10GEC THE 10 GIGABIT ETHERNET CONSORTIUM

PCS Test Suite V1.3b

Technical Document

Last Updated: March 16, 2009 2:30pm

10 Gigabit Ethernet Consortium

University of New Hampshire
InterOperability Laboratory

121 Technology Drive, Suite 2
Durham, NH 03824
Phone: +1-603- 862-0205
Fax: +1-603-862-4181
http://www.iol.unh.edu/consortiums/10gec

MODIFICATION RECORD

- March 16, 2009 Version 1.3b Released
  - Modified test 48.3.1 to include special code-groups from Table 36-2

- September 20, 2005 Version 1.3 Released
  - Modified the procedures for several test cases

- August 1, 2004 Version 1.2 Released
  - Changed some tests
  - Modified Annex for check_end
  - Editorial cleanup
  - Removed Group 5 tests to eventually add to new test suite

- August 13, 2003 Version 1.1 Released
  - Reformatted document with new style
  - Significant changes to check_end test (48.4.2)
  - Additional test cases for loss of alignment test (48.2.2)
  - Added test cases for acquire alignment (48.2.1)
  - Added Annexes on idle spacing and check_end interpretation

- February 28, 2003 Version 1.0 Released
  - Update of all tests with more generic procedures
  - Added Groups 3 and 4

- August 26, 2002 Version 0.1 Released
  - Synchronization and alignment tests

ACKNOWLEDGMENTS

The University of New Hampshire would like to acknowledge the efforts of the following individuals in the development of this test suite.

Eric Ackerson	University of New Hampshire
Jon Carrier	University of New Hampshire
David Estes	University of New Hampshire
Eric Lynskey	University of New Hampshire
Bob Noseworthy	University of New Hampshire
Vinod Venkatavaradan	University of New Hampshire

INTRODUCTION

Overview

The University of New Hampshire's InterOperability Laboratory (IOL) is an institution designed to improve the interoperability of standards based products by providing an environment where a product can be tested against other implementations of a standard. This suite of tests has been developed to help implementers evaluate the functioning of their 10GBASE-X based products. The tests do not determine if a product fully conforms to the IEEE Std. 802.3ae–2002 standard. Additionally, successful completion of all tests contained in this suite does not guarantee that the tested device will operate with other 10GBASE-X capable devices. However, combined with satisfactory operation in the IOL's interoperability test bed, these tests provide a reasonable level of confidence that the Device Under Test (DUT) will function well in many 10Gb/s environments.

Organization of Tests

The tests contained in this document are organized to simplify the identification of information related to a test and to facilitate in the actual testing process. Each test contains an identification section that describes the test and provides cross-reference information. The discussion section covers background information and specifies why the test is to be performed. Tests are grouped by similar functions and further organized by technology. Each test contains the following information:

Test Number

The Test Number associated with each test follows a simple grouping structure. Listed first is the Test Group Number followed by the test's number within the group. This allows for the addition of future tests to the appropriate groups of the test suite without requiring the renumbering of the subsequent tests.

Purpose

The purpose is a brief statement outlining what the test attempts to achieve. The test is written at the functional level.

References

The references section lists cross-references to the IEEE 802.3 standards and other documentation that might be helpful in understanding and evaluating the test and results.

Resource Requirements

The requirements section specifies the hardware, and test equipment that will be needed to perform the test. The items contained in this section are special test devices or other facilities, which may not be available on all devices.

Last Modification

This specifies the date of the last modification to this test.

Discussion
The discussion covers the assumptions made in the design or implementation of the test as well as known limitations. Other items specific to the test are covered here.

Test Setup
The setup section describes the configuration of the test environment. Small changes in the configuration should be included in the test procedure.

Procedure
The procedure section of the test description contains the step-by-step instructions for carrying out the test. It provides a cookbook approach to testing, and may be interspersed with observable results.

Observable Results
The observable results section lists specific items that can be examined by the tester to verify that the DUT is operating properly. When multiple values are possible for an observable result, this section provides a short discussion on how to interpret them. The determination of a pass or fail for a certain test is often based on the successful (or unsuccessful) detection of a certain observable result.

Possible Problems
This section contains a description of known issues with the test procedure, which may affect test results in certain situations.

TABLE OF CONTENTS

GROUP 1: Synchronization

Overview: These tests are designed to verify that the device under test properly synchronizes with the received bit stream. The PCS functions explored are defined in Clause 48 of IEEE Std. 802.3ae-2002, primarily in Figure 48-7.

Test 48.1.1 – COMMA code count

Purpose: To verify how many /COMMA/ codes that the device under test (DUT) needs to receive before it can acquire synchronization.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.1, 48.2.5, Figure 48-7
[2] IEEE Std. 802.3-2002 subclause 36.2.4.9

Resource Requirements:

- A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. When in the LOSS_OF_SYNC state, the PCS attempts to realign its current code-group boundary to the boundary defined by a comma. This process is called code-group alignment. In order for the DUT to acquire synchronization, all four lanes must individually acquire synchronization. Thus, if three lanes have received 4 /K/ and one lane has not received 4 /K/, then the DUT has not fully acquired synchronization. Additionally, as long as the DUT has not acquired synchronization, it is not capable of receiving frames.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (determine how many consecutive ||K|| cause DUT to gain synchronization)

1. Instruct the testing station to transmit a long stream of /Invalid/ code-groups to cause the DUT to enter the LOSS_OF_SYNC state.
2. Instruct the testing station to transmit a stream of valid non /K/ code-groups to the DUT.
3. Instruct the testing station to transmit 1 ||K|| to the DUT.
4. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
5. Instruct the testing station to send a valid frame to the DUT.
6. Repeat steps 1 through 5, increasing the number of ||K|| in step 3 until the DUT receives the frame sent in step 5. Set this number to *comma_count*.

