****
****

BIRD ID#: ?
ISSUE TITLE: IBIS-AMI Flow Correction
REQUESTER: Walter Katz, Signal Integrity Software, Inc.
Ken Willis, Sigrity, Inc.
Ambrish Varma, Cadence, Inc.
DATE SUBMITTED: August ~~10~~31, 2010
DATE REVISED:
DATE ACCEPTED BY IBIS OPEN FORUM:

****
****

STATEMENT OF THE ISSUE:

Section 2.3 of Section 10 (NOTES ON ALGORITHMIC MODELING INTERFACE AND PROGRAMMING GUIDE) describes a flawed reference flow. While the intent was to support non-LTI algorithms in the AMI_GetWave functions of the AMI models, Step 4 and Step 5, as described in Section 2.3 will only yield correct results with LTI AMI_GetWave algorithms.

Also, while Sections 2.1 and 2.2 allude to the existence of LTI (statistical) and non-LTI (Time Domain) flows, the specification contains only one detailed reference flow in Section 2.3 which does not differentiate between LTI Statistical, LTI Time Domain and non-LTI Time Domain flows.

Also, the IBIS ATM subcommittee, in attempting to incorporate Use_Init_Output into the correct flows concluded that Use_Init_Output added unnecessary complications to the flows, and decided to deprecate this AMI parameter.

****

Replace this text:

| This proposal breaks SERDES device modeling into two parts – electrical
| and algorithmic. The combination of the transmitter’s analog back-end, the
| serial channel and the receiver’s analog front-end are assumed to be linear
| and time invariant. There is no limitation that the equalization has to be
| linear and time invariant. The “analog” portion of the channel is
| characterized by means of an impulse response leveraging the pre-existing
| IBIS standard for device models.

with the following text:

(Due to the high percentage of modified or new text, the changes are not marked by the usual "*" characters at the beginning of each line).

| This proposal breaks SERDES device modeling into two parts - electrical
| and algorithmic. The combination of the transmitter's analog back-end, the
| serial channel and the receiver's analog front-end are assumed to be linear
| and time invariant. The algorithmic model of a Tx model represents the
| signal processing that is performed on the stimulus or input to the Tx
| model. This signal processing is also know as equalization or filtering.
| The "analog" portion of the channel is characterized by means of an impulse

```
| response leveraging the pre-existing IBIS standard for device models. There
| is no limitation that the equalization has to be linear and time invariant.
```

Replace this text:

```
| 2 APPLICATION SCENARIOS
| =======================
|
|
| 2.1 Linear, Time-invariant Equalization Model
| =============================================
|
| 1. From the system netlist, the EDA platform determines that a given
| [Model] is described by an IBIS file.
|
| 2. From the IBIS file, the EDA platform determines that the [Model] is
| described at least in part by an algorithmic model, and that the
| AMI_Init function of that model returns an impulse response for that
| [Model].
|
| 3. The EDA platform loads the shared library containing the algorithmic
| model, and obtains the addresses of the AMI_Init, AMI_GetWave, and
| AMI_Close functions.
|
| 4. The EDA platform assembles the arguments for AMI_Init. These arguments
| include the impulse response of the channel driving the [Model], a
| handle for the dynamic memory used by the [Model], the parameters for
| configuring the [Model], and optionally the impulse responses of any
| crosstalk interferers.
|
| 5. The EDA platform calls AMI_Init with the arguments previously prepared.
|
| 6. AMI_Init parses the configuration parameters, allocates dynamic
| memory, places the address of the start of the dynamic memory in
| the memory handle, computes the impulse response of the block and
| passes the modified impulse response to the EDA tool. The new
| impulse response is expected to represent the filtered response.
|
| 7. The EDA platform completes the rest of the simulation/analysis using
| the impulse response from AMI_Init as a complete representation of the
| behavior of the given [Model].
|
| 8. Before exiting, the EDA platform calls AMI_Close, giving it the address
| in the memory handle for the [Model].
|
| 9. AMI_Close de-allocates the dynamic memory for the block and performs
| whatever other clean-up actions are required.
|
| 10. The EDA platform terminates execution.
|
|
| 2.2 Nonlinear, and / or Time-variant Equalization Model
| =======================================================
|
| 1. From the system netlist, the EDA platform determines that a given block
| is described by an IBIS file.
```

2. From the IBIS file, the EDA platform determines that the block is described at least in part by an algorithmic model.

3. The EDA platform loads the shared library or shared object file containing the algorithmic model, and obtains the addresses of the AMI_Init, AMI_GetWave, and AMI_Close functions.

