AMI Models: How to tell a Peach from a Lemon

Dr. Michael Steinberger, SiSoft
msteinb@sisoft.com

Todd Westerhoff, SiSoft
twesterh@sisoft.com
Abstract

Most vendors now supply IBIS-AMI models of their SerDes macros. However, the accuracy, usability, and speed of execution of these models varies widely in ways that are not immediately obvious to many users. Using real models from un-named vendors and software tools that are publicly available free of charge, this tutorial demonstrates specific desirable and undesirable model characteristics, how they affect the user, and how to test or inspect for them.

Author(s) Biography

Michael Steinberger, Ph.D.
Lead Architect, Serial Channel Products
SiSoft

Michael Steinberger, Ph.D., has over 30 years experience designing very high speed electronic circuits. Dr. Steinberger holds a Ph.D. from the University of Southern California and has been awarded 14 patents.

He is currently responsible for the architecture of SiSoft's Quantum Channel Designer tool for high speed serial channel analysis. Before joining SiSoft, Dr. Steinberger led a group at Cray, Inc. performing SerDes design, high speed channel analysis, PCB design and custom RAM design.

Todd Westerhoff
Vice President of Software Products
SiSoft

Todd Westerhoff has over 29 years experience in the modeling and analysis of electronic systems, including 12 years of signal integrity experience. Prior to joining SiSoft, Todd managed a high speed design group that provided static timing, signal integrity and design rule consultation to various ASIC and system engineering groups within Cisco Systems, Inc. Todd holds a B.E. degree in electrical engineering from the Stevens Institute of Technology in Hoboken, New Jersey.
1.0 Users must be Choosers

Since its ratification in August 2008 as part of IBIS 5.0 [1], models using the IBIS algorithmic modeling interface (AMI) have become widespread and are now the primary means for modeling transmitters and receivers in high speed serial channels. With so many models being developed by so many different model developers, it is unfortunate but not at all surprising that the quality of IBIS-AMI models varies over a very wide range.

To drive IBIS-AMI models to a more consistent quality level, it would be nice if there were a group such as the IBIS Quality Task Group [2]; however, that group is just beginning to address the topic of AMI model quality.

In the absence of an industry-wide effort on defining AMI model quality, SiSoft has published the Opal™ resource document [3]. The copyright statement on this document specifically states that the document can be copied and modified by anyone as long as they don’t use the Opal™ name. Thus, users can use this material or refer to Opal™ if it says what they want to say.

Regardless whether or not there are industry documents which define AMI model quality, user feedback is the only mechanism that can regulate the quality of these models. Many suppliers only know that they are expected to produce an AMI model to go with their SerDes designs. They have very little information as to how these models will be used and therefore what characteristics these models must have. When provided with this information, suppliers have consistently shown that they can produce satisfactory AMI models.

In order to provide the feedback that is so essential to AMI model quality, users need to know how to relate the characteristics of AMI models back to the quality of the results these models will help generate, the effort required to produce these results, and the impact these results will have on the correctness of engineering decisions.

This tutorial demonstrates characteristics of an AMI model that can materially affect the user. Users can then decide which characteristics are the most critical and make sure their suppliers understand which characteristics those are. There may be other characteristics which users decide are at best “nice to have”. Either way, the goal is for these decisions to be informed ones.

2.0 User Experience

After providing a basic background in AMI modeling and describing the demonstration environment, this tutorial describes a number of problems encountered when using AMI models. The problems described in this tutorial follow the same sequence as a typical user experience.
1. Model won’t load.

Typically, this occurs either because of syntax errors in the IBIS file or because a file named in the IBIS file is not available.

2. Model loads but won’t run.

It may be that the AMI model requires additional data or executable files that are not available. In some cases, a license for a commercial product is required. In these cases, the simulation run crashes immediately. There are also some more subtle problems that cause an immediate crash.

There are also some models that run at least an order of magnitude slower than most models. For these models it’s difficult to know whether or not the simulation is hung or in an infinite loop.

3. Model runs but results don’t make sense.

Sometimes, the results from completed simulation runs are obviously flawed. For example, the data in the output waveform doesn’t match the data in the input waveform or there are unexplained discontinuities in the waveform.

Accurate AMI modeling depends on modeling the receiver’s recovered clock as well as the data. There is a surprising number of receiver models which don’t do this.

It can also happen that the AMI model didn’t implement the equalization that was expected. Often this is because the model’s configuration parameters were misunderstood or entered incorrectly. In order to detect this type of error, the user must know ahead of time what the output should look like.

4. Model produces results, but should I trust them?

AMI modeling results can be over-optimistic because significant impairments have been left out of the model. This can lead to incorrect engineering decisions.

Another problem is that different vendors define different evaluation conditions and acceptance criteria required to assure satisfactory performance. Other vendors do not address these topics, thus leaving a question as to how good is good enough.

Note that as one goes further along in the user experience, the problems become more subtle. They’re more difficult to detect, diagnose, and correct. Once one has an AMI model that loads, runs, and produces reasonable looking results, making sure the results are actually correct requires expertise and hard work. To summarize,

Just because it runs doesn’t mean it’s right.
3.0 AMI Modeling

3.1 Performance Analysis Methods

In the analysis of high speed serial channels, AMI models are used to determine the signal and clock at the receiver’s data decision latch, with the goal of estimating the channel’s bit error rate. The generic configuration is as shown in Figure 1.

In Figure 1, the transmitter AMI model consists of an algorithmic model followed by an IBIS analog buffer model. The analog model is a classic IBIS model that is solely responsible for defining the impedance at the interface to the passive interconnect network. This impedance is important because it defines the reflection coefficient presented to the passive interconnect network, and therefore it is essential for determining the standing waves that may be present in the interconnect network.

The voltage at the output of the model is the combined effect of the analog and algorithmic models. The algorithmic model is responsible for complex behaviors such as equalization that cannot be modeled in the analog model. The analog model has a voltage response as well, and the voltage response of the model is the cascade of these two responses. There is no clear rule for separating the voltage gain of the analog model from that of the algorithmic model, and the model developer is responsible for ensuring that the two parts of the model work together correctly.
The receiver AMI model consists of an IBIS analog buffer model followed by an algorithmic model. The receiver algorithmic model includes a model of the clock recovery loop which estimates the edge times of the recovered clock.

The EDA tool is responsible for estimating the error rate at the output of the data detection latch. In essence, the EDA tool simulates the latch. There are two complementary approaches that are used for bit error rate estimation: statistical analysis and time domain simulation. Each of these approaches has its advantages and disadvantages, and there are numerous variations to each approach. Neither approach is a very good substitute for the other; but when used together skillfully, they form a flexible and comprehensive tool set.

Statistical analysis [4], [5] computes the statistics of the data signal directly based on the assumption that the channel is linear and time invariant, and the data is completely uncorrelated. In the case of AMI modeling, the AMI model outputs an impulse response that represents the combined linear response of the transmit equalization, system interconnect, and receive equalization. For statistical analysis, the impulse response is converted to a pulse response or step response and the statistics of the data signal are computed directly from the pulse response.

Statistical analysis has the advantages that it’s very fast and provides complete coverage of the statistics of the signal. It is, however, limited by the assumptions it’s based on: linearity, time invariance, and uncorrelated data. That is:

- It is not a suitable method for time varying processes such as explicit simulation of adaptive loops.
- For nonlinear channels it’s at best an approximation.
- It doesn’t estimate the bit error rate for specific data patterns.