Part B (verify that DUT properly receives COMMA characters other than /K/)

1. Repeat *Part A*, but instead of using /K/ code-groups, use the following code-groups, each of which contains a comma: K28.1, K28.7. Each code-group should be sent *comma_count* times in step 3.

Observable results:

a. The DUT should exit the fault state and receive the frame when *comma_count* is 4 or higher.
b. The DUT should exit the fault state and receive the frame when *comma_count* is 4 or higher.

Possible Problems: When in the LOSS_OF_SYNC state, it is possible that the DUT may need to see several COMMA characters before it has properly aligned to the correct 10-bit boundary. Since the amount of time necessary to properly align is unbounded, it may be difficult to determine the exact number of COMMA characters necessary for the DUT to acquire synchronization. If this happens, then the procedure may be modified such that only a rough time estimate can be provided instead of an exact value for *comma_count*.

Test 48.1.2 – Reception of INVALID codes

Purpose: To verify that the DUT properly reacts to the reception of various invalid code-groups during the synchronization process.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.1, 48.2.5, Figure 48-7
[2] IEEE Std. 802.3-2002 subclause 36.2.4.6

Resource Requirements:

- A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. In order for the DUT to acquire synchronization, all four lanes must individually acquire synchronization. During the synchronization process, if the DUT receives invalid code-groups, then it will return to the LOSS_OF_SYNC state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (verify that the DUT reacts properly to invalid codes when in COMMA_DETECT)

1. Instruct the testing station to transmit a long stream of /Invalid/ code-groups to cause the DUT to enter the LOSS_OF_SYNC state.
2. Instruct the testing station to transmit a stream of valid non /K/ code-groups to the DUT.
3. Instruct the testing station to transmit 1 ||K|| to the DUT.
4. Instruct the testing station to transmit a single column, containing one of the patterns from the table below to the DUT:

1111111111	0000000000
code with RD error	Idle code with single bit error
other invalid codes	

5. Instruct the testing station to transmit *comma_count-1* ||K|| to the DUT.
6. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
7. Instruct the testing station to send valid frames separated by at least minimum IPG to the DUT.
8. Repeat steps 1 through 7, sending *comma_count* ||K|| in step 5.

9. Repeat steps 1 through 8, selecting a different pattern in step 4.

Part B (verify that the DUT reacts properly to invalid codes when in SYNC_ACQUIRED)

1. Send multiple (*invalid_count* and *invalid_count–1*) columns of the previous invalid code-groups, beginning each test case in the SYNC_ACQUIRED_1 state.

Observable results:

a. The DUT should properly transition to the LOSS_OF_SYNC state on each invalid code-group and then regain synchronization and receive the frames when *comma_count* ||K|| have been received.
b. The DUT should properly lose synchronization upon reception of *invalid_count* invalid code-groups. The value of *invalid_count* should be 4.

Possible Problems: The DUT may react differently to the different invalid code-groups. Also, when the DUT returns to the LOSS_OF_SYNC state, it may need to see more than *comma_count* ||K|| before it will acquire synchronization.

Test 48.1.3 – Acquire synchronization

Purpose: To verify that the device under test (DUT) acquires synchronization upon the reception of four columns of four identical Idle Sync code-groups corresponding to the Idle Sync function.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.1, 48.2.5, Figure 48-7
[2] IEEE Std. 802.3-2002 subclause 36.2.4.9

Resource Requirements:

- A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. When in the LOSS_OF_SYNC state, the PCS attempts to realign its current code-group boundary to the boundary defined by a comma. This process is called code-group alignment. In order for the DUT to acquire synchronization, all four lanes must individually acquire synchronization. Thus, if three lanes have received 4 /K/ and one lane has not received 4 /K/, then the DUT has not fully acquired synchronization. Additionally, as long as the DUT has not acquired synchronization, it is not capable of receiving frames.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (transition through COMMA_DETECT_1 with valid codes)

1. Instruct the testing station to transmit a long stream of /Invalid/ code-groups to cause the DUT to enter the LOSS_OF_SYNC state.
2. Instruct the testing station to transmit a stream of valid non /K/ code-groups to the DUT.
3. Using the value of *comma_count* obtained in Part A of test 48.1.1, instruct the testing station to send ||K||, ||R||, and then *n* columns of ||K|| to the DUT, where *n* is *comma_count–2*. For example, if *comma_count* is found to be 4, then initially set the number to 2, and transmit the pattern shown in the following table.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/R/	/R/	/R/	/R/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

4. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
5. Instruct the testing station to send valid frames to the DUT.
6. Repeat steps 1 through 5, increasing the value of *n* in step 3 to *comma_count–1*.
7. Repeat steps 1 through 6, substituting the following for the pattern in step 3, thus testing each lane's ability to acquire synchronization individually:

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/R/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/R/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/R/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/R/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Part B (transition through COMMA_DETECT_1 with invalid codes)

1. Repeat *Part A* substituting the following patterns for step 3 setting the initial value of *n* to *comma_count–1*, and then to *comma_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/Invalid/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/Invalid/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/Invalid/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/Invalid/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Part C (transition through COMMA_DETECT_2 with valid codes)

1. Repeat *Part A* substituting the following patterns for step 3 setting the initial value of *n* to *comma_count–3* and then to *comma_count–2*.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/R/	/R/	/R/	/R/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/R/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/R/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/R/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/R/
/K/	/K/	/K/	/K/

Part D (transition through COMMA_DETECT_2 with invalid codes)

