**BUFFER ISSUE RESOLUTION DOCUMENT (BIRD)**

**Draft 13**

**BIRD NUMBER:**  *147.1*

**ISSUE TITLE:** *Back-Channel Support*

**REQUESTOR:**  *Marcus Van Ierssel, Snowbush IP; Kumar Keshavan, Ambrish Varma,* *Ken Willis, Cadence Design Systems, Inc.; Bob Ross, Teraspeed Consulting Group*

**DATE SUBMITTED:** *October 18, 2011*

**DATE REVISED:** *August 12, 2014*

**DATE ACCEPTED BY IBIS OPEN FORUM:**

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

Back-channel communication is required for PCI Express Gen 3, 10GBASE-KR, and other emerging serial link standards. This communication ‘provides a mechanism through which the receiver can tune the transmitter equalizer to optimize performance’ [1]. Back-channel capability was initially developed by Sigrity (now Cadence Design Systems) and Snowbush (IP division of Gennum). It was deemed desirable to bring this capability to the IBIS standard in order to encourage other SerDes IP suppliers to enable back-channel functionality for their IP as well.

This BIRD defines how back-channel communications are to be handled in the IBIS specification. It requires BIRD128 (AMI\_GetWave passing AMI\_parameters\_out/in) as prerequisite. This BIRD also entails:

- new Reserved\_Parameters and reserved branches

- definition of a "back-channel" BCI file, with Protocol\_Specific parameters

- flow updates to enable the back-channel training to occur

[1] Section 5, IEEE Std 802.3.

**ANY OTHER BACKGROUND INFORMATION:**

The following documents are provided as supporting material for this BIRD:

- "Extending IBIS-AMI to Support Back-Channel Communications", by Marcus Van Ierssel of Snowbush, Kumar Keshavan of Sigrity, Inc., and Ken Willis of Sigrity, Inc., delivered at the IBIS Summit on Feb. 3, 2011:

 <http://www.sigrity.com/papers/2010/IBIS_AMI_Modeling_May_2010.pdf>

- "BIRD Proposal: Extending IBIS-AMI to Support Back-Channel Communications",

 by Marcus Van Ierssel of Snowbush, Kumar Keshavan of Sigrity, Inc., and Ken Willis of Sigrity, Inc., delivered at the IBIS-ATM subcommittee meeting on March 15, 2011:

<http://www.vhdl.org/pub/ibis/macromodel_wip/archive/20110315/kenwillis/Proposed%20BackChannel%20BIRD%20Modifications/Proposal_BackChannel_BIRD_mods.pdf>

- "BIRD Proposal: Extending IBIS-AMI to Support Back-Channel Communications",

 by Marcus Van Ierssel of Snowbush, Kumar Keshavan of Sigrity, Inc., Ken Willis of Sigrity, Inc., and Walter Katz of SiSoft, Inc, delivered at the IBIS Summit meeting on June 7, 2011:

 <http://www.sigrity.com/papers/2011/Backchannel_June_2011.pdf>

Based on syntactical concerns about Draft 9 and discussions between Ambrish Varma and Bob Ross, Draft 10 was issued to simplify (Type Bits), and remove three Formats: Bit\_Pattern, Bit\_Pattern\_File, and LFSR. Instead, a syntax expanded from an original SiSoft proposal was used to convert three reserved parameters (Preamble, Data, Postamble) to predefined branches and introduce five new reserved parameters: Bit\_Pattern, Bit\_Pattern\_File, Bit\_Pattern\_Instances, LFSR\_Seed, and LFSR\_Taps.

While (Type String) could have been used instead of (Type Bits), the latter Type supports Model\_Specific and Protocol\_Specific parameter applications with only bit data to enable string-specific checking for just a sequence of 0’s and 1’s or just “r” for random bit stream.

A tree diagram for the BCI file is added.

## Introduction (Section 10.1)

(Insert before

‘This section defines how the components of an algorithmic model are specified in an IBIS file.’)

There are scenarios when a receiver and transmitter circuits do not have prior information of the analog channels. Advanced models can perform back-channel communication to tune the transmitter equalizer parameters for optimized performance and adapt to the signature of any analog channel. This is done when transmitter tap parameters are re-configurable and receivers help them to be configured. Advanced communication specifications such as PCI express and IEEE 802.3ap define back-channel training protocols for transmitters and receivers. If both the transmitter and receiver AMI executable models support the same back-channel protocol encapsulated in a back-channel interface parameter definition file (also designated as BCI file) discussed later, the EDA tool will facilitate the channel for communication between the executable models and keep the channel open as long as necessary.

The BCI file for each supporting specification shall be created and managed by the IBIS Open Forum. Official BCI files for each supporting specification shall begin with the prefix ‘ibis\_’. Unofficial back-channel protocol files may be produced without the involvement of the IBIS Open Forum as long as they follow the rules of this specification. Such private protocol files should not use the ‘ibis\_’ prefix.