Time domain simulation generates a sample of the time domain waveform and accumulates an eye diagram in very much the same way that an eye diagram is measured in the laboratory. A clock is used to trigger the beginning of multiple segments of the time domain waveform and these segments are overlaid to produce the eye diagram. There are several ways to estimate errors from an eye diagram. One way is to count the errors directly. In the presence of thermal noise or crosstalk, it is also possible to estimate the error statistics by convolving the eye diagram with the probability density function of the noise process, thus producing a so-called semi-analytic bit error rate estimate.

When using AMI models, the AMI model supplies both the data waveform and the triggering clock. The AMI model is usually called many times to produce contiguous segments of its output waveform, and the clock recovery loop within the AMI model produces clock times which can be used to trigger the accumulation of the eye diagram.

Time domain simulation can be used to do all the things that statistical analysis can’t do. It can explicitly simulate time varying processes such as clock recovery, it can simulate nonlinear data paths, and it can simulate the response due to specific data patterns.
The disadvantage of time domain simulation is that it can take a long time to run—at least an order of magnitude longer than statistical analysis to produce any meaningful results. In fact, time domain simulation takes so long to run that it would be completely impractical to run a simulation which was long enough to produce statistically meaningful results at the low bit error rates such as $10^{-12}$ that are typically required for high speed serial channels.

### 3.2 Bit Error Rate Estimation Compared to Timing Analysis

As in any logic design, the channel will produce an error whenever the clock to data timing at the receiver’s data decision latch fails to meet the timing requirements of the circuit. There are, however, two distinct differences between bit error rate estimation and logic timing analysis:

1. The clock is provided by a clock recovery loop instead of being supplied directly from a core clock. In general, the recovered clock will have substantially more phase noise than a core clock used in a traditional logic design. 
   
   *The net result is that there is significant variability in the clock time at the data decision latch.*

2. In general, the data amplitude at the data decision latch varies over a relatively wide range. If the data amplitude is too low at the time of the recovered clock edge, then the latch will fail to meet the timing requirements of the circuit it’s in, and an error will occur. The minimum data amplitude is called the receiver sensitivity.
   
   *The net result is that there is significant variability in the effective data transition time at the data decision latch.*

Figure 2 illustrates this difference in clock time and data transition time distributions between logic timing analysis and high speed serial channel analysis. This figure also shows the receiver sensitivity for a typical data decision latch and illustrates how it increases the width of the data transition time distribution.
FIGURE 2. Comparison of clock times and data transition times for logic design and high speed channels.

In short, both the clock time and the data transition time become statistical variables instead of static design variables. The goal of the performance analysis is therefore to esti-
mate the probability that the clock to data timing at the receiver’s data decision latch will fail to meet the circuit’s timing requirements. This probability is reported as the estimated bit error rate.

Note that in order to obtain a reliable performance estimate, the statistical variation of both the clock times and the data transition times must be included in the analysis.

### 3.3 IBIS-AMI Interface

The interface to the algorithmic model is defined by the IBIS-AMI interface. The goals of this interface are

1. Enable the performance analysis of high speed serial channels with transmitters and receivers from different IP suppliers.
2. Ensure the portability of transmitter and receiver models between EDA tools.
3. Protect the proprietary details of vendor IP.

To achieve these goals, the IBIS-AMI interface is expressed as an applications programming interface (API). That is, it defines a set of function signatures, an expected content (semantics) for the variables in the function signatures, and an expected sequence of events in which the functions will be called. As long as both the EDA tool and the model accurately implement this interface, they can be expected to produce results. This is the basis of interoperability between models and portability between EDA tools.

IBIS-AMI models are delivered as compiled object code in the form of dynamically linked libraries (DLLs) in Windows or shared object libraries for Linux. This both protects vendor IP and makes model execution relatively efficient.

Because the IBIS-AMI interface is expressed as an API, it offers the model developer a great deal of flexibility. Although the API is expressed in the C language, this interface could be met using any of a number of programming languages. Furthermore, the only constraint on the internal structure of the model is that the structure must fit inside the three function signatures that have been defined. The model developer is free to use whichever algorithms they feel best expresses the behavior of the device.

### 3.4 Phases of Execution

There are four distinct phases in the execution of an algorithmic model. The first phase is mandatory and the rest of the phases are optional. However, all phases that are enabled must occur in the correct order:

1. **Construction** *(Mandatory)*
   
   The dynamic memory for the model is allocated and initialized. The amount of dynamic memory and its organization is entirely dependent on the model. Usually, the initial values in the dynamic memory will depend on the required model configuration, and may depend on the channel impulse response (e.g., adaptive equalization).
   
   This phase of execution is performed by the `AMI_Init()` function.
2. **Statistical Analysis (optional)**
   The model modifies the channel impulse response to represent the impulse response of the channel with the modeled device attached. In the EDA tool, this modified impulse response can be converted to a pulse or step response for statistical analysis. In some analysis flows, this modified impulse response is used to help generate time domain waveforms. 

   *This phase of execution is also performed by the AMI_Init() function.*

3. **Time Domain Simulation (optional)**
   Input waveform segments are presented to the model and the model modifies them to produce output waveform segments. In the EDA tool, these waveform segments can be combined and/or used for performance analysis. Receiver models also produce clock times that can be used to trigger the accumulation of the eye diagram.

   *This phase of execution is performed by the AMI_GetWave() function.*

   AMI_GetWave() is almost always called many times in a single time domain simulation.

4. **Destruction (optional but Recommended)**
   The dynamic memory for the model is de-allocated. Other actions related to the destruction of the model may be performed either by the model or the EDA tool. If this phase of execution is not performed, then the dynamic memory for the model will remain allocated until the end of the analysis run. Since the analysis run usually ends shortly after the algorithmic models have been destroyed, nothing very bad will happen if this phase is not executed. However, explicitly de-allocating dynamic memory is good programming practice.

   *This phase of execution is performed by the AMI_Close() function.*

### 3.5 Control Parameters and Messages

In addition to the data path (impulse response or waveform) signals and clock path (clock times) values returned by the model, there is a control path interface for controlling the configuration of the model and observing its state.

- Input parameters are provided to the AMI_Init() function in the form of a character string so that it can set the model in a configuration chosen by the user.

- Both the AMI_Init() and AMI_GetWave() functions can return output parameter character strings which indicate the state or status of the model. The output parameters can, for example, report the state of control loops in the model.

- The AMI_Init() function can return a message which may contain useful information about the model.

Each input and output parameter has a name and a value, and the parameters are organized into a tree or outline structure that has one or more levels. The parameter names, value types and tree structure depend entirely on the model, thus allowing the model developer to define controls and outputs that are uniquely suited to the model.
There is a specific, LISP-like syntax defined for the input and output parameter strings. This syntax is straightforward to read, even if one does not know the details of the syntax; and some EDA tools can read this syntax directly as well. These parameter strings are written by the model developer, the algorithmic model, and at least some EDA tools. Users should not be expected to write parameter strings directly, although that may vary between EDA tools.

In order for the EDA tool to form correct parameter strings for a given model, and to help assure that the EDA tool runs the model correctly, the model must be delivered with a separate ASCII file which declares the parameters for the model. This file is called an AMI file, and the file name usually has the extension “.ami”. The syntax for the AMI file is the same as that for the parameter strings themselves, and it is usually in the AMI file that the user may find it necessary to read this syntax.

For each parameter in the AMI file, the declaration contains much the same information as a declaration in any programming language:

- **Usage**
  Declares whether the parameter is an input to the model, an output of the model, or information for use by the EDA tool.

