```
***************************************************************************
***************************************************************************

BIRD ID#:
ISSUE TITLE:     IBIS-AMI Flow Correction
REQUESTER:       Arpad Muranyi, Mentor Graphics, Inc.
DATE SUBMITTED:
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 GetWave functions of the AMI
models, Step 4 and Step 5, as described in Section 2.3 will only yield
correct results with LTI 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.

***************************************************************************

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 mulitple 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.
|
| While the primary purpose of the AMI_Init functions 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 functions alone.
|
| Even though time domain simulations may be performed using the LTI AMI_Init
| functions as well as LTI AMI_GetWave functions, non-LTI algorithms may only
| be placed into AMI_GetWave functions and simulated in the time domain.
|
| When the AMI_Init function contains a duplicate or an approximation of an
| algorithm that is also present in the GetWave function of the same AMI
| model, double counting for the same filtering effects is avoided by the
| EDA tool observing the value of the associated GetWave_Exists Boolean
| parameter. When GetWave_Exists is True, the EDA tool will not allow the
| output of the AMI_Init functions to be propagated to the GetWave functions
| and be processed again (i.e. double counted).
|
| A special situation arises when the receiver's AMI_Init function contains
| an optimization algorithm and the transmitter's AMI_Init and GetWave
| functions contain algorithm duplications, because the optimizer relies on
| a modified impulse response from the transmitter's AMI_Init function, but
| due to the algorithm duplication the EDA tool is not supposed to allow the
| output of the transmitter's AMI_Init function to be propagated to the
| subsequent function calls. This is the reason that the impulse_matrix
| presented to the receiver's AMI_Init function must contain two impulse
| response matrices. The content of the first impulse response depends on
| the value of the transmitter's Boolean parameter GetWave_Exists and may be
| the input or the output of the transmitter's AMI_Init function. This is
| the impulse response which may be modified by the receiver's AMI_Init
| function if it contains an equalizer. The second impulse response contains
| the output of the transmitter's AMI_Init function at all times, and it
| serves as the input for the (optional) optimization algorithm in the
| receiver's AMI_Init function.

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 exists), giving it the digital input waveform and the address in the memory handle for the [Algorithmic Model].

9. The EDA platform combines the output of the transmitter AMI_GetWave function (if exists) or the digital input waveform (otherwise) with the modified or unmodified impulse response and passes this result to the receiver AMI_GetWave function (if exists) for each time segment of the simulation.

10. The output waveform of the receiver GetWave function (if exists) or the combined waveform generated by the EDA tool (otherwise) 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.

2.3 Statistical simulation reference flow

This section defines a reference simulation flow for statistical 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.

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 characteristics must not include any filtering effects, for example, equalization in this impulse response.

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 simulation platform assembles an impulse_matrix that contains two back-to-back copies of the same impulse response that is returned by the TX AMI_Init function in Step 2. (This is only necessary to have a consistent function interface for AMI_Init in both statistical and time domain simulations).

Step 4. The output of Step 3 is presented to the RX model's AMI_Init function. The impulse response returned by the RX AMI_Init function is passed onto Step 5.

Step 5. The EDA platform completes the rest of the simulation/analysis using the impulse response calculated in Step 4 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.

2.4 Time domain simulation reference flow

This section defines a reference simulation flow for 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.

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 characteristics must not include any filtering effects, for example, equalization in this impulse response.

Step 2. The output of Step 1 is presented to the TX model's AMI_Init function. When GetWave_Exists for the TX model is False, the impulse response returned by the TX AMI_Init function is passed onto Step 3. If GetWave_Exists for the TX model is True, the same impulse response that is passed into Step 2 is passed on to step 3.

Step 3. The simulation platform assembles an impulse_matrix that contains the impulse response returned by Step 2 in the first half of the impulse_matrix and the output of the TX AMI_Init function in the second half of the matrix. Depending on the value of the TX GetWave_Exists and TX Init_Returns_Impulse Boolean parameters, the two impulse responses stored in the impulse_matrix may or may not be identical.

Step 4. The output of Step 3 is presented to the RX model's AMI_Init function. When GetWave_Exists for the RX model is False, the impulse response returned by the RX AMI_Init function is passed onto Step 7. If GetWave_Exists for the RX model is True, the same impulse response that is passed into Step 4 is passed on to step 7.