4. The EDA platform assembles the arguments for AMI_Init. These arguments include the impulse response of the channel driving the block, a handle for the dynamic memory used by the block, the parameters for configuring the block, and optionally the impulse responses of any crosstalk interferers.

5. The EDA platform calls AMI_Init with the arguments previously prepared.

6. AMI_Init parses the configuration parameters, allocates dynamic memory and places the address of the start of the dynamic memory in the memory handle. AMI_Init may also compute the impulse response of the block and pass the modified impulse response to the EDA tool. The new impulse response is expected to represent the filtered response.

7. A long time simulation may be broken up into multiple time segments. For each time segment, the EDA platform computes the input waveform to the [Model] for that time segment. For example, if a million bits are to be run, there can be 1000 segments of 1000 bits each, i.e., one time segment comprises 1000 bits.

8. For each time segment, the EDA platform calls the AMI_GetWave function, giving it the input waveform and the address in the dynamic memory handle for the block.

9. The AMI_GetWave function computes the output waveform for the block. In the case of a transmitter, this is the Input voltage to the receiver. In the case of the receiver, this is the voltage waveform at the decision point of the receiver.

10. The EDA platform uses the output of the receiver AMI_GetWave function to complete the simulation/analysis.

11. Before exiting, the EDA platform calls AMI_Close, giving it the address in the memory handle for the block.

12. AMI_Close de-allocates the dynamic memory for the block and performs whatever other clean-up actions are required.

13. The EDA platform terminates execution.

2.3 Reference system analysis flow

System simulations will commonly involve both Tx and Rx algorithmic models, each of which may perform filtering in the AMI_Init call, the AMI_GetWave call, or both. Since both LTI and non-LTI behavior can be modeled with algorithmic models, the manner in which models are

```
| evaluated can affect simulation results. The following steps are
| defined as the reference simulation flow. Other methods of calling
| models and processing results may be employed, but the final simulation
| waveforms are expected to match the waveforms produced by the reference
| simulation flow.
|
| The steps in this flow are chained, with the input to each step being
| the output of the step that preceded it.
|
| Step 1. The simulation platform obtains the impulse response for the
| analog channel. This represents the combined impulse response
| of the transmitter's analog output, the channel and the
| receiver's analog front end. This impulse response represents
| the transmitter's output characteristics without filtering, for
| example, equalization.
|
| Step 2. The output of Step 1 is presented to the Tx model's AMI_Init
| call. If Use_Init_Output for the Tx model is set to True, the
| impulse response returned by the Tx AMI_Init call is passed
| onto Step 3. If Use_Init_Output for the Tx model is set to
| False, the same impulse response passed into Step 2 is passed
| on to step 3.
|
| Step 3. The output of Step 2 is presented to the Rx model's AMI_Init
| call. If Use_Init_Output for the Rx model is set to True, the
| impulse response returned by the Rx AMI_Init call is passed
| onto Step 4. If Use_Init_Output for the Rx model is set to
| False, the same impulse response passed into Step 3 is passed
| on to step 4.
|
| Step 4. The simulation platform takes the output of step 3 and combines
| (for example by convolution) the input bitstream and a unit
| pulse to produce an analog waveform.
|
| Step 5. The output of step 4 is presented to the Tx model's AMI_GetWave
| call. If the Tx model does not include an AMI_GetWave call,
| this step is a pass-through step, and the input to step 5 is
| passed directly to step 6.
|
| Step 6. The output of step 5 is presented to the Rx model's AMI_GetWave
| call. If the Rx model does not include an AMI_GetWave call,
| this step is a pass-through step, and the input to step 6 is
| passed directly to step 7.
|
| Step 7. The output of step 6 becomes the simulation waveform output at
| the Rx decision point, which may be post-processed by the
| simulation tool.
|
| Steps 4 though 7 can be called once or can be called multiple times to
| process the full analog waveform. Splitting up the full analog waveform
| into multiple calls minimized the memory requirement when doing long
| simulations, and allows AMI_GetWave to return model status every so many
| bits. Once all blocks of the input waveform have been processed, Tx
| AMI_Close and Rx AMI_Close are called to perform any final processing
| and release allocated memory.

----------------------------------------
```

with the following text:

