	3.4751
6	Eye Width Lower (ps)	3.2059
7	Eye Area Upper (V*ps)	0.045824
8	Eye Area Center (V*ps)	0.045766
9	Eye Area Lower (V*ps)	0.045824
10	COM	2.9572
11	VEC	10.7851
12	Eye Linearity	1
13		
14	CTLE:ConfigSelect	9
15		
16	DFECDR:TapWeights (V)	[0 0.0138 0.0016 -0.0013]
17	DFECDR:Interior:PAM4Threshold (V)	0.0560235

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

Part 2: 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-112G-VSR 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-112-VSR 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.
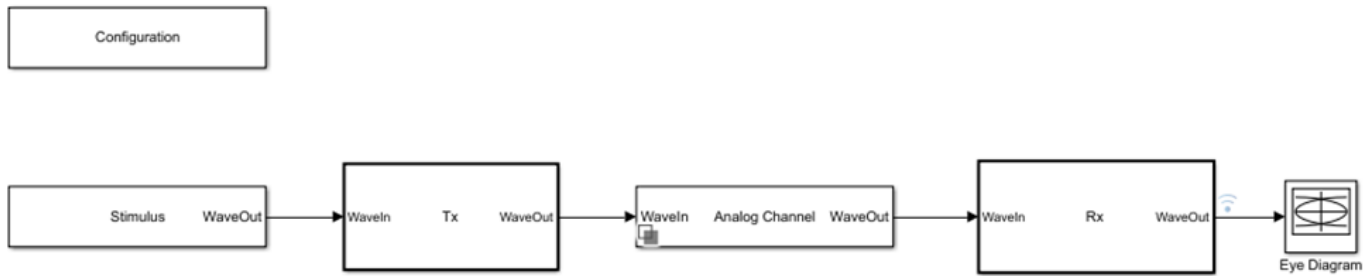
- Double click the Configuration block to open the Block Parameters dialog box. The parameter values for **Symbol time**, **Samples per symbol**, **Target BER**, **Modulation** and **Signaling** are carried over from the SerDes Designer app.

Double click the Stimulus block to open the Block Parameters dialog box. You can set the **Waveform Creation Method** to **Random Symbols** or other option to configure the stimulus pattern used during simulation. 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.

Configure Run-Time Parameters with IBIS-AMI Manager

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. They also contain run-time parameters that would be used by those EDA tools, and are also used by Simulink for this simulation.

Click on the **Export** tab and set the following:

AMI Model Settings – Tx:

- **Bits to ignore:** 4

AMI Model Settings – Rx:

- **Bits to ignore:** 20000 (Note: This must be fewer than the symbol count set in Stimulus block in order to display Time Domain Waveforms)

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.
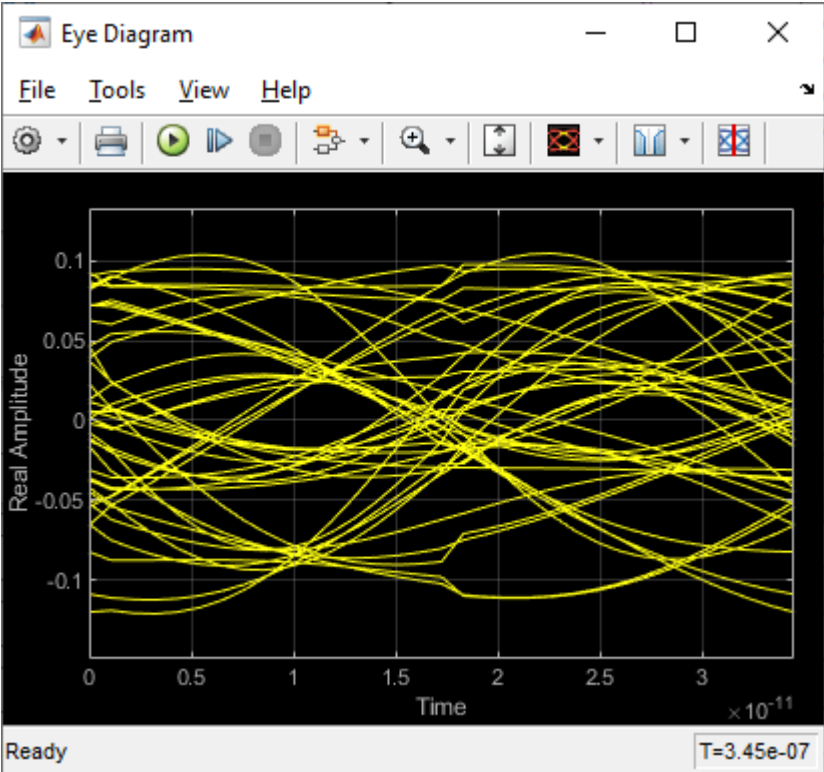
A second plot is displayed after simulation is completed that contains views of the statistical (Init) and time domain (GetWave) results, similar to what is available in the SerDes Designer App.

Init Statistical and Time Domain Analysis Results — □ ✕

File Edit View Insert Tools Desktop Window Help

Stat Analysis

Pulse Response

Unequalized
Equalized

[V] — [s] ×10⁻⁸

Waveform Derived from Pulse Response

Unequalized
Equalized

[V] — [s] ×10⁻⁹
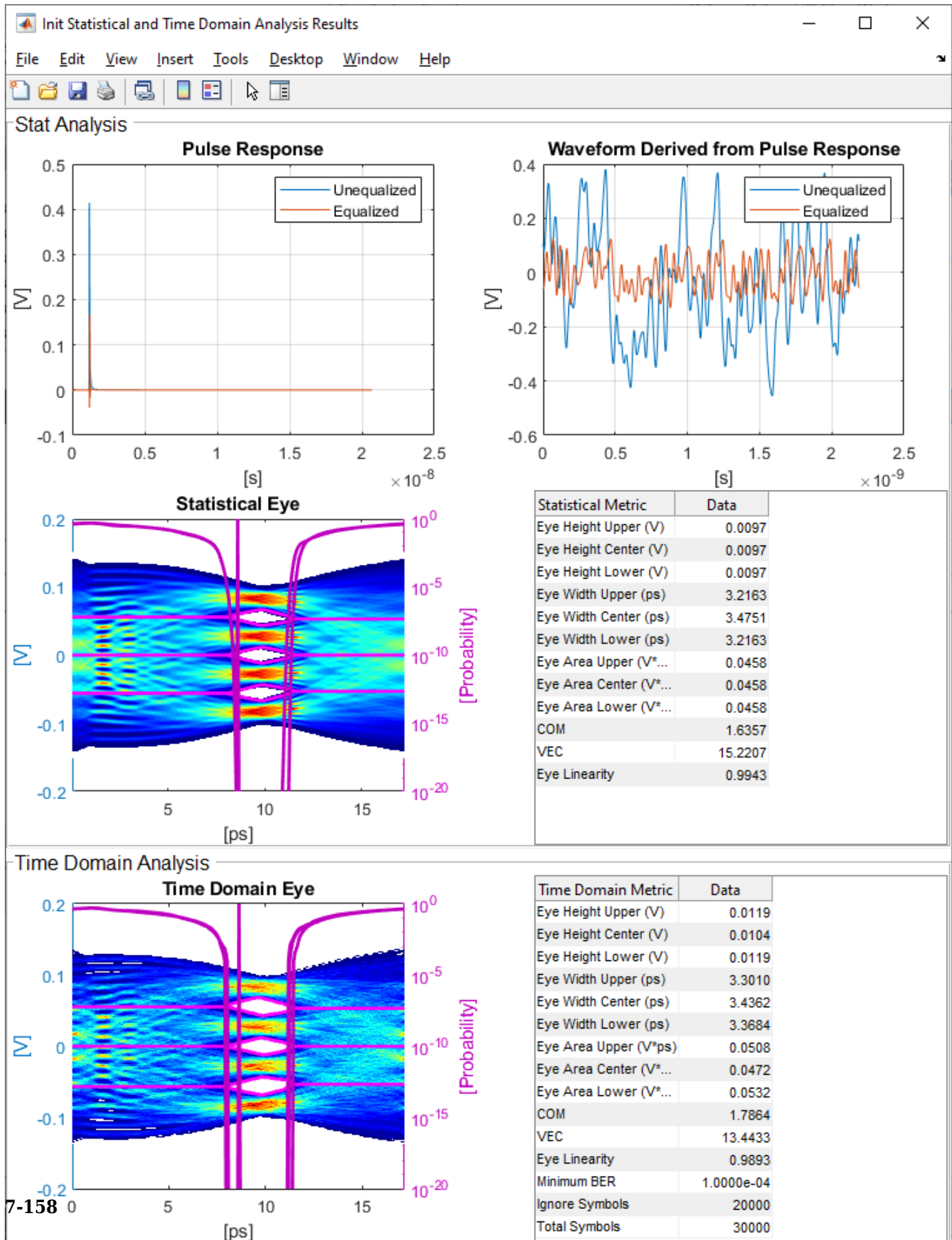
Statistical Eye

[V] — [ps] — [Probability]

Statistical Metric	Data
Eye Height Upper (V)	0.0097
Eye Height Center (V)	0.0097
Eye Height Lower (V)	0.0097
Eye Width Upper (ps)	3.2163
Eye Width Center (ps)	3.4751
Eye Width Lower (ps)	3.2163
Eye Area Upper (V*...	0.0458
Eye Area Center (V*...	0.0458
Eye Area Lower (V*...	0.0458
COM	1.6357
VEC	15.2207
Eye Linearity	0.9943

Time Domain Analysis

Time Domain Eye

[V] — [ps] — [Probability]

Time Domain Metric	Data
Eye Height Upper (V)	0.0119
Eye Height Center (V)	0.0104
Eye Height Lower (V)	0.0119
Eye Width Upper (ps)	3.3010
Eye Width Center (ps)	3.4362
Eye Width Lower (ps)	3.3684
Eye Area Upper (V*ps)	0.0508
Eye Area Center (V*...	0.0472
Eye Area Lower (V*...	0.0532
COM	1.7864
VEC	13.4433
Eye Linearity	0.9893
Minimum BER	1.0000e-04
Ignore Symbols	20000
Total Symbols	30000

Review 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
- Verify that **Mode** and **Tap Weights** are enabled

Review Rx CTLE Block

- Inside the Rx subsystem, double click the CTLE block to open the CTLE Block Parameters dialog box
- **Gain and Peaking Gain** 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

Review 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
- Verify check boxes are enabled for **Mode** and **Tap Weights**

Part 3: Generate CEI-112G-VSR Tx/Rx IBIS-AMI Models

Description of second code

The final part of this example takes the customized Simulink model, modifies the AMI parameters for CEI-112G-VSR, then generates compliant CEI-112G-VSR IBIS-AMI model files.

