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

BIRD ID#:       xx.x
ISSUE TITLE:    Algorithmic Modeling API (AMI) support in IBIS
REQUESTER:     Cadence, SiSoft, Mentor
DATE SUBMITTED: TBD
DATE REVISED:   05/01/2007 - Internal
DATE ACCEPTED BY IBIS OPEN FORUM: N/A

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

STATEMENT OF THE ISSUE:

1.   The Interoperability Problem

Advanced SERDES devices operating above 5Gbps often contain architectures that are difficult, if not impossible, to model using current modeling techniques. Such devices often include Feed Forward Equalization, Decision Feedback Equalization, Clock and Data Recovery modules as well as IP for noise cancellation using new and proprietary techniques. These complex modules are very difficult to model using traditional device/circuit level modeling techniques. In addition, the serial links that are driven by these devices often need to be simulated with extremely long bit streams (10s of millions of bits to accurately predict Bit Error Rates) making high simulation performance an important consideration.

Since current device/circuit level modeling techniques are inadequate, SERDES IP vendors have been forced to develop, distribute, and maintain proprietary software for the modeling and simulation of such devices. While this can address the initial device modeling issues, proprietary approaches from the SERDES suppliers result in a lack of interoperability between models, leaving the systems designers unable to simulate the serial links. This approach also adds a support burden for SERDES IP vendors. A new device modeling approach is needed to allow SERDES IP vendors to create algorithmic models that plug into commercially available PCB system interconnect simulators, while at the same time protecting the critical IP contained in the models.

2.   Key Requirements for Solution

The proposed modeling solution must:

- Allow SerDes models from a single semiconductor manufacturer to operate in multiple EDA tools
- Allow SerDes models from different semiconductor manufacturers to work together within a given EDA platform and simulation
- Expose key user-accessible device parameters [e.g. tap settings] to the EDA platform for control and observation by the user
- Support automatic optimization of device settings based on simulation
- Protect semiconductor vendor IP to the satisfaction of the IP vendors

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

STATEMENT OF THE RESOLVED SPECIFICATIONS:

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 “analog” portion of the channel is characterized by means of an impulse response leveraging the pre-existing IBIS standard for device models.

The transmitter equalization, receiver equalization and clock recovery circuits are assumed to be electrically isolated from the analog portion of the channel. The behavior of these circuits is modeled algorithmically through the use of executable code provided by the SerDes vendor. This proposal outlines the functions of the executable models, the standards for passing data to and from these executable modules and how the executable modules are called from the EDA platform.

Changes in Section 3a - K E Y W O R D   H I E R A R C H Y

```
*****   Original Text   *****


|  |-- [Model]                              Model_type, Polarity, Enable,
|  |   -------                              Vinl, Vinh, C_comp, C_comp_pullup,
|  |     |                                  C_comp_pulldown,
.
.
.
|  |     |
|  |     |-- [External Model]               Language, Corner, Parameters,
|  |         ----------------               Ports, D_to_A, A_to_D
|  |           |-- [End External Model]

*****    New Text (Changes  noted with |*) *****

|  |-- [Model]                              Model_type, Polarity, Enable,
|  |   -------                              Vinl, Vinh, C_comp, C_comp_pullup,
|  |     |                                  C_comp_pulldown,
.
.
.
|  |     |
|  |     |-- [External Model]               Language, Corner, Parameters,
|* |     |   ----------------               Ports, D_to_A, A_to_D
|* |     |     |-- [End External Model]
|* |     |
|* |     |-- [Algorithmic Model]            Executable,Number_of_Aggressors,
|* |     |                                  Ignore_bis, GetWave_present, Jitter,
|* |     |                                  DCD, Modified_IR_in_TX_INIT,       |
* |      |                                  Modified_IR_in_RX_INIT, Clock_PDF,
|* |     |                                  Receiver_Sensitivity, Description
|* |     |   -------------------
|* |     |     |-- Executable <type> <name>
|* |     |     |-- <param_name> <param_type> <data type> <data format> <values>
|* |     |     |-- [End Algorithmic Model]
```

