Section 6c

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 M I)

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.

| DEFINITIONS:
|
| 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.
|
| Usage: (required 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
| InOut   Required Input to executable. Executable may return different
| value.
|
| Type: (default is Float)
| Float
| Integer
| String
| Boolean (True/False)
| Tap (For use by TX and RX equalizers)
| 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

| 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>
| After expansion, the 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
| Steps   <typ> <min> <max> <# steps>
| Treat exactly like Increment with
| <delta> == (<max>-<min>)/<# steps>
| 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:

Formatted: Style2

```
|                  (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>:
|     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.
|
|   Description <string>:
|     ASCII string following Description describes a reserved parameter,
|     model specific parameter, or the Algorithmic model itself.  It is used
|     by the EDA platform to convey information to the end-user.  The entire
|     line has to be limited to IBIS line length specification.  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.
|
| 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.
|
| 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.
|
|=============================================================================
```

|==
|
| 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 be in the same directory as the IBIS (.ibs) file.

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.

```
| DEFINITIONS:
|
| 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.
|
|   Usage: (required 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
|     InOut  Required Input to executable.  Executable may return different
|            value.
|
|   Type: (default is Float)
|     Float
|     Integer
|     String
|     Boolean (True/False)
|     Tap (For use by TX and RX equalizers)
|     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

|     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>
|               After expansion, the 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
|     Steps       <typ> <min> <max> <# steps>
|               Treat exactly like Increment with
|               <delta> == (<max>-<min>)/<# steps>
|     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>:
|     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.
|
|   Description <string>:
|     ASCII string following Description describes a reserved parameter,
|     model specific parameter, or the Algorithmic model itself.  It is used
|     by the EDA platform to convey information to the end-user.  The entire
|     line has to be limited to IBIS line length specification.  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.
|
| 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.
|
| 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.
|
|================================================================================
```

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
|                   (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
|                   ...
|                     (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, Use_Init_Output, GetWave_Exists,
|               Max_Init_Aggressors and Ignore_Bits
|
|               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>))
|
|               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.  When this value is set to True,
|               the model returns the convolution of the input impulse
|               response with the impulse response of the equalization.
|
|               GetWave_Exists:
|
|               GetWave_Exists is of usage Info and type Boolean.  It tells
|               the EDA platform whether the "AMI_GetWave" function is
|               implemented in this model.  Note that if Init_Returns_Impulse
|               is set to "False", then Getwave_Exists MUST be set to "True".
|
|               Use_Init_Output:
|
|               Use_Init_Output is of usage Info and type Boolean.  When
|               Use_Init_Output is set to "True", the EDA tool is
|               instructed to use the output impulse response from the
```

AMI_Init function when creating the input waveform
presented to the AMI_Getwave function.

If the Reserved Parameter, Use_Init_Output, is set to
"False", EDA tools will use the original (unfiltered)
impulse response of the channel when creating the input
waveform presented to the AMI_Getwave function.

The algorithmic model is expected to modify the waveform in
place.

Use_Init_Output is optional. The default value for this
parameter is "True".

If Use_Init_Output is False, GetWave_Exists must be True.

The following reserved parameters are optional. If the
following parameters are not present, the values are
assumed as “0”.

Max_Init_Aggressors:

Max_Init_Aggressors is of usage Info and type Integer. It
tells the EDA platform how many aggressor Impulse Responses
the AMI_Init function is capable of processing.

Ignore_Bits:

Ignore_Bits is of usage Info and type Integer. 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.

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_DCD:
|
|               Tx_DCD (Transmit Duty Cycle Distortion) can be of Usage Info
|               and Out.  It can be of Type Float and UI and can have Data
|               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.  Example of
|               TX_DCD declaration is:
|
|               (Tx_DCD (Usage Info)(Type Float)
|                       (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>)(Type <data_type>)
|                                  (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_Receiver_Sensitivity:
|
|               Rx_Receiver_Sensitivity can be of Usage Info and Out and of
|               Type Float and of Data Format Value, Range and Corner.
|               Rx_Receiver_Sensitivity tells the EDA platform 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.  Examples of Rx_Clock_PDF
|               declarations are:
|
|               (Rx_Receiver_Sensitivity (Usage Info)(Type Float)
|                                        (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		Allowed Usage		
	Required	Default	Info	In	Out	InOut
Init_Returns_Impulse	Yes	NA	X			
GetWave_Exists	Yes	NA	X			
Use_Init_Output	No	True	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
Use_Init_Output					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	Value	Range	Corner	Listinc	Step	Gauss	Dual Dirac	Dj DR	Table
Init_Returns_Impulse	X								
GetWave_Exists	X								
Use_Init_Output	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 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>)
(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, this is an error condition
|              for which EDA platform behavior is not specified.
|
|=============================================================================
| Example of Parameter File
|=============================================================================
(mySampleAMI                          | Name given to the Parameter file
  (Description "Sample AMI File")
  (Reserved_Parameters                | Required heading

    (Ignore_Bits (Usage Info) (Type Integer) (Default 21)
      (Description "Ignore 21 Bits"))
    (Max_Init_Aggressors (Usage Info) (Type Integer)(Default 25))
    (Init_Returns_Impulse (Usage Info) (Type Boolean)(Default True))
    (GetWave_Exists (Usage Info) (Type Boolean) (Default True))
  )                                   | End Reserved_Parameters

  (Model_Specific                     | Required heading
    (txtaps
      (-2 (Usage Inout)(Type Tap) (Format Range 0.1 -0.1 0.2)(Default 0.1)
          (Description "Second Precursor Tap"))
      (-1 (Usage Inout)(Type Tap) (Format Range 0.2 -0.4 0.4)(Default 0.2)
          (Description "First Precursor Tap"))
      (0  (Usage Inout)(Type Tap) (Format Range 1 -1 2)(Default 1)
          (Description "Main Tap"))
      (1  (Usage Inout)(Type Tap) (Format Range 0.2 -0.4 0.4)(Default2 0.2)
          (Description "First Post cursor Tap"))
      (2  (Usage Inout)(Type Tap) (Format Range 0.1 -0.1 0.2)(Default 0.1)
          (Description "Second Post cursor Tap"))
    )                                 | End txtaps
    (tx_freq_offset (Format Range 1 0 150) (Type UI) (Default 0))
  )                                   | End Model_Specific
)                                     | End SampleAMI
|
|=============================================================================
| 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]
|
|=============================================================================
|=============================================================================
```