- **Format**
  States the range of valid values for the parameter. The valid values may be specified as being a constant, a member of a list, an element of a uniformly spaced array, or any value within a specified range. There are also a number of special purpose formats.

- **Type**
  States the type of value that the parameter can have. The supported types are integer, floating point, floating point scaled to the bit time (UI), floating point in an array (Tap), boolean, or string.

- **Default**
  States the default value of the parameter when there are multiple valid values the parameter can have.

- **Description**
  Describes the meaning of the parameter.

The content of the AMI file is organized into two branches: Reserved_Parameters and Model_Specific. The names and meanings of the parameters in the Reserved_Parameters branch are defined in the IBIS standard and contain parameters to be used by the EDA tool to make sure that it runs the model correctly. The parameters in the Model_Specific branch of the AMI file are defined entirely by the model developer.

There are two reserved parameters that users should be familiar with because they directly affect the ways in which the model can be used. They are Init_Returns_Impulse and GetWave_Exists.

When Init_Returns_Impulse is True, then the model can be expected to modify its input impulse response so that the output impulse response includes the response of the model.
Models which do this produce results that can be used for statistical analysis. When Init_Returns_Impulse is False, then the model will cause statistical analysis to produce results which do not reflect the effect of equalization, thus eliminating a valuable option for performance analysis.

When GetWave_Exists is True, then an AMI_GetWave() function is supplied with the model. This is, for example, the only way a receiver model can include a model of the clock recovery loop and therefore supply valid clock times to the time domain performance analysis. If GetWave_Exists is False, then no AMI_GetWave() function is supplied with the model, and the model can only represent the linear response of the data path. This can be an appropriate choice for the algorithmic model of a transmitter, but is not an appropriate choice for a receiver model unless jitter parameters describing the clock recovery behavior are included with the model.

### 3.6 A Complete IBIS-AMI Model

A complete IBIS-AMI model consists of at least the following three files:

1. IBIS file containing the analog model and [Algorithmic Model] statements referring to the other files.
2. DLL or shared object file.
3. AMI file

If one of these files is missing, the model is incomplete and cannot be run.

### 4.0 Demonstration Environment

To avoid commercial considerations that would detract from the information presented in this tutorial, all demonstrations in this tutorial use software that is publicly available, free of charge. For the participants’ convenience, either the software itself or links to the software have been collected at one location [6]. Manpages for many of the utilities are included as well.

Note that the models used in the demonstrations are actual vendor models. These models may or may not be freely available and are not included in the demonstration software package. We will not state the name of a vendor or model in any of the demonstrations, and wherever we show syntax, it’s representative of the problem but does not show the details of the actual model.

There are a number of steps in the interaction between an AMI model and the EDA tool it runs on. These steps are

1. Generate the impulse response of the channel.
2. Generate the stimulus.
3. Prepare the parameter string which configures the model.
4. Compute the impulse response of the passive interconnect channel.

5. Load and run the model.

6. Accumulate results.

7. View the results.

Both for convenience in collecting the demonstration software and to help participants understand the interaction between an AMI model and its environment, each of these steps is performed by a separate executable. Each of these executables is described in a separate subsection below.

While the demonstration software suite could be seen as a free signal integrity analysis environment, its purpose is solely to demonstrate specific features of AMI models that users should be aware of. Furthermore, by studying this software, participants may gain insight into the structure and algorithmic content of an EDA tool. However, this software suite is not suitable for product development.

### 4.1 Demonstration Execution

To the extent possible, the elements of the demonstration software suite were written in such a way that they can be piped together in a single command line. This eliminates unnecessary storage of intermediate results and speeds execution. For example, the command line

```
IBIS_AMI_prbs -f test_config.csv | IBIS_AMI_test -f IBIS_ATM_tx.dll -i tx_config.csv -g -c | IBIS_AMI_logger -c guide.csv -f lumber.csv -n -w waves.csv
```

Generates a pseudorandom sequence described in test_config.csv, passes that waveform through the model IBIS_ATM tx with the impulse response and configuration contained in tx_config.csv, records the model output parameters in lumber.csv and an extract of the output waveform in waves.csv.

To enable this mode of operation, many of the utilities use STDIN as their default input and STDOUT as their default output. There are command line options which over-ride these defaults.

The configuration file for IBIS_AMI_test contains the input parameters for the model and the input impulse response. These two pieces are obtained from separate programs and combined manually. Usually it’s most convenient to combine the input parameters and the impulse response generation into a single spreadsheet dedicated to the test to be performed, thus allowing input parameters and channel parameters to be read and modified in one location.

The typical steps in a demonstration are

1. In a spreadsheet program, set or verify the configuration of the data generator.
2. In a spreadsheet program, set or verify the model input parameters in the test configuration file.

3. While in the same spreadsheet, set or verify the channel parameters in the test configuration file.

4. Invoke an execution script file. This file contains a single piped command line as described above.

5. Copy the output impulse response and time domain waveform segment into the viewer spreadsheet. The viewer spreadsheet converts the impulse response to a step response and pulse response, prepares the eye diagrams, and plots the results.

6. Open the output parameters spreadsheet and plot the parameters as desired.

### 4.2 Stimulus

Time domain stimulus can be generated using the program IBISAMI_prbs. This program has a single command line option, -f, which specifies the test configuration file.

The test configuration file is CSV formatted. All argument values are placed on the same line as the name of the argument and separated from the name by a single comma (i.e., adjacent spreadsheet cell on the same row). Unrecognized text is ignored with no further action whatsoever. The keywords and default values are

- **register_length** 7
- **sample_interval** 25 ps
- **bit_time** 200 ps
- **stop_time** 1 us

IBISAMI_prbs generates an output file containing the sampled data waveform of the data pattern. The first lines of the output file are

```text
* Created by IBIS AMI PRBS Generator
Title PRBS Data Pattern
*,register_length,<7, 15, or 22>
*,sample_interval,<sample interval>
*,bit_time,<bit time>
*,stop_time,<stop time>
Time,wave_in
```

Subsequent lines contain the time and value for one sample.

If the register length has been set to zero, the program expects a user defined data pattern piped from STDIN. A ‘1’ in the file results in the output of a one in the data pattern. A ‘0’ in the file results in the output of a zero in the data pattern. A ‘*’ indicates a comment, in which case the rest of the line, including ‘1’s and ‘0’s, is ignored. All other characters are completely ignored.
4.3 Input Parameters

While it is possible to write the input parameter string for a model directly, this requires careful attention to syntax and the result may or may not be easy to read and modify. It’s much easier to read and modify parameters in a table or spreadsheet.

ami2csv is a utility which reads an AMI file and creates from it a text file and a CSV file. The text file presents an outline list of the contents of each parameter tree in the file. The CSV file presents the structure and default values for the Model_Specific section of each parameter tree in a format which can be inserted directly in an impulse response input file for the test program IBIS_AMI_test.

ami2csv should only be run once for any given AMI file. After that, the parameters should be read and modified in the test configuration CSV file.

4.4 Circuit Solution

As shown in Figure 3, the channel used in these demonstrations consists of a length of lossy differential transmission line driven by a source resistance with parasitic capacitance and terminated by a load resistance and parasitic capacitance. This relatively simple circuit is sufficient to introduce realistic transmission loss and reflections as needed. For the sake of clarity, it is assumed that transmission is entirely differential.

