**BUFFER ISSUE RESOLUTION DOCUMENT (BIRD)**

**Draft 7**

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

**REQUESTOR:**  *Marcus Van Ierssel, Snowbush IP; Kumar Keshavan, Ambrish Varma,* *Ken Willis, Cadence Design Systems, Inc.; Walter Katz, SiSoft*

**DATE SUBMITTED:** *October 18, 2011, June 19 2013, August 19, 2013*

**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 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

- 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>

## 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 models support the same back-channel protocol encapsulated in a Back-Channel Interface parameter definition file, the EDA tool will facilitate the channel for communication between the models and keep the channel open until necessary.

The back-channel parameter definition file for each supporting specification shall be a created by IBIS Open Forum with participation from interested members. This file will be stored at the same location as the IBIS specification itself.

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

**Bits**

Used to describe bit patterns that represents a sequence of individual integer bit values expressed in binary [0:1] numerical system .

If only the alphabet r is supplied, the EDA tool will use the binary equivalent of a random positive (decimal) integer for the bit value.

Type Bits is used only with Formats Bit\_Pattern, Bit\_Pattern\_File, and LFSR described later.

01010101010101

## New format types (On page 189, add new format types after DjRj:)

**Bit\_Pattern <bits> <repeat\_count>**

Bit\_Pattern defines a block of bits where “bits” are of type Bits followed by a “repeat\_count” which is a non negative (decimal) integer number and is the number of times the bits described in “bits” are to be inserted into the stimulus. If the value is 0, the EDA tool will repeat the bits forever.

Example: (bit\_pattern1 (Usage In) (Type Bits)

 (Bit\_Pattern 11110000111 2))

 (Description "Bit Pattern Sequence using format Bit\_Pattern")

 )

**Bit\_Pattern\_File <”file\_name”> <repeat\_count>**

Bit\_Pattern\_File defines a file named “file\_name” that contains a sequence of binarynumbers of Type Bits followed by a “repeat\_count” which is a non negative (decimal) integer number and is the number of times the bits described in “bits” are to be inserted into the stimulus. If the value is 0, the EDA tool will repeat the bits forever.

Example: (bit\_pattern2 (Usage In) (Type Bits)

 (Bit\_Pattern\_File “abc.bpi” 3))

 (Description "Bit Pattern Sequence using format Bit\_Pattern\_File")

)

**LFSR <LFSR\_taps> <seed> <data\_len>**

LFSR describes a Linear Feedback Shift Register used by the EDA tool for the PRBS generation. The first argument “LFSR\_taps” are integer (decimal) values separated by comma. LFSR\_taps determine which bit values are used to influence the future bit values. Please note that LFSR\_taps are not the same as taps specified for a digital filter such as FFE or DFE. The second argument “seed” is a non-negative binary number represented as Type Bits. At least 1 bit of the seed must be non-zero. The third argument “data\_len” 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.

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 are considered starting from the least significant bit.

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 ("LFSR\_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 as well as fed back to the first bit through the XORs determined by the LFSR\_taps.

Figures 1 and 2 are example implementations of an LFSR.



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



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

Example: (PRBS11 (Usage In) (Type Bits)

 (LFSR 1,9,11 r 4096) (Description "PRBS 11 Bit Pattern Sequence using LFSR with random seed value")

 )

Example: (PRBS31 (Usage In) (Type Bits) (LFSR 1,28,31 1110111001101011001001111111111 4096)

 (Description "PRBS 31 Bit Pattern Sequence using LFSR")

 )

## Parameter DEFINITIONs

Parameters **Training** and **Backchannel\_Protocol** are Reserved\_Parameters for the .AMI file.

*Parameter:* **Training**

*Required:* No.

*Descriptors*:

Usage: In

Type: Boolean

Format: Value.

Default: <Boolean\_literal>

Description:<string >

*Definition:* This parameter tells the EDA platform whether training for back-channel communication is enabled or not for the associated model. For the back-channel training to be enabled in the EDA tool, the **Training** parameter must be set to "True" indicating that Training is On for both the transmitter and receiver of a given through channel. When Training is “False” for either the transmitter or the receiver, Training will be considered Off.

*Usage Rules:* If Training is not present, its value is assumed “False”.

*Other Notes:*

*Examples:*

(Training (Usage In)(Type Boolean) ()

 (Default False) (Description "Turns training on or off")

)

*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 BCI file using a .bci file extension which tells the EDA platform which back-channel protocol is to be used for the back-channel training process. The protocol is defined in a standard-specific back-channel 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 "None" 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 simulation without Back Channel.

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:*

*Examples:*

(Backchannel\_Protocol (Usage In) (Type String) (List "None" "standard1.bci" "standard2.bci" "standard3.bci" "standard4.bci") (Default "standard1.bci") (Description "This Device can support back-channel training for multiple standards.")))

Parameters BCI\_Version, **Preamble**, **Data**, **Postamble**, **Max\_Train\_Bits**, and **Training\_Done** are Reserved\_Parameters that are solely for the purpose of enabling back-channel communication, in which a receiver provides information back to its associated transmitter in order to assist in optimizing that transmitter's equalization parameters, in the context of a particular industry standard. These additional back-channel Reserved Parameters are used only in a back-channel BCI file, using a .bci file extension and must not appear in the AMI parameter file.

