****
BIRD ID#: ~~?~~TBD
ISSUE TITLE: Algorithmic Modeling API (AMI) Improvements
REQUESTER: (in alphabetical order by company)
Adge Hawes, IBM;
Arpad Muranyi, Mentor Graphics;
Walter Katz, Mike Steinberger, Todd Westerhoff, SiSoft
DATE SUBMITTED:
Date of Draft: 11/17/2009
DATE REVISED:
DATE ACCEPTED BY IBIS OPEN FORUM: PENDING
****

STATEMENT OF THE ISSUE:

Based on the experiences of several EDA vendor and IC vendor implementations of AMI models and EDA software using AMI models it has become apparent that a number of changes to the document are required to correct the reference flow, clarify the specification and simplify both the development of AMI models and EDA software using AMI models.

Existing known AMI models and .ami files will work with these changes.

Section 6c and 10 are to be replaced with the following.

Summary of significant changes

- Change Reference Flows
- Remove Branches
  - Reserverd_Parameters
  - Model_Specific
- Remove Reserved Parameters
  - Tx_Jitter
  - Rx_Clock_PDF
- Add Reserved Parameters
  - Tx_Dj
  - Tx_Rj
  - Rx_Clock_Recovery_Mean
  - Rx_Clock_Recovery_Rj
  - Init_Returns_Filter
- Remove Keywords
  - Format
  - Gaussian
  - Table
  - DjRj
  - Dual-Dirac
- Add Keywords
  - Array
  - Scale
  - Limit
  - Labels
  - Value
  - Range
  - Increment
  - Step
- Support Environment Variables

```
|==============================================================================
|==============================================================================
|
|                                  Section 6c
|
|           A L G O R I T H M I C  M O D E L  A P I  S U P P O R T
|
|==============================================================================
|==============================================================================
|
| INTRODUCTION:
|
| Executable shared library files to model advanced Serializer-Deserializer
| (SERDES) devices are supported by IBIS.  This chapter describes how
| executable models written for these devices can be referenced and used by
| IBIS files.
|
| The shared objects use the following keywords within the IBIS framework:
|
|   [Algorithmic Model]
|   [End Algorithmic Model]
|
| The placement of these keywords within the hierarchy of IBIS is shown in
| the following diagram:
|
|   |-- [Component]
|   |    | ...
|   |    | ...
|   | ...
|   |-- [Model]
|   |    | ...
|   |    |-- [Algorithmic Model]
|   |    |-- [End Algorithmic Model]
|   |    | ...
|   | ...
|   | ...
|
| Figure 1: Partial keyword hierarchy
|
| GENERAL ASSUMPTIONS:
|
| 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.
|
| The transmitter equalization, receiver equalization and clock recovery
| circuits are assumed to have a high-impedance (electrically isolated)
| connection to the analog portion of the channel.  This makes it possible to
| model these circuits based on a characterization of the analog channel.
| The behavior of these circuits is modeled algorithmically through the use
```

```
| of executable code provided by the SERDES vendor.  This proposal defines the
| functions of the executable models, the methods for passing data to and
| from these executable models and how the executable models are called from
| the EDA platform.
|
| The parameter definition file (.ami) is an ASCII file that contains AMI-Parameters that
| are used to:
|
|       Specify flows that the model supports.
|       Configures the model for specific silicon implementations.
|       Configures the model for specific model programming.
|       Tells the EDA tool how to analyze the model output.
|       Tells the EDA tool what parameters are returned from the model.
|
| AMI-Parameters can either be Reserved Parameters or Model Specific Parameter.
|
| AMI-Parameters are passed into the model and returned from the model using a parameter tree
| syntax. The .ami file is in essentially the same format as the string passed into and returned
| from the model except that in place of the parameter value, the .ami file contains
| sub-parameters that describe the Type, Usage, Allowed Values, and Description of the parameter.
|
| The parameter tree contains a root, branches and leaves. Only leaves may have
| sub-parameters, except that the root and branches may have the sub-parameter Description.
|
| DEFINITIONS:
|
| The root and branch names are case sensitive, and must start with a letter [a-Z], and may
| contain letters [a-Z], numbers [0-9], and underscore [_].
|
| Parameters are case sensitive, and must start with a letter [a-Z], and may contain letters
| [a-Z], numbers [0-9], and underscore [_], except tap parameter leaves which must be a positive
| or negative integer. Reserved parameters will always be a leaf off the root and start with a
| capital letter [A-Z]. It is highly recommended that two parameters should not differ by case
| alone.
|
| If a parameter has more than one sub-parameter, the order of the sub-parameters is unimportant.
|
| The following sub-parameters are not allowed parameter names.
|
|      Type
|      Usage
|      Description
|
| The following sub-parameters describe the permitted parameter values and are not allowed
| parameter names.
|
|       Value
|       Range
|       List
|       Labels
|       Corner
|       Increment
|       Steps
|       Default
|
| The following sub-parameters describe how the EDA tool needs to adjust tap parameters and
| are not allowed parameter names.
```

```
|      Scale
|      Limit
|
| The following sub-parameter direct the EDA tool needs to pass the values of leaves of a
| branch as a single string of concatenated leaf values and are not allowed parameter names.
|     Array
|
| Environmental Variables
|      Parameters may reference a file name. These parameters must be of type String. If the
|      string begins with a "$", then the string between the $ and the first / in the string
|      shall be a computer system environmental parameter that must be defined by the EDA tool.
|      A special environmental variable AMISearchPath shall contain a list of directories that
|      the EDA tool shall search sequentially to find the file.
|
| The following 'Usage, Type Format and Default definitions are used
| throughout the following sections.
|
| Note: Usage, Type, Format and Default and their allowed values are
| reserved names in the parameter definition file (.ami) discussed in the
| "KEYWORD DEFINITION" section.
| Sub-Parameter Definitions
|
|   Usage: (Requiredrequired for model specific parameters)
|     In     Parameter is required Input to executable
|     Out    Parameter is Output only from executable
|     Info   Information for user or EDA platform
|     InOutInOut  Required Input to executable.  Executable may return a different
|            value.
|
|   Type: (Requireddefault is Float)
|     Float
|        Can be specified as an integer, decimal number, or in exponential format.
|     Integer
|        Can be positive, negative or zero.
|     String
|        Strings begin and end with a double quote (") and no double quotes are
|        allowed inside the string literals. The "s are optional if the string does not contain
|        white space, "(", ")","'" or ",". A blank string is denoted by " ". Adjacent double
|        quotes are not allowed, double quotes need to have at least one character or white
|        space between them. Carriage Return <CR> and Line Feed <LF> are explicitly allowed.
|        There shall be no limit on the length of a String, or the number of lines in a String.
|        If a string references a file name, then the Unix "/" shall delineate folders on all
|        platforms including Windows.
|     Boolean (True/False)
|        Values allowed are True and False.
|     Tap (For use by TX and RX equalizers)
|        The leaves of a branch represent Tx or Rx equalization coefficients.
|        Values shall be Float
|     UI
|         (Unit Interval, 1 UI is the inverse of the data rate frequency,
|         for example 1 UI of a channel operating at 10 Gb/s is 100-ps)
|
|   Format: (default is range)
|     Value:     <value> Single value data
|                     <value>=NA implies there is no constraint on the <value>
|                     Example  (Value 5.)
|     Range:     <typ value> <min value> <max value>
```

```
|                         <typ> >= <min>
|                         <typ> <= <max>
|                         <min> = NA means there is no lower limit to a value
|                         <max> = NA means there is no upper limit to a value
|                         Example  (Range -1. -2. 4.)
|      List:      <typ value> <value1> <value2> <value3> ... <valueN>
|                         Example  (List Xslow Slow Typ Fast Xfast)
|      Labels:     <label1> < label2> < label3> ... < labelN>
|                   Only allowed (and optional) when a parameter has a List sub-parameter
|                         Example  (List Xslow Slow Typ Fast Xfast)