```
|  Step 5. 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 6. The output of step 5 is presented to the TX model's AMI_GetWave
|          function.  If the TX model does not include an AMI_GetWave
|          function, this step is a pass-through step, and the output of
|          step 5 is passed directly to step 7.
|
|  Step 7. The EDA simulation platform combines (for example by convolution)
|          the output of step 4 with the output of step 6.
|
|  Step 8. The output of step 7 is presented to the RX model's AMI_GetWave
|          function.  If the RX model does not include an AMI_GetWave
|          function, this step is a pass-through step, and the input to
|          step 8 is passed directly to step 9.
|
|  Step 9. The output of step 8 becomes the simulation waveform output at
|          the RX decision point, which may be post-processed by the
|          simulation tool or presented to the user as is.
|
| Steps 5 though 9 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.


----------------------------------------


Replace this text:

| GetWave_Exists:
|
| GetWave_Exists is of usage Info and type Boolean.  It tells
| the EDA platform whether the “AMI_GetWave” function is
| implemented in this model. Note that if Init_Returns_Impulse
| is set to “False”, then GetWave_Exists MUST be set to “True”.


with the following text:


| GetWave_Exists:
|
| GetWave_Exists is of usage Info and type Boolean. It tells
| the EDA platform whether the “AMI_GetWave” function is
|* implemented in this model.  Note that if GetWave_Exists
|* is set to “False”, then both Use_Init_Output and
|* Init_Returns_Impulse MUST be set to “True”.
|


----------------------------------------


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:

```
| When
|* Use_Init_Output is set to "True", EDA tools are instructed
|* to make use of the impulse response returned by the AMI_Init
|* function for the steps that follow the AMI_Init function call.
|
| If the Reserved Parameter, Use_Init_Output, is set to
|* "False", EDA tools are instructed to discard the impulse response
|* returned by the AMI_Init function and use the original (unfiltered)
|* impulse response for the steps that follow the AMI_Init function call.
|*
|* For statistical simulations Use_Init_Output is ignored and is
|* treated as if it was set to True.
|
```

Remove this text:

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

Add this text immediately before section 3.1.2.2:

```
|
| In order to support (optional) optimization algorithms in the receiver's
| AMI_Init function under all circumstances in the reference flow, the EDA
| tool must assemble two impulse response matrices for the RX AMI_Init
| function call.  The first matrix contains the impulse response which may
| be modified by the RX AMI_Init function's filter (if it has one).  The
| second matrix is a read-only matrix which may contain a different impulse
| response for the RX AMI_Init function's (optional) optimization function.
| In case the optimizer and the filter use the same impulse response, the
| two matrices may contain identical impulse responses.
|
| The two matrices must have an identical format, i.e. they must have the
| same row_size, aggressors, sample_interval and bit_time parameters.  The
| second matrix is stored the same way in memory as the first one (using
| a single dimensional array of floating point numbers which is formed by
| concatenating the columns of the impulse response matrix, starting with
| the first column and ending with the last column), and it is placed
| immediately after the first matrix in memory.  This way the second matrix
| can be retrieved/identified using the same method as the first matrix
| with an offset corresponding to the length of the first matrix.
```

|

****

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. On November 17, 2009 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". It was also noted that several combinations of the Boolean parameters required de-convolution which has undesirable mathematical artifacts, and several EDA vendors expressed their desire to eliminate the need for any de-convolution from the flow. Numerous other ambiguities and interpretation differences of the existing specification were also discovered during the discussions. As a result, work started over in search for a flow diagram that was acceptable to the participants of the ATM Task Group. After several meetings and detailed discussions, a new flow diagram was accepted by the participants of the ATM Task Group on ???, 2010.

A graphical representation of the existing flow that is described in the IBIS v5.0 specification can be found on the first page of the following presentation:

http://www.vhdl.org/pub/ibis/macromodel_wip/archive/20090921/arpadmuranyi/AMI%20flows:%20IBIS%205.0%20and%202009%20Sept%208,15%20proposals/AMI_Flows.pdf

The yellow highlighted symbols on the second page indicate what the order should have been to achieve the goal of simulating non-LTI systems with the GetWave functions.

The graphical representation of the new proposed flow can be found in the following presentation:

http://www.vhdl.org/pub/ibis/macromodel_wip/archive/.../AMI_Flows_9.pdf

****

ANY OTHER BACKGROUND INFORMATION:

****