## New Types (On page 186, Section 10.3, add new type after UI:)

**Bits**

Describes bit patterns that represent a sequence of individual integer bit values expressed in the binary [0:1] numerical system. Data of Type Bits must be enclosed within double quotes to be passed on as strings. The least significant bit in the bit pattern is the right-most bit.

The alphabet “r” of Type Bits is allowed and interpreted as the binary equivalent of a random positive (decimal) integer for the bit value.

Examples of Bits are “01111111100000000” or “r”

**ADD TO SECTION 10.7 (MOVE SECTION 10.7 to SECTION 10.9?) A NEW SUB-SECTION**

## AMI Reserved Parameter DEFINITIONs For Back-Channel Communications

In this section, the parameters Training and Backchannel\_Protocol are documented to enable back-channel communication. These Reserved Parameters are in the AMI file and positioned under the Reserved\_Parameters branch.

*Parameter:* **Training**

*Required:* No.

*Descriptors*:

Usage: In

Type: Integer

Format: List

Default: <numeric\_literal>

Description:<string >

*Definition:* This parameter tells the EDA tool which training modes are available for back-channel communication for the associated executable models. To enable the Getwave based back-channel training in the EDA tool using the Increment/Decrement method, the Training parameter must be set to 1 for BOTH the Tx and Rx. To enable the Getwave based back-channel training in the EDA tool using the Coefficient Passing method, the **Training** parameter must be set to 2 for both Tx and Rx. For the AMI\_Init based back-channel training to be enabled in the EDA tool, the Training parameter must be set to 3 for BOTH Tx and Rx. This indicates that Init based Training is On for both the transmitter and receiver. When Training is 0 for either the transmitter or the receiver, Training will be considered Off. The Increment/Decrement method and the Coefficient Passing methods are described in section 1.7 “COMMUNICATION PROTOCOL BETWEEN THE TX AND RX FOR BACK-CHANNEL”

*Usage Rules:* If Training is not present, its value is assumed 0. Allowed values are only 0, 1, 2 and 3.

*Other Notes:* Training cannot be 1or 2 if the parameter GetWave\_Exists is set to “False”. Training cannot be 3 if the parameter Init\_Returns\_Impulse is set to “False”.

*Example:*

(Training (Usage In) (Type Integer) (List 0 1 2 3)(Default 0)

(Description "This model supports both Init based and Getwave based

back-channel training. Training Mode: 0 is off, 1 is Getwave based using the Increment/Decrement method, 2 is Getwave based using the Coefficient passing method, and 3 is Init based.")

)

Table 1 – Allowed values for Training.

|  |  |
| --- | --- |
| Training | Description |
| 0 | No Training available |
| 1 | Getwave Mode with Increment/Decrement |
| 2 | Getwave Mode with Coefficient Passing |
| 3 | Init Mode |

*Parameter:* **Backchannel\_Protocol**

*Required:* No.

*Descriptors*:

Usage: In

Type: String

Format: Value, List.

Default: <string literal>

Description:<string>

*Definition:* This parameter points to a back-channel parameter interface file (BCI file) using a .bci file extension. This parameter tells the EDA tool which back-channel protocol is to be used for the back-channel training process. The protocol is defined in a standard-specific BCI file. Both the transmitter and receiver for a given through channel must have identical settings for the Backchannel\_Protocol parameter for back-channel training to be enabled. If the settings are different, or if the parameter has "NA" specified for either the Tx, or Rx or both, the EDA tool will assume that back-channel communication is "Off" and will proceed to run the simulation without the back-channel communication.

The name of the BCI file will indicate the protocol described in the file. This name cannot be changed and must end with the .bci extension.

*Usage Rules:*

*Other Notes:*

*Example:*

(Backchannel\_Protocol (Usage In)(Type String)(List "NA" "standard1.bci"

"standard2.bci" "standard3.bci" "standard4.bci")

(Default "standard1.bci")

(Description "This Device can support back-channel training for

multiple standards.")

)

**(Section 10.8 The numbering here is wrong.)**

**11 BACK-CHANNEL PARAMETER INTERFACE FILE DESCRIPTION**

## BCI Parameter DEFINITION File Organization

**INTRODUCTION**

The BCI file (.bci file) follows a similar structure as the AMI file described in Section 10.3, except several new reserved branch names are needed: Preamble, Training\_Pattern, Postamble, Protocol\_Specific (instead of Model\_Specific), and BCI under Protocol\_Specific.

The file shall contain a distinct section or branch named “Reserved\_Parameters” beginning and ending with parentheses. The file may also contain another section or branch named “Protocol\_Specific”, beginning and ending with parentheses. “Reserved\_Parameters” and “Protocol\_Specific” are the only branches permitted to be connected to the root of the tree.

The back-channel parameter definition file shall be organized as shown below:

(my\_BCIname | Root name given to the Parameter file

 (Reserved\_Parameters | Required heading to start the

 | required Reserved\_Parameters

 | section

 ...