|                                  (Lables "Extremely Slow Process" "Slow Process"
|                                 "Typcial Process"  "Fast Process" "Extremely Fast Process")
|      Corner:    <typ value> <slow value> <fast value>
|                         Example  (Process 0 -1 1)

|      Increment: <typ> <min> <max> <delta>
|                         <typ> >= <min>
|                         <typ> <= <max>
|                         <min> = NA means there is no lower limit to a value
|                         <max> = NA means there is no upper limit to a value
|                   After expansion, theThe allowed values of the parameter are
|                   typ+N*delta where N is any positive or negative integer
|                   value such that:  min <= typ + N*delta <= max
|                         Example  (Increment 50 NA 100 5)
|      Steps:      <typ> <min> <max> <# steps>
|                         <typ> >= <min>
|                         <typ> <= <max>
|                   Treat exactly like Increment with
|                   <delta> == (<max>-<min>)/<# steps>
|                         Example  (Steps 50 0 100 20)
|      Table      The parameter name "Table" names a branch of the parameter
|                   tree rather than a single leaf.  One of the leaves of this
|                   branch can be named "Labels" and, if provided, is to be
|                   assigned a string value containing a list of column names.
|                   For example:
|
|                     (Rx_Clock_PDF
|                       (Usage Info)
|                       (Type Float)
|                       (Format Table
|                         (Labels Row_No Time_UI Density)
|                         (-50 -0.1 1e-35)
|                         (-49 -0.98 2e-35)
|                         ...
|                         (0 0 1e-2)
|                         ...
|                         (49 0.98 2e-35)
|                         (50 0.1 1e-35)
|                       ) | End Table
|                     ) | End Rx_Clock_PDF
|
|                   Gaussian <mean> <sigma>
|                   Dual-Dirac <mean> <mean> <sigma>
|                     Composite of two Gaussian
|                   DjRj <minDj> <maxDj> <sigma>
|                     Convolve Gaussian (sigma) with uniform Modulation PDF
|
```

```
|   Default <value>: (Optional)
|     Depending on the Type, <value> will provide a default value for the
|     parameter.  For example, if the Type is Boolean, <value> could be True
|     or False, if the Type is Integer, the <value> can be an integer value.
|     Default is not allowed if Allowed-Value is Value, or Corner.
|     If Default is not specified, then the default value of a parameter shall depend
|     on the Allowed-Value. If Default is specified then it must be a legal Allowed-Value.
|        Value:     NA
|        Range:     <typ>
|        List:      <value1>
|        Label:     NA
|        Corner:    NA
|        Increment: <typ>
|        Steps:     <typ>

|
|   Description <string>: (Required for Model Specific Parameters)
|     ASCII string following Description describes a reserved parameter,
|     model specific parameter, a branch within the parameter tree
|     or the Algorithmic model itself.  It is used
|     by the model make to convey information to the EDA platform and for the EDA platform
|     to convey information to the end-user.  The entire
|     line has to be limited to IBIS line length specification.
|     There shall be no limit on the length of a Description, or the number of lines
|     in a Description, however, the model developer should assume that the first
|     128 characters of the first line be at minimum an abstract of the full description.
|     The location of Description will determine what the parameter, branch or model
|     is being described.
|
String
|     literals begin and end with a double quote (") and no double quotes are
|     allowed inside the string literals.
|
|     The location of Description will determine what the parameter or model
|     is being described.
|
| Every parameter must have one, and only one of the following "Allowed-Value" sub-parameters:
|     Value
|     Range
|     List
|     Corner
|     Increment
|     Steps
|
| Note that in the context of Algorithmic Model for type 'Corner', <slow
| value> and <fast value> align implicitly to slow and fast corners, and
| <slow value> does not have to be less than <fast value>.  For type 'Range'
| and 'Increment', <min value>, <max value> does not imply slow and fast
| corners.
|
| If a Reserved Parameter must have one and only one Usage, Usage is optional.
| If a Reserved Parameter must have one and only one Type, Type is optional.
|
| Notes:
| 1. Throughout the section, text strings inside the symbols "<" and ">"
| should be considered to be supplied or substituted by the model maker.
| Text strings inside "<" and ">" are not reserved and can be replaced.
```

```
| 2. Throughout the document, terms "long", "double" etc. are used to
| indicate the data types in the C programming language as published in
| ISO/IEC 9899-1999.
| 3. Throughout the section, text strings inside the symbols "[(" and ")]"
| indicate that the parameter definition is optional.
| 4. Previous versions of the AMI spec used the two-word keyword sequence Format
| along with the Allowed-Value keyword. These AMI files can be corrected for the
| new format by simply removing the keyword Format from the AMI file. No change to
| the DLL is required. These AMI files can be parsed for the new format by simply
| ignoring the keyword Format from the AMI file.
| 5. Previous versions of the AMI spec required all parameter be either in a
| Reserved Parameter branch, or a Model Specific branch. These AMI files can be
| corrected for the new format by simply removing the Reserved Parameter branch and
| Model Specific branch from the AMI file, thus moving all parameter definition to
| the root branch of the parameter tree. No change to the DLL is required. These
| AMI files can be parsed for the new format by simply moving all parameters in the
| the Reserved Parameter branch and Model Specific branch into the root branch of the
| tree.
|
|==============================================================================
|
| KEYWORD DEFINITIONS:
|
|==============================================================================
|    Keywords: [Algorithmic Model], [End Algorithmic Model]
|    Required: No
|
| Description: Used to reference an external compiled model.  This compiled
|              model encapsulates signal processing functions.  In the case
|              of a receiver it may additionally include clock and data
|              recovery functions.  The compiled model can receive and modify
|              waveforms with the analog channel, where the analog channel
|              consists of the transmitter output stage, the transmission
|              channel itself and the receiver input stage.  This data
|              exchange is implemented through a set of software functions.
|              The signature of these functions is elaborated in section 10
|              of this document.  The function interface must comply with
|              ANSI 'C' language.
|
|  Sub-Params: Executable
| Usage Rules: The [Algorithmic Model] keyword must be positioned within a
|              [Model] section and it may appear only once for each [Model]
|              keyword in a .ibs file.  It is not permitted under the
|              [Submodel] keyword.
|
|              The [Algorithmic Model] always processes a single waveform
|              regardless whether the model is single ended or differential.
|              When the model is differential the waveform passed to the
|              [Algorithmic Model] must be a difference waveform.
|
|              [Algorithmic Model], [End Algorithmic Model]
|              Begins and ends an Algorithmic Model section, respectively.
|
|
|              Subparameter Definitions:
|
|              Executable:
```

Three entries follow the Executable subparameter on each line:

```
  Platform_Compiler_Bits  File_Name  Parameter_File
```

The Platform_Compiler_Bits entry provides the name of the operating system, compiler and its version and the number of bits the shared object library is compiled for. It is a string without white spaces, consisting of three fields separated by an underscore '_'. The first field consists of the name of the operating system followed optionally by its version. The second field consists of the name of the compiler followed by optionally by its version. The third field is an integer indicating the platform architecture. If the version for either the operating system or the compiler contains an underscore, it must be converted to a hyphen '-'. This is so that an underscore is only present as a separation character in the entry.