Open the **SerDes IBIS-AMI Manager** again. In the **Export** Tab you can set the model name for the Tx/Rx and the .ibs file. Verify that the **Model Configuration** is set to "Tx and Rx", that **AMI Model Settings** for Tx and Rx are set to "Dual Model," and that **File Creation Options** is set to "Both Tx and Rx." Configure the Tx Ignore Bits to 4, because the FFE has 4 taps. Configure Rx Ignore Bits to 20000 to ensure the DFE has sufficient symbols to converge during adaptation.

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 IBIS-AMI simulator. In the **AMI-Tx** and **AMI-Rx** tabs in the SerDes IBIS-AMI manager, verify the following are configured to match parameters defined or determined from Table 23-14:

Set Tx FFE Tap Ranges

- Highlight **TapWeights** and click on the Edit button to set:
- Verify **Tap Weights** is set to [0 0 0 0] because the FFE is defined to be 4 UI in length
- Set the **Min Range** to [0 -0.2 0.65 -0.1]
- Set the **Max Range** to [0.10 0 1.2 0]
- Click **OK** to save the changes.

Set Rx CTLE Tap Ranges

- Highlight **ConfigSelect** and click on the Edit button:
- Verify the **List values** are set to [0 1 2 3 4 5 6 7 8 9 10 11 12 13]
- Set the List_Tip values to describe each settings ["-0dB" "-1dB" "-2dB" "-3dB" "-4dB" "-5dB" "-6dB" "-7dB" "-8dB" "-9dB" "-10dB" "-11dB" "-12dB" "-13dB"]
- Click **OK** to save the changes.

Set Rx DFECDR Tap Ranges

- Highlight **TapWeight** and click the Edit button to set the following:
- Verify **Tap Weights** is set to [0 0 1 0] (because the DFE is defined to be 4 UI in length)
- Set the **Min Range** to [0 -0.1 -0.1 -0.05]
- Set the **Max Range** to [0.4 0.15 0.15 0.1]
- 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_112g_vsr_tx
- Update the **Rx model name** to cei_112g_vsr_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 20000 (20000 UI will allow sufficient time for the Rx DFE taps to converge and adapt 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_112g_vsr.ibs
- Press the **Export** button to generate models in your **Target directory**.

Test Generated IBIS-AMI Models

The CEI-112G-VSR transmitter and receiver IBIS-AMI models are now complete and ready to be operated in any industry standard AMI model simulator.

References

[1] IBIS (I/O Buffer Information Specification), Version 7.0, https://ibis.org/ver7.0/ver7_0.pdf.

[2] OIF Forum, CEI-112G-VSR, Contribution Number oif2017.346.22-1.pdf, https://www.oiforum.com.

[3] "Channel Operating Margin (COM)" (Signal Integrity Toolbox)

See Also

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

Related Examples

- "Managing AMI Parameters" on page 6-2
- "CEI 112G-VSR Compliance Kit" (Signal Integrity Toolbox)

USB4 V2 Transmitter/Receiver IBIS-AMI Model

This example shows how to implement USB4 V2 (80Gbps PAM3) Transmitter and Receiver architectures with SerDes Designer and generate IBIS-AMI models using the library blocks in SerDes Toolbox™. The generated models conform to the IBIS-AMI and USB4 V2 specifications.

Transmitter and Receiver Setup in SerDes Designer App

The first part of this example sets up the target transmitter and receiver model architecture for Captive Device transmitter compliance testing. The second part of this example will demonstrate the construction of a reference receiver for a router assembly based on USB4 V2 eCOM (enhanced Channel Operating Margin). You will see how to create these data path blocks for USB4 V2 with the SerDes Designer app. In the third part of this example, you will export the system to Simulink® for further customization. The fourth part concludes this example with exporting IBIS-AMI models for use in any EDA tool capable of IBIS simulation such as Serial Link Designer from Signal Integrity Toolbox™.

Reference Files for SerDes Designer in MATLAB

This example uses the SerDes Designer models `usb4p0v2_tx_comp.mat` and `usb4p0v2_rx_ref.mat` as example systems you may reference. The first is to test Transmitter compliance (from USB4 V2 Specification, section 3.2.5.1) and the second is to implement a reference Receiver based on eCOM (see "USB 80G PHY background.pdf" in the reference section). Type the following command in the MATLAB® command window to open the first model with SerDes Designer:

```
serdesDesigner('usb4p0v2_tx_comp');
```

Part 1: System Configuration for Captive Device

The SerDes Designer app will load a system representing USB4 V2 for Captive Device transmitter and receiver compliance. This includes a Transmitter having an FFE, a Captive Device Receiver with CTLE and CDR along with analog characteristics and an example channel as shown below.
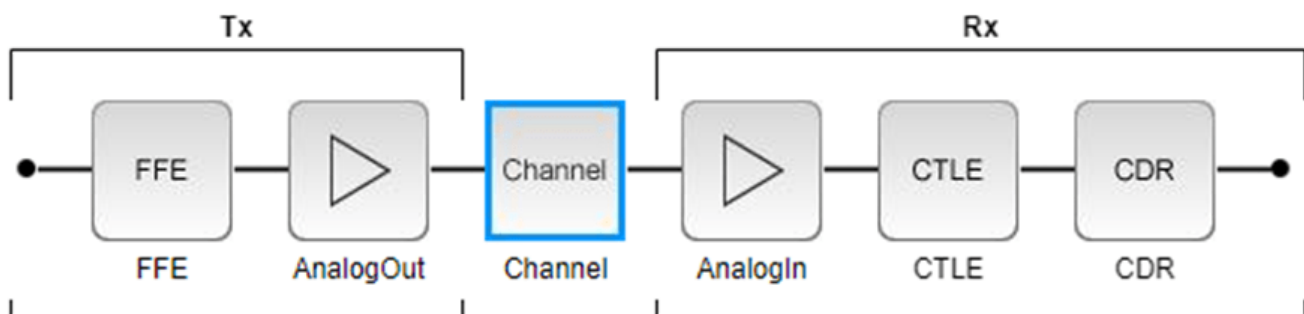
You can see from the SerDes Designer app options that the following are already configured based on the USB4 V2 specification (USB4 Version 2.0, October 2022 edition):

- **Symbol Time** is set to 39.0625ps (From 25.6 Giga-baud data rate given by Section 3.2.3, Table 3-22 and eCOM table)

- **Target BER** is set to 15.7e-09
- Note: Trit Error Ratio (TER) of 1e-08 is specified (Section 3.2.4.2) and you can find Bit Error Ratio as follows:
- $\mathrm{BER} = \mathrm{TER} * \left(\dfrac{\mathrm{Bits}}{\mathrm{Symbols}} \right) = 1.0e-08 * \dfrac{(11\ \mathrm{bits})}{(7\ \mathrm{trits})} = 1.57e-08$
- **Samples per Symbol** is set to 32 (Section 3.2.3, Table 3-23)
- **Modulation** is set to PAM3 (Section 3.2.3, Figure 3-29)
- **Signaling** is set to Differential (Section 3.2.3, Table 3-22)

Channel Model

You can click on the channel model block to observe the following configuration settings:

- **Channel loss** is set to 9.5dB (Section 3.2.5.3, Figure 3-40).
- **Differential impedance** is set to 85 Ohms (From 42.5 Ohms single-ended, Section 3.1.2.6, Compliance Boards).
- **Target Frequency** is set to the Nyquist frequency of 12.8 GHz (From 25.6 Giga-baud data rate given by Section 3.2.3, Table 3-22 and eCOM table).

Transmitter Model: Configure AnalogOut Block

You click on the Tx AnalogOut block to observe the following configuration settings:

- **Voltage** is 0.5 V (Section 3.2.3, Table 3-22)
- **Rise time** is 7.8125 ps (From Rt defined as 0.2*UI, Section 3.2.3, Table 3-22)
- **R** (single-ended output resistance) is 42.5 Ohms (Section 3.1.2.6, Compliance Boards)
- **C** (capacitance) is 0.1 pF (default value, not defined in specification).

Transmitter Model: Configure FFE Block

The Tx FFE block is set up for two pre-taps and one post-tap by including four tap weights, with presets as defined in the USB4 V2 specification, Table 3-24. This is done with the vector [0 0 1 0], where the main tap is specified by the largest value in the vector. You can click on the Tx FFE block to observe the following configuration settings:

- **FFE Tap** is [0 -0.05 0.9 -0.05] (Section 3.2.3.8, Table 3-24, Preset Number 5)
- **Normalize Taps** is Checked / Enabled (required per USB4 V2 specification, Table 3-24 notes)
- **Note:** This is a tap selected manually to represent reasonable performance given the configuration of the channel block above.

Receiver Model for Captive Device: Configure AnalogIn Block

You can click on the Rx AnalogIn block to observe the following configuration settings:

- **R** (single-ended input resistance) is 42.5 Ohms (Section 3.2.4.1, Table 3-26)
- **C** (capacitance) is 0.2 pF (default value, not defined in specification).

Receiver Model for Captive Device: Find CTLE Response

The Rx CTLE for TX compliance is defined according to the USB4 V2 specification, section 3.2.5.1:

$$H(s) = \frac{(s + A_{DC} \cdot \omega_{p1})}{(s + w_{p1})}$$

Where:

A_{DC} is the DC Gain

$$\omega_{p1} = 2 \cdot \pi \cdot 4e^9 \frac{\text{rad}}{\text{sec}}$$

The following code block shows how to find a GPZ matrix that represents the Rx CTLE used for Captive Device transmitter compliance. You will see how to visualize its response curve plots. After finding a GPZ matrix, the next step will be to add this configuration to the CTLE block that is part of the Receiver structure.