Additions to Section 6 – M O D E L   S T A T E M E N T

```
|==============================================================================
|
| KEYWORD DEFINITIONS:
|
|==============================================================================
|    Keywords:  [Algorithmic Model]  [End Algorithmic Model]
|    Required:  No
| Description:
|  Sub-Params:  Executable, Description, Number_of_Aggressors, Ignore_Bits,
                GetWave_present, Jitter, DCD, Modified_IR_in_TX_INIT,
                Modified_IR_in_RX_INIT, Clock_PDF, RX_Sensitivity,
                <parameters>
| Usage Rules:  Executable: (required if [Algorithmic Model] is present)
                        Name of exe or dll
                  Executable dll windows name.dll

                Number_of_Aggressors: Integer (Optional, Default 0)
                        Tells EDA tools how to handle Tx and Rx settling

                Ignore Bits: (Optional, Default 0)
                        How many bits of GetWave output to ignore

                GetWave_present: Boolean (Optional, Default True)
                        Does this model have a GetWave

                Jitter: PDF (Optional, Default 0)
                        Jitter of Tx stimulus

                DCD: Real (Optional, Default 50% pulses)
                        Active high/low pulse width difference

                Modified_IR_in_TX_INIT: Boolean (Optional, Default False)
                         Does Tx.Init create an h_TE?

                Modified_IR_in_RX_INIT: Boolean (Optional, Default False)
                         Does Rx.Init create an h_RE?

                Clock_PDF: (Optional, Default 0)
                         Probability Density Function of Recovered Clock

                Receiver_Sensitivity: (Optional, Default 0)
                        “Vdiff” at data decision point

                Description: (Optional)
                        ASCII description of parameters

                        Description <parameter_name> <Description>

                <parameter_name> <usage> <data type> <data format> <values>

                <parameter_name>
                       Arbitrary Names (IP Vendors choose what names their
                       models accept; The IBIS Parser does not verify the names.
                       It would, however, check the usage, data type, data
```

```
             format and values.) The names, types and description of
             the parameters used for algorithmic modeling will not be
             specified in the IBIS standard.

             Published List of Standard Parameter Names
             The IBIS Librarian will publish a list of parameter
             names. Each entry will include a unique data type and
             description. Model creators wishing to create new
             parameters may submit requests to the IBIS Librarian for
             approval

      <usage>

             In      Parameter is required Input to executable
             Out     Parameter is Output only from executable
             Info    Information for user or EDA platform
             InOut  Required Input to executable. Executable may
                    return different value. If not 'optimized' then
                    executable will use input value of parameter.
                    If 'optimized' then executable may change value of
                    parameter. Default is optimize.

      <data type> (default is float)
             float
             integer
             string
             Boolean (True/False)

      <data format> (default to range)
             Range   <typ value> <min value> <max value>
             List <typ value> <value> <value> <value> ... <value>
             Corner <typ value> <slow value> <fast value>
             Increment    <typ> <min> <max> <delta>
             Steps  <typ> <min> <max> <# steps>
             Tap_Equalizer

             Table <columns names> ... < columns names >
             EndTable

             Gaussian <mean> <sigma>
             Dual-Dirac <mean> <mean> <sigma>
                     Composite of two Gaussian
             DjRj <minDj> <maxDj> <sigma>
                    Convolve Gaussian (sigma) with uniform Modulation
                    PDF


| Other Notes:
|-------------------------------------------------------------------------------
|
|Example 1
|This is an example of an RX model with init to initialize GetWave and all the |
work is done inside the GetWave.

[Algorithmic Model]

Executable dll Windows example_rx.dll
```

```
Receiver_Sensitivity 10mV
| Same as Vdiff, but at decision point not at input pin.

Ignore_Bits UI 0.5e6

Number_of_Aggressors 0 0 8

Modified_IR_in_RX_INIT False
| init does not return h_RE(t)

GetWave_present True

Clock_ticks 500

[End Algorithmic Model]

|Example 2
[Algorithmic Model] Tx_GetWave

Executable dll Windows tx_getwave.dll

Executable dll Solaris tx_getwave.so

Modified_IR_in_TX_INIT False

Ignore_Bits 4

Number_of_Aggressors 0

Jitter DjRj mks 0 3ps 2ps

DCD .05 0. .1

Strength in range real 1. 0. 1.

FFE InOut tap_equalizer FFE 4 Synch

Tap -1 real increment 0. -1. 1. 32
Tap 0 real increment 1. -1. 1. 32
Tap 1 real increment 0. -1. 1. 32
Tap 2 real increment 0. -.25 .25 8
Tap_Sum real range 1. 0. 1.

End_Tap_equalizer

[End Algorithmic Model
|
|==============================================================================
```

****

ANALYSIS PATH/DATA THAT LED TO SPECIFICATION:

This specification has been driven primarily by the following factors:

1. The interaction between a SerDes and the system surrounding it is quite complex, thus requiring sophisticated and detailed modeling.