1. Repeat *Part A* substituting the following patterns for step 3 setting the initial value of *n* to *comma_count–3*, and then to *comma_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/Invalid/	/K/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/Invalid/	/K/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/Invalid/	/K/
/K/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/Invalid/
/K/	/K/	/K/	/K/

Part E (transition through COMMA_DETECT_3 with valid codes)

1. Repeat *Part A* substituting the following patterns for step 3 setting the initial value of *n* *comma_count*–4, and then to *comma_count–3*.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/R/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/R/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/R/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/R/

Part F (transition through COMMA_DETECT_3 with invalid codes)

1. Repeat *Part A* substituting the following patterns for step 3 setting the initial value of *n* to *comma_count– 4*, and then to *comma_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/Invalid/	/K/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/Invalid/	/K/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/Invalid/	/K/

Lane 0	Lane 1	Lane 2	Lane 3
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/K/
/K/	/K/	/K/	/Invalid/

Observable results:

a. The DUT should receive frames after receiving a total of *comma_count* /K/ on each lane.
b. The DUT should receive frames only after each lane receives *comma_count* /K/ code-groups without any intervening invalid code-groups.
c. The DUT should receive frames after receiving a total of *comma_count* /K/ on each lane.
d. The DUT should receive frames only after each lane receives *comma_count* /K/ code-groups without any intervening invalid code-groups.
e. The DUT should receive frames after receiving a total of *comma_count* /K/ on each lane.
f. The DUT should receive frames only after each lane receives *comma_count* /K/ code-groups without any intervening invalid code-groups.

Possible Problems: The DUT may respond differently to different /Invalid/ code-groups.

Test 48.1.4 – INVALID code count

Purpose: To verify the number of consecutive /Invalid/ codes that will cause the DUT to lose synchronization once it has arrived in the SYNC_ACQUIRED_1 state.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.5, 48.2.5.2.2, Figure 48-7

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. After acquiring synchronization, the DUT tests received code-groups and uses multiple sub-states, effecting hysteresis, to move between the SYNC_ACQUIRED_1 and LOSS_OF_SYNC states. After receiving multiple invalid code-groups, the DUT may be forced into the LOSS_OF_SYNC state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (determine how many consecutive invalid codes cause DUT to lose synchronization)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to transmit a valid frame to the DUT.
3. Instruct the testing station to transmit the following test pattern to the DUT:

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/Invalid/	/Invalid/	/Invalid/

4. Instruct the testing station to transmit at least one valid non-||K|| column of idle.
5. Instruct the testing station to send a valid frame to the DUT.
6. Repeat steps 1 through 5, increasing the number of ||Invalid|| in step 3 until the DUT no longer receives the frame in step 5, and set this to *invalid_count*.
7. Repeat steps 1 through 6, using each of the following test patterns in step 3, and incrementing the number of invalid code-groups as necessary (at a minimum, use *invalid_count-1* and *invalid_count* /Invalid/ codes).

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/

Observable results:

a. The DUT should discard the middle frame in the sequence when *invalid_count* is set to the value of 4 or higher.

Possible Problems: None

Test 48.1.5 – good_cgs code count

Purpose: To verify the number of valid code groups that are necessary to move the DUT from one of the SYNC_ACQUIRED_xA states to the SYNC_ACQUIRED_(x-1) states.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.5, 48.2.5.2.2, Figure 48-7

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. After acquiring synchronization, the DUT tests received code-groups and uses multiple sub-states, effecting hysteresis, to move between the SYNC_ACQUIRED_1 and LOSS_OF_SYNC states. After receiving multiple invalid code-groups, the DUT may be forced into the LOSS_OF_SYNC state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to transmit the following test pattern to the DUT, where the total number of /Invalid/ columns is *invalid_count*:

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

3. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
4. Instruct the testing station to send a valid frame to the DUT.

5. Repeat steps 1 through 4, increasing the number of ||R|| in step 3 until the DUT receives the frame, and set this to *good_cgs_count*.
6. Repeat steps 1 through 5, using each of the following test patterns in step 3, and incrementing the number of invalid code-groups as necessary (at a minimum, use *good_cgs_count-1* and *good_cgs_count* ||R||).

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

7. Repeat steps 1 through 6, adding additional columns containing /Invalid/ codes so that the good_cgs count is tested in SYNC_ACQUIRED_3A and SYNC_ACQUIRED_4A.

Observable results:

a. The DUT should receive the frame when *good_cgs_count* is set to the value of 4 or higher.

Possible Problems: None

Test 48.1.6 – Loss of synchronization

Purpose: To verify that a DUT will lose synchronization after the reception of code-group sequences which should cause it to return to the LOSS_OF_SYNC state.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.5, 48.2.5.2.2, Figure 48-7

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 28, 2004

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. After acquiring synchronization, the DUT tests received code-groups and uses multiple sub-states, effecting hysteresis, to move between the SYNC_ACQUIRED_1 and LOSS_OF_SYNC states. After receiving multiple invalid code-groups, the DUT may be forced into the LOSS_OF_SYNC state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A1 (transition through SYNC_ACQUIRED_2A to SYNC_ACQUIRED_3)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to transmit *comma_count* ||K|| followed by this test pattern to the DUT, where the number of full ||R|| columns is *good_cgs_count–1*. For example, if the value for *good_cgs_count* is found to be 4, then use 3 ||R||, as shown below. The number of consecutive /Invalid/ columns at the end of the sequence should be *invalid_count–1*, as determined in test 48.1.2. For example, if *invalid_count* is found to be 4 in, then use 3 /Invalid/ columns, as shown below:

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

3. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
4. Instruct the testing station to send a valid frame to the DUT.
5. Repeat steps 1 through 4, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count–1*. The number of consecutive /Invalid/ columns at the end of the sequence should be *invalid_count–1*, as determined in test 48.1.2.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part A2 (transition through SYNC_ACQUIRED_2A to SYNC_ACQUIRED_1)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part B1 (transition through SYNC_ACQUIRED_3A to SYNC_ACQUIRED_4)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count–1*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–2*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part B2 (transition through SYNC_ACQUIRED_3A to SYNC_ACQUIRED_2)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–1*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part C1 (transition through SYNC_ACQUIRED_4A to LOSS_OF_SYNC)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count–1*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–3*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/

Part C2 (transition through SYNC_ACQUIRED_4A to SYNC_ACQUIRED_3)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–2*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part D1 (transition through all states)

1. Repeat *Part A1* substituting the following patterns for step 2. The DUT should not respond to the middle frame for these test cases, as it should lose synchronization. The total number of /Invalid/ columns should be *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/

Observable results:

a. The DUT should never receive the frame in the sequence.
b. The DUT should never receive the frame in the sequence.
c. The DUT should never receive the frame in the sequence.
d. The DUT should never receive the frame in the sequence.

Possible Problems: None

Test 48.1.7 – Maintain Synchronization

Purpose: To verify that the DUT is able to maintain synchronization for a specific set of invalid code-group sequences.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.5.2.2, Figure 48-7

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The synchronization process determines whether the underlying receive channel is ready for operation. The PCS synchronization process continuously monitors the code-groups conveyed through the PMA_UNITDATA.indicate primitive and conveys these code-groups to the PCS Deskew process via the SYNC_UNITDATA.indicate primitive. Four independent processes are run within the PCS, one for each lane. After acquiring synchronization, the DUT tests received code-groups and uses multiple sub-states, effecting hysteresis, to move between the SYNC_ACQUIRED_1 and LOSS_OF_SYNC states. After receiving multiple invalid code-groups, the DUT may be forced into the LOSS_OF_SYNC state and will be unable to properly respond to frames while in this state. However, while going through the hysteresis, the DUT will be in one of the SYNC_ACQUIRED_X or SYNC_ACQUIRED_XA states. While in one of these states, the DUT should maintain synchronization and be able to reply to frames.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A1 (transition through SYNC_ACQUIRED_2)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to transmit the following test pattern to the DUT:

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/

3. Instruct the testing station to transmit a combination of ||A|| and ||R|| to the DUT such that the DUT will be able to set align_status <= OK. This would include sending at least 4 columns of ||A|| spaced 16-31 columns apart and transmitting ||R|| in the gaps between the ||A||, but not sending any ||K|| code-groups.
4. Instruct the testing station to send a valid frame to the DUT.
5. Repeat steps 1 through 4 with the following patterns in step 3:

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/

Part A2 (transition through SYNC_ACQUIRED_2 and 2A)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count-1*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–2*, such that the total number of /Invalid/ columns is *invalid_count–1*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part A3 (transition through SYNC_ACQUIRED_2A back to SYNC_ACQUIRED_1)

1. Repeat *Part A1*, substituting the following patterns for step 3, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–1*, such that the total number of /Invalid/ columns is *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part B1 (transition through SYNC_ACQUIRED_3)

1. Repeat *Part A1* substituting the following patterns for step 2. The number of /Invalid/ codes should be *invalid_count-2*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part B2 (transition through SYNC_ACQUIRED_3 and 3A)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count-1*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–3*, such that the total number of /Invalid/ columns is *invalid_count–1*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/

Part B3 (transition through SYNC_ACQUIRED_3A back to SYNC_ACQUIRED_2)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–2*, such that the total number of /Invalid/ columns is *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part C1 (transition through SYNC_ACQUIRED_4 and 4A)

1. Repeat *Part A1* substituting the following patterns for step 2. The number of /Invalid/ codes should be *invalid_count-1*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part C2 (transition through SYNC_ACQUIRED_4A back to SYNC_ACQUIRED_3)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of full ||R|| columns is *good_cgs_count*. The number of consecutive /Invalid/ columns at the end of the test sequence should be *invalid_count–1*, such that the total number of /Invalid/ columns is *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/

Part D1 (transition through all states)

1. Repeat *Part A1* substituting the following patterns for step 2. The total number of /Invalid/ columns should be *invalid_count-1*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/

Part D2 (transition through all states)

1. Repeat *Part A1*, substituting the following patterns for step 2, where the number of consecutive ||R|| columns is *good_cgs_count –1* in the first two parts of the test vector and *good_cgs_count* in the third. The total number of /Invalid/ columns should be *invalid_count.*

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/Invalid/	/Invalid/	/Invalid/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/Invalid/	/R/	/R/	/R/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/Invalid/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/

Part E

1. Repeat *Part A1*, substituting the following patterns for step 2. The total number of /Invalid/ codes should be *invalid_count*.

Lane 0	Lane 1	Lane 2	Lane 3
/Invalid/	/R/	/R/	/R/
/R/	/Invalid/	/R/	/R/
/R/	/R/	/Invalid/	/R/
/R/	/R/	/R/	/Invalid/

Observable results:

a. The DUT should receive the frame.
b. The DUT should receive the frame.
c. The DUT should receive the frame.
d. The DUT should receive the frame.
e. The DUT should receive the frame.

Possible Problems: None

Test 48.1.8 – Reacquire synchronization

Purpose: To verify that the DUT properly reacquires synchronization once it has established a valid link.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.1, 48.2.5, Figure 48-7
[2] IEEE Std. 802.3-2002 subclause 36.2.4.6

Resource Requirements:

- A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 29, 2004

Discussion: Once the synchronization process is complete, if the entire received bit stream were to be shifted by several UI, then the DUT would begin to see /Invalid/ code-groups and misaligned commas. The DUT would lose synchronization and then attempt to align to the new commas, and reacquire synchronization.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

1. Bring the DUT to a state such that it has established a link and can transmit frames.
2. Instruct the testing station to introduce several UI of skew on each lane.
3. Instruct the testing station to transmit valid idle and frames to the DUT (although the entire stream is now skewed).
4. Observe all transmissions and indications from the DUT.