Note: The SerDes CTLE model requires that more poles be defined than zeros. In the code below, you will see we will pad an extra pole at high enough frequency to not disturb the signal but allow the model to have more poles than zeros and be stable.

```matlab
%Reference RX CTLE for Captive Device transmitter measurements, USB4 V2 specification, section 3
clear;
DCgain = 0:10;
ADC = 10.^(-DCgain/20);
p1 = 4e9; %Hz
wp1 = 2*pi*p1;
f = linspace(0,100e9,1001);
w = 2*pi*f;
s = 1j*w;
H = zeros(length(ADC),length(f));
for ii = 1:length(ADC)
    H(ii,:) = (s + ADC(ii)*wp1)./(s + wp1);
end
figure(1)
    subplot(211);
        semilogx(f,db(H))
        grid on
        xlabel('Hz')
        ylabel('dB')
        title('Reference CTLE from 3.2.5.1')
    subplot(212);
        semilogx(f,unwrap(angle(H)))
        grid on
        xlabel('Hz')
        ylabel('Radians')
```

Reference CTLE from 3.2.5.1
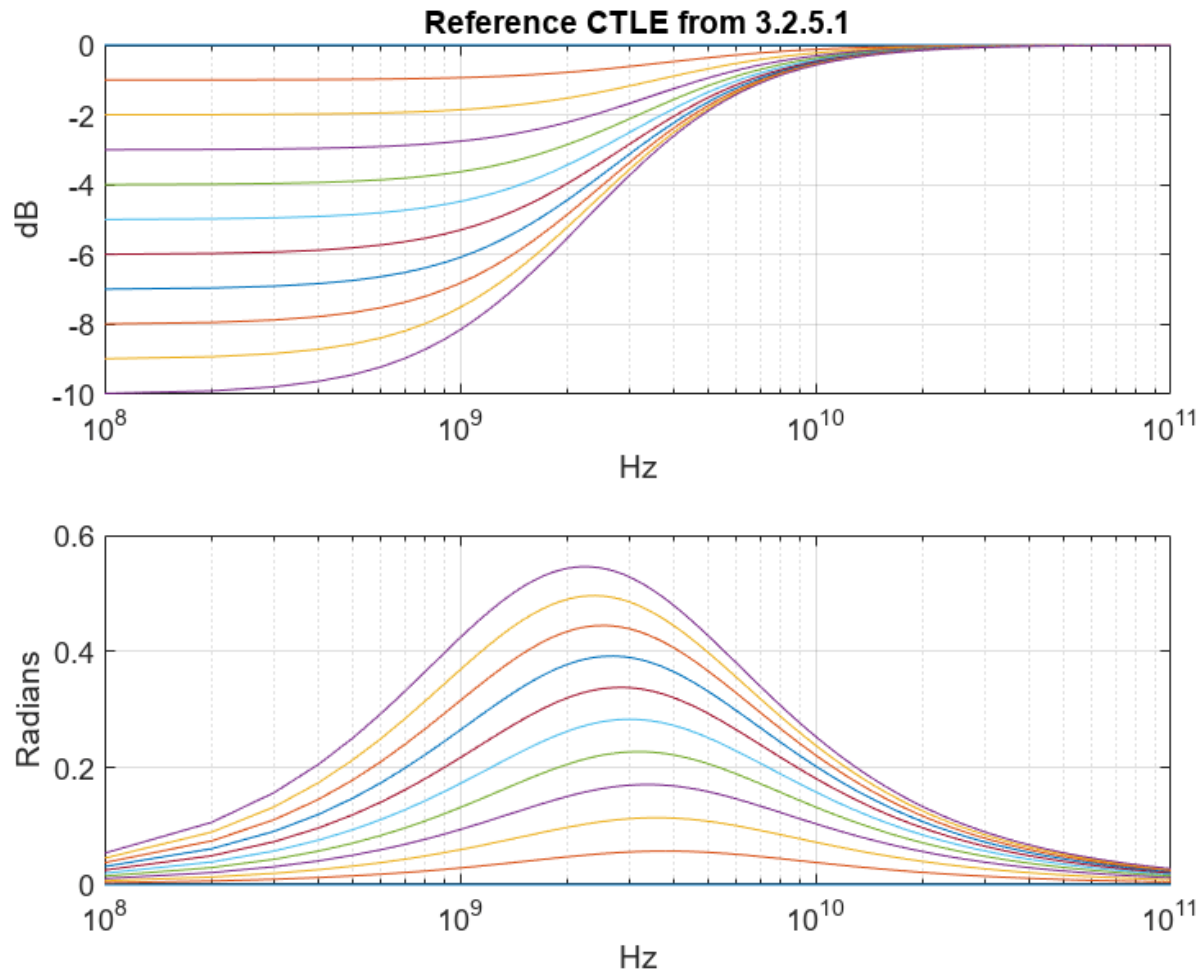
```
ppad = 200e9;

gpz = zeros(length(ADC),4);
gpz(:,1) = -DCgain;
gpz(:,2) = -p1;
gpz(:,3) = -p1*ADC;
gpz(:,4) = -ppad;
```

Receiver Model for Captive Device: Configure CTLE Block

In the Rx CTLE block, you can set the **Specification** to **GPZ Matrix** and insert the workspace variable **gpz** into the **Gain pole zero matrix** dialog (Note: you can use the workspace variable name, or the contents of that variable in this dialog). Then set the CTLE **Mode** to adapt. The Rx CTLE block is set up for 11 configurations (representing Gains of 0 to -10dB in steps of -1dB). The **GPZ** (Gain Pole Zero) matrix data is derived from the transfer function of the behavioral CTLE (USB4 V2 specification, section 3.2.5).

Receiver Model for Captive Device: Configure CDR Block

You can see a CDR block has been placed within the Receiver structure, this will enable clock-domain recovery to be configured at a later time.

Receiver Model for Captive Device: Plot Statistical-Mode Results

Note that a minimum compliance for the Receiver (which may be used to evaluate Eye Diagrams) is defined by the USB4 V2 specification (Section 3.2.6.2, Table 3-31 and Figure 3-42) as follows:

- **Voltage Margin Range, Minimum:** +/- 74/2 mV or 37mVpp
- **Voltage Margin Range, Maximum:** +/- 200/2 mV or 100mVpp
- **Timing Margin Range, Minimum:** +/- 0.2 UI
- **Timing Margin Range, Maximum:** +/- 0.5 UI

You can use the SerDes Designer plots to visualize the performance of the USB4 V2 system in Statistical Mode. You can then select the options **BER** and **Report** from **Add Plots** drop down in the toolstrip and observe the results to compare performance against the Voltage and Timing margin ranges given above.

Note: You can also add a plot to visualize the CTLE Transfer Function.

Jitter Parameters for Transmitter

You can add Jitter parameters based on the eCOM table for the USB4 V2 by selecting the Tx/Rx Jitter button and setting the values as follows:

Note: These parameters will export as type "Float" with format "Value." After exporting to Simulink, you can change these to format "Range" using the IBIS-AMI Manager, to set `0` UI as default value for simulation.

Tx Rj Jitter Value

- Set value to `0.0085` (Note: this is the maximum allowed per eCOM table v1.1, where `0` UI is nominal).
- Set units to UI.
- Enable **Tx_Rj** by checking the box and observe in real time how the eye diagram and report values change.

Tx Sj Jitter Value

- Set value to `0.024704` (Note: this is the maximum allowed per eCOM table v1.1, where `0` UI is nominal).
- Set units to UI.
- Enable **Tx_Sj**.

Jitter Parameters for Receiver

Rx Rj Jitter Value

- Set value to `0.013124405` (Note: this is the maximum allowed per eCOM table v1.1, where `0` UI is nominal).
- Set units to UI.
- Enable **Rx_Rj.**

Jitter Parameters

Tx/Rx Jitter

Eye Diagram Clock:

◉ Clocked   ○ Ideal

Tx Jitter:

Name	Value	Unit
☐ Tx_DCD	0	UI ▼
☑ Tx_Rj	0.0085	UI ▼
☐ Tx_Dj	0	UI ▼
☑ Tx_Sj	0.024704	UI ▼
☐ Tx_Sj_Frequency*	1000000	Hz

Rx Jitter:

Name	Value	Unit
☐ Rx_DCD	0	UI ▼
☑ Rx_Rj	0.013124405	UI ▼
☐ Rx_Dj	0	UI ▼
☐ Rx_Sj	0	UI ▼

Rx Clock Recovery Jitter:

Name	Value	Unit
☐ Rx_Clock_Recovery_Mean	0	UI ▼
☐ Rx_Clock_Recovery_Rj	0	UI ▼
☐ Rx_Clock_Recovery_Dj	0	UI ▼
☐ Rx_Clock_Recovery_Sj	0	UI ▼
☐ Rx_Clock_Recovery_DCD	0	UI ▼

Rx Noise:

Name	Value	Unit
☐ Rx_Receiver_Sensitivity*	0	Volts

You may observe that the Statistical Eye and BER plot appear to be more closed, but this is expected. The jitter parameters from the specification are maximum allowed values and there is no expectation they would each be maximum at the same time in a real-world system. The reason you are enabling these jitter parameters is so they are automatically included in this SerDes System when exporting to Simulink. Later you will see how to change these to format "Range" in the IBIS-AMI manager and set each one to 0 UI as a default value.

Note: After exporting to Simulink, you can also edit their Type, Usage, Format, and Value using the IBIS-AMI manager.

Part 2: System Configuration for Router Assembly with Reference Receiver

The SerDes Designer app will load a system representing a router assembly described by the USB4 V2 specification: The Transmitter is retained from the Captive Device topology discussed in the first part of this example and shows a Reference Receiver for a Router Assembly based on the eCOM table for USB4 V2 (see reference section). This includes a Transmitter having an FFE, a Router Assembly Receiver with CTLE and DFECDR along with analog characteristics and an example channel as shown below.

Use the MATLAB® command window to open the second model usb4p0v2_rx_ref.mat with SerDes Designer:

serdesDesigner('usb4p0v2_rx_ref');
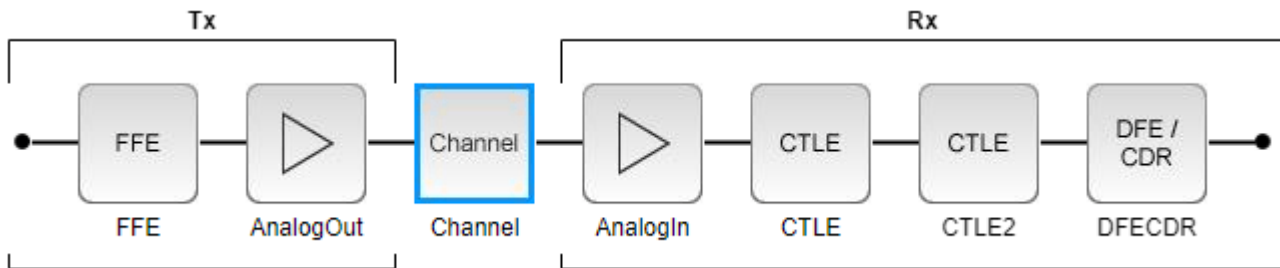
You can see from the SerDes Designer app options that the following remain configured the same as Part 1 above:

- **Symbol Time:** 39.0625ps
- **Target BER:** 1.57e-08
- **Samples per Symbol:** 32
- **Modulation:** PAM3
- **Signaling:** Differential

Channel Model

This also retains the configuration from Part 1 above:

- **Channel loss:** 9.5dB
- **Differential impedance:** 85 Ohms
- **Target Frequency:** 12.8 GHz

Transmitter Model: AnalogOut Block

In the transmitter section, the Tx AnalogOut block remains the same from Part 1 above:

- **Voltage:** 0.5 V
- **Rise time:** 7.8125 ps
- **R:** 42.5 Ohms
- **C:** 0.1 pF (default value, not defined in specification).