The architecture entry can be either "32" or "64". Examples of Platform_Compiler_Bits:

```
  Linux_gcc3.2.3_32
  Solaris5.10_gcc4.1.1_64
  Solaris_cc5.7_32
  Windows_VisualStudio7.1.3088_32
  HP-UX_accA.03.52_32
```

The EDA tool will check for the compiler information and verify if the shared object library is compatible with the operating system and platform.

Multiple occurrences, without duplication, of Executable are permitted to allow for providing shared object libraries for as many combinations of operating system platforms and compilers for the same algorithmic model.

The File_Name provides the name of the shared library file. The shared object library ~~should~~ can be in the same directory as the IBIS (.ibs) file, or in directories defined in an environmental variable AMISearchPath.

The Parameter_File entry provides the name of the parameter file with an extension of .ami. This must be an external file and should reside in the same directory as the .ibs file and the shared object library file. It will consist of reserved and model specific (user defined) parameters for use by the EDA tool and for passing parameter values to the model. If there are multiple Executable lines in a [Algorithmic Model] they all must have the same Parameter File entry.

The model parameter file must be organized in the parameter tree format as discussed in section 3.1.2.6 of "NOTES ON ALGORITHMIC MODELING INTERFACE AND PROGRAMMING GUIDE", Section 10 of this document. ~~The file must have 2 distinct sections, or sub-trees, 'Reserved_Parameters' section and 'Model_Specific' section with sections beginning and ending~~

```
|                  with parentheses.  The complete tree format is described in
|                  the section 3.1.2.6 of the Section 10 of this document.
+
|                  The 'Reserved_Parameter' section is required while the
|                  'Model_Specific' section is optional.  The sub-trees can be
|                  in any order in the parameter file.  The '|' character is the
|                  comment character.  Any text after the '|' character will be
|                  ignored by the parser.
|
|                  The Model Parameter File must be organized in the following
|                  way:
|
|                    (<my_AMIname>            --| Name given to the Parameter file
|                                               |(This need not be the same as the basename
|                                               | of the .ami file)
|                      (Reserved_Parameters    | Required heading to start the
|                                               | required Reserve_Parameters
|                                               | section
|                      ...
|                        (Reserved parameter text)
|                      ...
|                      )                        | End of Reserved_Parameters
|                                               | section
|                      (Model_Specific          | Required heading to start the
|                                               | optional Model_Specific section
|                      ...
|                        (Parameter Text)Model specific parameter text)
|                      ...
|                      )                        | End of Model_Specific section
|                      (Description <string>)  | description of the model
|                                               | (optional)
|                    )                          | End my_AMIname parameter file
|
|                  Reserved Parameters:
|
|                  Init_Returns_Impulse, GetWave_Exists, Max_Init_Aggressors and
|                  Ignore_Bits
|                       AMI_Version                  (Required)
|                       Init_Returns_Impulse         (Required)
|                       GetWave_Exists               (Required)
|                       Max_Init_Aggressors          (Optional)
|                       Ignore_Bits                  (Optional)
|                       Use_Init_Output              (Optional)
|                       Init_Returns_Filter          (Optional)
|                       Tx_Dj                        (Tx only, Optional)
|                       Tx_Rj                        (Tx only, Optional)
|                       Tx_DCD                       (Tx only, Optional)
|                       Rx_Clock_Recovery_Mean       (Rx only, Optional)
|                       Rx_Clock_Recovery_Rj         (Rx only, Optional)
|                       Rx_Receiver_Sensitivity      (Rx only, Optional)
|
|
|                  All reserved parameters must contain sub-parameters Usage, Type,
|                  and one of the Allowed-Values sub-parameters. Description is optional.
+
|                  The model parameter file must have a sub-tree with the
|                  heading 'Reserved_Parameters'.  This sub-tree shall contain
```

```
|                 all the reserved parameters for the model.
|
|                 The following reserved parameters are used by the EDA tool
|                 and are required if the [Algorithmic Model] keyword is
|                 present.  The entries following the reserved parameters
|                 points to its usage, type and default value.  All reserved
|                 parameters must be in the following format:
|
|                 (parameter_name (Usage <usage>)(Type <data_type>)
|                                 (Default <values>) (Description <string>))
|
|                 AMI_Version:
|
|                 AMI_Version is of usage Info, type Float, and Value 5.1.
|                 Example of AMI_Version declaration is:
|                    (AMI_Version (Usage Info)(Type Float) (Value 5.1))
|
|                 Init_Returns_Impulse:
|
|                 Init_Returns_Impulse is of usage Info and type Boolean.  It
|                 tells the EDA platform whether the AMI_Init function returns
|                 a modified impulse response.  Allowed-Values must be Value.
|                 When this value is set to True, the model returns either the impulse
|                 response of the filter, or the impulse response of the channel including
|                 the equalization of the filter depending on the value of Init_Returns_Filter.
|                 If GetWave_Exists is False, AMI_Init always returns a modified impulse response.
|                 If GetWave_Exists is True, the model writer may set
|                 Init_Returns_Impulse to False, and not return an impulse response. It is
|                 highly recommended that Tx models that have GetWave_Exists set to True
|                 also have Init_Returns_Impulse set to True and return a best estimate
|                 modified impulse response in order to maximize the effectiveness of
|                 Rx Ami_Init function that do internal optimization based on the channel and
|                 Tx equalization.
|                 the model returns the convolution of the input impulse
|                 response with the impulse response of the equalization.
|                 Example of Init_Returns_Impulse declaration is:
|                    (Init_Returns_Impulse (Usage Info)(Type Boolean) (Value True))
|
|                 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.  Allowed-Value must be Value.
|                 Note that if Init_Returns_Impulse
|                 is set to "False", then Getwave_Exists MUST be set to "True".
|                 Examples of GetWave_Exists declaration are:
|                    (GetWave_Exists (Usage Info)(Type Boolean) (Value True))
|                    (GetWave_Exists (Value True))
|
|                 Use_Init_Output:
|
|                 Use_Init_Output is of usage Info and type Boolean.  It