2. There is considerable variation in the architectures and circuit techniques used in SerDes devices.

3. There is not a commonly accepted set of parameters that can be measured to fully and reliably characterize the performance of a given SerDes device independently from the system that surrounds it.

Because of these factors, IP vendors' experience has been that customers use the models delivered by the IP vendor as a form of performance specification. If the model predicts a level of performance in a given application, then the IP is held to that level of performance or better when the system is tested.

For this reason, IP vendors are reluctant to supply any but most detailed and accurate models they can produce. This is a fundamental shift in that in the past, the models that were presumed to be utterly complete and reliable were SPICE models, and IBIS models were understood to be a useful approximation that could be shared without divulging sensitive proprietary information.

By setting the algorithmic model as the primary deliverable, this specification maximizes the flexibility available to the model developers and also maximizes the degree of protection for proprietary information. By standardizing the interface to these algorithmic models, this specification also enables the required degree of interoperability.
****

ANY OTHER BACKGROUND INFORMATION:

1. AMI Function Calls and Description

TABLE OF CONTENTS

1.1 OVERVIEW

The algorithmic model of a SerDes consists of three functions: Init, GetWave and Close. The interfaces to these functions are designed to support three different phases of the simulation process: initialization, simulation during a segment of time, and termination of the simulation.

The interfaces to these functions are defined from three different perspectives. In addition to specifying signature of the functions to provide a software coding perspective, anticipated application scenarios provide a functional and dynamic execution perspective, and a specification of the software infrastructure provides a software architecture perspective. Each of these perspectives is required to obtain interoperable software models.

1.2 APPLICATION SCENARIOS

1.2.1 Linear, Time-invariant 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, and that the AMI_Init function of that model returns an impulse response for that block.

3. The EDA platform loads the DLL 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 interferors.

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 for the block and passes it back to the EDA platform.

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.

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.

1.2.2 Nonlinear, Time-variant 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 DLL 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 IR of the block and pass the modified IR to the EDA tool.

7. For each of many time segments, the EDA platform computes the input waveform to the block for that time segment. For example, if a million bits are to be run, there can be 1000 segments of 1000 bits each.

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. For transmitter, it simply passes the output to the receiver GetWave.

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.

1.3 SOFTWARE INFRASTRUCTURE

1.3.1 Name Resolution

The primary mechanism used to distinguish between different algorithmic models is the name of the DLL or shared object file. Each such file can contain at most one AMI_Init, one AMI_GetWave, and one AMI_Close function.

If a given file does not contain one of these functions, then at the point in the execution of the simulation and/or analysis when such a function would normally be called, no function is called and no action is performed. The EDA platform which is coordinating the simulation/analysis is responsible for determining whether it can complete its task without the function, and the model developer is responsible for determining whether or not the function is required to perform the modeling that will be delivered, and for providing the function if it is required. If there is no GetWave call, close can be empty. Inside the DLL, only AMI_Init, or all 3 (AMI_Init, AMI_GetWave and AMI_Close) should exist.

The DLL or shared object file can, and usually will contain other functions in addition to AMI_Init, AMI_GetWave, and AMI_Close. The EDA platform will have no standardized mechanism for knowing the names of these other function or their signatures, and therefore will not call them directly.

1.3.2 Dynamic Linking

<THIS SECTON TO BE ADDED>

1.3.3 External Execution

<THIS SECTON TO BE ADDED>

1.3.4 Memory Management

In all cases, the program entity which allocates heap memory (e.g., through malloc and free) is responsible for de-allocating that memory. For example, if the EDA platform allocates memory for an array to be passed to the algorithmic model, then the EDA platform is responsible for de-allocating that memory. Similarly, if an algorithmic model allocates memory (e.g., as part of the execution of the AMI_Init function), then that algorithmic model is responsible for de-allocating it, usually as part of the execution of the AMI_Close function.

There, however, numerous instances in which memory allocated by one program entity will be written to by another program entity. Each such instance is stated explicitly in the description of the function argument for which it occurs.