Transmitter Model: Configure FFE Block

The Tx FFE remains the same as Part 1 above:

- **FFE Tap** is [0 -0.05 0.9 -0.05] (Preset 5)

Receiver Model for Router Assembly: Summary

Note that the **CTLE** block portion of the Receiver model differs from Part 1 above, in that there are 2 stages represented by 2 blocks, and there is the addition of a **DFECDR** block as well.

Receiver Model for Router Assembly: Configure AnalogIn Block

You can click on the Rx AnalogIn block to observe the following configuration settings, which are identical to Part 1 above:

- **R:** 42.5 Ohms
- **C:** 0.2 pF (default value, not defined in specification).

Receiver Model for Router Assembly: Find CTLE Response based on eCOM

Note that from this point forward, the Receiver model differs from Part 1 above, in that a reference eCOM receiver is described as having a 2-stage CTLE with 12-tap DFE and CDR (see "eCOM_general_config_rev1p1.xlsx" reference below). The Rx CTLE is defined as having 2 stages with the following transfer equation (see "USB 80G PHY Analysis v1p1.pptx" reference below):

$$H(s) = \frac{\left(f_{p1} \cdot \left(s + 2\pi f_{z1} \cdot A_{DC\_1}\right)\right)}{\left(f_{z1} \cdot \left(s + 2\pi f_{p1}\right)\right)} \cdot \frac{\left(2\pi f_{p2}\right)}{\left(s + 2\pi f_{p2}\right)} \cdot \frac{\left(s + 2\pi f_{LP} \cdot A_{DC\_LP}\right)}{\left(s + 2\pi f_{LP}\right)}$$

Where:

A_{DC_1} : DC Gain for CTLE stage 1 ("high pole" stage): 0 to -12dB in steps of 1dB

f_{p1}: Pole location for CTLE stage 1: 10.24GHz

f_{z1}: Zero location for CTLE stage 1: 10.24GHz

f_{p2}: 2nd Pole location for CTLE stage 1: 25.6GHz

A_{DC_LP} : DC Gain for CTLE stage 2 ("low pole" stage): 0 to -6dB in steps of 1dB
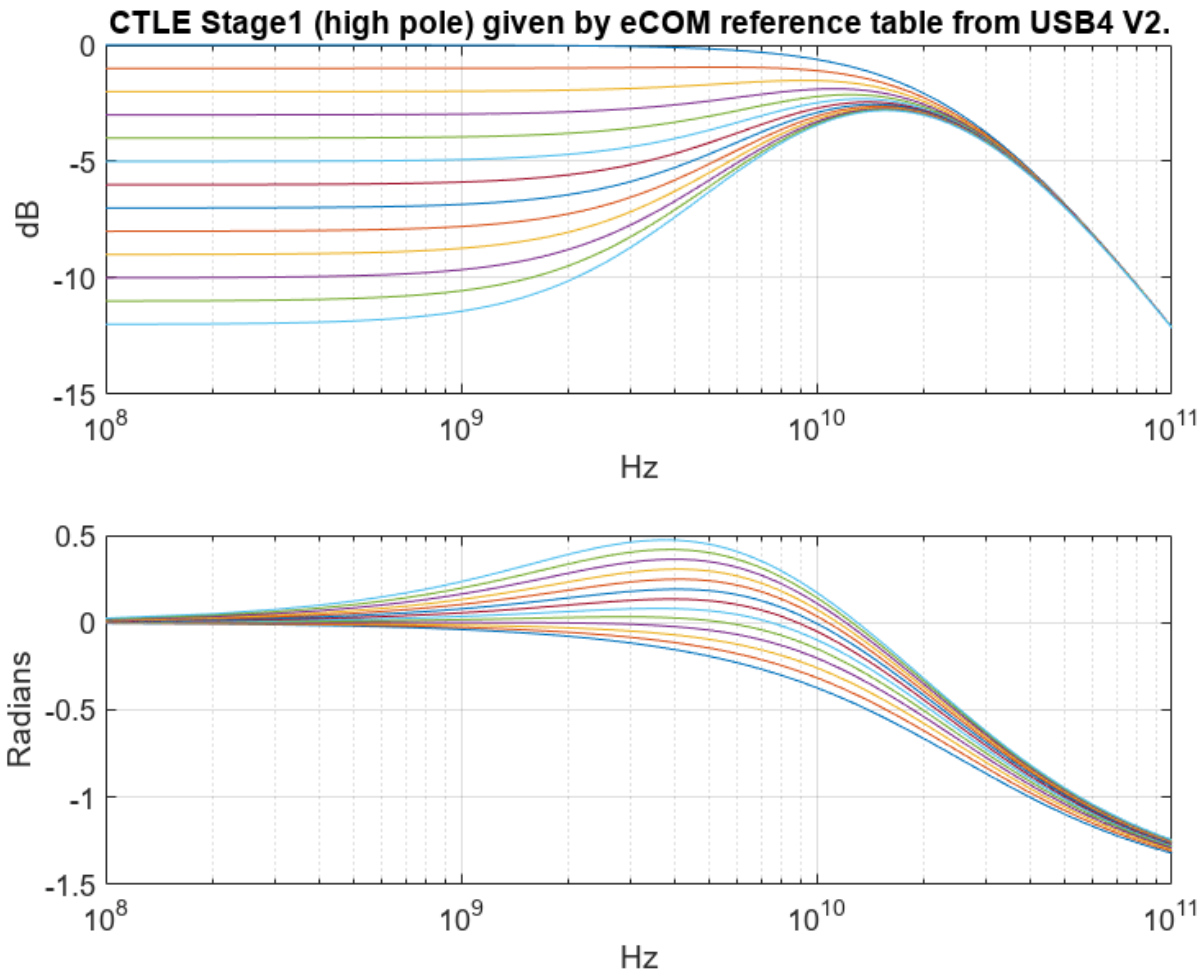
f_{LP} : Pole location for CTLE stage 2: 0.32GHz

The following code block shows how to find a GPZ matrix that represents the Rx CTLE stage 1 ("high pole"). You can then add this GPZ matrix (as you were shown in Part 1 above) to the first CTLE block (named CTLE) in the Receiver structure.

```matlab
% CTLE Stage1 (high pole) given by eCOM reference table for USB4 V2.
DCgain_hp = 0:12;
ADC_hp = 10.^(-DCgain_hp/20);
hp_z1 = 10.24e9; %Hz
hp_wz1 = 2*pi*hp_z1;
hp_p1 = 10.24e9; %Hz
hp_wp1 = 2*pi*hp_p1;
hp_p2 = 25.6e9; %Hz
hp_wp2 = 2*pi*hp_p2;

f = linspace(0,100e9,1001);
w = 2*pi*f;
s = 1j*w;

H = zeros(length(ADC_hp),length(f));
for ii = 1:length(ADC_hp)
    H(ii,:) = (hp_p1*(s+hp_wz1*ADC_hp(ii)))./(hp_z1*(s+hp_wz1)).*((hp_wp2)./(s+hp_wp2));
end

figure(10)
    subplot(211);
        semilogx(f,db(H))
        grid on
        xlabel('Hz')
        ylabel('dB')
        title('CTLE Stage1 (high pole) given by eCOM reference table from USB4 V2.')
    subplot(212);
        semilogx(f,unwrap(angle(H)))
        grid on
        xlabel('Hz')
        ylabel('Radians')
```

CTLE Stage1 (high pole) given by eCOM reference table from USB4 V2.

```
% Create a GPZ matrix for use in a CTLE block:
gpz_hp = zeros(length(ADC_hp),4);
gpz_hp(:,1) = -DCgain_hp;
gpz_hp(:,2) = -hp_p1;
gpz_hp(:,3) = -hp_z1*ADC_hp; %z1
gpz_hp(:,4) = -hp_p2;
```

The following code block shows how to find a GPZ matrix that represents the Rx CTLE stage 2 ("low pole"). You can then add this GPZ matrix (as you were shown in Part 1 above) to the second CTLE block (named CTLE2) in the Receiver structure (where there are two CTLE blocks: one CTLE block per stage).

```
%CTLE Stage2 (low pole) given by eCOM reference table for USB4 V2.
DCgain_lp = 0:12;
ADC_lp = 10.^(-DCgain_lp/20);
lp = 0.32e9; %Hz
wlp = 2*pi*lp;

f = linspace(0,100e9,1001);
w = 2*pi*f;
s = 1j*w;
```

```
H = zeros(length(ADC_lp),length(f));
for ii = 1:length(ADC_lp)
    H(ii,:) = (s + wlp*ADC_lp(ii))./(s + wlp);
end

figure(3)
    subplot(211);
        semilogx(f,db(H))
        grid on
        xlabel('Hz')
        ylabel('dB')
        title('CTLE Stage2 (low pole) given by eCOM reference table from USB4 V2.')
    subplot(212);
        semilogx(f,unwrap(angle(H)))
        grid on
        xlabel('Hz')
        ylabel('Radians')
```