|                 tells the EDA platform if it needs to combine the output of
|                 AMI_Init with the waveform. If the model AMI_GetWave is False
|                 the value of Use_Init_Ouput parameter must be True.
|                 If Use_Init_Output=True in a Tx model with an AMI_GetWave, then the
|                 output of the Tx AMI_GetWave needs to be convolved with the output of Tx
```

```
|               AMI_Init, instead of convolved with the impulse response of the channel
|               alone. If Use_Init_Output=True in a Tx model without an AMI_GetWave, then the
|               output of the Tx stimulus waveform must be convolved with an impulse response
|               that contains both the channel and the output of AMI_INIT.
|               AMI_GetWave needs to be convolved with the output of Tx
|               AMI_Init, instead of convolved with the impulse response of the channel
|               alone.

Use_Init_Output=True in an Rx model with an AMI_GetWave, then the
|               input to the Rx AMI_GetWave needs to be convolved with the filter only
|               component of the output of Rx AMI_Init. If Use_Init_Output is not
|               specified in the .ami file then it is assumed to be False.
|               Allowed-Values must be Value.
|               Examples of Use_Init_Output declaration is:
|                  (Use_Init_Output (Usage Info)(Type Boolean) (Value True))
|                  (Use_Init_Output (Value False))
|
|               The following reserved parameters are optional.  If these
|               parameters are not present, the values are assumed as "0".
|
|               Init_Returns_Filter:
|               (WMK, although most often the EDA tool will use the combination of the impulse
|               response input to AMI_Init and the response of the filter block within the model,
|               there are flows in which the EDA tool needs to use the impulse response of just
|               the filter. To avoid the EDA tool from doing a deconvolution, it is advisable that
|               the AMI_Init function have the ability to return either the impulse
|               of just the filter, or the combination of the filter and the input impulse
|               response.)
|
|               Init_Returns_Filter is of usage Info and type Boolean. If it is True, then
|               The AMI_Init function will return just the impulse response of the filter.
|               If not set, or is False, AMI_Init will return only the impulse response
|               of the filter convolved with the channel.
|               Examples of Init_Returns_Filter declaration are:
|                  (Init_Returns_Filter (Usage Info)(Type Boolean) (Value False))
|                  (Init_Returns_Filter (Value True))
|
|               Max_Init_Aggressors:
|
|               Max_Init_Aggressors is of usage Info and type Integer.  Allowed-Values must be
|               Value. It
|               tells the EDA platform how many aggressor Impulse Responses
|               the AMI_Init function is capable of processing.
|               Example of Max_Init_Aggressors declaration is:
|                  (Max_Init_Aggressors (Usage Info)(Type Integer) (Value 25))
|
|               Ignore_Bits:
|
|               Ignore_Bits is of usage Info and type Float or Integer.  Allowed-Value
|               must be Value. It tells the
|               EDA platform how long the time variant model takes to complete
|               initialization.  This parameter is meant for AMI_GetWave
|               functions that model how equalization adapts to the input
|               stream.  The value in this field tells the EDA platform how
|               many bits of the AMI_Getwave output should be ignored.
|               Examples of Ignore_Bits declaration are:
|                  (Ignore_Bits (Usage Info)(Type Integer) (Value 100))
```

```
              (Ignore_Bits (Usage Info)(Type Float) (Value 1.e5))

          The following reserved parameter provides textual description
          to the user defined parameters.

          Tx-only reserved parameters:

          Tx_Jitter and Tx_DCD

          These reserved parameters only apply to Tx models.  These
          parameters are optional; if the parameters are not specified,
          the values default to "no jitter specified in the model ("0"
          jitter).  If specified, they must be in the following format:

          (<parameter_name> (Usage <usage>)(Type <data_type>)
                            (Format <data format>) (Default <values>)
                            (Description <string>))

          Tx_Jitter:

          Tx_Jitter can of Usage Info and Out and can be of Type Float
          or UI.  It can be of Data Format Gaussian, Dual-Dirac, DjRj
          or Table.  It tells the EDA platform how much jitter exists
          at the input to the transmitter's analog output buffer.
          Several different data formats are allowed as listed.
          Examples of Tx_Jitter declarations are:

          (Tx_Jitter (Usage Info)(Type Float)
                     (Format Gaussian <mean> <sigma>))

          (Tx_Jitter (Usage Info)(Type Float)
                     (Format Dual-Dirac <mean> <mean> <sigma>))

          (Tx_Jitter (Usage Info)(Type Float)
                     (Format DjRj <minDj> <maxDj> <sigma>))

          (Tx_Jitter (Usage Info)(Type Float)
                     (Format Table
                       (Labels Row_No Time Probability)
                       (-5  -5e-12  1e-10)
                       (-4  -4e-12  3e-7)
                       (-3  -3e-12  1e-4)
                       (-2  -2e-12  1e-2)
                       (-1  -1e-12  0.29)
                       (0    0      0.4)
                       (1    1e-12  0.29)
                       (2    2e-12  1e-2)
                       (3    3e-12  1e-4)
                       (4    4e-12  3e-7)
                       (5    5e-12  1e-10) ))

          Tx_Dj:

          Tx_Dj (Transmit Deterministic Jitter) can be of Usage Info, In or Out,
          of Type Float or UI and of Allowed-Value Value, Range and Corner.
          Tx_Dj is the deterministic jitter of the transmit clock. If the Usage is
          In, then the EDA tool can assume that Tx_Dj will be implemented in the
```

Tx AMI_GetWave funtion. If of type Float, then its units are in seconds.
Example of Tx_Dj declaration is:
(Tx_Dj (Usage In)(Type UI) (Value .1))

Tx_Rj:

Tx_Rj (Transmit Random Jitter) can be of Usage Info, In or Out,
of Type Float or UI and of Allowed-Value Value, Range and Corner.
Tx_Rj is the random jitter of the transmit clock. If the Usage is
In, then the EDA tool can assume that Tx_Rj will be implemented in the
Tx AMI_GetWave funtion. If of type Float, then its units are in seconds.
Example of Tx_Rj declaration is:
(Tx_Rj (Usage Info)(Type UI) (Value .05))

Tx_DCD:

Tx_DCD (Transmit Duty Cycle Distortion) can be of Usage Info, In
~~and~~ or Out. It can be of Type Float and UI and can have ~~Data~~
Allowed-Value ~~Format~~ of Value, Range and Corner. It tells the EDA platform
the maximum percentage deviation of the duration of a
transmitted pulse from the nominal pulse width. If the Usage is
In, then the EDA tool can assume that Tx_DCD will be implemented in the
Tx AMI_GetWave funtion. If of type Float, then its units are in seconds.
Example of Tx_DCD declaration is:~~Example of~~
~~TX_DCD declaration is:~~

(Tx_DCD (Usage Info)(Type ~~Float~~UI) (Value .15))
~~(Format Range <typ> <min> <max>))~~

~~Rx-only reserved parameters:~~

~~Rx_Clock_PDF and Rx_Receiver_Sensitivity~~

~~These reserved parameters only apply to Rx models. These~~
~~parameters are optional; if the parameters are not specified,~~
~~the values default to "0". If specified, they must be in the~~
~~following format:~~