Observable results:

a. The DUT should properly lose synchronization on the misaligned stream and then acquire synchronization on this new stream.

Possible Problems: If the DUT cannot lose and acquire synchronization and alignment on a small number of columns, then this test may not be able to be performed due to limitations of the testing station.

GROUP 2: Alignment

Overview: These tests are designed to verify that the device under test properly achieves alignment on the received bit stream. The PCS functions explored are defined in Clause 48 of IEEE Std. 802.3ae-2002, primarily in Figure 48-8.

Test 48.2.1 – Determination of align_count

Purpose: To determine the number of ||A|| the DUT needs to receive before it will acquire alignment.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The ||A|| ordered_set contains a special code-group known as the Align, or /A/ code-group in all four lanes. ||A|| ordered_sets are sent in the idle stream, and are transmitted 16-31 columns apart. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After acquiring alignment, the DUT tests received ordered_sets and uses multiple sub-states, effecting hysteresis, to move between the ALIGN_ACQUIRED_1 and LOSS_OF_ALIGNMENT states. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (determine number of ||A|| necessary to achieve alignment)

1. Bring the DUT to a state in which the DUT has achieved synchronization on all four lanes and is in the SYNC_ACQUIRED_1 state and the LOSS_OF_ALIGNMENT state. This can be accomplished by sending the DUT ||K|| and ||R|| but no ||A||.
2. Instruct the testing station to send zero columns of ||A|| to the DUT followed by 16-31 columns of ||K|| and ||R||.
3. Instruct the testing station to transmit a valid frame to the DUT.
4. Repeat steps 1 through 3 increasing the number of ||A|| columns being sent in step 2 until the DUT receives the frame (each ||A|| will be separated by 16-31 non ||A|| columns). Set this number to *align_count*.

Observable results:

a. The DUT should receive the frame when the value of *align_count* is 4.

Possible Problems: The deskew process, initialized by the enable_deskew variable is completely unbounded. This means that an arbitrary number of valid ||A|| columns can be received in the LOSS_OF_ALIGNMENT state before the DUT transitions to the ALIGN_DETECT_1 state. It is possible that the procedure may need to be modified to account for these issues, and that the value of *align_count* may rise above 4.

Test 48.2.2 – Acquire alignment

Purpose: To determine that a DUT will acquire alignment after the reception of 4 identical Idle Align code-groups corresponding to the Idle Align function.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The ||A|| ordered_set contains a special code-group known as the Align, or /A/ code-group in all four lanes. ||A|| ordered_sets are sent in the idle stream, and are transmitted 16-31 columns apart. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After acquiring alignment, the DUT tests received ordered_sets and uses multiple sub-states, effecting hysteresis, to move between the ALIGN_ACQUIRED_1 and LOSS_OF_ALIGNMENT states. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, or backplane).

Procedure:

Part A (acquire alignment through ALIGN_DETECT_1)

1. Bring the DUT to a state in which the DUT has achieved synchronization on all four lanes and is both the SYNC_ACQUIRED_1 state and the LOSS_OF_ALIGNMENT state.
2. Using the value of *align_count* obtained in test 48.2.1, instruct the testing station to send ||A||, deskew error(s), *n* columns of ||A|| to the DUT, where *n* is initially set to *align_count*–1. For example, if 4 columns were needed in Part A, then initially set *n* to 3, as shown in the following table. Following every column or partial column of ||A||, the testing station will transmit an additional 16-31 columns of ||R|| and ||K||.

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/R/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

3. Instruct the testing station to transmit a valid frame to the DUT.
4. Repeat steps 1 through 3, increasing *n* in step 2 to *align_count*.
5. Repeat steps 1 through 4, substituting the following for the pattern in step 2:

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/R/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/R/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/R/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Part B (acquire alignment through ALIGN_DETECT_2)

1. Repeat *Part A* substituting the following patterns for step 2. Instruct the testing station to send ||A||, ||A||, deskew errors, *n* columns of /A/ to the DUT, where *n* is initially set to *align_count*–1.

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/R/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/R/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/R/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Part C (acquire alignment through ALIGN_DETECT_3)

1. Repeat *Part A* substituting the following patterns for step 2. Instruct the testing station to send ||A||, ||A||, ||A||, deskew errors, *n* columns of /A/ to the DUT, where *n* is initially set to *align_count–1*.

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/R/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/R/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/R/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/R/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/
/A/	/A/	/A/	/A/

Observable results:

a. The DUT should receive the frame after receiving 4 consecutive aligned ||A||.
b. The DUT should receive the frame after receiving 4 consecutive aligned ||A||.
c. The DUT should receive the frame after receiving 4 consecutive aligned ||A||.

Possible Problems: The deskew process, initialized by the enable_deskew variable is completely unbounded. This means that an arbitrary number of valid ||A|| columns can be received in the LOSS_OF_ALIGNMENT state before the DUT transitions to the ALIGN_DETECT_1 state. It is possible that the procedure may need to be modified to account for these issues.