CTLE Stage2 (low pole) given by eCOM reference table from USB4 V2.
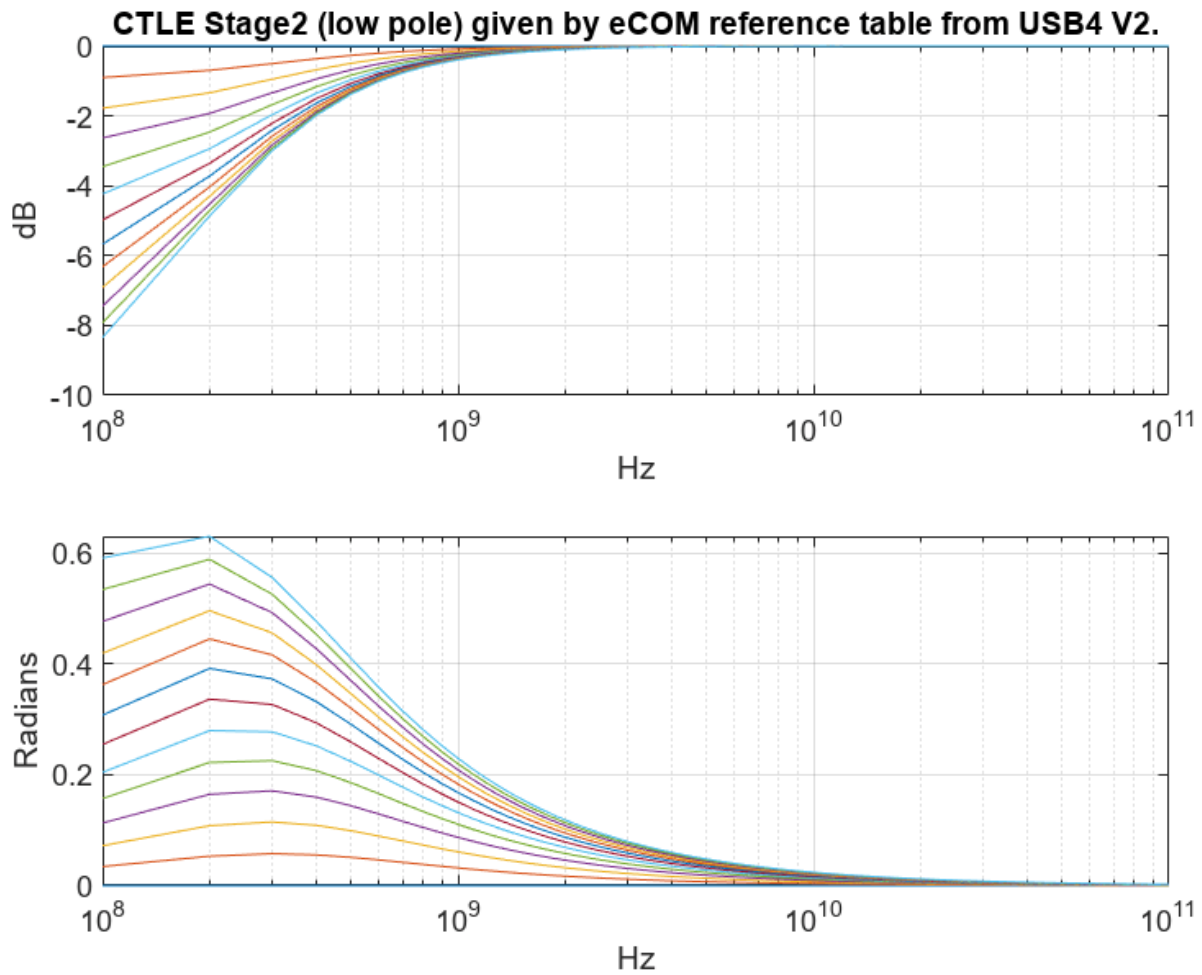
```
ppad = 200e9;
```

```
gpz_lp = zeros(length(ADC_lp),4);
gpz_lp(:,1) = -DCgain_lp;
gpz_lp(:,2) = -lp;
gpz_lp(:,3) = -lp*ADC_lp;
gpz_lp(:,4) = -ppad;
```

Receiver Model for Router Assembly: Define DFE Taps

The Rx **DFECDR** block is defined according to the eCOM table from USB4 V2 as having the following characteristics:

- **Tap Count:** 12
- **Range of Tap 1:** 0 to -0.75
- **Range of Taps 2 through 12:** 0.2 to -0.2

Part 3: Export SerDes System to Simulink

Click on the **Export** button to export the second of the above configurations to Simulink for further customization and generation of the AMI model executables. This configuration contains both the Transmitter and reference Receiver models.

Review Simulink Model Configuration

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.
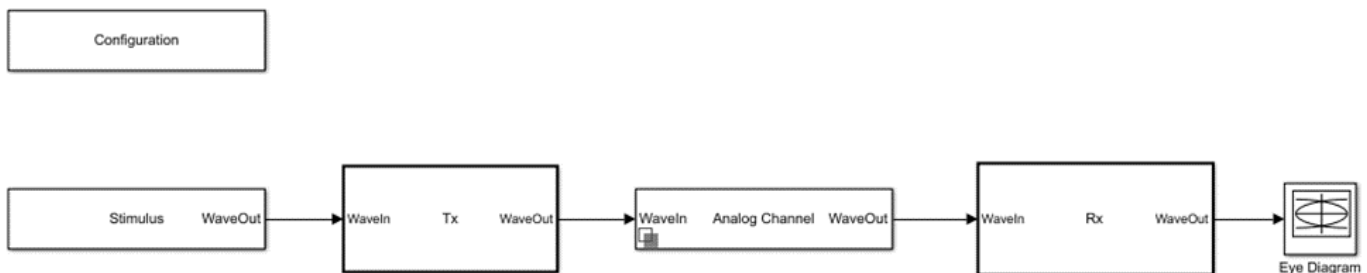
You can explore the configuration of the SerDes system in Simulink as follows:

- 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 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 2 CTLEs and 1 DFECDR block carried over from the SerDes Designer app. An Init block is also introduced to model the Statistical(Init) portion of the AMI model.

- Double click the Stimulus block to open its Block Parameters dialog. You will see how to configure this in the next step below.

Configure Stimulus Pattern

Open the Stimulus block and select the appropriate stimulus pattern for your system. For USB4 V2, set **Waveform creation method** to **Serial PRBS** and then **Binary to PAMn mapping** to **USB4V2** to set the mapping as defined in section 4.3.2.2.1 of the USB4 V2 specification.

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.

Simulink supports more sophisticated AMI model development compared to Serdes Designer, so you can create a selector for the Tx FFE to allow selection of the values from the presets specified in the USB4 V2 specification (Section 3.2.3, Table 3-24).

Setup Tx FFE

In the **AMI - TX** tab, under the **Model_Specific** parameters, you can click on **FFE** and then **Add Parameter** button to create a new AMI parameter "ConfigSelect."
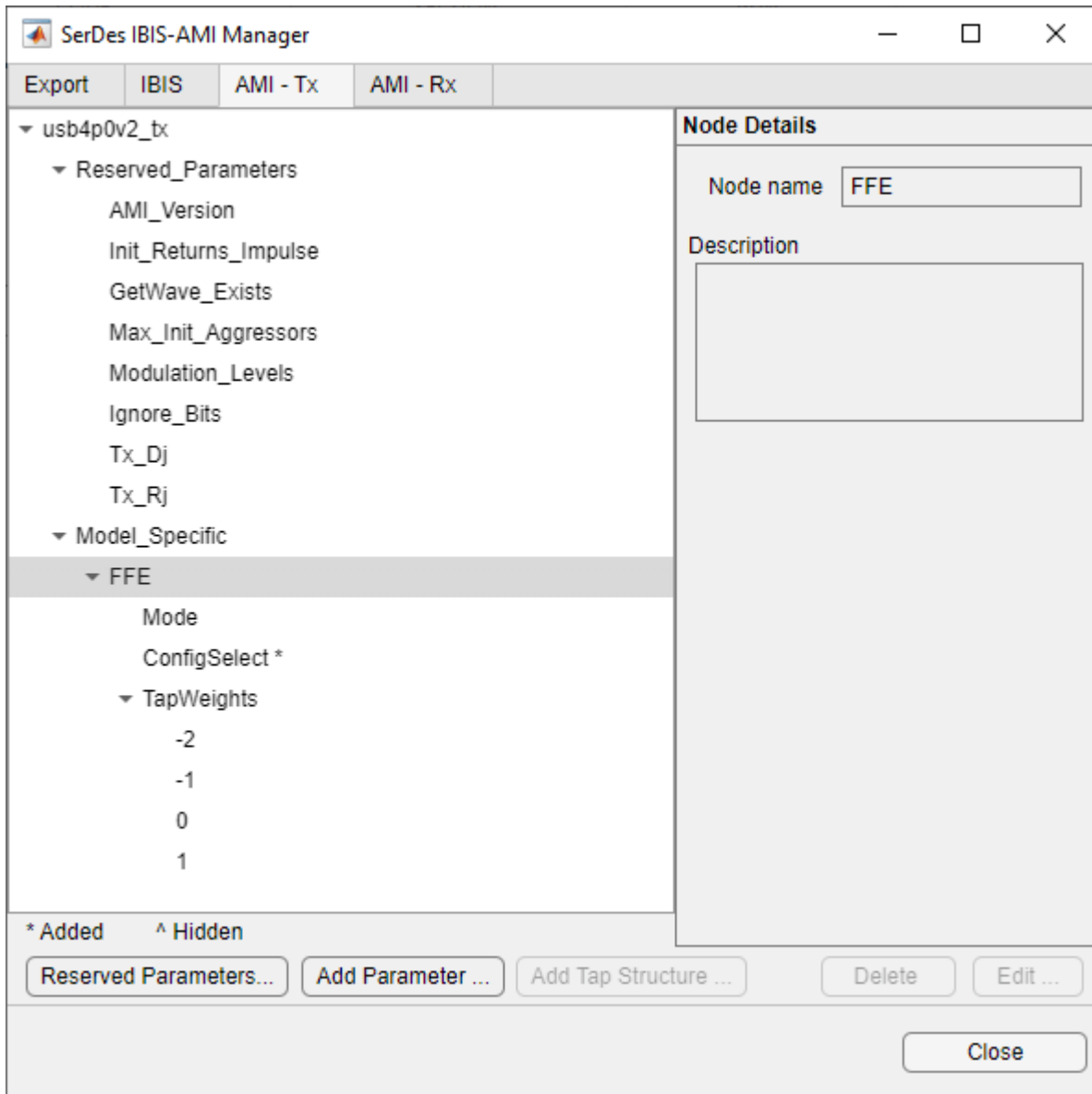
Then select the new parameter **ConfigSelect** and click the edit button. In the next dialog window, set the following values that represent **Tx FFE Presets** P0 through P41 (as defined by the PCIe Gen5 Base Specification).

Settings for the Add/Edit AMI Parameter **ConfigSelect**:

- Parent Node **FFE**
- Parameter name **ConfigSelect**
- Current value **P5**
- Description **Transmitter Equalization Presets**
- Usage **In**
- Type **Integer**

- Format **List**
- Default **0**
- List values **[-1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41]**
- List_Tip values **["CUSTOM" "P0" "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9" "P10" "P11" "P12" "P13" "P14" "P15" "P16" "P17" "P18" "P19" "P20" "P21" "P22" "P23" "P24" "P25" "P26" "P27" "P28" "P29" "P30" "P31" "P32" "P33" "P34" "P35" "P36" "P37" "P38" "P39" "P40" "P41"]**
- Hidden = **False** (unchecked)

### Tx FFE: Enable Tap-Preset Selector for Statistical (Init) Analysis

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.