~~(<parameter_name> (usage <usage>)(data type <data_type>)~~
~~(data format <data format> (Default <values>)~~
~~(Description <string>))~~

~~Rx_Clock_PDF:~~

~~Rx_Clock_PDF can be of Usage Info and Out and of Type Float~~
~~and UI and of Data Format Gaussian, Dual-Dirac, DjRj or~~
~~Table. Rx_Clock_PDF tells the EDA platform the Probability~~
~~Density Function of the recovered clock. Several different~~
~~data formats are allowed as listed. Examples of Rx_Clock_PDF~~
~~declarations are:~~

~~(Rx_Clock_PDF (Usage Info)(Type Float)~~
~~(Format Gaussian <mean> <sigma>))~~

~~(Rx_Clock_PDF (Usage Info)(Type Float)~~
~~(Format Dual-Dirac <mean> <mean> <sigma>))~~

```
+
|               (Rx_Clock_PDF (Usage Info)(Type Float)
|                             (Format DjRj <minDj> <maxDj> <sigma>))
+
|               (Rx_Clock_PDF (Usage Info)(Type Float)
|                             (Format Table
|                               (Labels Row_No Time Probability)
|                               (-5  -5e-12  1e-10)
|                               (-4  -4e-12  3e-7)
|                               (-3  -3e-12  1e-4)
|                               (-2  -2e-12  1e-2)
|                               (-1  -1e-12  0.29)
|                               (0    0      0.4)
|                               (1    1e-12  0.29)
|                               (2    2e-12  1e-2)
|                               (3    3e-12  1e-4)
|                               (4    4e-12  3e-7)
|                               (5    5e-12  1e-10) ))
|               Rx_Clock_Recovery_Mean:
|
|               Rx_Clock_Recovery_Mean can be of Usage Info, In or Out, of
|               Type Float or UI and of Allowed-Value Value, Range and Corner.
|               Rx_Clock_Recovery_Mean is the offset of the recovered clock from
|               the center of an average bit. If the Usage is In, then the EDA tool
|               can assume that Rx_Clock_Recovery_Mean will be implemented in the
|               Rx AMI_GetWave funtion. If of type Float, then its units are in seconds.
|               Example of Rx_Clock_Recovery_Mean declaration is:
|                  (Rx_Clock_Recovery_Mean (Usage Info)(Type UI) (Value .15))
|
|               Rx_Clock_Recovery_Rj:
|
|               Rx_Clock_Recovery_Rj can be of Usage Info, In and Out, of
|               Type Float or UI and of Allowed-Value Value, Range and Corner.
|               Rx_Clock_Recovery_Rj is sigma of the Gaussian distribution of
|               the recovered clock around the clock mean. If the Usage is In, then
|               the EDA tool can assume that Rx_Clock_Recovery_Rj will be implemented in the
|               Rx AMI_GetWave funtion. If of type Float, then its units are in seconds.
|               Example of Rx_Clock_Recovery_Rj declaration is:
|                  (Rx_Clock_Recovery_Rj (Usage Info)(Type UI) (Value .05))
|
|               Rx_Receiver_Sensitivity:
|
|               Rx_Receiver_Sensitivity can be of Usage Info, In,  and or Out and of
|               Type Float and of Data FormatAllowed-Value Value, Range and Corner.
|               Rx_Receiver_Sensitivity tells the EDA platformis the voltage
|               needed at the receiver data decision point to ensure proper
|               sampling of the equalized signal.  In this example, 100 mV
|               (above +100 mV or below -100 mV) is needed to ensure the
|               signal is sampled correctly.  Units are in Volts.
Examples of Rx_Clock_PDF
|               declarations are:
+
|               Example of Rx_Receiver_Sensitivity declaration is:
|               (Rx_Receiver_Sensitivity (Usage Info)(Type Float) (Value .1))
|                                        (Format Value <value>))
+
|               (Rx_Receiver_Sensitivity (Usage Info)(Type Float)
```

(Format Range <typ> <min> <max>))

(Rx_Receiver_Sensitivity (Usage Info)(Type Float)
(Format Corner <slow> <fast>))

The general rules, allowed usage and a brief summary of the data types and data formats allowed for each reserved parameter is presented in the following tables.

Reserved Parameter	General Rules: Required	General Rules: Default	Allowed Usage: Info	In	Out	InOutInOut
Init Returns Impulse	Yes	NA	X			
GetWave Exists	Yes	NA	X			
Ignore Bits	No	0	X	X		
Max Init Aggressors	No	0	X			
Tx Jitter	No	No Jitter	X	X		
Tx DCD	No	0	X	X		
Rx Receiver Sensitivity	No	0	X	X		
Rx Clock PDF	No	Clock Centered	X	X		

Table 1: General Rules and Allowed Usage for Reserved Parameters

Reserved Parameter	Data Type: Float	UI	Integer	String	Boolean
Init Returns Impulse					X
GetWave Exists					X
Ignore Bits			X		
Max Init Aggressors			X		
Tx Jitter	X	X			
Tx DCD	X	X			
Rx Receiver Sensitivity	X				
Rx Clock PDF	X	X			

Table 2: Allowed Data Types for Reserved Parameters

Reserved Parameter	Data Format: Value	Range	Corner	List	Incr	Steps	Gaus	DualDira	DjRj	Table

```
| |                          |   |   |   |   |   |   |   | c |   |   |
| |--------------------------|---|---|---|---|---|---|---|---|---|---|
| | Init Returns Impulse     | X                                      |
| | GetWave Exists           | X                                      |
| | Ignore Bits              | X                                      |
| | Max Init Aggressors      | X                                      |
| | Tx Jitter                |                           X   X   X   X |
| | Tx DCD                   | X   X   X                              |
| | Rx Receiver Sensitivity  | X   X   X                              |
| | Rx Clock PDF             |                           X   X   X   X |
| |--------------------------|----------------------------------------|
|
| Table 3: Allowed Data Format for Reserved Parameters
```

Model Specific Parameters:

The Following section describes the ~~user defined~~user-defined parameters. The algorithmic model expects these parameters and their values to function appropriately. The model maker can specify any number of user defined parameters for their model. ~~The user defined parameter section subtree must begin with the reserved parameters 'Model_Specific'.~~

The user defined parameters must be in the following format:

```
(<parameter_name> (usage <usage>) (Type <data type>)
                  (Allowed Values)Format <data format>) [(Default <values>)]
                  (Description <string>))
```

A tapped delay line can be described by creating a separate parameter for each tap weight and grouping all the tap weights for a given tapped delay line in a single parameter group which is given the name of the tapped delay line. If in addition the individual tap weights are each given a name which is their tap number (i.e., "-1" is the name of the first precursor tap, "0" is the name of the main tap, "1" is the name of the first postcursor tap, etc.) and the tap weights are declared to be of type Tap, then the EDA platform can assume that the individual parameters are tap weights in a tapped delay line, and use that assumption to perform tasks such as optimization. The model developer is responsible for choosing whether or not to follow this convention.