Test 48.2.3 – Loss of alignment count

Purpose: To determine the number of misaligned ||A|| code-group sequences which should cause the DUT to return to the LOSS_OF_ALIGNMENT state.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 28, 2004

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The ||A|| ordered_set contains a special code-group known as the Align, or /A/ code-group in all four lanes. ||A|| ordered_sets are sent in the idle stream, and are transmitted 16-31 columns apart. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to send a valid frame to the DUT followed by valid idle.
3. Instruct the testing station to send *n* columns (the initial value of *n* is 1) containing a deskew error to the DUT, with each misaligned ||A|| being separated by 16-31 columns of ||K|| and ||R||, followed by a valid frame.
4. If the DUT does not lose alignment, then repeat steps 1 through 6, increasing the value of *n* in step 3 until the DUT does lose alignment, and set this value to *lose_align_count*.

Observable results:

a. The value of *lose_align_count* should be 4.

Possible Problems: None

Test 48.2.4 – Deskew error identification

Purpose: To determine what the DUT considers to be a deskew error.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to send a valid frame to the DUT followed by valid idle.
3. Instruct the testing station to send *lose_align_count* columns containing a deskew error to the DUT, with each misaligned ||A|| being separated by 16-31 columns of ||K|| and ||R||, followed by a valid frame. Choose the deskew error from the following table:

/A/A/R/R/	/R/A/A/R/	/A/R/A/A/	/R/R/A/A/	/A/R/R/A/	/A/R/A/R/	/R/A/R/A/
/A/A/A/R	R/A/A/A/	/A/A/R/A/	/A/A/A/*/	/A/A/*/A/	/A/*/A/A/	/*/A/A/A/
/A/A/A/#/	/A/A/#/A/	/A/#/A/A/	/#/A/A/A/			

Notes: * represents any code that is one bit different than /A/. # represents an otherwise valid /A/ that has the incorrect running disparity.

4. Repeat steps 1-3 for each deskew error.
5. Repeat steps 1-4 for each deskew error, but with no columns of ||K|| and ||R|| separating the errors.

Observable results:

a. The DUT should lose alignment on each deskew error and not receive the second frame.

Possible Problems: None

Test 48.2.5 – Loss of alignment

Purpose: To verify that a DUT will lose alignment after the reception of code-group sequences which should cause it to return to the LOSS_OF_ALIGNMENT state.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The ||A|| ordered_set contains a special code-group known as the Align, or /A/ code-group in all four lanes. ||A|| ordered_sets are sent in the idle stream, and are transmitted 16-31 columns apart. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After acquiring alignment, the DUT tests received ordered_sets and uses multiple sub-states, effecting hysteresis, to move between the ALIGN_ACQUIRED_1 and LOSS_OF_ALIGNMENT states. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A (deskew errors on lane 0)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the testing station to send a valid frame to the DUT followed by valid idle.
3. Instruct the testing station to send the following pattern to the DUT, with each misaligned ||A|| being separated by 16-31 columns of ||K|| and ||R||, and the total number of deskew errors is *lose_align_count*:

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

4. Instruct the testing station to send a valid frame to the DUT.

5. Repeat steps 1 through 4 with the following patterns in step 3:

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

Part B (deskew errors on lane 1)

1. Repeat *Part A* substituting the following patterns for step 2.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Part C (deskew errors on lane 2)

1. Repeat *Part A* substituting the following patterns for step 2.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Part D deskew errors on lane 3)

1. Repeat *Part A* substituting the following patterns for step 2.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Observable results:

a. The DUT should not receive the second frame.
b. The DUT should not receive the second frame.
c. The DUT should not receive the second frame.
d. The DUT should not receive the second frame.

Possible Problems: If the patterns listed above do not cause a deskew error to be recognized by the DUT, another pattern may be substituted.

Test 48.2.6 – Maintain alignment

Purpose: To verify that the DUT is able to maintain alignment for receiving a specific set of deskew errors.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 28, 2004

Discussion: The deskew process determines whether the underlying receive channel is ready to send coherent data to the XGMII. The deskew process asserts the align_status flag when the PCS has successfully deskewed and aligned all four lanes. Skew is introduced by both active and passive elements of the link. The ||A|| ordered_set contains a special code-group known as the Align, or /A/ code-group in all four lanes. ||A|| ordered_sets are sent in the idle stream, and are transmitted 16-31 columns apart. The DUT uses the ||A|| ordered_sets to remove the skew from and align the four lanes. After acquiring alignment, the DUT tests received ordered_sets and uses multiple sub-states, effecting hysteresis, to move between the ALIGN_ACQUIRED_1 and LOSS_OF_ALIGNMENT states. After receiving multiple deskew errors, the DUT may be forced into the LOSS_OF_ALIGNMENT state and will be unable to properly respond to frames while in this state.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A (maintain alignment on lane 0)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the Testing Station to send a valid frame followed by valid idle to the DUT.
3. Instruct the testing station to send the following pattern to the DUT, where each alignment error is separated by 16-31 columns of ||R|| and ||K||:

Lane 0	**Lane 1**	**Lane 2**	**Lane 3**
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/R/	/R/	/R/	/R/

4. Instruct the testing station to transmit a valid frame to the DUT.
5. Repeat steps 1 through 4 with the following patterns in step 3, increasing the total number of alignment errors from 1 to *lose_align_count–1*in the test cases that do not contain a valid ||A||, and from 1 to *lose_align_count* in the test cases that do contain a valid ||A||:

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/R/	/R/	/R/
/A/	/A/	/A/	/A/
/A/	/R/	/R/	/R/

Part B (maintain alignment on lane 1)

1. Repeat *Part A*, using the following patterns for step 3.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/K/	/K/
/R/	/A/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/R/	/A/	/R/	/R/
/A/	/A/	/A/	/A/
/R/	/A/	/R/	/R/

Part C (maintain alignment on lane 2)

1. Repeat *Part A*, using the following patterns for step 3.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/
/A/	/A/	/A/	/A/
/R/	/R/	/A/	/R/
/R/	/R/	/A/	/R/

Part D (maintain alignment on lane 3)

2. Repeat *Part A*, using the following patterns for step 3.

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/R/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/

Lane 0	Lane 1	Lane 2	Lane 3
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/R/	/R/	/R/	/A/
/A/	/A/	/A/	/A/
/R/	/R/	/R/	/A/

Observable results:

a. The DUT should receive both frames.
b. The DUT should receive both frames.
c. The DUT should receive both frames.
d. The DUT should receive both frames.

