**BUFFER ISSUE RESOLUTION DOCUMENT (BIRD)**

**BIRD ID#: 155.1**

**ISSUE TITLE:** *New AMI API to Resolve Dependent Model Parameter*

**REQUESTER:** *Fangyi Rao and Radek Biernacki, Agilent Technologies, Inc.*

 *Adge Hawes, IBM*

**DATE SUBMITTED:** December 13, 2012

**DATE REVISED:** *August26, 2013*

**DATE ACCEPTED BY IBIS OPEN FORUM:**

**ANALYSIS PATH/DATA THAT LED TO SPECIFICATION:**

AMI model parameters that are used by EDA tools can depend on other model parameters and simulation parameters including data rate, IBIS corner and IBIS model name. The form of such dependency relation varies from IC vendor to IC vendor and from device to device. The number of possible variations among vendors and devices is infinite. Model vendors need a flexible mechanism to implement parameter dependency according to their proprietary formula and pass the dependent parameter values to EDA tools. It’s foreseeable that certain vendors need to conceal the dependency formula.

The proposed approach does not require any ad hoc syntax or rule to be added for new dependency forms. The same DLL can resolve dependent parameters for different IBIS models according to the new reserved parameter Model\_Name. The API is a sensible partition between EDA tool and model, allowing model vendors to have full control on dependency definition as well as implementation.

Two new functions are added to the AMI API, and two new reserved parameters are introduced.

In Section 10.2.1, replace

“The executable model file of a Serializer-Deserializer (SERDES) transmitter or receiver contains up to three functions: “AMI\_Init”, “AMI\_GetWave” and “AMI\_Close”.”

with

“The executable model file of a Serializer-Deserializer (SERDES) transmitter or receiver contains up to five functions: “AMI\_Resolve”, “AMI\_Resolve\_Close”, “AMI\_Init”, “AMI\_GetWave” and “AMI\_Close”.”

Replace

“These functions (AMI\_Init, AMI\_GetWave and AMI\_Close)”

with

“These functions (AMI\_Resolve, AMI\_Resolve\_Close, AMI\_Init, AMI\_GetWave and AMI\_Close)”

Replace

“The three functions can be included in the executable model file in one of the following two combinations:

Case 1: Executable model file has AMI\_Init, AMI\_GetWave and AMI\_Close.

Case 2: Executable model file has AMI\_Init and AMI\_Close.”

with

“The five functions can be included in the executable model file in one of the following four combinations:

Case 1: Executable model file has AMI\_Init, AMI\_GetWave and AMI\_Close.

Case 2: Executable model file has AMI\_Init and AMI\_Close.

Case 3: Executable model file has AMI\_Resolve, AMI\_Resolve\_Close, AMI\_Init, AMI\_GetWave and AMI\_Close.

Case 4: Executable model file has AMI\_Resolve, AMI\_Resolve\_Close, AMI\_Init and AMI\_Close.”

In Section 10.2.3 add:

*Function:* **AMI\_Resolve**

*Required:* No

*Declaration:* AMI\_Resolve (double bit\_time,

 char \* AMI\_parameters\_in,

 char \*\* AMI\_paramters\_out);

*Arguments:*

**bit\_time**

Input argument, in second, equals 1/data rate.

**AMI\_parameters\_in**

Input argument. The format and content of this string are the same as that of the AMI\_parameters\_in argument in AMI\_Init.

**AMI\_parameters\_out**

Output argument, pointer to a string that contains name-value pairs of dependent parameters of Usage Dep. The format of this string is the same as that of the AMI\_parameters\_out argument in AMI\_Init.

*Function:* **AMI\_Resolve\_Close**

*Required:* Yes if AMI\_Resolve exists

*Declaration:* AMI\_Resolve\_Close (char \* AMI\_paramters\_out);

*Arguments:*

**AMI\_parameters\_out**

The AMI\_parameters\_out pointer returned by AMI\_Resolve.

In Section 10.3, add:

Add under “Usage”

“**Dep**

Parameter value is to be assigned by the AMI\_Resolve function”

Also in Section 10.3, replace

“**Out**

Parameter value is coming from the AMI model

**InOut**

Parameter value is a required input to the AMI model. The AMI model may return a different value.”

with

“**Out**

Parameter value is coming from the AMI\_Init and/or AMI\_GetWave functions

**InOut**

Parameter value is a required input to the AMI model. The AMI\_Init and/or AMI\_GetWave functions may return a different value.”

Add to “**RESERVED PARAMETERS REFERENCE”**

*Parameter:*      **Resolve\_Exists**

*Required:*        No

*Descriptors*:

Usage:                   Info

Type:                     Boolean

Format:                  Value

Default:<Boolean\_literal*>*

Description:<string>

*Definition:*       Tells EDA tool whether the model implements the AMI\_Resolve/AMI\_Resolve\_Close function pair

*Usage Rules:*   If omitted, the default is False.

*Other Notes:* Independent parameters must be of Usage In or InOut.

Usage Dep is allowed in .ami files in which the parameter “Resolve\_Exists” is True.