Parameters **Preamble, Data** and **Postamble** are used to describe the bit pattern sent from the transmitter to the receiver during the back-channel training. These three parameters shall be contained in a distinct section or branch within the Reserved\_Parameters branch named “**Training\_Pattern**” beginning and ending with parentheses.

Note that the branch named “Training\_Pattern” is only needed if any or all of the parameters **Preamble, Data** and **Postamble** are present in the BCI file.

A BCI file may also contain additional parameters in the "Protocol\_Specific" section which will be under the reserved root name “BCI”. This section is analogous to the "Model\_Specific" section of an AMI file, and must abide by the same rules and syntax. The purpose of this section is to describe the protocol-specific parameters that are to be passed back and forth between the Tx and Rx AMI models during the backchannel training process. 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.

The Tx AMI model will create a parameter string based on the supported protocol indicated by the 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 model through the using the AMI\_parameters\_out argument in the AMI\_Getwave function.

The Rx AMI model will also create a parameter string based on the supported protocol indicated by the 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 model through the using the AMI\_parameters\_out argument in the AMI\_Getwave function.

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

*Parameter:* **BCI\_Version**

*Required:* Yes for AMI\_Version 6.1 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 the parameter definition files of AMI models which are

written in compliance with the IBIS Version 6.1 or later specification(s). When required, this parameter shall be the first parameter defined in the Reserved\_Parameters branch of the BCI parameter definition file.

*Parameter:* **Preamble**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

Format: Bit\_Pattern, Bit\_Pattern\_File, LFSR

Default: <illegal>

Description:<string>

*Definition:* Preamble defines the leading bit pattern that starts a back-channel training Frame.

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

*Other Notes:* This Reserved\_Parameter must be positioned under the Training\_Pattern branch.

*Examples:* (Preamble (Usage Info) (Type Bits) (Bit\_Pattern 11111111111111110000000000000000 1))

*Parameter:* **Data**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

Format: Bit\_Pattern, Bit\_Pattern\_File, LFSR

Default: <illegal>

Description:<string>

*Definition:* This parameter describes the bit pattern that the EDA tool should generate to serve as the body of the Frame.

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

*Other Notes:* This Reserved\_Parameter must be positioned under the Training\_Pattern branch.

*Examples:* (Data (Usage Info) (Type Bits) (LFSR 1,9,11 r 50000))

*Parameter:* **Postamble**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

Format: Bit\_Pattern, Bit\_Pattern\_File, LFSR

Default: <illegal>

Description:<string>

*Definition:* Postamble describes 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.

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

*Other Notes:* This Reserved\_Parameter must be positioned under the Training\_Pattern branch.

*Examples:* (Postamble (Usage Info) (Type Bits) (Bit\_Pattern 1010 1))

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

*Required:* No.

*Descriptors*:

Usage: Info

Type: Integer

Format: Value

Default: <illegal>

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, if the receiver does not indicate it first with the Training\_Done parameter.

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

*Other Notes:*

*Examples:* (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 of usage InOut and is issued by the receiver 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 model. Then the receiver model will re-issue the parameter Training\_Done=True to the transmitter 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 writes out for Tx only when it wants to communicate to the EDA tool that the BackChannel training is complete.

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

*Other Notes:*

*Examples:* (Training\_Done (Usage Info) (Type Boolean) (Default False))

Total number of training bits will equal to the lesser of 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. Corollary of this rule is that if Ignore\_Bits is less than the total number of training bits, no further bits will be ignored.

An example template for a back-channel BCI file is given below:

(802.3KR

 (Reserved\_Parameters

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

 (Training\_Pattern

(Preamble (Usage Info) (Type Bits) (Bit\_Pattern 11111111111111110000000000000000 1) (Description "Leading preamble pattern."))

(Data (Usage Info) (Type Bits) (LFSR 1,9,11 11010101011 4096) (Description "Training pattern with a seed of 11010101011."))

 (Postamble (Usage Info) (Type Bits) (Bit\_Pattern 00 1) (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\_inc\_dec

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

 (Description "Parameter name is standard-specific, and can be any legal Type"))

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

 (Description "Parameter name is standard-specific, and can be any legal Type"))

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

 (Description "Parameter name is standard-specific, and can be any legal Type"))

 )

 )

)

)

## 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, 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.

10.2.2.3.1 Back-Channel Training Reference Flow

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

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 simulation is performed per the "Time domain simulation reference flow" defined later in this specification.

Note that back-channel training does not apply to statistical simulation, as back-channel training utilizes the AMI\_GetWave function in both the Tx and Rx, and is therefore not applicable to statistical simulation.

To enable the back-channel training to occur, 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 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.

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.

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 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.

*Example of BackChannel Communication:*

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\_inc\_dec (-1 0)(0 0)(1 0)))”

1. The Rx sends a string back to the Tx

“(BCI (taps\_inc\_dec (-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 coefficient become (-1 -0.0625) (0 0.84375)(1 -0.09375)

1. The Tx sends back a new string to Rx

 “(BCI (taps\_inc\_dec (-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\_inc\_dec (-1 -1) (0 0) (1 -1)))”

 it may mean, 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\_inc\_dec (-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.