1.3.5 Object-oriented Description

One way to describe the algorithmic model is as an object oriented model with a C language interface. AMI_Init is the constructor for the object class, AMI_Close is the destructor for the object class, and AMI_GetWave is the only other method supported by the object class. The memory handle supplied to the AMI_Init function is a pointer to an instance of the object class, and the memory pointer passed to the AMI_GetWave and

AMI_Close functions points to the data members of an instance to the object class.

1.4 FUNCTION SIGNATURES

1.4.1 AMI_Init

1.4.1.1 Declaration

```
long AMI_Init (double *impulse_matrix,
               long row_size,
               long aggressors,
               double sample_interval,
               double bit_time,
               char *AMI_dll_parameters_in,
               char *AMI_dll_parameters_out,
               void **AMI_dll_memory_handle,
               AMI_dll_msg_type **msg)
```

1.4.1.2 Arguments

1.4.1.2.1 impulse_matrix: Impulse matrix is the channel impulse response matrix. The impulse values are in volt/s. The sample spacing is given by the parameter 'sample_interval'.

The 'impulse_matrix' is stored in single double array. The matrix elements can be retrieved /identified using

```
Impulse_matrix[idx] = element (row, col)

 Idx = col * row_size + row

Row – row index , ranges from 0 to row_size-1
Col – column index, ranges from 0 to aggressors
```

1.4.1.2.2 row_size: the number of rows in the impulse matrix

1.4.1.2.3 aggressors: the number of aggressors in the impulse matrix. The first column of the impulse matrix is the primary channel. The rest are aggressors.

1.4.1.2.4 sample_interval: sample_interval of the impulse matrix. Sample_interval will usually be a fraction of the highest data rate (lowest bit_time) of the device.

E.g.,

```
Sample_interval = (lowest_bit_time/64)
```

1.4.1.2.5 bit_time: the bit time of the current data. E.g 100ps, 200ps etc. The dll may use this Information along with the impulse matrix to initialize the filter coefficients.

1.4.1.2.6 AMI_dll_parameters (_in and _out)

Memory allocated by EDA platform. This is a pointer to a string. All the input parameters from the IBIS file Algorithmic model section are passed using a string that has been formatted as a parameter tree. The syntax for this string is:

```
<tree>:
   <branch>
```

```
<branch>:
   ( <branch name> <branch> )
   ( <branch name> <leaf> )

<leaf>:
   ( <parameter name> whitespace <value> )
```

White space is ignored, except as a delimiter between the parameter name and
value.

An example of the tree parameter passing is:

```
 (dll
   (tx
      (taps 4)
      (spacing sync)
   )
)
```

The EDA tool will convert the parameter information from the IBIS file and convert them to tree format for dll consumption.

1.4.1.2.7 AMI_dll_memory_handle: Used to store local storage for the dll and will be passed back during the GetWave call. e.g. a code snippet may look like the following

```
my_space = allocate_space (sizeof_space);
status = store_all_kinds_of_things (my_space);
*sedes_dll_memory_handle = my_space;
```

1.4.1.2.8 msg

1.4.1.3 Return Value
    1 for success
    0 for failure

1.4.2 AMI_GetWave
1.4.2.1 Declaration

```
long AMI_GetWave (double *wave_in,
                  long wave_size,
                  double *clock_times,
                  void *AMI_dll_memory);
```

1.4.2.2 Arguments
1.4.2.2.1 wave_in: A vector of time domain wave form, sampled at 'sample_interval' soecified   during the init call. The wave_in is both input and output. The EDA platform provides the wave_in. The dll can modify the wave form in place.

1.4.2.2.2 wave_size: A size of the wave_in vector, range from 1 – unlimited.

1.4.2.2.3 clock_times: Vector to return clock times. The clock times are referenced to the start of the simulation (the first GetWave call). The time is always greater than zero. The last clock is indicated by putting a value of -1 at the end of clocks for the current wave simple. The clock_time vector is allocated by the eda platform and is guaranteed to be greater than the number of clocks expected during this wave_in call.