The type Tap implies that the parameter takes on floating point values. Note that if the type Tap is used and the parameter name is not a number (except for Limit and Scale, this is an error condition for which EDA platform behavior is not specified.

Tap parameter names Limit and Scale are used to adjust Tap values.
A Tap may have a Scale, a Limit, or neither. If Scale is specified then the sum of the absolute values of all of the Taps must equal Scale.
If Limit is specified then the Taps are scaled by Limit/(maximum value of the absolute values of all of the Taps).
Scale and Limit must be Usage Info, Type Tap, and Allowed-Value Value)

```
|  Array
|  If a branch has multiple leaves, and one of the leafs is Array, and the value of
|  Array is True, then the parameter tree that is passed into the model, and returned
|  from the model will use the branch as the parameter name, and the values of all of the leaves
|  of the branch (except the Array, Scale, and Limit leaves) will be passed as a white space
|  delimited string.
|
|==============================================================================
| Example of Parameter File
|==============================================================================
(mySampleAMI                                | Root Name given toof the Parameter fileTree
    (Description "Sample AMI File")
  (Reserved_Parameters                      | Required heading
    (AMI_Version (Usage Info) (Type Float) (Value 5.1))
    (Ignore_Bits (Usage Info) (Type Integer) (Default Value 21))
      (Description "Ignore 21 Bits"))
    (Max_Init_Aggressors (Usage Info) (Type Integer)(Default Value 25))
    (Init_Returns_Impulse (Usage Info) (Type Boolean)( Value Default True))
    (GetWave_Exists (Usage Info) (Type Boolean) (Value Default True))
  )                                         | End Reserved_Parameters

  (Model_Specific                           | Required heading
    (txtaps
      (-2 (Usage InoutInOut)(Type Tap) (Format Range 0.1 -0.1 0.2)(Default 0.1)
          (Description "Second Precursor Tap"))
      (-1 (Usage InoutInOut)(Type Tap) (Format Range -0.2 -0.4 0.4)(Default 0.2)
          (Description "First Precursor Tap"))
      (0  (Usage InoutInOut)(Type Tap) (Format Range 1.4 -1 2)(Default 1)
          (Description "Main Tap"))
      (1  (Usage InoutInOut)(Type Tap) (Format Range 0.2 -0.4 0.4)(Default2 0.2)
          (Description "First Post cursor Tap"))
      (2  (Usage InoutInOut)(Type Tap) (Format Range -0.1 -0.1 0.2)(Default 0.1)
          (Description "Second Post cursor Tap"))
      (Scale (Usage Info)(Type Tap) (Value 1.))
    )                                       | End txtaps
    (tx_freq_offset (Format Range 1 0 150) (Type UI) (Default 0))
    (framis (Value NA) (Usage Out) (Type String) (Description "state of Tx framis"))
    (strength (Range 6 0 7) (Usage In) (Type Integer) (Description "IC Strength Register"))
  )                                         | End Model_Specific
)                                           | End SampleAMI
|
| The EDA tool would pass the following string to the Model in the string pointed to by
| *AMI_parameters_in for the default value of each In and InOut parameter.
|
    (mySampleAMI (txtaps (-2 .05) (-1 -.1) (0 .7) (1 .1) (2 -.05)) (strength 6))
|
|
| The Model would pass the following string back to the EDA tool in the string pointed to by
| *AMI_parameters_out for the value of each Out and InOut parameter.
|
    (mySampleAMI (txtaps (-2 .06) (-1 -.05) (0 .8) (1 .05) (2 -.04)) (framis "Overloaded"))
|
|
| Array Example:
|
| The same as the previous example, but the txtaps branch is:
|
```

```
    (txtaps
      (-2 (Usage InOut)(Type Tap) (Range 0.1 -0.1 0.2)
          (Description "Second Precursor Tap"))
      (-1 (Usage InOut)(Type Tap) (Range -0.2 -0.4 0.4)
          (Description "First Precursor Tap"))
      (0  (Usage InOut)(Type Tap) (Range 1.4 -1 2)
          (Description "Main Tap"))
      (1  (Usage InOut)(Type Tap) (Range 0.2 -0.4 0.4)
          (Description "First Post cursor Tap"))
      (2  (Usage InOut)(Type Tap) (Range -0.1 -0.1 0.2)
          (Description "Second Post cursor Tap"))
      (Scale (Usage Info)(Type Tap) (Value 1.))
      (Array (Usage Info)(Type Boolean) (Value True))
    )                                       | End txtaps
|
|
| The EDA tool would pass the following string to the Model in the string pointed to by
| *AMI_parameters_in for the default value of each In and InOut parameter.
|
    (mySampleAMI (txtaps .05 -.1 .7 .1 -.05) (strength 6))
|
|
| The Model would pass the following string back to the EDA tool in the string pointed to by
| *AMI_parameters_out for the value of each Out and InOut parameter.
|
    (mySampleAMI (txtaps .06 -.05 .8 .05 -.04) (framis "Overloaded"))
|
|
|
|
|==============================================================================
| Example of RX model in [Algorithmic Model]
|==============================================================================
[Algorithmic Model]
|
Executable Windows_VisualStudio_32 example_rx.dll example_rx_params.ami
|
[End Algorithmic Model]
|
|==============================================================================
| Example of TX model in [Algorithmic Model]:
|==============================================================================
[Algorithmic Model]
|
Executable Windows_VisualStudio_32 tx_getwave.dll tx_getwave_params.ami
Executable Solaris_cc_32 libtx_getwave.so tx_getwave_params.ami
|
[End Algorithmic Model
|
|==============================================================================
```