(Due to the high percentage of modified or new text, the changes are not marked by the usual "*" characters at the beginning of each line).

```
| 2 APPLICATION SCENARIOS
| =======================
|
| The next two sections provide an overview of the two simulation types
| supported by the IBIS-AMI specification. Statistical simulations require
| that the algorithm in the [Algorithmic Model] is linear and time-invariant
| (LTI). Time domain simulations do not have this requirement, therefore
| [Algorithmic Model]-s used in time domain simulations may also contain
| non-linear and/or time-variant (non-LTI) algorithms.
|
| System simulations will commonly involve a transmitter (Tx) and a receiver
| (Rx) [Algorithmic Model], each of which may perform filtering in the
| AMI_Init function, the AMI_GetWave function, or both (i.e., a “dual”
| algorithmic model). In the case of a “dual” algorithmic model, the
| filtering functionality in the AMI_Init and AMI_GetWave functions are each
| intended to be independent representations of the device’s equalization.
| Users of a dual model can elect to use either the AMI_Init or AMI_GetWave
| filtering functionality, but not combine both simultaneously.
|
| While the primary purpose of the AMI_Init function is to perform the
| required initialization steps, they may also include LTI signal
| processing algorithms. Therefore, statistical simulations may be
| performed using the AMI_Init function alone.
|
| Even though time domain simulations may also be performed with the LTI
| AMI_Init and/or LTI AMI_GetWave functions, AMI_GetWave functions containing
| non-LTI algorithms may only be simulated in the time domain.
|
|
| 2.1 Statistical simulations
| ===========================
|
| 1. A system simulation usually involves a transmitter (Tx) and a receiver
| (Rx) model with a passive channel placed between them. From the
| system netlist, the EDA platform determines that a given buffer is
| described by an IBIS [Model].
|
| 2. From the IBIS [Model], the EDA platform determines that the buffer
| is described in part by an [Algorithmic Model].
|
| 3. The EDA platform loads the shared library or shared object file
| containing the [Algorithmic Model], and obtains the addresses of the
| AMI_Init, AMI_GetWave, and AMI_Close functions.
|
| 4. The EDA platform loads the corresponding parameter file (.ami file)
| and assembles the arguments for the AMI_Init function. These arguments
| include an impulse response matrix, a memory handle for the dynamic
| memory used by the [Algorithmic Model], the parameters for configuring
| the [Algorithmic Model], and optionally the impulse response(s) of any
| crosstalk interferers.
|
```

```
| 5. The EDA platform calls the AMI_Init function with the arguments
| previously prepared. The AMI_Init function of the transmitter and
| receiver [Algorithmic Model]-s are called separately as described in
| the reference flow below.
|
| 6. The AMI_Init function parses the configuration parameters, allocates
| dynamic memory, places the address of the start of the dynamic memory
| into the memory handle and modifies the impulse response by the filter
| response of the [Algorithmic Model].
|
| 7. The EDA platform completes the rest of the simulation/analysis using
| the impulse response calculated by the AMI_Init function which is a
| complete representation of the behavior of a given [Algorithmic Model]
| combined with the channel.
|
| 8. Before exiting, the EDA platform calls the AMI_Close function, giving
| it the address in the memory handle for the [Algorithmic Model].
|
| 9. The AMI_Close function de-allocates the dynamic memory used by the
| [Algorithmic Model] and performs whatever other clean-up actions are
| required.
|
| 10. The EDA platform terminates execution.
|
|
| 2.2 Time domain simulations
| ===========================
|
| 1. A system simulation usually involves a transmitter (Tx) and a receiver
| (Rx) model with a passive channel placed between them. From the
| system netlist, the EDA platform determines that a given buffer is
| described by an IBIS [Model].
|
| 2. From the IBIS [Model], the EDA platform determines that the buffer
| is described in part by an [Algorithmic Model].
|
| 3. The EDA platform loads the shared library or shared object file
| containing the [Algorithmic Model], and obtains the addresses of the
| AMI_Init, AMI_GetWave, and AMI_Close functions.
|
| 4. The EDA platform loads the corresponding parameter file (.ami file)
| and assembles the arguments for the AMI_Init function. These arguments
| include an impulse response matrix, a memory handle for the dynamic
| memory used by the [Algorithmic Model], the parameters for configuring
| the [Algorithmic Model], and optionally the impulse response(s) of any
| crosstalk interferers.
|
| 5. The EDA platform calls the AMI_Init function with the arguments
| previously prepared. The AMI_Init function of the transmitter and
| receiver [Algorithmic Model]-s are called separately as described in
| the reference flow below.
|
| 6. The AMI_Init function parses the configuration parameters, allocates
| dynamic memory, places the address of the start of the dynamic memory
| into the memory handle and (optionally) modifies the impulse response
| by the filter response of the [Algorithmic Model]. The EDA platform
| may make use of the impulse response returned by the AMI_Init function
```

```
| in its further analysis if needed.
|
| 7. The EDA platform generates a time domain digital input waveform bit
| pattern (stimulus). A long bit pattern (and simulation) may be broken
| up into multiple time segments by the EDA platform. For example, if
| one million bits are to be simulated, there can be 1000 segments of
| 1000 bits each, i.e., one time segment comprises 1000 bits.
|
| 8. For each time segment, the EDA platform calls the AMI_GetWave function
| of the transmitter (if it exists), giving it the digital input waveform
| and the address in the memory handle for the [Algorithmic Model].
|
| 9. For the AMI_GetWave function of the receiver, the EDA platform takes the
| output from the transmitter AMI_GetWave (if it exists) or the output from
| step 7 and combines it (for example by convolution) with the channel
| impulse response to produce an analog waveform and passes this result to
| the receiver AMI_GetWave function for each time segment of the simulation.
|
|
| 10. The output waveform of the receiver AMI_GetWave function
| represents the voltage waveform at the decision point of the receiver.
| The EDA platform completes the simulation/analysis with this waveform.
|
| 11. Before exiting, the EDA platform calls the AMI_Close function, giving
| it the address in the memory handle for the [Algorithmic Model].
|
| 12. The AMI_Close function de-allocates the dynamic memory used by the
| [Algorithmic Model] and performs whatever other clean-up actions are
| required.
|
| 13. The EDA platform terminates execution.
|
| 3 Reference Flows
| =================
|
| The next two sections define a reference simulation flow for statistical
| and time domain system analysis simulations. Other methods of calling
| models and processing results may be employed, but the final simulation
| waveforms are expected to match the waveforms produced by this reference
| simulation flow.
|
| A system simulation usually involves a transmitter (Tx) and a receiver
| (Rx) model with a passive channel placed between them.
|
| 3.1 Statistical simulation reference flow
| =========================================
|
| Step 1. The simulation platform obtains the impulse response for the
| analog channel. This represents the combined impulse response
| of the transmitter's analog output, the channel and the
| receiver's analog front end. The transmitter's output or receiver's
| input characteristics must not include any filtering effects, for
| example, equalization, in this impulse response, although it may include
| any parasitics which are included in the
+ Tx or Rx analog model.
|
| Step 2. The output of Step 1 is presented to the Tx model's AMI_Init
```

```
| function. The impulse response returned by the Tx AMI_Init
| function is passed onto Step 3.
|
| Step 3. The output of Step 2 is presented to the Rx model's AMI_Init
| function. The impulse response returned by the Rx AMI_Init
| function is passed onto Step 4.
|
| Step 4. The EDA platform completes the rest of the simulation/analysis
| using the impulse response calculated in Step 3 by the Rx
| model's AMI_Init function which is a complete representation
| of the behavior of a given [Algorithmic Model] combined with
| the channel.
|
|
| 3.2 Time domain simulation reference flow
| =========================================
|
| Step 1. The simulation platform obtains the impulse response for the
| analog channel. This represents the combined impulse response
| of the transmitter's analog output, the channel and the
| receiver's analog front end. The transmitter's output or receiver's
| input characteristics must not include any filtering effects, for
| example equalization, in this impulse response, although it may include
| any parasitics which are included in the Tx or Rx analog model.
|
| Step 2. The output of Step 1 is presented to the Tx model's AMI_Init
| function.
|
| Step 3. The output of Step 2 is presented to the Rx model's AMI_Init
| function.
|
| Step 4. The simulation platform produces a digital stimulus waveform. A
| digital stimulus waveform is 0.5 when the stimulus is "high",
| -0.5 when the stimulus is "low", and may have a value between
| -0.5 and 0.5 such that transitions occur when the stimulus
| crosses 0.
|
| Step 5. Preparing the stimulus modified by the Tx Equalization:
|    Step 5a. If the Tx GetWave_Exists is True,
|          the output of Step 4 is presented to the Tx model's AMI_GetWave
|          function. The output of Step 5a is convolved with the output of
|          step 1.
|    Step 5b. If the Tx GetWave_Exists is False,
|          the output of Step 4 convolved with the output of Step 2.
|
|
| Step 6. Preparing the input to Rx AMI_GetWave:
|    Step 6a. If the Rx GetWave_Exists is True,
|         the output of Step 5 is passed directly into the Rx AMI_GetWave
|         function.
|    Step 6b. If the Rx GetWave_Exists is False,
|         the output of Step 5 is convolved with the Rx filter.
|
| Note the problematic combination for time domain simulation where Tx
| GetWave_Exists is True, Tx Init_Returns_Impulse is False, and the Rx model
| performs filter optimization. In this case, the Rx algorithmic model
| expects to be provided with the impulse response that has been modified by
```