Possible Problems: If the patterns listed above do not cause a deskew error to be recognized by the DUT, another pattern may be substituted.

Test 48.2.7 – Reacquire alignment

Purpose: To verify that the DUT is able to maintain alignment for receiving a specific set of deskew errors.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, 48.2.5.2.3, Figure 48-8

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: Once a device has properly acquired alignment and synchronization, certain network behaviors may cause a certain amount of instantaneous skew to be introduced into the link. This skew should cause the DUT to lose alignment and then acquire a new alignment on the incoming data stream.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the Testing Station to send a stream of valid frames to the DUT.
3. Instruct the testing station to introduce and maintain a minimum of 10 UI of skew to the idle stream and transmit a frame.
4. Instruct the testing station to correct the 10 UI of skew and continue to transmit valid frames to the DUT.

Observable results:

a. The DUT should lose alignment when skew is introduced into the idle stream and then regain alignment on the new stream. The DUT should repeat this process when the skew is removed from the idle stream.

Possible Problems: If the DUT cannot lose and regain alignment with a small number of columns, then it may not be possible to perform this test due to testing station capabilities.

Test 48.2.8 – Skew tolerance

Purpose: To verify that the DUT is able to properly achieve alignment and receive frames when skew is introduced to the XAUI link.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2.2, Table 48-5

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 29, 2004

Discussion: Both active and passive elements of a 10GBASE-X link may introduce skew between any of the four lanes. The deskew function has the ability to compensate for this skew at the receiver. The ||A|| ordered_set is a unique code-group, or /A/, on each lane that is not used in any other ordered_set. Since each lane transmits an /A/ simultaneously, there is little skew introduced at the transmitter. The PMA of the transmitter, the PCB the signals traverse, the medium, and the PMA of the receiver are all allowed to introduce various amounts of skew, measured in UI, to the 10GBASE-X link.

Lane0	Lane1	Lane2	Lane3	Lane0	Lane1	Lane2	Lane3	Lane0	Lane1	Lane2	Lane3
+1	0	0	0	-1	0	0	0	0	+1	0	0
+2	0	0	0	-2	0	0	0	0	+2	0	0
+3	0	0	0	-3	0	0	0	0	+3	0	0
+X	0	0	0	-Y	0	0	0	0	+X	0	0

This table shows how the skew can be introduced in either direction on any of the lanes. For this test, only a single lane will be tested at a time. The skew will be increased in one direction on a single lane until the DUT no longer receives frames. Then, the skew will be increased in the other direction on the same lane. The same process will be repeated on each of the other lanes.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multimode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Connect the DUT to the testing station, but turn off the transmitters of the testing station so that the DUT is receiving no signal.
2. Turn on the transmitters of the testing station, and transmit idle followed by a frame to the DUT.
3. Observe counters and other indicators on the DUT.
4. Repeat steps 1-2 and increase the skew on Lane 0 by +1 UI until the DUT no longer receives the frame.

5. Repeat steps 1-4 and increase the skew on Lane 0 by -1 UI until the DUT no longer receives the frame.
6. Repeat steps 1-5 on Lane 1, Lane 2, and Lane 3.

Observable results:

a. The DUT should be tolerant of at least 21 UI total skew on its receiver.

Possible Problems: It may be difficult to determine the amount of skew generated at the receiver of the DUT. The skew can only be calibrated to the point at which the testing station connects to the DUT.

GROUP 3: Transmit Related

Overview: These tests are designed to verify that the device under test transmits properly encoded and formed data and idle streams. The PCS functions explored are defined in Clause 48 of IEEE Std. 802.3ae-2002.

Test 48.3.1 – 8B/10B Encoding

Purpose: To verify that the device under test (DUT) selects the proper encoding for transmitted code-groups.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.3, Figure 48-3, 48.2.4

[2] IEEE Std. 802.3-2002 subclauses 36.2.4.4, 36.2.4.5, Table 36-1, Table 36-2

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: September 20, 2005

Discussion: The PCS transmit process updates its running disparity value after each code-group is sent. The current value of the running disparity is used to select the proper encoding of each transmitted code-group.

In order to adequately test the 8B/10B encoding process, it is necessary to have the device under test transmit all valid data code-groups and all valid special code-groups. This includes /K/, /R/, /A/, /S/, /T/, /K28.1/, /K28.6/, and /K23.7/ for both the positive and negative running disparity. Both forms of each valid data code-group can be generated as part of normal packet data.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A: (Valid data code-groups)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Force the DUT to transmit packets containing both forms of every valid data code-group listed in Table 36-1.
3. Observe all transmissions from the DUT.

Part B: (Valid special codes groups)

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Force the DUT to transmit code-groups containing both forms of every valid special code-group listed in Table 36-2.
3. Observe all transmissions from the DUT.

Observable Results:

a. At all times, the DUT should transmit the correct encoding of the appropriate code-group as specified in Part A step 2.

b. At all times, the DUT should transmit the correct encoding of the appropriate code-group as specified in Part B step 2.

Possible Problems: Part B can only be performed if the device under test has direct access to the XGMII.

Test 48.3.2 – Idle Sequencing

Purpose: To verify that the DUT follows the proper rules for the sequencing of Idle.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2, 48.2.4.2.1, 48.2.4.2.2, 48.2.4.2.3, Figure 48-6.