![Circuit Diagram](image)

**FIGURE 3. Demonstration channel**

This circuit was solved using general circuit parameters [7] and coupled transmission line equations [8]. While the solution is expressed entirely as closed form equations, these equations are clearer if they are broken into multiple steps. The steps are

1. Compute the transmission line series and shunt impedances as a function of frequency.
2. Compute the transmission line’s differential propagation constant.
3. Compute the transmission line’s general circuit parameters.
4. Compute those general circuit parameters for the entire channel that are required to compute the voltage transfer function.
5. Invert the matrix of general circuit parameters to get the voltage transfer function.
6. Apply the inverse discrete Fourier transform \([9], [10]\) to obtain the impulse response in the time domain.

These steps are all implemented in the spreadsheet channel.xls. The first page of this spreadsheet can be output as a CSV file and used as the input impulse response for the AMI model.

Since none of the techniques mentioned in this section are a fundamental aspect of evaluating AMI models, we will include only minimal explanation of the derivation of the equations.

From \([11]\), the complex valued dielectric constant, including the effects of loss tangent, is

\[
\varepsilon_r(\omega) = \varepsilon_r \left(1 + \frac{2 \tan \delta}{\pi} \ln \left(\frac{\omega_2 + j \omega}{\omega_1 + j \omega}\right) \right)
\]  

(EQ 1)

where \(\omega_1\) is a lower frequency limit, typically \(2\pi \times 10^4\) Hz, and \(\omega_2\) is an upper frequency limit, typically \(2\pi \times 10^{12}\) Hz.

The shunt admittance per unit length therefore becomes

\[
Y(\omega) = j \omega \varepsilon_r(\omega) \frac{1}{Z_{0o} c}
\]  

(EQ 2)

where \(Z_{0o}\) is the odd mode (differential) impedance and \(c\) is the speed of light in a vacuum. Later equations will also use the even mode (common mode) impedance \(Z_{0e}\). For edge coupled stripline, \(Z_{0o}\) and \(Z_{0e}\) can be calculated using the equations found in \([12]\).

The series impedance per unit length is a combination of the inductance per unit length due to the magnetic fields external to the signal conductors and the resistance and inductance due to currents flowing inside the signal conductors, the so-called internal impedance. As explained in \([13]\), a good approximation for the internal impedance is

\[
R_i(\omega) + j \omega L_i(\omega) = \sqrt{(R_{hf} + j \omega L_{hf})^2 + R_{DC}^2}
\]  

(EQ 3)

where
and $p$ is the perimeter of the signal conductor.

The total series impedance per unit length is therefore

$$Z(\omega) = Z_e(\omega) + Z_i(\omega) = j\omega \sqrt{\epsilon_r(\omega)} \frac{Z_0}{c} + j\omega L_i(\omega) + R_i(\omega)$$  \hspace{1cm} (EQ 5)

and the differential propagation constant is

$$\gamma = \alpha + j\beta = \sqrt{ZY}$$  \hspace{1cm} (EQ 6)

**FIGURE 4. Differential transmission line voltages and currents**

Given the node voltages and currents defined in Figure 4 and the propagation constant from equation 6, the equations which describe a differential transmission line of length $l$ are

$$V_1 = V_3 \cosh(\gamma l) - I_3 \frac{Z_{0e} + Z_{0o}}{2} \sinh(\gamma l) - I_4 \frac{Z_{0e} - Z_{0o}}{2} \sinh(\gamma l)$$  \hspace{1cm} (EQ 7)

$$I_1 = V_3 \left( \frac{1}{2Z_{0e}} + \frac{1}{2Z_{0o'}} \right) \sinh(\gamma l) + V_4 \left( \frac{1}{2Z_{0e}} - \frac{1}{2Z_{0o'}} \right) \sinh(\gamma l) - I_3 \cosh(\gamma l)$$  \hspace{1cm} (EQ 8)

$$V_2 = V_4 \cosh(\gamma l) - I_4 \frac{Z_{0e} + Z_{0o}}{2} \sinh(\gamma l) - I_3 \frac{Z_{0e} - Z_{0o}}{2} \sinh(\gamma l)$$  \hspace{1cm} (EQ 9)

$$I_2 = V_4 \left( \frac{1}{2Z_{0e}} + \frac{1}{2Z_{0o'}} \right) \sinh(\gamma l) + V_3 \left( \frac{1}{2Z_{0e}} - \frac{1}{2Z_{0o'}} \right) \sinh(\gamma l) - I_4 \cosh(\gamma l)$$  \hspace{1cm} (EQ 10)

For the sake of clarity, define

$$A \equiv \cosh(\gamma l)$$  \hspace{1cm} (EQ 11)
Therefore, solving the circuit in Figure 3,

\[
V_1 = \left( (1 + j\omega R_SC_S)A + R_SC_1 + ((1 + j\omega R_SC_S)B_1 + R_SA)\frac{(1 + j\omega RLCL)}{RL} \right) V_3 + \\
\left( R_SC_2 + ((1 + j\omega R_SC_S)B_2)\frac{(1 + j\omega RLCL)}{RL} \right) V_4
\]

\[
V_2 = \left( R_SC_2 + ((1 + j\omega R_SC_S)B_2)\frac{(1 + j\omega RLCL)}{RL} \right) V_3 + \\
\left( (1 + j\omega R_SC_S)A + R_SC_1 + ((1 + j\omega R_SC_S)B_1 + R_SA)\frac{(1 + j\omega RLCL)}{RL} \right) V_4
\]

(EQ 16)

This pair of linear equations must be solved to get the differential output voltage \((V_3 - V_4)\) in terms of the input differential voltage \((V_1 - V_2)\):

\[
V_3 - V_4 = \frac{V_1 - V_2}{(1 + j\omega R_SC_S)A + R_SC_1 + ((1 + j\omega R_SC_S)B_1 + R_SA)\frac{(1 + j\omega RLCL)}{RL}}
\]

(EQ 17)

### 4.5 Model Execution

IBIS files and AMI files are text files and so, at least in theory, they can be verified by inspection. However, a DLL or shared object library contains compiled object code; and so the contents of these files cannot be inspected directly. The IBIS_AMI_test utility was donated to the IBIS ATM subcommittee in July 2007 as a public stand-alone utility for verifying that the DLL or shared object library for a model correctly implements the IBIS-AMI interface. It tests four elements of the interface:

1. Load DLL or shared object library.
2. AMI_Init() function
3. AMI_GetWave() function
4. AMI_Close() function

By making inputs and outputs directly visible to the user, IBIS_AMI_test makes it possible to test each of the elements of the interface. These elements include

- Input impulse response
- Output impulse response
- Input parameter string
- Output parameter string
- Output message
- Input waveform
- Output waveform
- Clock times
- Function return value

The command line arguments are designed to support independent testing of individual functions. Thus, if only the -i option is used, then the AMI_Init() function is the only one called and tested. If both the -i and -g options are invoked, then both the AMI_Init() and AMI_GetWave() functions are tested; and if all three options (-i, -g and -c) options are invoked, then AMI_Init(), AMI_GetWave(), and AMI_Close() are tested. Other combinations can be invoked if the dependencies will be met for the function(s) being called.

Function arguments are supplied in CVS files, with the argument value placed in the cell(s) beside the argument name. Model input parameters are supplied in an outline list form and converted to the parameter tree string syntax by IBIS_AMI_test. Comment lines are supported.

Results are stored in files whose names are derived from the input file name. For example, if the input file is froboz.csv, then the output file will be froboz_out.csv. The default input file is STDIN, with the corresponding output file STDOUT.