| the Tx filter. Since the Tx AMI_Init function does not modify the impulse
| response in this case, the Rx AMI_Init function does not get what it needs
| to optimize itself correctly. One possible option is that the user chooses
| not to employ the Rx optimization functionality, and that the user instead
| derives optimized Rx algorithmic model settings through the use of their
| EDA tool.
|
| Note the other problematic combination for time domain simulation where Tx
| GetWave_Exists is True, Tx Init_Returns_Impulse is True (i.e., "dual" Tx
| model), and the Rx model has GetWave_Exists is False. performs filter optimization. In this case, the Rx | algorithmic model is correctly provided with the impulse response that has | been modified by the Tx AMI_Init function, but the Tx AMI_GetWave function | provides redundant Tx equalization, resulting in a "double counting" of the | Tx equalization effects. One option to address this is that the user not | utilize the Tx AMI_GetWave functionality in this case, and set the Tx | GetWave_Exists to False for time domain simulations. Another option is to | use deconvolution to be able to properly combine the Rx filtering from the | Rx AMI_Init function with the output of the Tx AMI_GetWave function and the | channel itself.
|
| One option to address this is that the EDA tool not utilize the Tx
| AMI_GetWave functionality, by treating the Tx AMI model as if
| GetWave_Exists was False.
|
| Another option is to use deconvolution to isolate the impulse response
| of the Rx filter. Since AMI_Init is a linear and time
| invariant representation, the Rx equalization can be represented
| as an impulse response. Since the output of the Rx AMI_Init
| function (output of Step 3) is the channel modified by the
| Rx equalization (e.g. by convolving the impulse response of the
| output of Tx AMI_Init with the impulse response of the Rx filter),
| the impulse response of the Rx filter can be obtained by deconvolving the
| output of Step 3 by the input to Step 3.
|
| Step 7. The output of step 6 becomes the simulation waveform output at
| the Rx decision point, and optionally also returns clock ticks, which may
| be post-processed by the simulation tool or presented to the user as is.
|
| Steps 4 though 7 can be called once or can be called multiple times to
| process the full analog waveform. Splitting up the full analog waveform
| into multiple calls reduces the memory requirements when doing long
| simulations, and allows AMI_GetWave to return model status every so many
| bits. Once all blocks of the input waveform have been processed, Tx
| AMI_Close and Rx AMI_Close are called to perform any final processing and
| release allocated memory.