Look inside the Tx subsystem and you will see an FFE block along with a block called "Init."
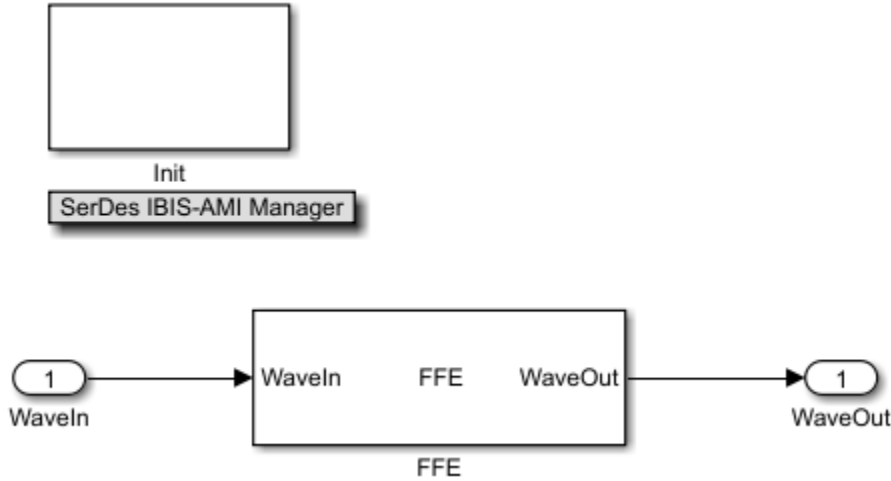
To edit the code that executes during statistical AMI simulation, double-click the **Init** block to open the Block Parameters dialog box and click the **Refresh Init** button to propagate the new AMI parameter to the Initialize sub-system.

Then click on **Show Init** to display the code in an editor window.
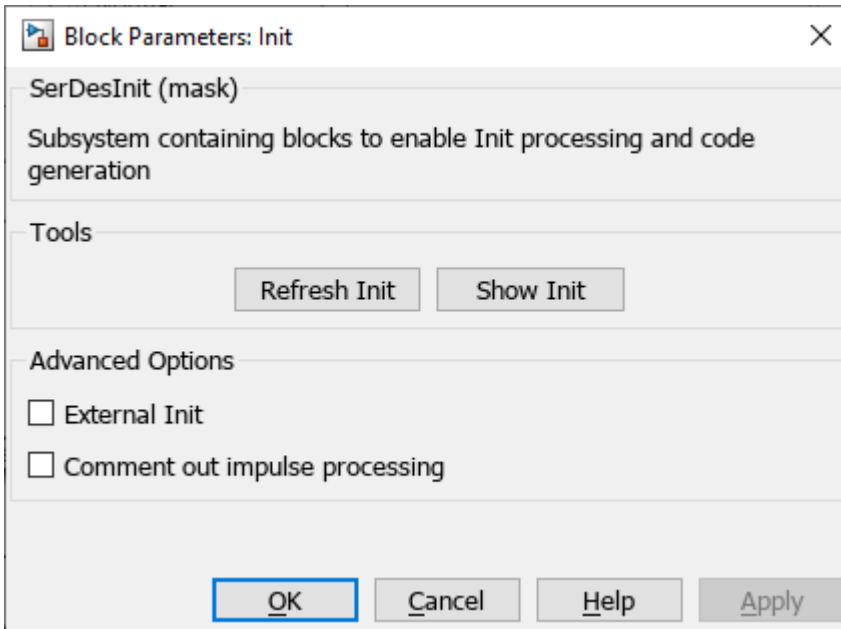
You will see the section for the Custom user code area shown thusly:

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

% END: Custom user code area (retained when 'Refresh Init' button is pressed)
```
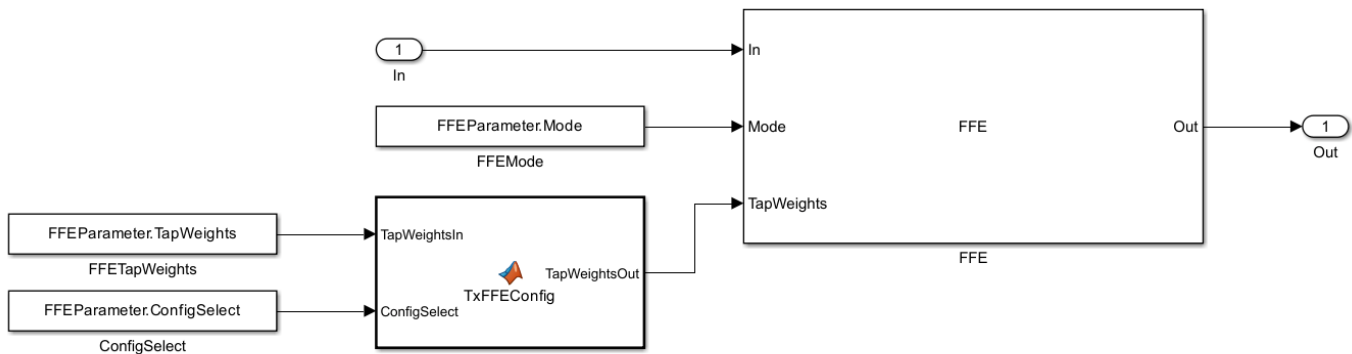
Add code for your Tx FFE Tap selector between the %% Begin and % End portions of Custom user code area. Or, you can copy the example code from the file Tx_FFE_Presets_Init.m attached to

this example. This will allow for case/select for different Tx FFE Tap Preset values already configured in the IBIS-AMI manager.

Tx FFE: Enable Tap-Preset Selector for Time Domain (GetWave) Analysis

Enable Tx FFE tap presets in GetWave analysis with 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.

You can see that a new constant block has been added called **FFEParameter.ConfigSelect.** This is created automatically by the IBIS-AMI Manager when a new Reserved Parameter is added. Next, you can follow these steps to re-configure the selection of tap weight presets for time domain (GetWave) simulation:

- Add a MATLAB Function block to the canvas from the Simulink/User-Defined library.
- Rename the MATLAB Function block to `TxFFEConfig`.
- Double-click the MATLAB Function block and observe the template code now contains a function allowing case/select.

```
function TapWeightsOut = TxFFEConfig(TapWeightsIn, ConfigSelect)