 (Reserved Parameter text starting with BCI\_Version)

 (Preamble

 (Preamble Reserved Parameters)

 )

 (Training\_Pattern

 (Training\_Pattern Reserved Parameters)

 )

 (Postamble

 (Postamble Reserved Parameters)

 )

 ...

 ) | End of Reserved\_Parameters

 | section

 (Protocol\_Specific | Required heading to start the

 | optional Protocol\_Specific section

 (BCI | BCI Branch

 (Protocol Specific Parameter text)

 ) | End of BCI Branch

 ) | End of Protocol\_Specific section

 (Description <string>) | description of the model

 | (optional)

 ) | End my\_AMIname parameter file

## BCI Parameter DEFINITION File BranchES and Operation

The Reserved Parameters documented below are solely for the purpose of enabling back-channel communication. A receiver provides information back to its associated transmitter to assist in optimizing that transmitter's equalization parameters in compliance with a particular industry standard protocol. The additional back-channel Reserved Parameters are used only in a BCI file, using a .bci file extension and must not appear in the AMI parameter file.

The Reserved Parameters BCI\_Version, Max\_Train\_Bits, and Training\_Done documented later are positioned directly under the Reserved\_Parameters branch.

Zero or one reserved training branches (Preamble, Training\_Pattern, and Postamble) are permitted in a .bci file. The Preamble branch contains the leading bit pattern that the EDA tool should generate to start back-channel training. The Training\_Pattern branch contains the bit pattern to serve as the body of the frame (containing all of the training pattern bits). The Postamble branch contains the trailing bits used to indicate the end of the training pattern. This is used by the EDA tool to determine the end of the particular training pattern.

If all reserved branches are missing, the training pattern defaults to a random pattern (Type Bits) (Value “r”).

Only the Reserved Parameters Bit\_Pattern\_Instances, Bit\_Pattern, Bit\_Pattern\_File, LFSR\_Seed and LFSR\_Taps can be positioned under the Preamble, Training\_Pattern, and Postamble training branches according to the rules documented later. These Reserved Parameters are used to describe the bit pattern sent from the transmitter to the receiver during the back-channel training.

A .bci file may also contain additional parameters in the "Protocol\_Specific" section. These parameters shall be under the reserved branch “BCI”. The “Protocol\_Specific” section is analogous to the "Model\_Specific" section of an AMI file, and must abide by the same rules and syntax documented in section 10.3. The purpose of the BCI branch is to describe the protocol-specific parameters that are to be passed back and forth between the Tx and Rx AMI models during the back-channel training process.

Any Protocol\_Specific parameter that is outside of the branch named “BCI” shall be ignored by the AMI models and the EDA tool.

Note that the Tx and Rx AMI models utilizing a particular .bci file must support the Protocol\_Specific parameters defined in that BCI file.

The .bci file sets the minimum standard for back-channel communication for a particular protocol. This specification does not restrict the Tx and Rx from implementing and supporting extra taps or any other relevant information.

The Tx AMI executable model will create a parameter string based on the supported protocol indicated by the .ami file reserved parameter Backchannel\_Protocol. This tree string will contain a BCI branch with the branch name “BCI” and will be passed to the Rx AMI executable model using the AMI\_parameters\_out argument in the AMI\_Getwave function.

The Rx AMI executable model will also create a parameter string based on the supported protocol indicated by the .ami file reserved parameter Backchannel\_Protocol. This tree string will contain a BCI branch with the branch name “BCI” and will be passed to the Tx AMI executable model using the AMI\_parameters\_out argument in the AMI\_Getwave function.

The AMI model receiving the tree string will look for the “BCI” root name to extract the BCI tree string.

In the case of a statistical simulation or a time domain simulation without the AMI\_Getwave function (GetWave\_Exists parameter set as "False" and Init\_Return\_Impulse set as “True”) the parameters string from the Tx AMI\_Init function will be passed to the Rx AMI\_Init function and from Rx AMI\_Init function to the the Tx AMI\_Init function through the AMI\_parameters\_out argument in the AMI\_Init function for the Tx and Rx.

Both Tx and Rx AMI model may support multiple protocols in the same model, but both have to point to the same .bci file before training can begin.

## BCI Parameter DEFINITION File Reserved Parameters

*Parameter:* BCI\_Version

*Required:* Yes for BMI\_Version 7.0 and above.

*Descriptors*:

Usage: Info

Type: String

Format: Value

Default:<string\_literal>

Description:<string>

*Definition:* Tells EDA tool the version of the BCI file.

*Usage Rules:* BCI\_Version is required in BCI files for AMI models written in compliance with the IBIS Version 7.0 or later specification(s). This parameter shall be the first parameter defined in the Reserved\_Parameters branch of the BCI file.

The version numbers of .ibs files and AMI models do not have to match. The EDA tool is expected to execute the BCI model according to the rules of the specification which corresponds to its version number.

*Other Notes:* In this document, the shorthand, BCI\_Version <version\_number>, is used to indicate the minimum BCI\_Version level that is supported.