The command line arguments are

- **-f <file>**
  Load the dynamically loadable module found in file. Only one module will be loaded, and only the functions AMI_Init(), AMI_GetWave(), and AMI_Close() will be loaded from that module. Functions which are not loaded successfully will be noted with a WARNING message, but will have no other effect except for any effects on subsequent function calls.

- **-i <file>**
  Execute the AMI_Init() function using the arguments found in file. file can be omitted, in which case the default value is stdin.
<file> is assumed to be CSV-formatted, with arguments named as shown in the IBIS AMI API specification and with argument values placed in the cells beside the argument name. For pointer arguments, the contents of the pointer are placed in the cell beside the pointer name. If the pointer points to an array of numerical values, then that array is placed underneath the pointer name, occupying multiple columns for two dimensional arrays.

-g <file>
Execute the AMI_GetWave() function using the arguments found in file. file can be omitted, in which case the default value is stdin. The file format is the same as for the -i option.

-c
Execute the AMI_Close() function.

-n <block_size>
Number of samples per block to be used when running AMI_GetWave(). Default is 1024.

4.6 Accumulate results

While it is possible to view the results from an IBIS_AMI_test time domain simulation directly, this typically isn’t very practical because of the volume of data and the fact that the output parameters are not separated from the output time domain waveform.

IBIS_ATM_logger is a parameter logger for testing AMI_GetWave functions. It accepts input from STDIN and repeats it to STDOUT without modification. It can also output a segment of the waveform to a separate file.

More importantly, IBIS_ATM_logger looks for parameter strings in the input, modifies the format of those strings to CSV, and outputs them to a separate file. Since IBIS_ATM_test echos the parameter string once each time it calls AMI_GetWave, piping to IBIS_ATM_logger will result in a CSV-formatted record of the parameter values that can then be loaded into a spreadsheet program for further analysis.

IBIS_ATM_logger also collects the statistics from the T_clk output of the AMI_GetWave functions. The modulo of each clock tick relative to the bit time is computed. A moving average is then taken over the last 1000 clock ticks, and the statistics of the difference between each clock tick and this average are accumulated, resulting in an average clock timing (at the edge of the eye) and a probability density function (PDF) for the clock phase noise. The average clock timing is output in the log file and the PDF is output as a separate file.

The command line arguments for IBIS_AMI_logger are

-c <file>
File containing record_start and record_stop control parameters. NOTE: This command line option is mandatory.
The test configuration file is CSV formatted. All argument values are placed on the same line as the name of the argument and separated from the name by a single comma (i.e., adjacent spreadsheet cell on the same row). Unrecognized text is ignored with no further action whatsoever. The keywords and default values are

```
sample_interval 25 ps
record_start 0.9 us
record_stop 1 us
```

```
-f <file>
```

File in which to write the parameter output strings.

The first two lines of the parameter output file are

```
*Created by IBIS ATM Parameter Logger
Title Output Parameters
```

Lines which start with the character “*” are interpreted as comment lines and echoed to the output file. The next line starts with the word “Time”, followed by a name for each parameter value. The first parameter is always the average clock timing as derived from the clock ticks in the input and has the label “clock_avg”. Subsequent lines contain the sample time followed by one sample for each parameter. The parameter output string is detected by the presence of an open parenthesis. The result is a CSV file in which each parameter is named by a dot delimited concatenation of its parameter group names and the parameter name itself. Each parameter is therefore identified uniquely, and its value can be plotted as a function of time.

There is also a clock PDF output file. The clock PDF file name is the base name of the output file with “_pdf” appended to it. The file extension is preserved.

The first two lines of the clock PDF file are

```
*Created by IBIS ATM Parameter Logger
Title Clock PDF
```

Lines which start with the character “*” are interpreted as comment lines and echoed to the output file. The next line starts with “*Clock time standard deviation,” followed by the standard deviation of the clock times accumulated over the course of the simulation. The next line is

```
Time,Clock PDF
```

followed by a line for each offset and the probability for that offset. The scale is set at the beginning of the run, but is intended to be about +- ten sigma.

```
-n
```

No echo. Do not echo the input to STDOUT.
-t <testfile>
Obtain input from testfile instead of stdin. Primarily a way to test the program.

-w <file>
File in which to write the waveform segment.

The first lines of the waveform output file are

*Created by IBIS ATM Waveform Logger
Title Output Waveform
*,sample_interval,<sample interval>
*,record_start,<record start time>
*,record_stop,<record stop time>
* AMI_dll_parameters_out,<AMI_Init parameter output string>
Time,wave_in,Tclk(N),AMI_dll_parameters_out

Subsequent lines contain one sample each of the sample time, output waveform value, and
clock tick. For each sample block (as defined through the IBIS_ATM_test command line
option “-n”), the first line contains the parameter output string. Output clock ticks for the
sample block are output at the beginning of the block, followed by zero values.

4.7 View Results

The following results are available directly from the test configuration spreadsheets and
execution utilities:

• Input impulse response
• Channel transfer function
• Output impulse response
• Input waveform
• Output waveform
• Output parameters
• Clock PDF

In addition, it would be useful to have

• Output step response
• Output pulse response
• Statistical eye diagram
• Persistent (time domain) eye diagram

Of these outputs, the step response is the integral of the impulse response, and therefore
quite easy to compute. The rest of these outputs depend on the number of samples per bit,
which is a quantity that will be varied in some of the demonstrations. These outputs are
more difficult to compute in a spreadsheet primarily because the spreadsheet language has no concept of pointers, and so it isn’t possible to have a variable index into an array.

Rather than write a compiled utility to compute the pulse response and eye diagrams, we chose to write a viewer spreadsheet that has a separate set of sheets for each number of samples per bit that is to be used in a demonstration.

The equation for the pulse response is

\[ p[kt] = \sum_{i = k}^{k+n} \tau \delta[i] \]  

(EQ 18)

where \( n \) is the number of samples per bit, \( \tau \) is the sample interval, and \( \delta \) is the impulse response.

While it is not practical to compute a statistical eye in a spreadsheet, it is practical to compute the extreme inner and outer contours of the eye diagram using peak distortion analysis [4]. Given that the pulse response peaks at index \( m \), the inner and outer contours of the statistical eye are

\[ x[kt] = \frac{1}{2} \left( p[kt] - \sum_{i \neq 0} |p[(k + i \cdot n)t]| \right) \quad m - \frac{n}{2} \leq k \leq m + \frac{n}{2} \]  

(EQ 19)

\[ X[kt] = \frac{1}{2} \left( p[kt] + \sum_{i \neq 0} |p[(k + i \cdot n)t]| \right) \quad m - \frac{n}{2} \leq k \leq m + \frac{n}{2} \]  

(EQ 20)

In the spreadsheet, the \( i \neq 0 \) condition is satisfied by subtracting \( p[kt] \) from \( \sum_{i} |p[(k + i \cdot n)t]| \).

The persistent eye is plotted as the overlay of individual segments of the time domain waveform.

For both the statistical eye PDA contours and the persistent eye, no attempt is made to recover or use the clock phase, again because of the static nature of spreadsheet indexing. Instead, the eye diagram is plotted 2 UI wide.

4.8 Other Utilities

ibischk5 [14] is a parser which checks the syntactic correctness of not only IBIS files but AMI files as well. It’s very useful for verifying an AMI model’s compliance to IBIS 5.0 [1].
Dependency Walker [15] is a Windows utility that determines which DLLs must be resident in order for a given program or DLL to run. For Linux, equivalent information can be obtained using the system command `nm`.

### 5.0 No Documentation/No-one to Talk To

AMI models are seldom distributed with useful documentation. Usually, the only file that resembles documentation is the AMI file; and that only tells you what the simulator knows. This is not enough information to make sure that the model can be relied upon to produce results that lead to correct engineering decisions.