The Section 10 addition is below:

```
|==============================================================================
|==============================================================================
|
```

```
|                                Section 10
|
|                             N O T E S   O N
|        A L G O R I T H M I C   M O D E L I N G   I N T E R F A C E
|                A N D   P R O G R A M M I N G   G U I D E
|
|=============================================================================
|=============================================================================
|
| INTRODUCTION:
|
| This section is organized as an interface and programming guide for
| writing the executable code to be interfaced by the [Algorithmic Model]
| keyword described in Section 6c.  Section 10 is structured as a reference
| document for the software engineer.
|
| TABLE OF CONTENTS
|
| 1 OVERVIEW
| 2 APPLICATION SCENARIOS
|   2.1 Linear, Time-invariant equalization Model
|   2.2 Nonlinear, and / or Time-variant equalization Model
|   2.3 Reference system analysis flow
|
| 3 FUNCTION SIGNATURES
|   3.1 AMI_Init
|     3.1.1 Declaration
|     3.1.2 Arguments
|       3.1.1 impulse_matrix
|       3.1.2 row_size
|       3.1.3 aggressors
|       3.1.4 sample_interval
|       3.1.5 bit_time
|       3.1.6 AMI_parameters (_in and _out)
|       3.1.7 AMI_memory_handle
|       3.1.8 msg
|     3.1.3 Return Value
|   3.2 AMI_GetWave
|     3.2.1 Declaration
|     3.2.2 Arguments
|       3.2.10 wave
|       3.2.11 wave_size
|       3.2.12 clock_times
|       3.2.13 AMI_memory
|     3.2.3 Return Value
|   3.3 AMI_Close
|     3.3.1 Declaration
|     3.3.2 Arguments
|     3.3.3 Return Value
|       3.3.13 AMI_memory
|
| 4 CODE SEGMENT EXAMPLES
|
|=============================================================================
|
| 1 OVERVIEW
| ==========
```

```
|
| The algorithmic model of a Serializer-Deserializer (SERDES) transmitter or
| receiver consists of three functions: 'AMI_Init', 'AMI_GetWave' and
| 'AMI_Close'.  The interfaces to these functions are designed to support
| three different phases of the simulation process: initialization,
| simulation of a segment of time, and termination of the simulation.
|
| These functions ('AMI_Init', 'AMI_GetWave' and 'AMI_Close') should all be
| supplied in a single shared library, and their names and signatures must be
| as described in this section.  If they are not supplied in the shared
| library named by the Executable sub-parameter, then they shall be ignored.
| This is acceptable so long as
|
|  1. The entire functionality of the model is supplied in the shared
|     library.
|  2. All termination actions required by the model are included in the
|     shared library.
|
| The three functions can be included in the shared object library in one of
| the two following combinations:
|
|  Case 1: Shared library has AMI_Init, AMI_Getwave and AMI_Close.
|  Case 2: shared library has AMI_Init and AMI_Close.
|  Case 3: Shared library has only AMI_Init.
|
| Please note that the function 'AMI_Init' is always required.
|
| The interfaces to these functions are defined from three different
| perspectives.  In addition to specifying the 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.
|
|
| 2 APPLICATION SCENARIOS
| =======================
|
|
| Arpad to rewrite this section.
|
| 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 for the [Model] 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 [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.
|
|   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.  For transmitter, it simply passes
|    the output to the receiver AMI_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.

|
|
|
| 3 FUNCTION SIGNATURES
| =====================
|
| 3.1 AMI_Init
| ============
|
| 3.1.1 Declaration
| =================
|
| long AMI_Init (double *impulse_matrix,
|                long row_size,
|                long aggressors,
|                double sample_interval,
|                double bit_time,
|                char *AMI_parameters_in,
|                char **AMI_parameters_out,
|                void **AMI_memory_handle,
|                char **msg)
|
| 3.1.2 Arguments
| ===============
|
| 3.1.2.1 impulse_matrix
| ======================
|
| Impulse matrix is the channel impulse response matrix.  The impulse values
| are in volts and are uniformly spaced in time.  The sample spacing is given
| by the parameter 'sample_interval'.
|
| AMI_Init should be written to support a null **AMI_memory_handle. In this case,
| the AMI_Init will ignore any data, if any, in *impulse_matrix and will still return
| **AMI_parameters_out, and **msg. If **AMI_memory_handle is null then AM_Close will
```

```
| still need to be called before the next call to AMI_Init. The intent of this entry
| is to allow the model developer to implement computational expressions to recalculate
| AMI parameters base on the selected value of other AMI parameters. (WMK: I suspect we
| will need a new Reserved Parameter Computational_Entry that tells the EDA
| tool that AMI_Init has this computational capability.)
|
| The 'impulse_matrix' is stored in 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.  The matrix elements can be retrieved /identified using
|
|   impulse_matrix[idx] = element (row, col)
|   idx = col * number_of_rows + row
|   row - row index , ranges from 0 to row_size-1
|   col - column index, ranges from 0 to aggressors
|
| The first column of the impulse matrix is the impulse response for the
| primary channel.  The rest are the impulse responses from aggressor drivers
| to the victim receiver.
|
| The impulse response of a short lossless channel is a rectangle with a
| width equal to sample_interval (in other words, one discrete sample) and
| a height of 1/sample_interval (to get the unit area).
| The impulse response of a short lossless channel would be element[0,0]=
| 1/sample_interval, element[n] = 0 for all n != 0. If the channel was lossless
| but had a length of 30.3 sample_intervals, then element[30,0]=.667/sample_interval,
| element[31,0]=.333/sample_interval, element[n] = 0 for all n != 30 and 31.
|
| The AMI_Init function may return a modified impulse response by modifying
| the first column of impulse_matrix. If the impulse response is modified,
| the new impulse response is expected to represent the concatenation of the
| model block with the blocks represented by the input impulse response.
| if Init_Returns_Filter is False, or is not specified. If Init_Returns_Filter is True
| the AMI_Init function will return an impulse response of the model block only.
|
| The aggressor columns of the matrix should not be modified.
|
| 3.1.2.2 row_size
| ================
|
| The number of rows in the impulse matrix.
|
| 3.1.2.3 aggressors
| ==================
|
| The number of aggressors in the impulse matrix.
|
| 3.1.2.4 sample_interval:
| ========================
|
| This is the sampling interval of the impulse matrix. Sample_interval is
| usually a fraction of the highest data rate (lowest bit_time) of the
| device.  Example:
|
|   Sample_interval = (lowest_bit_time/64)
|
| 3.1.2.5 bit_time
```

```
| ================
|
| The bit time or unit interval (UI) of the current data, e.g., 100 ps, 200
| ps etc.  The shared library may use this information along with the impulse
| matrix to initialize the filter coefficients.
|
| 3.1.2.6 AMI_parameters (_in and _out)
| =====================================
|
| Memory for AMI_parameters_in is allocated and de-allocate by the EDA.  The
| memory pointed to by AMI_parameters_out is allocated and by the model.
| Memory for AMI_parameters_in is allocated and de-allocated by the EDA platform. The
| memory pointed to by AMI_parameters_out is allocated and de-allocated by the model.
| This is a pointer to a string.  All the input from the IBIS AMI parameter
| file are passed using a string that been formatted as a parameter tree.
|
| Examples of the tree parameter passing is:
|
|    (dll
|      (tx
|        (taps 4)
|        (spacing sync)
|      )
|    )
|
| and
|
|    (root
|      (branch1
|        (leaf1 parameter1 value1)
|        (leaf2 parameter2 value2)
|        (branch2
|          (leaf3 parameter3 value3)
|          (leaf4 parameter4 value4)
|        )
|        (leaf5 parameter5 value5 value6 value7)
|      )
|    )
|
| Note that the only way a parameter can pass more than one value is if the
| parameter is a branch with the sub-parameter Array True.
|