*Examples:*

(BCI\_Version (Usage Info) (Type String) (Value “7.0”)

 (Description “Valid for BCI\_Version 7.0 and above”)

)

*Parameter:* **Max\_Train\_Bits**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Integer

Format: Value

Default: <numeric\_literal>

Description:<string>

*Definition:* Max\_Train\_Bits defines the total number of training bits that can be sent by a transmitter during the back-channel communication. This tells the EDA tool when the back-channel training is complete, unless the receiver does not indicate completion first with the Training\_Done parameter described next.

*Usage Rules:* For Back-Channel Communication. To be used in a BCI file only.

*Other Notes:*

*Example:*

(Max\_Train\_Bits (Usage Info) (Type Integer)(Value100000)

)

*Parameter:* **Training\_Done**

*Required:* No.

*Descriptors*:

Usage: InOut

Type: Boolean

Format: Value

Default: <Boolean\_literal>

Description:<string>

*Definition:* Training\_Done is issued by the receiver executable model to signify the completion of back-channel training. Training\_Done can also be initiated by the EDA tool. In this case the parameter Training\_Done “True” can be passed from the EDA tool to the receiver executable model. Then the receiver executable model will re-issue the parameter Training\_Done “True” to the transmitter executable model to end the training process. The starting point for this parameter is False.

The Rx will append the parameter Training\_Done to the string it issues for Tx when it wants to communicate to the EDA tool that the back-channel training is complete.

*Usage Rules:* For Back-Channel Communication. To be used in a BCI file only.

*Other Notes:*

*Example:*

(Training\_Done (Usage Info)(Type Boolean)(ValueFalse)

)

For time domain simulations, total number of training bits will limited by whichever occurs first: Max\_Train\_Bits or when Rx indicates Training\_Done “True”. If this total number of bits is less than Ignore\_Bits set in the .ami file, the EDA tool will further ignore the balance number of bits before it starts collecting data for analysis. The corollary of this rule is that if Ignore\_Bits is less than the total number of training bits, the remaining balance of bits shall be used for analysis.

RESERVED PARAMETERS FOR THE TRAINING BRANCHES

Two methods to generate bit patterns to be inserted into a stimulus are documented with Reserved Parameters. The first method uses Bit\_Pattern\_Instances, Bit\_Pattern, and Bit\_Pattern\_File, to describe bit patterns directly. The second method uses LFSR\_Seed and LFSR\_Taps to generate a Pseudo Random Bit Sequence (PRBS) with a Linear Feedback Shift Register (LFSR).

Direct Bit Pattern Description Parameters:

*Parameter:* **Bit\_Pattern\_Instances**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Integer

Format: Value

Default: <numeric\_literal>

Description:<string>

*Definition:* Tells the EDA tool how many times the bits described in either the Bit\_Pattern or the Bit\_Pattern\_File parameter documented below is inserted into the stimulus.

*Usage Rules:* If missing, Bit\_Pattern\_Instances defaults to 1. A value of 0 means that the bit pattern repeats forever. The Value must be a non-negative integer. For back-channel communication. To be used in a BCI file only.

*Other Notes:* Bit\_Pattern\_Instances shall be positioned under the Preamble, Training\_Pattern and/or Postamble branches. Bit\_Pattern\_Instances shall not be used with LFSR\_Seed and/or LFSR\_Taps Reserved Parameters documented later. Bit\_Pattern\_Instances require either Bit\_Pattern or Bit\_Pattern\_File parameters documented later and under the same branch.

*Examples:*

(Bit\_Pattern\_Instances (Usage Info)(Type Integer) (Value 2)

(Description “Two instances of the Bit\_Pattern” or the content of

“Bit\_Pattern\_File is used”)

)

(Bit\_Pattern\_Instances (Usage Info)(Type Integer) (Value 0)

(Description “Contents of Bit\_Pattern” or “Bit\_Pattern\_File repeat

forever”)

)

*Parameter:* **Bit\_Pattern**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

Format: Value

Default: <bits\_literal>

Description:<string>

*Definition:* Tells the EDA tool the bit pattern to be inserted in the stimulus.

*Usage Rules:* For back-channel communication. To be used in a BCI file only.

*Other Notes:* Bit\_Pattern shall be positioned under the Preamble, Training\_Pattern, and/or Postamble branches. Bit\_Pattern shall not be used with Bit\_Pattern\_File documented later. Bit\_Pattern shall not be used with LFSR\_Seed and/or LFSR\_Taps Reserved Parameters documented later.

*Examples:*

(Bit\_Pattern (Usage Info)(Type Bits) (Value “0110101001”)

(Description “Bit pattern binary string”)

)

(Bit\_Pattern (Usage Info)(Type Bits) (Value “r”)

(Description “random decimal number converted to binary bit pattern”)

)

*Parameter:* **Bit\_Pattern\_File**

*Required:* No.

*Descriptors*:

Usage: Info

Type: String

Format: Value

Default: <string\_literal>

Description:<string>

*Definition:* Tells the EDA tool the bit pattern file where a pattern of Type Bits is located. This pattern is to be inserted as a stimulus.