1.4.2.2.4 AMI_dll_memory: this is the memory which was allocated during the init call

1.4.2.2.3 Return Value
1 for success
0 for failure

1.4.3 AMI_Close
1.4.3.1 Declaration
```
long AMI_Close(void * AMI_dll_memory);
```

1.4.3.2 Arguments
1.4.3.2.1 AMI_dll_memory
Same as for AMI_GetWave. See section 1.4.2.2.4

1.4.3.3 Return Value
1 for success
0 for failure

1.4.4 AMI_Init and AMI_GetWave Function Calls in Plain English
:AMI_TX_Init:

```
      In the AMI Init function, the model is expected to do the following

      INPUT
            : Parameters (model, required eda sim parameters)
            : Impulse Response of Channel (h_tcr(t))
                  - Channel = Tx FrontEnd + Package + PCB Interconnect + Package
                              + Rx FrontEnd

            : Impulse Response of Aggressors (optional)
                  - h_an_TE(t) X h_an_vTCR(t) (an_v = nth aggressor to victim)

            (- Where h_an_vTCR(t) = Impulse Response of the aggressor TX front
            end + coupled aggressor victim circuit + victim RX front end

            - It is not intended to simultaneously optimize the aggressor TXs.)

            : Sampling Interval
            : Bit time
            : Number of Aggressors

      OUTPUTS
            : Parameters (optional, depends on parameter_type)
            (Process and store parameters passed by EDA Platform)
            : Data for GetWave (optional)
            (Use Impulse Response to initialize filter and derive tap
            coefficients)
            : Modified Channel Impulse Response (optional)
                     h_TE(t) X h_tcr(t)
                     May be provided when getwave is not provided
```

:AMI_TX_GetWave:

In the GetWave call, the model is expected to do the following
INPUTS
: Stimulus (Required)
- V T table, with logic values.(OR PWL?)
: Data for GetWave (output of TX_INIT) (Required)
: Start of New Msg Flag

OUTPUTS
: Stimulus X h_TE(t) X h_tcr(t)
-This is a continuous waveform with time step specified by the TX_INIT call.

:AMI_RX_Init:

In the AMI Init function, the model is expected to do the following

INPUT
: Parameters
-Impulse Response of Channel
= h_TE(t) X h_tcr(t)
: Impulse Response of Aggressors (optional)
- h_an_TE(t) X h_an_vTCR(t) (an_v = nth aggressor to victim)
(Where h_an_vTCR(t) = h_an_TX(t) X h_an_v(t) X h_RX(t))
(It is not intended to simultaneously optimize the aggressor TXs.)
: Sampling Interval
: Bit time
: Number of Aggressors

OUTPUTS
: Parameters (optional, depends on parameter_type)
(Process and store parameters passed by EDA Platform)
: Data for GetWave (optional)
(Use Impulse Response to initialize filter and derive tap coefficients)
: Modified Channel Impulse Response (optional)
H_TE(t)X h_TCR(t) X h_RE(t), may be provided when getwave is not provided

:AMI_RX_GetWave:

In the GetWave call, the model is expected to do the following
INPUTS
: Waveform at the input of RX (X(t) = Output of circuit simulation.)
- Examples:
X(t) = Stimulus(t) X h_TE(t) X p(t) X h_tcr(t)
X(t) = Stimulus(t) X h_TE(t) X p(t) X h_tcr(t)
+ crosstalk contribution + amplitude noise
X(t) = Output of TX GetWave X p(t) X h_tcr(t)
Output of (SPICE) Simulation.

```
          : Data for GetWave (o/p of RX_INIT)
          : Start of New Msg Flag

    OUTPUTS
          : Equalized RX data
                - Y(t)
          : Clock Ticks
```

1.5 CODE SEGMENT EXAMPLES

```
      extern long AMI_GetWave (wave_in, wave_size, clock_times, AMI_dll_memory)

      my_space = AMI_dll_memory

       clk_idx=0;
       time = my_space->prev_time + my_space->sample_intevel
       for (i=0; i<wave_size; i++)
         {
           Wave_in = filterandmodify (wave_in, my_space);
           If (clock_times && found_clock (my_space, time))
           Clock_times[clk_idx++] = getclocktime (my_space, time);
           Time += my_space->sample_intevel
         }

      Clock_times[clk_idx] = -1;   //terminate the clock vector

     Return 1;
```

SUPPLEMENTARY MATERIAL (To be resolved before final submission.)

1.6. Definition of Impulse Response:

Impulse Response of a system is the response of a system when excited with a dirac delta.

Practically the Impulse response can be derived by

• stimulating the system with a narrow pulse and by dividing the result by the area of the input pulse.
OR

• stimulating the system with a step and by taking the time derivative of the result.

Impulse Response is independent of bit period

Note:

An X in this document represents the operation of convolution. For ex: A X B is the convolution of A with B.

REVISION HISTORY CHANGES:

****