Usage Dep distinguishes parameters returned by AMI\_Resolve, which are of Usage Dep, from those by AMI\_Init and/or AMI\_GetWave, which are of Usage Out or Usage InOut, preventing a parameter from being returned by both AMI\_Resolve and AMI\_Init/AMI\_GetWave.

Tables 17-19 will be modified to add Resolve\_Exists and to include Dep in allowed usage types of jitter parameters.

*Parameter:*      **Model\_Name**

*Required:*        No

*Descriptors*:

Usage:                   In

Type:                     String

Format:                  Value

Default:String\_literal

Description:<string>

*Definition:*       Name of the IBIS [Model] keyword that is being used.

*Usage Rules:* Value specified in the .ami file is ignored. The EDA tool must pass the name of the IBIS [Model] keyword that is being instantiated by the EDA tool through the input parameter strings to AMI\_Resolve and AMI\_Init functions as the value of this parameter.

The usage of the new API is described below.

1. User selects ibis model and specifies corner and data rate.
2. EDA tool initializes AMI\_parameters\_out to NULL.
3. If Resolve\_Exists is False, go to step 9.
4. If Resolve\_Exists is True, EDA tool allocates memory for the AMI\_parameters\_in string and writes to it name-value pairs of all parameters of Usage type In.
5. EDA tool calls AMI\_Resolve before analog channel impulse characterization.
6. DLL computes dependent parameter values according to independent parameter values in AMI\_parameters\_in, bit\_time, corner and model\_name.
7. DLL allocates memory for the AMI\_parameters\_out string and writes to it name-value pairs of dependent parameters.
8. EDA tool sets/adjusts analog model parameters if their values are returned by AMI\_Resolve in AMI\_parameters\_out. EDA tool calls AMI\_Resolve\_Close to release the memory allocated by the DLL in AMI\_Resolve.
9. EDA tool characterizes analog channel impulse responses and finishes the rest of the simulation.

Note that dependent parameters are of Usage Dep, and their values used in the simulation are set by the call to AMI\_Resolve before the call to AMI\_Init. Values of parameters of Usage InOut returned by the AMI\_Init and AMI\_GetWave functions shall not affect the dependent parameter values used in the simulation.

In page 97 (IBIS Version 6.0 Draft):

Replace:

When the extension of the external parameter‘s file name ends with .ami:

a) only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or

the [External Circuit] keywords

with

When the extension of the external parameter‘s file name ends with .ami:

a) in general, only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or the [External Circuit] keywords,

b) for [External Model] models instantiated from within a [Model] that contains the [Algorithmic Model] keyword pointing to the same .ami file, parameters of Usage Dep  are also allowed to be passed to such models.

The new API provides model vendors infinite scalability, extensibility and flexibility to implement dependency relations. It also conceals the dependency formula. It allows any complex dependency relation. A few examples are listed below.

Examples:

Example 1: multi-dimensional functions such as y = f(x1, x2, x3)

Example 2: various interpolation methods

Example 3: various extrapolation methods

Example 4: expression in condition statement such as

 

Example 5: advanced functions such as

 y(tap1, tap2, tap3) = FIR(tap1, tap2, tap3) spectrum at data rate

Example 6:

(Rx\_model

 (Reserved\_Parameters

 (Resolve\_Exists (Usage Info) (Type Boolean) (Value True)

 (Description “Indicates whether DLL implements AMI\_Resolve.”))

 (Model\_Name (Usage In) (Type String) (Value “ignore\_me”)

 (Description “IBIS model name”))

 (Rx\_Receiver\_Sensitivity (Usage Out) (Type Float) (Range 0.0 0.0 0.01)

 (Description “Value depends on OP\_mode and data rate”)) …

 )

 (Model\_Specific

 (Tstonefile (Usage Dep) (Type String) (Value “ignore\_me.s4p”)

 (Description “Rx analog model. Value depends on OP\_mode”))

 (my\_corner (Usage In) (Type String) (Corner “Typ” “Min” “Max”)

 (Description “Informs DLL what corner is selected by user”)) (OP\_mode (Usage In) (Type Integer) (List 0 1 2 3)

 (Description “Operation mode”))

 …

 )

)

In this example, the Rx analog model is represented with a 4-port touchstone file specified by parameter Tstonefile, Both Rx\_Receiver\_Sensitivity and Tstonefile depend on the legacy IBIS model name, parameter my\_corner, and parameter OP\_mode, which specifies the device operation mode. Rx\_Receiver\_Sensitivity also depends on bit\_time. Parameters Model\_Name, my\_corner and OP\_mode, having usage type In, are included in both input parameter strings to AMI\_Resolve and AMI\_Init. Tstonefile is of usage type Dep, and its dependency on Model\_Name, my\_corner and OP\_mode is resolved in AMI\_Resolve, which returns the value of Tstonefile. Rx\_Receiver\_Sensitivity is of usage type Out, and its dependency on Model\_Name, my\_corner, OP\_mode and bit\_time is resolved in AMI\_Init, which returns the value of Rx\_Receiver\_Sensitivity.