*Usage Rules:* For back-channel communication. To be used in a BCI file only.

*Other Notes:* Bit\_Pattern\_File shall be positioned under the Preamble, Training\_Pattern, and/or Postamble branches. Bit\_Pattern\_File shall not be used with Bit\_Pattern. Bit\_Pattern\_File shall not be used with LFSR\_Seed and/or LFSR\_Taps Reserved Parameters documented later.

Any file name extension or no extension is allowed as long as the extension does not conflict with extensions in this document (such as .ibs, .pkg, .ebd, and .ami).

*Example:*

(Bit\_Pattern\_File (Usage Info) (Type String) (Value “bit\_pattern\_1.bpf”)

(Description “Bit pattern file contains a quoted string of Type Bits”)

)

PRBS Generation Using an LFSR:

The LFSR generates the pseudo random bits using the exclusive-or (XOR) based external feedback mechanism, where the XORs are external from the shift register.

An LFSR consists of a series of shift registers where some registers ("taps") feed the XOR gates in its feedback network. The PRBS output is taken from the last stage. An L-stage LFSR produces a repetitive PRBS of length 2L-1.

The last bit is output as the PRBS and also is fed back to the first bit through the XORs determined by the LFSR taps.

A binary seed is used to initialize the LFSR. If the binary seed value is less than the number of LFSR bits, the leading bits will be padded with 0’s. If the seed value is more than the number of LFSR bits, only the required number of bits is used, starting from the least significant bit.

See Figures 1 and 2 below for example implementations of an LFSR.

*Parameter:* **LFSR\_Seed**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

Format: Value

Default: <bits\_literal>

Description:<string>

*Definition:* Tells the EDA tool the seed of type Bits to initialize an LFSR that is used for generating the bit pattern to be inserted in the stimulus.

*Usage Rules:* At least one bit of LFSR\_Seed must be non-zero. For back-channel communication. To be used in a BCI file only.

*Other Notes:* LFSR\_Seed shall be positioned under the Preamble, Training\_Pattern, and/or Postamble branches. LFSR\_Seed shall not be used with Bit\_Pattern\_Instances, Bit\_Pattern, or Bit\_Pattern\_File Reserved\_Parameters.

If LFSR\_Seed is missing, a random seed “r” is assumed.

*Examples:*

(LFSR\_Seed (Usage Info) (Type Bits) (Value “100101101”)

(Description “Bit pattern binary seed containing 9 bits”)

)

(LFSR\_Seed (Usage Info) (Type Bits) (Value “1110111001101011001001111111111”)

(Description “Bit pattern binary seed containing 31 bits”)

)

(LFSR\_Seed (Usage Info) (Type Bits) (Value “r”)

(Description “random decimal number converted to binary LFSR seed”)

)

*Parameter:* **LFSR\_Taps**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Integer

Format: Table

Default: (Illegal)

Description:<string>

*Definition:* Describes the LFSR used for Pseudo Random Bit Stream (PRBS) generation to be inserted in the stimulus.

*Usage Rules:* For back-channel communication. To be used in a BCI file only.

A single row Table shall contain column entries: (<data\_length> <tap1> <tap2> … <tapn>).

The first table column entry <data\_length> is a non-negative (decimal) integer number signifying the length of the data pattern generated by this LFSR in bits. If the value is 0, the LFSR will generate bits forever.

The remaining table column entries <tap1> … <tapn> determine which bit values are used to influence the future bit values. Note that <tap1> … <tapn> are not the same as taps specified for a digital filter such as FFE or DFE. <tap1> must be 1 or greater, and each successive tap entry must be greater than the previous entry. At least two tap entries are required.

*Other Notes:* LFSR\_Taps shall be positioned under the Preamble, Training\_Pattern, and/or Postamble branches. LFSR\_Taps shall not be used with Bit\_Pattern\_Instances, Bit\_Pattern, or Bit\_Pattern\_File Reserved\_Parameters.

*Examples:*



Figure 1: LFSR with 2 taps at the 6th and the 9th bits

(LFSR\_Taps (Usage Info) (Type Integer)

(Table

(Labels “data\_length” “tap1” “tap2”)

(4096 6 9)

)

(Description “LFSR taps for Figure 1 with a seed of 100101101”)

)



Figure 2: LFSR with taps at the 2nd, 6th and 9th bits

(LFSR\_Taps (Usage Info) (Type Integer)

(Table

(Labels “data\_length” “tap1” “tap2” “tap3”)

(4096 2 6 9)

)

(Description “LFSR taps for Figure 2 with a seed of 100101101”)

)

(LFSR\_Taps (Usage Info) (Type Integer)

(Table

(Labels “data\_length” “tap1” “tap2” “tap3” “tap4”)

(4096 3 5 7 11)

)

(Description “PRBS11 Bit Pattern Sequence with LFSR)

)

(LFSR\_Taps (Usage Info) (Type Integer)

(Table

(Labels “data\_length” “tap1” “tap2” “tap3” “tap4”)

(4096 7 19 27 31)