****

ANALYSIS PATH/DATA THAT LED TO SPECIFICATION

The IBIS-ATM Task Group spent several meetings to discuss the problems discovered in the AMI flow in the months of September, October and November of 2009. In November the IBIS-ATM Task Group arrived to a solution which was then considered the final version of the flow proposal.

When the topic was revisited in April 2010, several EDA vendors opposed the addition of the new Boolean parameter "Init_Returns_Filter", and it was also discovered that the flow diagram did not show the existing Boolean parameter "Init_Returns_Impulse". Several additional ambiguities and interpretation differences of the existing specification were discovered during these discussions. As a result, work started over in search for a flow diagram that was acceptable to the participants of the ATM Task Group.

The following will be handled when IBIS 5.1 is written.

Replace this text:

```
| When
| Use_Init_Output is set to "True", the EDA tool is
| instructed to use the output impulse response from the
| AMI_Init function when creating the input waveform
| presented to the AMI_GetWave function.
|
| If the Reserved Parameter, Use_Init_Output, is set to
| "False", EDA tools will use the original (unfiltered)
| impulse response of the channel when creating the input
| waveform presented to the AMI_GetWave function.
```

with the following text:

```
| Use_Init_Output is of usage Info and type Boolean. The use of
| Use_Init_Output is hereby deprecated. The value of Use_Init_Output
| should have no affect on the way an EDA tool shall use this model.
```

Remove this text:

```
| If Use_Init_Output is False, GetWave_Exists must be True.
```

****

ANY OTHER BACKGROUND INFORMATION:

Documents AMI_Flows.ppt and AMI_Flows.xls are supporting documents that describe the flows using standard AMI Flow Conventions.

****