switch ConfigSelect

    case 0

    otherwise

end
```
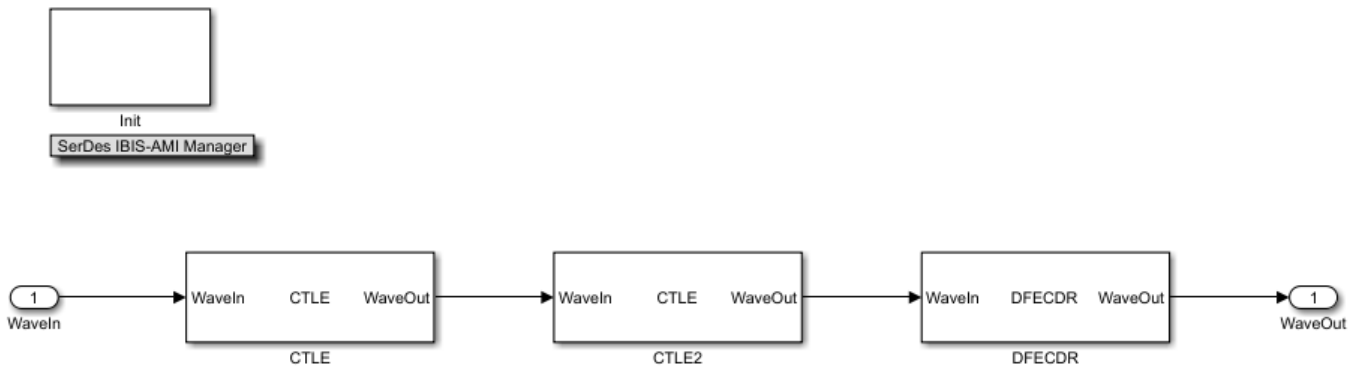
- You can manually create the case/select values, or you can copy from the example code provided in the file `TxFFEConfig.m` attached to this example.
- Finally, wire up the Data Stores and new MATLAB function block as shown in the figure above.

Update Receiver AMI Parameters
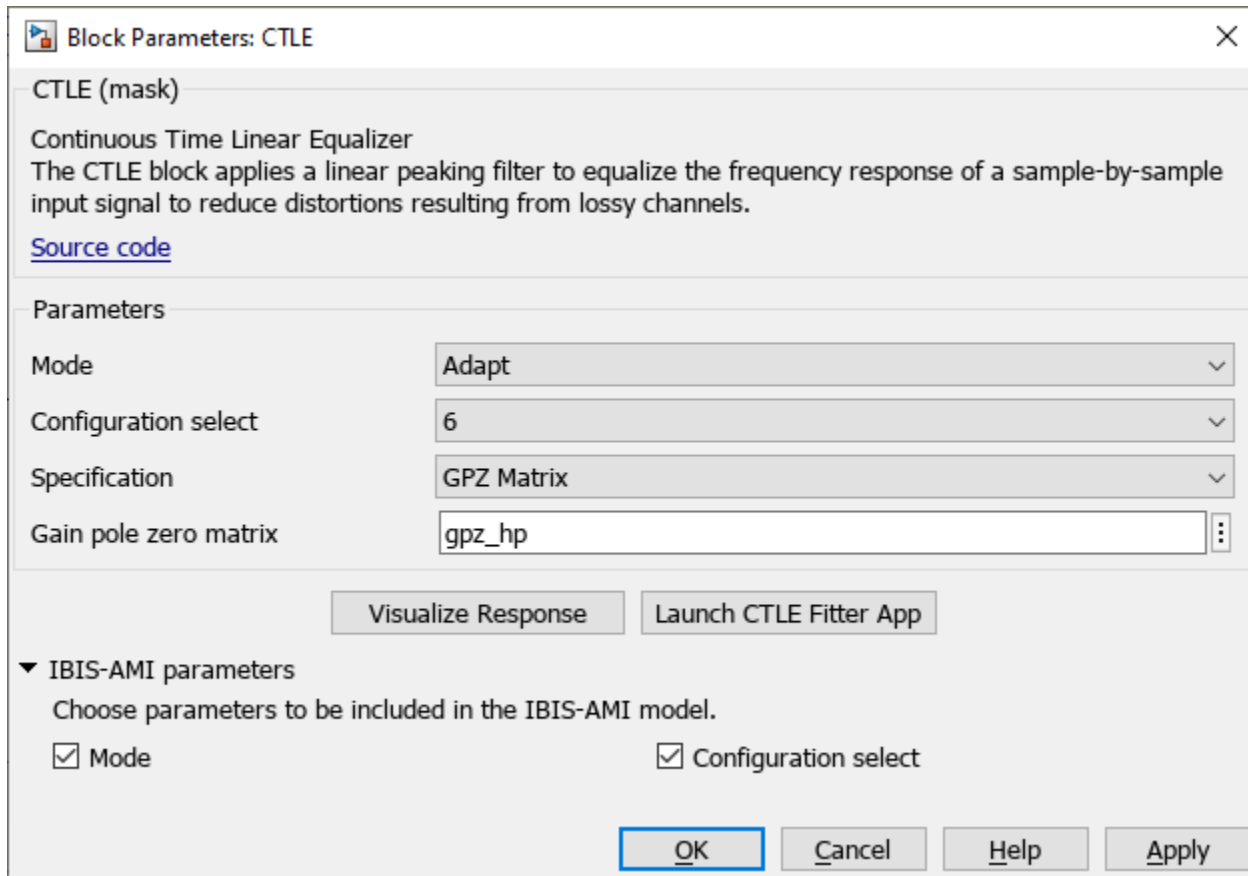

Configure Rx CTLE Blocks

You can configure the **CTLE** blocks to contain the appropriate GPZ matrices. Each block should have one GPZ matrix. Due to the principle of Super Position, you can use either one. For the first CTLE block (CTLE), configure the following settings:

Mode: **Adapt**

Configuration select: **(any)**

Specification: **GPZ Matrix**

GPZ Matrix: **gpz_hp** (Note: this references the MATLAB workspace variable created with the script above)

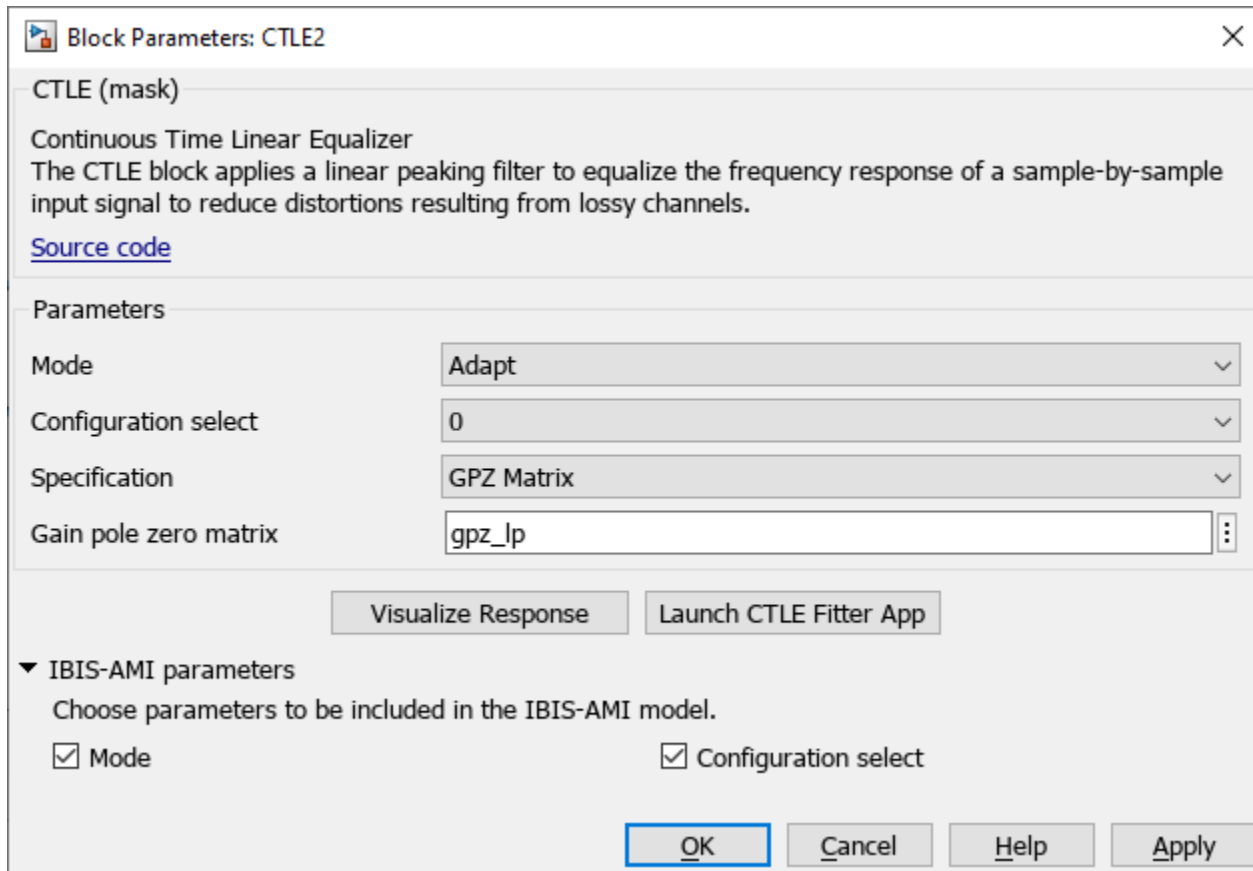Then for the second CTLE block (CTLE2), configure the following settings:

Mode: **Adapt**

Configuration select: **(any)**

Specification: **GPZ Matrix**

GPZ Matrix: **gpz_lp** (Note: this references the MATLAB workspace variable created with the script above)

Configure Rx DFECDR Block

Simulink supports Time Domain simulation, so you can configure the **DFECDR** block to perform clock recovery based on UI and other parameters depending on your particular implementation. You will see how to configure the **DFE** portion and **CDR** sections of this block below:

Configure values in the **DFE** tab as follows:

- Mode: **Adapt**
- Initial tap weights (V): **[0 0 0 0 0 0 0 0 0 0 0 0]**
- Adaptive gain: **9.6e-05**
- Adaptive step size (V): **1e-06**
- Minimum DFE tap values (V): **[-0.75 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2]**
- Maximum DFE tap values (V): **[0 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2]**
- 2x Tap Weights = **False** (unchecked):

IBIS-AMI Parameters:

- Mode: **Enabled**
- Tap weights: **Enabled**
- Phase offset: **Enabled**

7-183

- Reference offset: **Enabled**

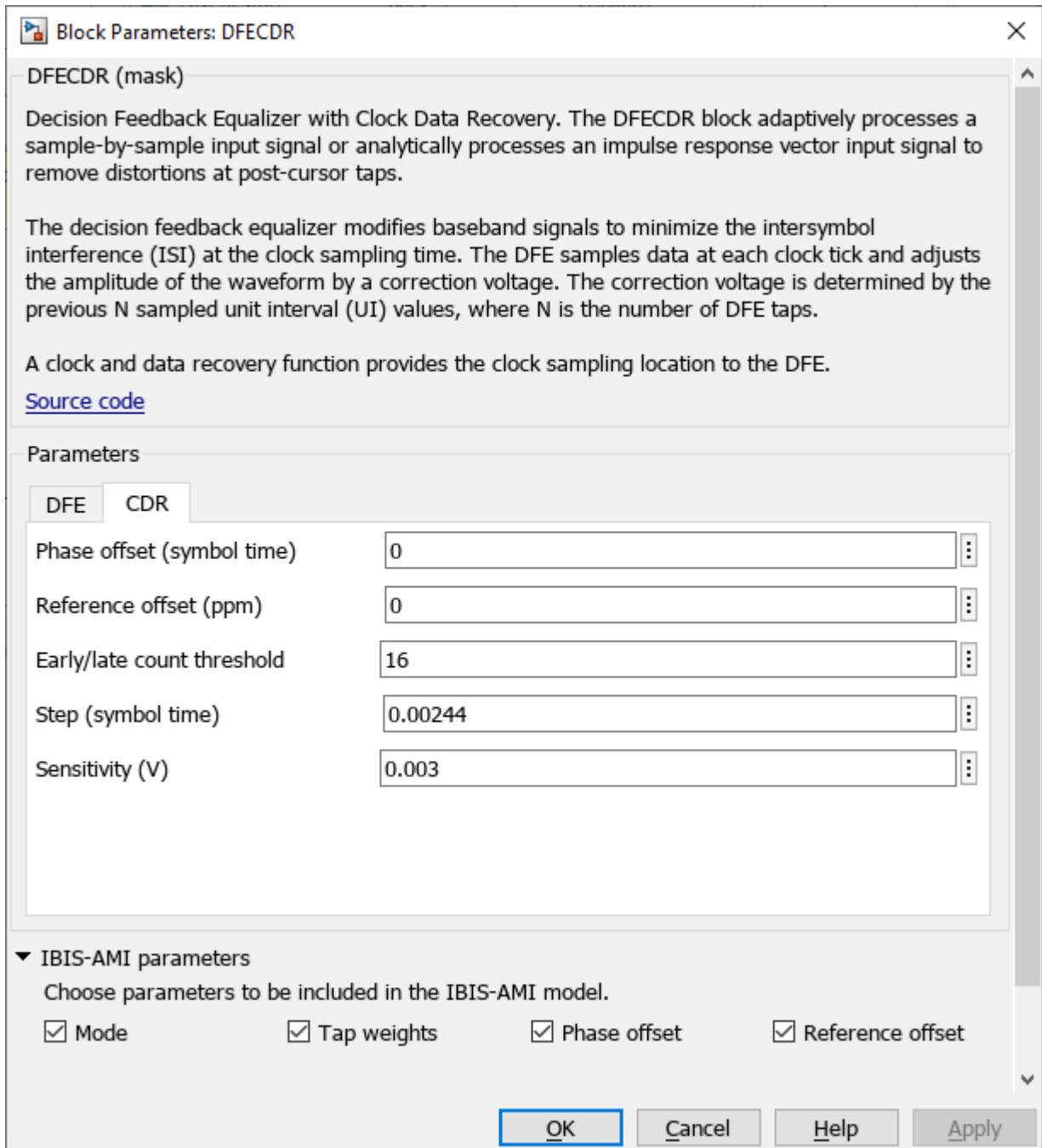Next, click on the **CDR** tab. Assuming that the **CDR** transfer function and the jitter transfer function are compliments to each other and that they touch at the 3dB point, we can infer that the CDR bandwidth needs to be 6 MHz (Section 3.2.2.2, USB4 V2 Specification).

The following is an approximate way of determining the bandwidth of the digital SerDes **CDR** (see **Count** under **serdes.CDR** in the MATLAB documentation).

$$\text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Early Threshold Count} \cdot \text{Step}} \textbf{ or } \text{Bandwidth} = \frac{1}{\text{Symbol time} \cdot \text{Late Threshold Count} \cdot \text{Step}}$$

So given the Bandwidth and Symbol time of the link, we could adjust the count and step parameters to provide a reasonable response. With this in mind, configure values in the **CDR** tab as follows:

- **Phase offset** is set to 0 UI (symbol time)
- **Reference offset** is set to 0 ppm
- **Early/late count threshold** is set to 0 (default)
- **Step** is set to 0.00244 (symbol time)
- **Sensitivity** is set to 0.003 V (For purposes of this example, this is set to approximately 10% of a valid eye height- you may adjust this to represent the actual value for receiver sensitivity in your system design)

Block Parameters: DFECDR ✕

DFECDR (mask)

Decision Feedback Equalizer with Clock Data Recovery. The DFECDR block adaptively processes a sample-by-sample input signal or analytically processes an impulse response vector input signal to remove distortions at post-cursor taps.

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

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

