****
****

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:

In Section 10, "NOTES ON ALGORITHMIC MODELING INTERFACE AND PROGRAMMING GUIDE", sub-section 2 describes a flawed reference flow. The intent was to make non-LTI simulations possible using the GetWave functions of AMI models, however the order of Step 4 and Step 5, as described in the IBIS v5.0 specification will only work properly with LTI GetWave functions.

****

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

| ======================

| 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.
|
| 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, places the address of the start of the dynamic memory in the
| memory handle. Depending on the value of Init_Returns_Filter, it
| either computes and returns the filter response of the block, or
| computes the impulse response of the channel modified by the filter
| response of the block.
|
| 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 block combined with the channel, or makes
| use of the filter response returned by AMI_Init to compute the
| behavior of the given block combined with the channel.
|
| 8. Before exiting, the EDA platform calls AMI_Close, giving it the
| address in the memory handle for the block.
|
| 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. Depending on the value of Init_Returns_Filter, it either computes and returns the filter response of the block, or computes the impulse response of the channel modified by the filter response of the block. The EDA platform may make use of the impulse response or the filter response returned by AMI_Init in its further analysis if needed.

7. The EDA platform generates a time domain input waveform (stimulus) bit pattern. A long simulation may be broken up into multiple time segments by the EDA platform. For example, if a 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 transmitter AMI_GetWave function, giving it the input waveform and the address in the dynamic memory handle for the block.

9. Depending on the value stored in the Use_Init_Output parameters, the EDA platform combines the output of the transmitter AMI_GetWave function with the output(s) of the AMI_Init function(s) with the impulse response of the channel and passes this result to the receiver AMI_GetWave function for each time segment of the simulation.

10. The output waveform of the receiver GetWave function represents the voltage waveform at the decision point of the receiver. The EDA platform uses this waveform 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.

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 Init_Returns_Filter for the TX model is set to True, the model returns the impulse response of the TX filter. If it is set to False, the TX AMI_Init call returns the modified impulse response of the channel. The output of TX AMI_Init is

| returned to the EDA tool which decides how to make use of it
| depending on the transmitter's Init_Returns_Filter and
| Use_Init_Output parameters.
|
| Step 3. If the transmitter's Init_Returns_Filter parameter is set to
| False, the output of Step 2 is presented to the RX model's
| AMI_Init call. If the Init_Returns_Filter is set to True, the
| EDA tool will combine the output of Step 2 with the output of
| Step 1 (for example by convolution) before presenting it to the
| RX model's AMI_Init call.
|
| Step 4. The output of Step 3 is presented to the RX model's AMI_Init
| call. If Init_Returns_Filter for the RX model is set to True,
| the model returns the impulse response of the RX filter. If it
| is set to False, the RX AMI_Init call returns the filtered
| response of the channel. The output of RX AMI_Init is returned
| to the EDA tool which decides how to make use of it depending
| on the receiver's Init_Returns_Filter and Use_Init_Output
| parameters.
|
| Step 5. If the receiver's Init_Returns_Filter parameter is set to
| False, the output of Step 4 may be presented to the user of the
| EDA tool, or the EDA tool can further process the results using
| statistical algorithms. If the Init_Returns_Filter is set to
| True, the EDA tool will combine the output of Step 4 with the
| output of Step 3 (for example by convolution) before presenting
| it to the user of the EDA tool, or before continuing with the
| statistical processing of these results.
|
| Step 6. 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 7. The output of step 6 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 7 is
| passed directly to step 8.
|
| Step 8. The EDA simulation platform combines (for example by
| convolution) the output of step 7 with the output of several
| different previous steps depending on the value of the
| transmitter's and receiver's Init_Returns_Filter and
| Use_Init_Output settings as follows:
|
| If TX Use_Init_Output = False, combine the outputs of Step 7
| and Step 1.
|
| If TX Use_Init_Output = True and TX Retuns_Filter = False,
| combine the outputs of Step 7 and Step 2.
|
| If TX Use_Init_Output = True and TX Retuns_Filter = True,
| combine the outputs of Step 7, Step 1 and Step 2.
|
| In addition, the EDA simulation platform will also combine the
| output of Step 4 with the above if RX Use_Init_Output = True.
| When RX Init_Returns_Filter = True, this is a relatively straight
| forward operation, but when RX Init_Returns_Filter = False, the
| EDA simulation platform will have to take additional steps to
| prevent the duplication of the content that is present in the
| output of Steps 2 and/or 3 (for example by deconvolution). This
| is why RX Init_Returns_Filter = True is preferred when RX
| Use_Init_Output = True.
|
| Step 9. The output of step 8 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 9 is
|          passed directly to step 10.
|
| Step 10. The output of step 9 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 6 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.
```

****

ANALYSIS PATH/DATA THAT LED TO SPECIFICATION

The IBIS-ATM Task Group spent several meetings to discuss the AMI flow problem and the best solution for it in the months of September, October and November of 2009. The IBIS-ATM Task Group arrived to the final version of the proposed flow on November 17, 2009.

A graphical representation of the flow that is in 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 highligted symbols on the second page indicate what the order should have been to achieve the goal of simulating non-LTI sysmtems with the GetWave functions.

The ATM Task Group also discovered during the discussions that certain conditions would require a deconvolution operation which is known to be a challange due to its numerical instability. To remedy this oversight, a new Boolean parameter "Init_Returns_Filter" was also introduced in the proposed flow to provide a mechanizm to eliminate the need for using deconvolution in the flow.

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

http://www.vhdl.org/pub/ibis/macromodel_wip/archive/20091118/arpadmuranyi/Final%20AMI%20flow%20for%20IBIS%205.1/AMI_Flows_5_final.pdf

****

ANY OTHER BACKGROUND INFORMATION:

****