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
|
| 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 three 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
|
|
| 2.1 Linear, Time-invariant Equalization Model
|
|   1. From the system netlist, the EDA platform determines that a given
|      [Model] is described by an IBIS file.
|
|   2. From the IBIS file, the EDA platform determines that the [Model] is
|      described at least in part by an algorithmic model, and that the
|      AMI_Init function of that model returns an impulse response for that
|      [Model].
|
|   3. The EDA platform loads the shared library containing the algorithmic
|      model, and obtains the addresses of the AMI_Init, AMI_GetWave, and
|      AMI_Close functions.
|
|   4. The EDA platform assembles the arguments for AMI_Init. These arguments
```

include the impulse response of the channel driving the [Model], a handle for the dynamic memory used by the [Model], the parameters for configuring the [Model], and optionally the impulse responses of any crosstalk interferers.

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

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

7. The EDA platform completes the rest of the simulation/analysis using the impulse response from AMI_Init as a complete representation of the behavior of the given [Model].

8. Before exiting, the EDA platform calls AMI_Close, giving it the address in the memory handle for the [Model].

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

10. The EDA platform terminates execution.

2.2 Nonlinear, and / or Time-variant Equalization Model

1. From the system netlist, the EDA platform determines that a given block is described by an IBIS file.

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

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

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

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

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

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

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

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

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

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

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

13. The EDA platform terminates execution.

2.3 Reference system analysis flow

System simulations will commonly involve both TX and RX algorithmic models, each of which may perform filtering in the AMI_Init call, the AMI_Getwave call, or both. Since both LTI and non-LTI behavior can be modeled with algorithmic models, the manner in which models are evaluated can affect simulation results. The following steps are defined as the reference simulation flow. Other methods of calling models and processing results may be employed, but the final simulation waveforms are expected to match the waveforms produced by the reference simulation flow.

The steps in this flow are chained, with the input to each step being the output of the step that preceded it.

Step 1. The simulation platform obtains the impulse response for the analog channel. This represents the combined impulse response of the transmitter's analog output, the channel and the receiver's analog front end. This impulse response represents the transmitter's output characteristics without filtering, for example, equalization.

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

| Step 3. The output of Step 2 is presented to the RX model's AMI_Init
| call. If Use_Init_Output for the RX model is set to True, the
| impulse response returned by the RX AMI_Init call is passed
| onto Step 4. If Use_Init_Output for the RX model is set to
| False, the same impulse response passed into Step 3 is passed
| on to step 4.
|
| Step 4. The simulation platform takes the output of step 3 and combines
| (for example by convolution) the input bitstream and a unit
| pulse to produce an analog waveform.
|
| Step 5. The output of step 4 is presented to the TX model's AMI_Getwave
| call. If the TX model does not include an AMI_Getwave call,
| this step is a pass-through step, and the input to step 5 is
| passed directly to step 6.
|
| Step 6. The output of step 5 is presented to the RX model's AMI_Getwave
| call. If the RX model does not include an AMI_Getwave call,
| this step is a pass-through step, and the input to step 6 is
| passed directly to step 7.
|
| Step 7. The output of step 6 becomes the simulation waveform output at
| the RX decision point, which may be post-processed by the
| simulation tool.
|
| Steps 4 though 7 can be called once or can be called multiple times to
| process the full analog waveform. Splitting up the full analog waveform
| into mulitple calls minimized the memory requirement when doing long
| simulations, and allows AMI_Getwave to return model status every so many
| bits. Once all blocks of the input waveform have been processed, TX
| AMI_Close and RX AMI_close are called to perform any final processing
| and release allocated memory.

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'.

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 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 filtered response. The number of items in the matrix should remain unchanged.

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-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 value1)
    (leaf2 value2)
    (branch2
      (leaf3 value3)
      (leaf4 value4)
    )
    (leaf5 value5 value6 value7)
  )
)
```

The syntax for this string is:

1. Neither names nor individual values can contain white space characters.
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.
| 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);
| *serdes_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 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 vector.

3.2.2.3 clock_times

Vector 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 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 clock period after a clock time.

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 3.2.2.5.

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
|   Return 1;
|
|================================================================================
|================================================================================
```