Furthermore, it is often difficult to find a vendor representative who can provide useful information or help resolve a problem.

The net result is that lack of documentation has a serious negative impact on every phase of the user’s experience, and is the single most critical problem in AMI modeling. This is a recurring theme in the problem scenarios and demonstrations in this tutorial.

One of the better documents actually distributed with a vendor’s AMI model is a 4k README file that has the following outline:

- Copyright statement
- I. Name
- II. Parameters and controls
- III. Limitations
- IV. Additional information

There is one consulting company that supplies a high level design document with every AMI model they develop for an IP vendor. Generally, these high level design documents contain the following information:

- Parameter definitions (input and output)
- Block diagram
- Algorithmic description
- Design decisions

This may or may not be enough information to support a model, but this information seldom if ever appears in any user documentation that the IP vendor distributes with the model.

Opal™[3] recommends the following documentation outline:

- 1.0 Description
  - 1.1 Device Modeled
  - 1.2 Model Contents
  - 1.3 Analog Model(s)
  - 1.4 Opal™ Compliance
- 2.0 Installation
6.0 Can’t Install Model

When installing an AMI model, and before attempting to run it, the user should run ibischk5 [14]. While this procedure is not foolproof, it does catch most syntax errors and makes sure that files named in the IBIS file are actually available (see Section 3.6 on page 12).

For example,

```
[localhost ibis]$ ibischk5 xyz.ibs
IBISCHK5 V5.0.6
Checking xyz.ibs for IBIS 5.0 Compatibility...
ERROR - Code file xyz_rx_win.dll not found. It was defined in [Algorithmic Model] for Model xyz rx
Checking xyz_rx.ami for Compatibility...
```

In this case, ibischk5 detected that the DLL file xyz_rx_win.dll was missing. This error can be resolved by copying the file xyz_rx_win.dll into the same directory as the IBIS file.

ibischk5 often issues warnings concerning file names that are not all lower case. Having upper case letters in the file names does not usually affect the running of the model.

7.0 Model Doesn’t Run

7.1 Missing supporting files

The following demonstration was run on a model whose IBIS file passed ibischk5:

```
[localhost Linux]$ ./IBIS_AMI_prbs -f rx_config16.csv | ./IBIS_AMI_test -f ./xyz_rx.dll -i rx_config16.csv -g -c &> rx_results.csv
Segmentation fault
```

The problem is that the directory include_files/ had been moved to not_include_files/. Moving the directory back fixes the problem. The DLL xyz_rx.dll was written to assume
that the directory include_files/ would be present in the directory that the simulation was run in, and that directory would contain certain files. When those assumptions were violated, the model crashed, taking the simulation with it. There were no error messages.

There are a number of AMI models that depend on auxiliary files, and many depend on having those auxiliary files in a specific location. When installing these models, it’s important to make sure that the auxiliary files are copied along with the primary files required by IBIS 5.0. If the right auxiliary files are not in the right locations, the model will very likely crash without leaving behind any messages to indicate what happened.

Another variant of this problem is that there are models which run on a separate executable such as a commercial analysis program. When the EDA tool calls the model, the DLL invokes this separate executable as a separate process. Several problems have been encountered with this arrangement:

- The executable was not installed on the machine that was trying to run the simulation.
- The executable was installed, but there was a problem with the licensing.
- The model needed to run on a different version of the executable than the one that was installed on the machine.
- The simulation was trying to run two models: one required one version of the executable and the other model required a different version.

**Documentation:** The documentation distributed with the IBIS-AMI model should list all auxiliary files that are required, and state specifically where those files need to be located.

### 7.2 Environment dependencies

Consider the following demonstration:

```
C:\PeachLemon\test>IBIS_AMI_test -f IBIS_AMI_Rx.dll -f tx_impulse_no_eq.csv -g waveform.csv -c
ERROR: Failed to load dynamically loadable module IBIS_AMI_Rx.dll
* ERROR: Unable to load module. Aborting.
C:\PeachLemon\test>
```

The error message makes it clear that the DLL did not load, but it does not provide any additional information that would point toward a root cause.

The test case runs successfully on some machines but fails, as above, on other machines with exactly the same operating system. The module does exist in the directory from which the demonstration is run, and the model developer states that they have tested the model. Furthermore, the model was delivered without any support files and the model developer states that the model does not need any support files.

The result of invoking the Dependency Walker utility [15] is shown in Figure 5.
FIGURE 5. Dependency Walker screen shot

The missing DLL is a Microsoft C++ DLL, but it is not the standard one that is typically resident in the Windows operating system. The “D” at the end of the DLL’s base file name indicates that it is the debug version that is included with Microsoft Visual Studio, but is not part of the standard Windows installation. Microsoft has more information on their Visual Studio web site [16]:

FIGURE 6. Microsoft Visual Studio web site

This problem was caused by the way in which the AMI model’s DLL was compiled. Almost every compiler, including the one in Microsoft Visual Studio, offers the software developer the option of including debug symbols in the compiled object files. When debugging and testing the software, these debug symbols make it possible to relate loca-
tions in the compiled object code back to line numbers in the source code. This information is essential to the debug process. Once the software has been tested, however, the software should be compiled and distributed without debug symbols, both to make the software run much faster and to make the file size of the distributed product smaller. In this case, the model developer failed to follow this procedure.

In addition to missing libraries, there are obvious dependencies on operating system and processor bus width. These are addressed by the “Executable” subparameter of the [Algorithmic Model] statement in the IBIS file. Nonetheless, these dependencies can be a problem. For example, there are models that are only available for Windows and other models that are only available for Linux. Thus there is no operating system that can run both sets of models.

Bus width tends to pose more subtle problems. While AMI model simulations never come close to using all of a 32 bit address space, some users still seem to insist on having 64 bit models. This poses a problem for the EDA tools in that although an EDA tool compiled for 32 bits can run on a 64 bit platform, it cannot load and run a 64 bit model. Similarly, an EDA tool compiled for 64 bits cannot load and run a 32 bit model. This again limits the combinations of models that can be run. In this case, however, the problem could be easily avoided with no disadvantages to anyone by using only 32 bit EDA tools and models.

If an EDA tool compiled for one bus width tries to load and run a model compiled for the other bus width, the problem can often be difficult to diagnose. The model will fail to load and run, but the error message will depend on the platform. Some platforms provide no error message at all, while other platforms provide a cryptic error message such as

*Wrong ELF type.*

If a model fails to load and run, and no meaningful error message is given, look for an environment dependency such as a missing library or a bus width incompatibility.

### 7.3 Performance

There are some models which run so slowly that it isn’t clear that they’re running at all. One of the earliest AMI models took four hours to simulate two million bits. Although the worst performers have since been dramatically improved, there is still a substantial difference in the run times for various models.