Source code

Parameters

| DFE | CDR |

Phase offset (symbol time)   0

Reference offset (ppm)   0

Early/late count threshold   16

Step (symbol time)   0.00244

Sensitivity (V)   0.003

▼ IBIS-AMI parameters
Choose parameters to be included in the IBIS-AMI model.
☑ Mode   ☑ Tap weights   ☑ Phase offset   ☑ Reference offset

OK   Cancel   Help   Apply

Configure Run-Time Parameters with IBIS-AMI Manager

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. They also contain run-time parameters that would be used by those EDA tools, and are also used by Simulink for this simulation.

Click on the **Export** tab and set the following:

**AMI Model Settings – Tx:**

- **Bits to ignore:** 4

AMI Model Settings – Rx:

- **Bits to ignore:** 30000 (Note: This must be fewer than the symbol count set in Stimulus block in order to display Time Domain Waveforms)

Test the USB4 V2 System with Simulink

You can click on the **Run** button in the Simulink toolstrip simulate the SerDes System representing USB4 V2. Two plots are generated. The first is a time domain (GetWave) eye diagram that is updated as the model is running.

Note: This may appear differently in your Simulink session depending on your specific viewer settings or SerDes system configuration.
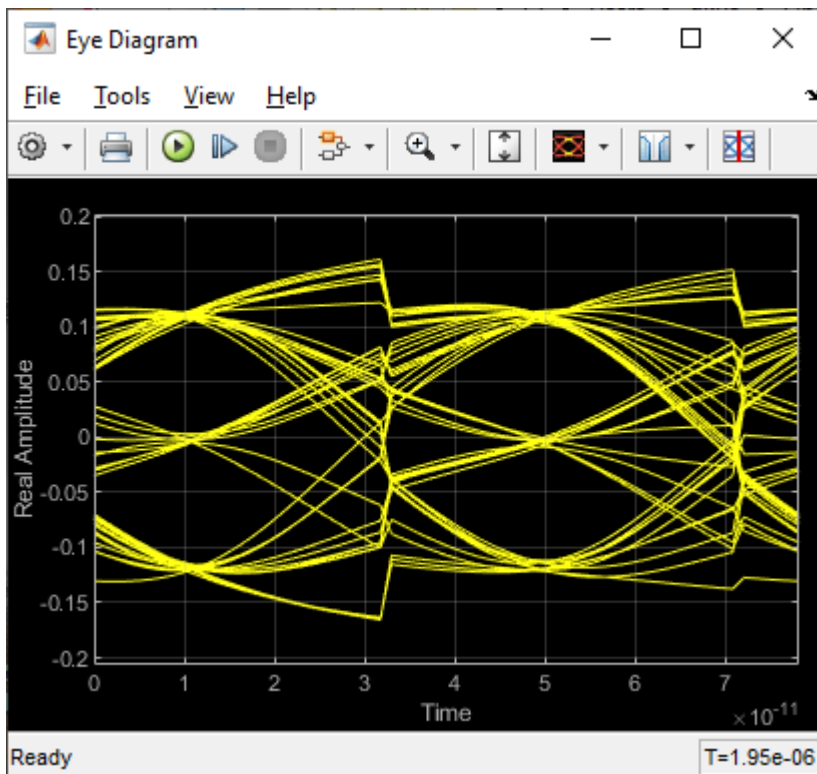
A second plot is presented after simulation has completed which contains views of the Statistical (Init) and Time Domain (GetWave) results:
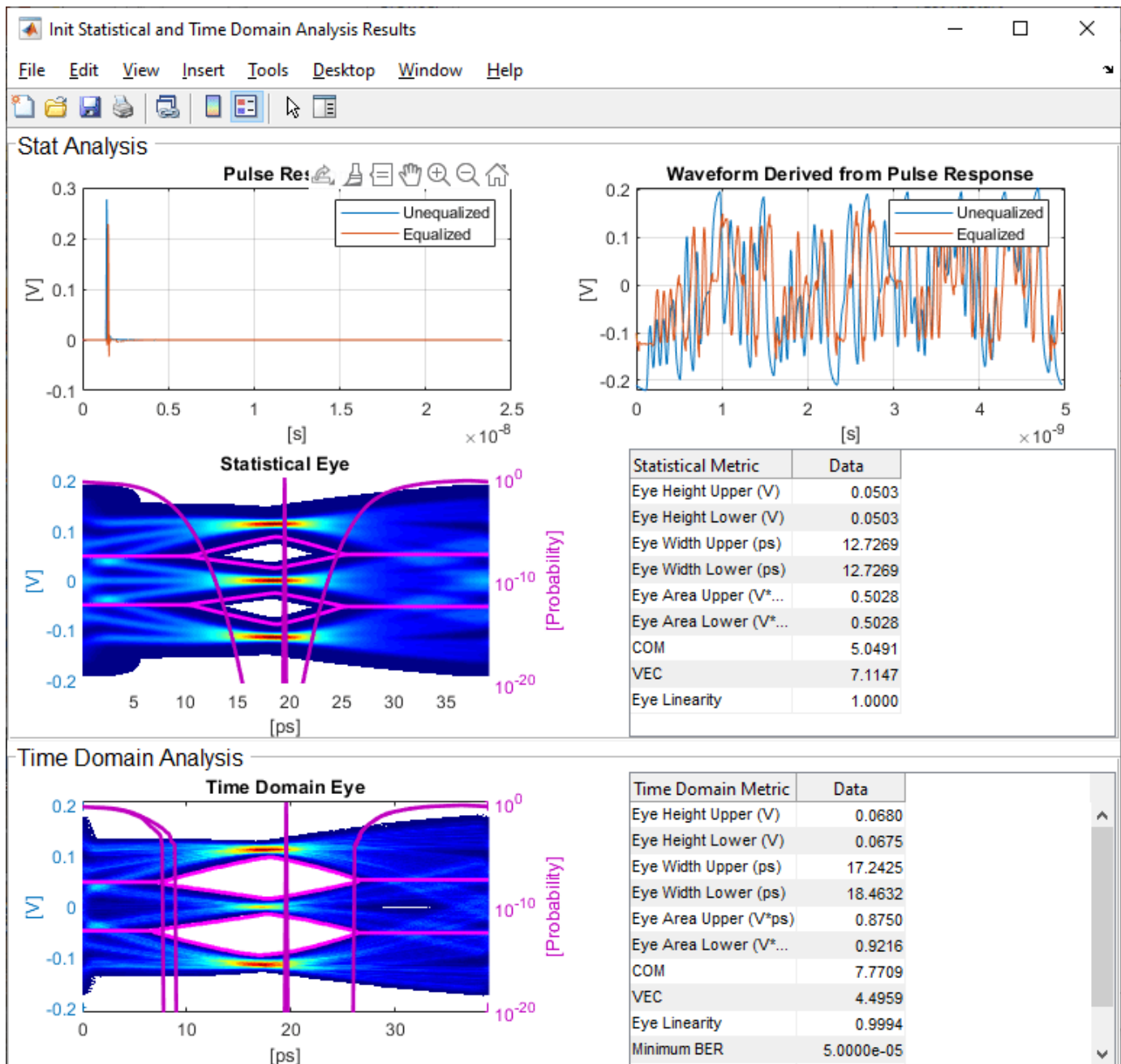
Note: You may observe a difference between Statistical and Time Domain waveforms. In Time Domain simulation, the effects of symbol encoding (e.g. to control ISI) and also real time DFE (Decision Feedback Equalizer) behavior may contribute to improved performance. Conversely, Statistical simulation utilizes frequency-domain Convolution rather than symbol-by-symbol stimulus, and does not include the effects of symbol encoding or signal conditioning by a DFE. Further discussion is beyond the scope of this example, but you may consider other differences between the simulation types as well by researching Communications Theory and the IBIS specification reference provided below.

Part 4: Generate and Export IBIS-AMI Models

The fourth part of this example takes the customized Simulink model, modifies the AMI parameters for USB4 V2, then generates IBIS-AMI compliant 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.

Export IBIS-AMI Models

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

- Update the **Tx model name** to usb4p0v2_tx
- Update the **Rx model name** to usb4p0v2_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 three taps in the Tx FFE.
- Set the **Rx model Bits to ignore value** to 30000 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, .dll files [Windows] and/or .so files [Unix/Linux]).
- Set the **IBIS file name** to be usb4p0v2.ibs

You can now observe the IBIS-AMI manager looks as follows, where your system is ready to export IBIS-AMI models:

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

Test Generated IBIS-AMI Models

The USB4V2 transmitter and receiver IBIS-AMI models are now complete and ready to be utilized within any industry EDA tool capable of IBIS-AMI simulation, such as Serial Link Designer from Signal Integrity Toolbox.

References

[1] USB-IF USB4 V2 Specification, October 2022 Edition, https://usb.org/.

[2] USB4 V2 Artifact, "USB 80G PHY background.pdf", https://usb.org/.

[3] USB4 V2 Artifact, "eCOM_general_config_rev1p1.xlsx", https://usb.org/.

[4] IBIS 7.1 Specification, https://ibis.org/ver7.1/ver7_1.pdf.

[5] "Channel Operating Margin (COM)" (Signal Integrity Toolbox)

See Also
SerDes Designer | FFE | CTLE | CDR

Related Examples
-