| The syntax for this string is:
|
| 1. Neither names nor individual values can contain white space characters.
|    White space is allowed between a pair of double quotes that delimitate a string.
| 2. Parameter name/value pairs are always enclosed in parentheses, with the
|    value separated from the name by white space.
| 3. A parameter value in a name/value pair can be either a single value or a
|    list of values separated by whitespace.
| 4. Parameter name/value pairs can be grouped together into parameter groups
|    by starting with an open parenthesis followed by the group name followed
|    by the concatenation of one or more name/value pairs followed by a close
|    parenthesis.
| 5. Parameter name/values pairs and parameter groups can be freely
|    intermixed inside a parameter group.
| 6. The top level parameter string must be a parameter group.
```

```
| 7. White space is ignored, except as a delimiter between the parameter name
|    and value.
| 8. Parameter values can be expressed either as a string literal, decimal
|    number or in the standard ANCI 'C' notation for floating point numbers
|    (e.g., 2.0e-9).  String literal values are delimited using a double
|    quote (") and no double quotes are allowed inside the string literals.
|    White space is allowed between the double quotes delimiting a string.
|    Strings that do not contain white space, "(", ")", "'" or "," do not require
|    double quotes.
| 9. A parameter can be assigned an array of values by enclosing the
|    parameter name and the array of values inside a single set of
|    parentheses, with the parameter name and the individual values all
|    separated by white space.
|
| The modified BNF specification for the syntax is:
|
|     <tree>:
|       <branch>
|
|     <branch>:
|       ( <branch name> <leaf list> )
|
|     <leaf list>:
|       <branch>
|       <leaf>
|       <leaf list> <branch>
|       <leaf list> <leaf>
|
|     <leaf>:
|       ( <parameter name> whitespace <value list> )
|
|     <value list>:
|       <value>
|       <value list> whitespace <value>
|     <value>:
|       <string literals>
|       <decimal number>
|       <decimal number>e<exponent>
|       <decimal number>E<exponent>
|
| 3.1.2.7 AMI_memory_handle
| =========================
|
| Used to point to local storage for the algorithmic block being modeled and
| shall be passed back during the AMI_GetWave calls. 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_memory_handle = my_space;
|
| The memory pointed to by AMI_handle is allocated and de-allocated by the
| model.
|
| 3.1.2.8 msg (optional)
| ======================
|
```

```
| Provides descriptive, textual message from the algorithmic model to the EDA
| platform.  It must provide a character string message that can be used by
| EDA platform to update log file or display in user interface.
|
| 3.1.3 Return Value
| ==================
|
| 1 for success
| 0 for failure
|
| 3.2 AMI_GetWave
| ===============
|
| 3.2.1 Declaration
| =================
|
| long AMI_GetWave (double *wave,
|                   long wave_size,
|                   double *clock_times,
|                   char **AMI_parameters_out,
|                   void *AMI_memory);
|
| 3.2.2 Arguments
| ===============
|
| 3.2.2.1 wave
| ============
|
| A vector of a time domain waveform, sampled uniformly at an interval
| specified by the 'sample_interval' specified during the init call.  The
| wave is both input and output.  The EDA platform provides the wave.  The
| algorithmic model is expected to modify the waveform in place.
| An array of a time domain waveform, sampled uniformly at an interval
| specified by the 'sample_interval' specified during the init call. The
| wave is both input and output. The EDA platform provides the wave.
| The algorithmic model is expected to modify the waveform in place by
| applying a filtering behavior, for example, an equalization function,
| being modeled in the AMI_Getwave call.
|
| Depending on the EDA platform and the analysis/simulation method chosen,
| the input waveform could include many components.  For example, the input
| waveform could include:
|
| - The waveform for the primary channel only.
| - The waveform for the primary channel plus crosstalk and amplitude noise.
| - The output of a time domain circuit simulator such as SPICE.
|
| It is assumed that the electrical interface to either the driver or the
| receiver is differential.  Therefore, the sample values are assumed to be
| differential voltages centered nominally around zero volts.  The
| algorithmic model's logic threshold may be non-zero, for example to model
| the differential offset of a receiver; however that offset will usually be
| small compared to the input or output differential voltage.
|
| The output waveform is expected to be the waveform at the decision point of
| the receiver (that is, the point in the receiver where the choice is made
| as to whether the data bit is a "1" or a "0").  It is understood that for
```

```
| some receiver architectures, there is no one circuit node which is the
| decision point for the receiver.  In such a case, the output waveform is
| expected to be the equivalent waveform that would exist at such a node
| were it to exist.
|
| 3.2.2.2 wave_size
| =================
|
| Number of samples in the waveform vectorarray.
|
| 3.2.2.3 clock_times
| ===================
|
| Vector Array to return clock times.  The clock times are referenced to the start
| of the simulation (the first AMI_GetWave call).  The time is always
| greater or equal to zero.  The last clock is indicated by putting a value
| of -1 at the end of clocks for the current wave sample.  The clock_time
| vector array is allocated by the EDA platform and is guaranteed to be greater
| than the number of clocks expected during the AMI GetWave call.  The clock
| times are the times at which clock signal at the output of the clock
| recovery loop crosses the logic threshold.  It is to be assumed that the
| input data signal is sampled at exactly one half bit_time clock period after a
| clock time.
| (WMK Arpad may want to incorporate comments on the care that is needed to calculate
| clock_times because of numerical precision accumulation errors if not done carefully.
| Incrementing times by sample_interval can introduce errors in excess of
| one bit time after simulations > 10**8 bits.
|
|
| 3.2.2.4 AMI_parameters_out (optional)
| =====================================
|
| A handle to a 'tree string' as described in 1.3.1.2.6.  This is used by the
| algorithmic model to return dynamic information and parameters.  The memory
| for this string is to be allocated and deleted by the algorithmic model.
|
| 3.2.2.5 AMI_memory
| ==================
|
| This is the memory which was allocated during the init call.
|
| 3.2.2.6 Return Value
| ====================
|
| 1 for success
| 0 for failure
|
| 3.3 AMI_Close
| =============
|
| 3.3.1 Declaration
| =================
|
|   long AMI_Close(void * AMI_memory);
|
| 3.3.2 Arguments
| ===============
```

```
|
| 3.3.2.1 AMI_memory
| ==================
|
| Same as for AMI_GetWave.  See section 3.2.2.4.
|
| 3.3.3 Return Value
| ==================
|
| 1 for success
| 0 for failure
|
|
| 4 CODE SEGMENT EXAMPLES
| =======================
|
| extern long AMI_GetWave (wave, wave_size, clock_times, AMI_memory);
|
|   my_space = AMI_memory;
|
|    clk_idx=0;
|    time = my_space->prev_time + my_space->sample_interval;
|    for(i=0; i<wave_size; i++)
|      {
|      wave = filterandmodify(wave, my_space);
|      if (clock_times && found_clock (my_space, time))
|        clock_times[clk_idx++] = getclocktime (my_space, time);
|      time += my_space->sample_interval;
|      }
|   clock_times[clk_idx] = -1;   //terminate the clock ~~vector~~array
|   Return 1;
|
|*****************************************************************************
```

ANALYSIS PATH/DATA THAT LED TO SPECIFICATION:

This section of the IBIS 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

Reviewers: Bob Ross, Teraspeed; Michael Mirmak, Intel

REVISION HISTORY CHANGES:

Changes for Bird104.1

```
The text in Notes section just above the KEYWORD DEFINITION
| 2. Throughout the document, terms "long", "double" etc. are used to
| indicate the data types in the ANSI 'C' programming language.
is replaced by
| 2. Throughout the document, terms "long", "double" etc. are used to
| indicate the data types in the C programming language as published in
| ISO/IEC 9899-1999.
```

****