)

(Description “PRBS31 Bit Pattern Sequence with LFSR)

)

FULL EXAMPLE OF A BCI FILE:

(802.3KR

 (Reserved\_Parameters

 (BCI\_Version (Usage Info) (Type String) (Value "6.1")

 )

 (Preamble

 (Bit\_Pattern (Usage Info) (Type Bits)

 (Value “11111111111111110000000000000000”)

 (Description "One Instance of leading preamble pattern")

 )

 }

 (Training\_Pattern

 (LFSR\_Seed (Usage Info) (Type Bits) (Value “11010101011”)

 (Description "Training pattern seed of 11010101011")

 )

 (LFSR\_Taps (Usage Info) (Type Integer)

 (Table

Labels “data\_length” “tap1” “tap2” “tap3”)

 (4096 1 9 11)

 )

 (Description “Training pattern of length 4096 and taps 1, 9, 11”)

 )

 )

 (Postamble

 (Bit\_Pattern\_Instances (Usage Info) (Type Integer) (Value 1)

 (Description “Trailing Postamble pattern instances”)

 )

 (Bit\_Pattern (Usage Info) (Type Bits) (Value “00”)

 (Description “Trailing Postamble Pattern”)

 )

 )

 (Max\_Train\_Bits (Usage Info) (Type Integer) (Value 500000)

 (Description "Number of total training bits allowed")

 )

 (Training\_Done (Usage InOut) (Type Boolean) (Default False)

 (Description "If True then training is done")

 )

 )

 (Protocol\_Specific

 (BCI

 (taps

 (-1 (Usage InOut) (Type Tap) (Range 0 -1 1) (Default 0)

 (Description "Tap parameter -1”))

 (0 (Usage InOut) (Type Tap) (Range 0 -1 1) (Default 0)

 (Description "Tap parameter 0”))

 (1 (Usage InOut) (Type Tap) (Range 0 -1 1) (Default 0)

 (Description "Tap parameter 1”))

 )

 )

 )

)

## Communication Protocol between the Tx and Rx for Back-channel

### Time Domain, AMI\_Getwave flow

For the time domain, Getwave flow, depending on the Training mode the Tx and Rx are set to, the Tx will construct a string with the information about the taps.

If the training mode is 1, the string going from Tx to Rx will instruct the Rx whether the Tx tap coefficient can be incremented or decremented, or if it has reached its upper or lower limits. This is done by specifying the parameter values to be

* 0 for open to be changed
* -1 for reaching its lower limit and
* 1 for reaching its upper limit.

Examples of BCI parameter string that come from the Tx and their brief explanation are provided below:

1. “(BCI (taps (-1 0) (0 0) (1 0)))”: The 3 taps names are -1, 0 and 1 and they are open to be changed by the Rx.
2. “(BCI (taps (-1 -1) (0 0) (1 1)))”: The pre tap (-1) has reached its lower limits indicated by the value -1 and the post tap (1) has reached its upper limit indicated by the value 1

The string coming back from the Rx to the Tx will include instructions for the Tx to increment or decrement a specific tap coefficient by a specified number of units. Each tap instructions will be independent of each other. The Rx can send the instructions in the following manner:

* 0 for no change
* +n for incrementing the tap coefficient by n units, depending on the resolution of the tap coefficient
* -n for decrementing the tap coefficient by n units, depending on the resolution of the tap coefficient.

The Rx can also include the Training\_Done parameter in the BCI string to indicate that training is done.

Examples of BCI parameter string that come from the Rx and their brief explanation are provided below:

1. “(BCI (taps (-1 -1) (0 0) (1 -2)))”: The Rx instructs the Tx to decrement the pre tap by 1 unit and post tap by 2 units
2. “(BCI (Training\_Done True) (taps (-1 0) (0 0) (1 0)))”: The Rx instructs the EDA tool that training is complete and the communication channel between the Tx and Rx back-channel can be closed.

If the training mode is 2, the string going from Tx to Rx will instruct the Rx whether the Tx tap coefficient can be adjusted. The Tx AMI model, based on the BCI file, will create a string that will convey the allowable range for the tap values to the Rx AMI model. The range is specified as value for each tap. The first value is the minimum followed by the maximum value for that tap. If the tap value is a single floating point number, then the Rx cannot change the tap value. This protocol for sending one or two values for tap parameters is unique for back-channel communications.

Note that the constraint specification for each tap is relative to the main tap value of 1.

Please refer to the example provided in the AMI\_Init flow for a sample of the string that passes from Tx to the Rx and back in order to pass coefficients in the Getwave mode.

Private back-channel protocols may choose to use this communication method or use it in conjunction with their own embellishments or use their own communication methods as long as the back-channel string is under the reserved root name “BCI” and follow the parameter tree structure defined in section 10.2. The embellishments and rules (for example, introducing new Protocol\_Specific parameters and processing rules) need to be understood by both the Rx and Tx executable models.