Table 1 shows the results of a recent experiment. Models from vendor A, vendor B, and a set of generic models. The models from vendor A and vendor B have similar complexity while the generic models are somewhat simpler. The models from vendor A could only be run at 32 samples per bit, while the models from vendor B and the generic models could be run over a wide range of samples per bit. The generic models were run with both 32 samples per bit and 8 samples per bit, but with the same number of bits, to get some estimate of the time consumed by the simulation environment.
TABLE 1. Run time comparison between three models

<table>
<thead>
<tr>
<th>Model Description</th>
<th>Simulated bits</th>
<th>Samples/bit</th>
<th>Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor A</td>
<td>3500000</td>
<td>32</td>
<td>4:13:00</td>
</tr>
<tr>
<td>Vendor B</td>
<td>3500000</td>
<td>32</td>
<td>1:23:00</td>
</tr>
<tr>
<td>Generic</td>
<td>3500000</td>
<td>32</td>
<td>1:11:00</td>
</tr>
<tr>
<td>Generic</td>
<td>3500000</td>
<td>8</td>
<td>0:40:00</td>
</tr>
</tbody>
</table>

From Table 1, it is clear that there is still significant variation in the run time for different models. Beyond the concern over whether the simulation is running or not, this difference in run time can affect the usability of the model. For example, there are engineering studies that can be performed using the models from vendor B that would not be practical using the models from vendor A.

8.0 Results are Obviously Incorrect

8.1 Samples per Bit

IBIS 5.0 does not state any constraints on either the sample interval or the ratio of the bit time to the sample interval; that is, samples per bit. The only constraint on the number of samples per bit is the Nyquist sampling theorem. Nonetheless, some AMI models have considerably tighter requirements.

Figure 7 shows a segment of a time domain waveform from a receiver model for different values of samples per bit. For comparison, Figure 8 shows the same waveform segment for another receiver model.
While the output waveforms for the second receiver model varies somewhat as a function of samples per bit, the output waveforms for the first receiver model indicate a strong sensitivity to samples per bit.

As is evident from the discontinuities in both sets of waveforms, both receivers include decision feedback equalization (DFE). For the first receiver, the variation with samples per bit appears to be related to these discontinuities, and therefore to the behavior of the DFE.
The root cause for the sensitivity to samples per bit is explained by looking at the clock times returned by the models. Figure 9 shows the clock times for the first receiver model as a function of samples per bit while Figure 10 shows the same result for the second receiver model.

**FIGURE 9. Vendor receiver model clock times vs. samples per bit**

**FIGURE 10. Generic receiver model clock times vs. samples per bit**

For the first receiver model, the clock recovery model is only functioning correctly for 32 samples per bit. The slope of clock time vs. bit number (i.e., the clock period) for the other values of samples per bit suggest that 32 samples per bit has been hard coded into some portion of the clock recovery model.
Conversely, the second receiver model returns essentially the same clock times under all conditions.

The conclusion is that the first receiver model is only valid if the sample interval for the simulation is set to 32 samples per bit. This is not unusual. There is another receiver model that has been in use for a long time that requires 48 samples per bit. There is yet another receiver model that requires the sample interval to be an integer multiple of 1pS.

Models that impose constraints on the sample interval can limit the types of analyses that can be performed. For example, a simulation incorporating one model that requires 32 samples per bit with another model that requires 48 samples per bit can only be performed in an EDA tool that can convert waveforms from one sample interval to another. Such EDA tools do exist, but an EDA tool does not have to have this feature in order to comply with IBIS 5.0.

**Documentation:** At the very least, any requirements imposed on the sample interval should be documented in a form that is clear and readily accessible to the user. This assumes that the vendor understands the limitations of their model in the first place, and that they supply documentation with their model. There have been a number of instances in which neither assumption was valid.

### 8.2 Block Size

One of the variables that is under user control is the length of a waveform segment during a time domain simulation. This is referred to as the block size. Block size has some effect on the amount of memory a simulation takes and, to a much lesser extent, the amount of time it takes to run the simulation. Smaller block size produces a smaller memory footprint, but also causes the simulation to incur the overhead of calling AMI_GetWave() more often. When the model outputs parameters, the block size also determines the granularity with which the parameters are sampled.

Figure 11 shows the input and output waveforms for a receiver model running with a block size of 1024.
FIGURE 11. Receiver time domain waveforms with a block size of 1024

In a time domain simulation, the data at the output of the AMI model is expected to match the data at its input. However, in the case shown in Figure 11, the output data is clearly different from the input data. For example, in the input data, a long string of “zeros” is followed by a single “one”; whereas in the output data, a somewhat shorter string of “zeros” is followed by at least four “ones”. After some investigation, it was found that the behavior of the model was a function of block size.

Figure 12 shows the input and output waveforms when the model is run with a block size of 1280.

FIGURE 12. Receiver time domain waveforms with a block size of 1280
In this case, the output data does match the input data. It turned out that the model only produces correct results when the block size is an integer multiple of 20 bits long.

**Documentation:** A dependency on block size should not be considered acceptable. However, if a dependency on block size exists for some reason, that limitation should be clearly stated in the documentation distributed with the model.

### 8.3 Invalid Clock Times

Section 8.1 on page 29 has already provided one example of invalid clock times, in that the clock times depended on the number of samples per bit.

Another example of invalid clock times was encountered when testing a receiver model at multiple data rates. When simulating at 10 Gb/s, the period between clock times was 100 pS, as it should have been. However, when the data rate in the simulation was 6.25 Gb/s, the period between clock times was still 100 pS.

Yet another example of invalid clock times was encountered when testing a receiver model with a sequence of gradually increasing path lengths. Even though the time domain waveform was delayed in a way that was completely consistent with the path delay, the clock times output from the model remained exactly the same. Clearly, there was no function clock recovery in this receiver model.

All of these models were tested by the vendor before delivery to the user, and yet they had very obvious problems in the clock times they output. Furthermore, many of these vendors were not aware that their models were outputting clock times, or that their models should output clock times. As explained in Section 3.1 on page 5, clock times are an essential part of the performance analysis of a high speed serial channel. Not having valid clock times is like not having information on clock distribution delays when analyzing the timing of a logic design.

Invalid clock times appears to be a problem that is far more common than one would normally expect. One hypothesis is that there is a single problem that is common to the test environment for all of these models.

### 8.4 Statistical Analysis vs. Time Domain Simulation

As explained in Section 3.1 on page 5, statistical analysis and time domain simulation are complementary analysis methods, and both are required to produce a reliable performance estimate. Unfortunately, not all models support both analysis methods.

There are those who argue that a model should not support both analysis methods because the model may give different results from the different methods, leaving the user with a challenge to determine what they choose to believe. It is true that having only one answer would be less confusing. It is also true that a model which supports both analysis methods is often two models in one: one model to support statistical analysis and one model to support time domain simulation. However, the fact is that the questions the two analysis
methods address directly are different, and they therefore offer different views of the same problem.

**Documentation:** Perhaps the most serious problem is that some AMI models are distributed without documentation that describes the analysis flows in which the model should be used. Unless the user looks at the reserved parameters in the AMI file (Section 3.5 on page 10), they may run an analysis and use the results when in fact the results are not valid.

### 8.5 Simultaneous Simulations

Most computers used for signal integrity analysis have multi-core processors; and most signal integrity analyses require many cases to be simulated. It is therefore natural that users will want to run multiple simulations simultaneously. If all of these simulations are run from the same directory, as is usually the case, some models can create unexpected problems and/or produce unexpected results. These behaviors and results are usually not reproducible, and so are not suitable for demonstration.

There are two types of problems that can occur in simultaneous simulations:

- **Access to a common resource is denied.**
  If multiple simulations are trying to read the same file at the same time, one of them may be denied access, in which case the simulation usually crashes. Since the simulations are typically not running in their own console window, any console messages are lost and so there are usually no error messages indicating what went wrong.

- **Multiple simulations write to the same file.**
  Some models write to auxiliary files, usually as a way to output information directly from the model. If the name of the auxiliary file is fixed, then multiple simulations may end up writing to the same file. No error messages are generated, so the user has no warning; and yet the results will be confusing at best, and possibly misleading.

