**BUFFER ISSUE RESOLUTION DOCUMENT (BIRD)**

**Draft2**

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

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

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

**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. 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.

**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

## New Types (On page 177, add new type after UI:)

**Bits**

Used to describe bit patterns in Binary (b), Hex (h), Octal (o) or decimal (d) (base 10) format. Strings that begin with b,h,o and d denote Binary, Hex, Octal and decimal respectively.

Examples of Bits are b01111111100000000, h0123456789ABCDEF0123456789ABCDEF, o01234567012345670123456701234567 and d399999. If only the alphabet r is supplied, the EDA tool will use a random positive integer for the bit value.

## New format types (On page 181, 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 integer number and is the number of times the bits described in “bits” are to be inserted into the stimulus. If the value is zero, the EDA tool will repeat the bits forever.

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

 (Bit\_Pattern b11110000111 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 binary, octal or hex numbers of Type Bits followed by a “repeat count” which is a non negative integer number and is the number of times the bits described in “bits” are to be inserted into the stimulus. If the value is zero, 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")

)

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

LFSR is of usage Info and describes the value associated with a linear feedback shift register used by the EDA tool for the PRBS generation. The first argument “taps” are integer values separated by comma. The second argument “seed” is a non-negative number represented as Type Bits. The third argument “data len” is a non negative integer number signifying the length of the data pattern generated by this LFSR in bits. If the value is zero, the LFSR will generate bits forever.

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 d3999999999 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: String

Format: Value, List.

Default: “Off”

Description:<string literal>

*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, theTraining parameter must be set to "On" for both the transmitter and receiver of a given through channel.

*Usage Rules:*

*Other Notes:*

*Examples:*

(Training (Usage In)(Type String) (List "Off" "On")

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

)

*Parameter:* **Backchannel\_Protocol**

*Required:* No.

*Descriptors*:

Usage: In

Type: String

Format: Value, List.

Default: <string literal>

Description:<string literal>

*Definition:* This parameter tells the EDA platform what back-channel protocol is to be used for the back-channel training process. This 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. When calling the Tx and Rx AMI\_Init function, the EDA tool shall pass the value: <full\_path\_to>/<protocol>.bci. The EDA tool is responsible for determining <full\_path\_to>. This file may be located in the same directory as the .ibs, .ami, dll files or may be located in library folders controlled by the EDA tool.

*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 **Preamble**, **Data**, **Postamble**, **Max\_Train\_Bits**, and **TrainingDone** 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.

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.

Also, a BCI file may contain additional parameters in the "Protocol\_Specific" section. 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 define 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 provides all the information that a Model or the EDA tool needs and there is no need for the EDA tool to pass a parameter string to the AMI model like done for Model\_Specific parameters in the .ami file.

*Parameter:* **Preamble**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

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

Default: <illegal>

Description:<string literal>

*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. Part of Training\_Pattern.

*Other Notes:*

*Examples:* (Preamble (Usage Info) (Type Bits) (Bit\_Pattern b11111111111111110000000000000000 1))

*Parameter:* **Data**

*Required:* No.

*Descriptors*:

Usage: Info

Type: Bits

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

Default: <illegal>

Description:<string literal>

*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. Part of Training\_Pattern.

*Other Notes:*

*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 literal>

*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. Part of Training\_Pattern.

*Other Notes:*

*Examples:* (Postamble (Usage Info) (Type Bits) (Bit\_Pattern b1010 1))

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

*Required:* No.

*Descriptors*:

Usage: Info

Type: Integer

Format: Value

Default: <illegal>

Description:<Integer literal>

*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 TrainingDone 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: List

Default: False

Description:<Integer literal>

*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 TrainingDone=True to the transmitter model to end the training process.

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

*Other Notes:*

*Examples:* (Training\_Done (Usage Info) (Type Boolean) (ListFalse True))

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

(802.3KR

 (Reserved\_Parameters

 (Training\_Pattern

(Preamble (Usage Info) (Type Bits) (Bit\_Pattern b11111111111111110000000000000000 1) (Description "Leading preamble pattern."))

(Data (Usage Info) (Type Bits) (LFSR 1,9,11 hEE6B27FF 4096) (Description "Training pattern."))

 (Postamble (Usage Info) (Type Bits) (Bit\_Pattern b00 1) (Description "Trailing postamble pattern."))

 )

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

 (Description "Number of total training bits allowed"))

 (TrainingDone (Usage InOut) (Type Boolean) (List False True)

 (Description "If True then training is done"))

 )

 (Protocol\_Specific

 (PreTap (Usage InOut) (Type Integer) (List -1 0 1) (Default 0)

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

 (MainTap (Usage InOut) (Type Integer) (List -1 0 1) (Default 0)

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

 (PostTap (Usage InOut) (Type Integer) (List -1 0 1) (Default 0)

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

 )

)

## Reference FLOW change (ReplAce section 10.2.3 REFERENCE FLOWS, Paragraph 1, add section 10.2.3.1 and advance subsequent bullet numbers)

10.2.3 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.

10.2.3.1 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, these parameters are read in. Then the Tx AMI\_GetWave function is executed.

The output of the Tx AMI\_GetWave function is passed on to Step 4. The Protocol\_Specific parameters defined in the back-channel BCI file are written out by the Tx model's AMI\_GetWave function.

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, and the Rx AMI\_GetWave function is executed. The Protocol\_Specific parameters are modified and output by the Rx AMI\_GetWave function.

Step 6. Steps 2-5 are executed iteratively until the Rx model's AMI\_GetWave function returns the value of the TrainingDone parameter as "1", or until the Length 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.