### AMI\_INIT/STatistical Flow

For statistical simulations or time domain simulation using only the AMI\_Init function, the Tx AMI model will create a parameter string which will contain a tree string with the branch name “BCI”. The Tx AMI model, based on the BCI file, will create a string that will convey the allowable range for the tap values to the Rx AMI model. The range is specified as value for each tap. The first value is the minimum followed by the maximum value for that tap. If the tap value is a single floating point number, then the Rx cannot change the tap value. This protocol for sending one or two values for tap parameters is unique for back-channel communications.

Note that the constraint specification for each tap is relative to the main tap value of 1.

Example for the string created by the Tx AMI\_Init and a brief description are included below:

1. “(BCI (taps (-1 -0.25 0) (0 1) (1 -0.3 0.3)))”: The main tap is specified by the tap number 0 with a value of 1. The pre tap (-1) cannot be lower than -0.25 and higher than 0 (-0.25 <= value <= 0). The post tap (1) can have a value between -0.3 and 0.3 (-0.3 <= value <= 0.3).
2. “(BCI (taps (-1 -0.15) (0 0.75) (1 -0.1)))”: The Tx AMI\_Init is conveying to the Rx AMI\_Init the tap values for the 3 taps. In this example the Rx AMI\_Init cannot change the values.

The string coming back from the Rx to the Tx will include the suggested relative values of the taps.

An example for the string created by the Rx AMI\_Init with a brief description are included below:

1. “(BCI (taps (-1 -0.2) (0 1) (1 -0.1)))”: The Rx AMI\_Init is conveying to the Tx the suggested relative tap values that are used to modify the impulse response.

## Reference FLOW change (ReplAce section 10.2.2.3 REFERENCE FLOWS, Paragraph 1, add section 10.2.2.3.1 and advance subsequent bullet numbers)

10.2.2.3 Reference Flows

=================

The next several sections define reference flows for back-channel training (both AMI\_Init and AMI\_Getwave based flows) , statistical analysis, and time domain system analysis simulations. Other methods of calling models and processing results may be employed, but the final simulation waveforms are expected to match the waveforms produced by these reference flows.

A system simulation usually involves a transmitter (Tx) and a receiver (Rx) model with a passive channel placed between them.

Some industry standards for serial link interfaces utilize back-channel communications as a means by which the Rx can communicate back to the Tx to provide guidance as to the equalization settings of the Tx, to optimize for the given channel. Once the back-channel training is completed and the Tx equalization settings are optimized, then time domain or statistical simulation is performed per the reference flows defined later in this specification.

Note that the back-channel AMI\_Init flow describes how the impulse response is modified and handed over to the EDA tool for further processing. The EDA tool does not have any more functional interaction with the AMI models.

The back-channel Getwave flow has two phases. In the first phase, the Tx and Rx AMI models co-optimize their equalization settings. Once that is completed, the standard time domain Getwave flow takes place described in the “TIME DOMAIN SIMULATION REFERENCE FLOW”

10.2.2.3.1 Back-Channel Reference Flow for AMI\_Init based simulation

To enable the back-channel training to occur using the AMI\_Init interface, the .ami files for both Tx and Rx of a given through channel must have the Init\_Returns\_Impulse parameter set as "True", the Training parameter set to "on" and the Backchannel\_Protocol parameter specifying the same back-channel BCI file.

Step 1. The EDA tool 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. The transmitter’s output or receiver’s input characteristics must not include any

filtering effects, for example equalization, in this impulse response, although it may include any

parasitics which are included in the Tx or Rx analog model.

Step 2. The output of Step 1 is presented to the Tx executable model file’s AMI\_Init function.

The Tx AMI\_Init function returns a string conveying the tap constraints described in the section “Communication Protocol between the Tx and Rx for Back-channel”.

Step 3. The output of Step 2 is presented to the Rx executable model file’s AMI\_Init function. Based on the constraints provided by the Tx, the Rx will return a string containing suggested Tx tap settings as described in the section “Communication Protocol between the Tx and Rx for Back-channel”.

Step 4. The output of Step 3 is presented to the Tx AMI\_Init function in the second pass. The Tx AMI\_Init may use the information from the Rx AMI\_Init and modify the impulse response. This impulse response is passed onto Step 5. It will also create a string showing the actual tap values used and pass it to Step 5. This string is formatted as described in the section “Communication Protocol between the Tx and Rx for Back-channel”.

Step 5. The output of Step 4 is presented to the Rx AMI\_Init in the second and final pass. The Rx AMI\_Init will modify the impulse response.

Step 6. The EDA tool completes the rest of the simulation/analysis using the impulse response

calculated in Step 5 by the Rx executable model file’s AMI\_Init function which is a complete

representation of the behavior of a given [Algorithmic Model] combined with the channel.

*Example of Back-Channel Communication for AMI\_Init /Statistical Simulation:*

This section contains an example of an entire cycle of communication between the Tx and the Rx for AMI\_Init based flow.