**Documentation:** This is another problem for which documentation is an essential part of the solution. The user needs to know if the model uses any auxiliary files and if so, where those files are and how they’re used.

### 9.0 Can I Trust the Results?

So far, this tutorial has presented obvious problems that can be detected by even a minimal inspection of the results. This raises several questions:

1. Presumably, the vendors tested these models. What environment and test conditions did they use to perform these tests?
2. Even if a vendor model does not have obvious problems, are there more subtle problems that might have been missed?
3. Even if the model functions exactly as the model developer intended, does the model fail to include impairments that would materially affect the performance of the system?
AMI models model complex behaviors in channels for which the interaction between circuit elements is complex. For channel designs with minimal performance margin, it’s important to understand the limitations of the models used in the analysis.

### 9.1 Analog Modeling

As mentioned in Section 3.1 on page 5, an AMI model is the combination of an analog model and an algorithmic model, and there are no clear rules for partitioning between these two models. One of the choices is to make the analog model transparent and assign all of the behavior to the algorithmic model. This is an approximation in that it ignores the frequency variation of the circuit’s input or output impedance.

Figure 13 compares the pulse responses for two receiver models driven by a 20 inch interconnect. The first receiver model sets the input capacitance in the analog model to zero, thus ignoring the variation of the receiver’s input impedance with frequency. The second model assumes a receiver input capacitance of 0.5 pF on each input pin. The two receivers have been configured to equalize the channel as well as possible, and the two pulse responses are nearly identical.

![Figure 13. Receiver output pulse response at the end of a 20 inch channel](image)

Similarly, compares the pulse responses of the two receiver models when driven by a 2 inch interconnect. While there is still a great deal of similarity between the two pulse responses, the pulse response from the second model has additional ripples.
FIGURE 14. Receiver output pulse response at the end of a 2 inch channel

The ripples in second receiver’s pulse response are due to standing waves on the interconnect. The wave front from the transmitter is reflected by the receiver’s input capacitance, travels back to the transmitter, where it is reflected by the transmitter’s output capacitance, and then travels back to the receiver.

While these standing waves are not a serious problem when the interconnect is truly only 2 inches long, they can be the primary source of system degradation when two discontinuities separated by a short distance are part of a longer path [17],[18]. Eliminating the receiver input capacitance could cause one of these resonant sections to be overlooked, resulting in system performance that is much worse than was predicted by simulation.

The conclusion is that assigning all of the behavior to the algorithmic model may be a reasonable approximation for lossy paths with no discontinuities, but it can be an unacceptably optimistic approximation when discontinuities such as vias and connectors are separated by short distances.

Even the parasitic input capacitance in an IBIS model is an approximation that may or may not be suitable for a given application. Typically, the reflection coefficient up to 5 GHz or so is well modeled by an input parasitic capacitor. However, at higher frequencies this approximation can become inaccurate because of parasitic inductances and resistances in the termination network. On-die S parameters [19] is another way to model the impedance of a driver or receiver. In this approach, the input or output impedance is specified in an S parameter file as the reflection coefficient as a function of frequency. This approach has the advantage of complete flexibility and the potential for high accuracy. The disadvantage is that a separate file is required.

There has been considerable difference of opinion over whether on-die S parameters uniquely define the reflection coefficient of a driver or receiver. Note, however, that the
very first commercial AMI model used on-die S parameters because the model developers were unwilling to accept the approximation offered by the standard IBIS constructs. This model has been in extensive production use for a long time and has consistently produced results which match the supplier’s internal correlation data. Note also that on-die S parameters can easily be run in a SPICE-type circuit simulator and give correct results.

**Documentation:** The documentation distributed with the AMI model should state how the analog behavior of the circuit is modeled, what the analog model has been correlated to, and any limitations of the analog model.

### 9.2 Performance Budgets

There are impairments which are usually modeled by the EDA tool and external to the AMI model. Jitter budgets are the most typical examples; but shot noise at a receiver input is another example. IBIS 5.0 defines some of the parameters needed, and additional parameters described in other sources [3] have been submitted to IBIS. In order to produce completely accurate and reliable results, the complete impairment budget must be included in the performance analysis.

Transmit jitter is one example for which it is relatively easy to compare model results to measured data. When most AMI transmitter models are simulated driving an ideal load, the eye diagram is extremely clean, and it’s questionable whether any measured eye could ever been that clean. The question, therefore, is to what extent jitter components present in the real system have been left out of the simulation and to what extent the additional impairment is due to noise in the measurement setup. Interestingly enough, there is seldom a characterization of the measurement noise that would turn this into a controlled experiment.

**Documentation:** The documentation distributed with the AMI model should describe any impairments, such as jitter budgets, that are not part of the model but should be included in the performance analysis.

### 9.3 Model Correlation

Most AMI models are correlated to reference data from some source. However, the nature of that data source varies over a wide range. In order of increasing value, these sources are:

1. **Standards Document**
   - Some models only implement the behavior that is defined in a particular standards document. In essence, the model developer is stating that the IP complies with the standard.

2. **Another Simulator**
   - The AMI model is written to reproduce the behavior of another simulator. The other simulator is believed to be correlated to measured data, and so the AMI model is correlated indirectly to measured data. Often the other simulator is a circuit simulation such as a SPICE simulation. In other cases, the simulator was developed internally by the vendor.
3. **Measured Data**

   There has been a recent trend to correlate AMI models directly to measured data. For example, one receiver model was correlated to measured jitter tolerance data to within less than 0.02 UI for bit error rates from $10^{-11}$ to $10^{-5}$. There have also been several transmitter models correlated to measured eye diagrams or waveforms, and there are many other ways to measure receiver correlation data.

   The quality of the correlation data required for a given application depends on the amount of performance margin the system is expected to have. Smaller performance margins require higher quality correlation data.

**Documentation:** The documentation distributed with the AMI model should include a correlation report which states the source of the correlation data, the methods used to perform the correlation study, and the level of correlation achieved.

### 9.4 Levels of Model Quality

There is a wide variation in the quality of AMI models delivered by IP vendors to system developers. While the quality range is continuous, it is useful to identify levels of AMI model quality. The following three definitions have proven useful:

1. **IBIS 5.0 compliant**
   - The model passes ibischk5, loads and runs. The reliability of the results is unknown.

2. **Certified.**
   - Somebody has worked with the vendor to make sure that the model runs as well as could be expected.
   - Someone who knows what they’re doing has looked carefully at results produced by the model and says that the results seem reasonable.
   - Since this quality level depends on results, and results vary between EDA tools, this level is probably limited to a single tool.

3. **Correlated.**
   - Results from the model have been compared to measured laboratory data.
   - A correlation report has been distributed with the model. This correlation report states the test conditions and compares the model results to the measured results.

### 10.0 Conclusion

High speed serial channels involve the complex interaction of many different circuit components. While AMI models hide some of that complexity, modeling high speed serial channels accurately requires meticulous attention to detail. Users should not only make sure that model runs and the results look reasonable; they should make sure they understand what is being modeled and what the limitations of the models are.

Users cannot do their job properly without adequate documentation; and unfortunately the documentation distributed with AMI models is seldom adequate. This is the area in which AMI models are most in need of improvement.
In short, **just because it runs doesn’t mean it’s right.**

### 11.0 Acknowledgement

This tutorial represents the collected experience of many people at SiSoft. Special thanks go to Mike LaBonte, Mike Mayer, and Paul Wildes for helping to prepare most of the demonstrations presented in this tutorial.

### 12.0 References