1. The Tx sends a string to the Rx

“(BCI (taps (-1 -0.2 0.2) (0 1) (1 -0.3 0.4)))”

1. The Rx sends a string back to the Tx

“(BCI (taps (-1 -0.2) (0 1) (1 -0.1)))”

The Tx construes that the pre cursor tap is 20% of the main tap and the post cursor is 10% of the main tap. Another constraint that the Tx may have is to maintain the sum of the coefficients to be 1.

With this additional constraint, the new Tx tap coefficients become (-1 -0.153) (0 0.77) (1 -0.077). These are the actual coefficients used to modify the impulse response.

1. The Tx sends back a new string to Rx

 “(BCI (taps (-1 -0.153) (0 0.77) (1 -0.077)))”

The EDA tool completes the rest of the simulation/analysis in the standard statistical reference flow.

10.2.2.3.1 Back-Channel Training Reference Flow for AMI\_Getwave/Time Domain Simulation

====================================================================

To enable the back-channel training to occur using the Getwave flow, the .ami files for both Tx and Rx of a given through channel must have the GetWave\_Exists parameter set as "True", the Training parameter set to "on" and the Backchannel\_Protocol parameter specifying the same back-channel BCI file.

Step 1. The simulation platform obtains the impulse response for the analog channel, as described in the statistical and time domain simulation flows.

Step 2. The simulation platform produces a digital stimulus waveform as defined per the back-channel BCI file. A digital stimulus waveform is 0.5 when the stimulus is "high", -0.5 when the stimulus is "low", and may have a value between -0.5 and 0.5 such that transitions occur when the stimulus crosses 0.

Step 3. The output of Step 2 is presented to the Tx model's AMI\_GetWave function. If the Rx model's AMI\_GetWave function has written out the Protocol\_Specific parameters from a previous training sequence using the AMI\_parameters\_out argument of the AMI\_Getwave function, these parameters are read in using the AMI\_parameters\_out argument. Then the Tx AMI\_GetWave function is executed. The parameter string is created as described in the section “Communication Protocol between the Tx and Rx for Back-channel” in the “Time Domain, AMI\_Getwave flow”.

The output of the Tx AMI\_GetWave function is passed on to Step 4. The parameters based on the back-channel BCI file are written out by the Tx model's AMI\_GetWave function using the AMI\_parameters\_out argument.

Step 4. The output of Step 3 is convolved with the output of Step 1 by the simulation platform and the result is passed on to Step 5.

Step 5. The output of Step 4 is presented to the Rx model's AMI\_GetWave function, the Protocol\_Specific parameters from the Tx are read in using the AMI\_GetWave’s AMI\_parameters\_out argument, and the Rx AMI\_GetWave function is executed. The Protocol\_Specific parameters are modified and output by the Rx AMI\_GetWave function through the AMI\_parameters\_out argument. The parameter string is created as described in the section “Communication Protocol between the Tx and Rx for Back-channel” in the “Time Domain, AMI\_Getwave flow”.

Step 6. Steps 2-5 are executed iteratively until the Rx model's AMI\_GetWave function returns the value of the Training\_Done parameter as "True", or until the Max\_Train\_Bits parameter defined in the back-channel BCI file is exceeded, whichever occurs first.

Step 7. With the Tx equalization settings optimized through back-channel communication, the "Time domain simulation reference flow" is executed directly.

*Example of Back-Channel Communication for AMI\_Getwave/Time Domain Simulation:*

This section contains an example of an entire cycle of communication between the Rx and the Tx assuming the resolution of all the taps as implemented by the Tx is 1/32 and the starting coefficient for the 3 taps are (-1 -0.03125) (0 0.9375) (1 -0.03125).

1. The Tx sends a string to the Rx

“(BCI (taps (-1 0) (0 0) (1 0)))”

1. The Rx sends a string back to the Tx

“(BCI (taps (-1 -1) (0 0) (1 -2)))”

The Tx construes that the pre cursor tap needs to be decremented by 1/32 and the post cursor needs to be decremented by 2/32. The main cursor will also be reduced by 3/32 in order to maintain the sum of the coefficients to be 1.

The new Tx tap coefficients become (-1 -0.0625) (0 0.84375) (1 -0.09375)

1. The Tx sends back a new string to Rx

 “(BCI (taps (-1 0) (0 0) (1 0)))”

This string would communicate to the Rx that there is still room for more adjustments in the Tx FFE filter if need be.

If after some time, the Tx sends the following string to the Rx:

 “(BCI (taps (-1 -1) (0 0) (1 -1)))”

This may mean that the Tx pre cursor has reached its internal limit of -0.3125 as set by the Tx: (-1 -0.3125) (0 0.375) (1 -0.3125).

This cycle continues till the Rx determines if no more adjustment is needed or if total number of bits for back-channel communication runs out.

The Rx may conclude that training is done by sending the following string back to the Tx:

 “(BCI (Training\_Done True) (taps (-1 0) (0 0) (1 0)))”

The EDA tool intercepts this parameter string and finds the Training\_Done parameter and terminates the back-channel communication by ceasing to transmit the parameters between the Tx and Rx.