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 26, 2004

Discussion: Idle ordered_sets, ||I||, are transmitted continuously by the DUT whenever frames are not being transmitted. All idle ordered_sets come across the XGMII as TXD<31:0>=0x07070707 and TXC<3:0>=0xF. The PCS then encodes the idle into the appropriate column of /R/, /K/, or /A/, depending on the rules in 48.2.4.2. Ignoring skew that may be present at the transmitter, the DUT always transmits a full column of the same idle ordered_set. This helps to maintain lane synchronization and alignment at the receiver of the link partner. Additionally, a device can manipulate the idle stream in order to perform clock rate compensation in the PCS layer. The idle sequencing begins with the first full column following a ||T||. This will be the first column to contain idle characters in each of the four lanes. There are several explicit rules that need to be followed with regard to idle sequencing.

The first ||I|| following ||T|| alternates between ||A|| or ||K|| (maintaining ||A|| spacing)
||R|| is always the second full column of idle after ||T||
||A|| is sent after *r* non-||A|| columns, where *r* is uniformly distributed integer between 16 and 31, inclusive. This establishes a minimum and maximum ||A|| spacing.
||K|| and ||R|| are sent with a uniform distribution in the absence of ||A||

These rules, with the help of Figure 48-6 define what the idle stream should look like. It is also specified that one of two 7^{th} order polynomials must be used to generate the pseudo-random idle sequence: X^7+X^6+1, or X^7+X^3+1.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A (check first two columns of idle):

1. Establish a valid link with the DUT.
2. Instruct the DUT to send frames that are at least 116 bytes in length to the testing station.
3. Observe all transmissions from the DUT.
4. Repeat steps 1-2, with frames that are 64 bytes in length.
5. Repeat step 4, attempting to get the DUT to transmit ||Q|| between the frames.

Part B (check ||A|| spacing and ||K||, ||R|| spacing):

1. Establish a valid link with the DUT such that the DUT will send nothing but idle.
2. Observe all transmissions from the DUT.

Observable results:

a1. In Part A, the first column of idle after the ||T|| should alternate between ||A|| and ||K|| when sending frames of at least 116 bytes in length.

a2. In Part A, the first column of idle after the ||T|| should follow the pattern of ||A||, ||K||, ||K||, when sending frames of 64 bytes in length.

a3. In Part A, the second column of idle after the ||T|| should always be ||R|| or ||Q||.

b1. In Part B, the ||A|| spacing should be between 16 and 31 columns, inclusive.

b2. In Part B, the distribution of ||K|| and ||R|| should follow one of the two defined polynomials.

b3. In Part B, the ||A|| spacing should be uniformly distributed, or randomly distributed in the manner fitting one of the two defined polynomials. (See ||A|| spacing Appendix for more details. This part is INFORMATIVE.)

Possible Problems: None.

Test 48.3.3 – cvtx_terminate

Purpose: To verify that the DUT properly fills in the column containing the /T/ with /K/ code-groups.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.3.2, 48.2.6.1.4, Figure 48-6

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: February 28, 2003

Discussion: The XGMII is allowed to transmit a frame of any length, bounded, of course, by the limits set on the MAC in Clause 4. Within those limits, however, any length frame can be transmitted, and therefore the /T/ control character can land on any of the four lanes. If the length of the frame is divisible by 4, then the /T/ code-group will be transmitted on lane 0, and so on for lanes 1, 2, and 3. When the /T/ is transmitted on any lane but lane 3, there exist one or more lanes that need to be filled with idle code-groups before the first full column of idle can be transmitted. In these cases, the DUT must transmit /K/ code-groups to complete the column containing the /T/. For example, if the /T/ code-group was on lane 1, then the DUT would transmit /K/ on lanes 2 and 3.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Bring the DUT to a state in which the DUT is able to receive and transmit data frames.
2. Instruct the DUT to transmit frames on length *n*, where *n* is an integer multiple of 4.
3. Observe all transmissions from the DUT.
4. Repeat steps 1-3 with frames of size *n+1*, and *n+2*.

Observable results:

a. The DUT should complete the column containing the /T/ with /K/ code-groups.

Possible Problems: None

Test 48.3.4 – Fault Transmission

Purpose: To verify that the DUT properly transmits sequence ordered_sets.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 46.3.4, 48.2.4.5.1, 48.2.6.1.4, Figure 48-6.

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 26, 2004

Discussion: Sequence ordered_sets are defined in subclause 46.3.4. When sequence ordered_sets are being sent across the XGMII, the PCS is only allowed to transmit them over the PMA service interface in the column following an ||A|| ordered_set. In most cases, the sequence ordered_sets will be coming across the XGMII continuously, and the PCS will have to discard many of them since an ||A|| is only sent once every 16-31 columns. Sequence ordered_sets do not otherwise interfere with the randomized ||I|| sequence, and are not acted upon or modified by the PCS.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Bring the DUT to a state in which the PCS should be transmitting continuous local fault sequence ordered_sets.
2. Observe transmissions from the DUT.
3. Repeat steps 1-2 with continuous remote fault sequence ordered_sets.

Observable results:

a. When continuous sequence ordered_sets are being transmitted, the DUT should only place ||Q|| ordered_sets after ||A|| ordered_sets.

Possible Problems: None

GROUP 4: Receive Related

Overview: These tests are designed to verify that the device under test follows the rules for the reception of data and idle streams. The PCS functions explored are defined in Clause 48 of IEEE Std. 802.3ae-2002.

Test 48.4.1 – Fault Reception

Purpose: To verify that the DUT properly receives sequence ordered_sets.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 46.3.4, 48.2.4.5.1, 48.2.6.2.4, Figure 48-9.

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: February 28, 2003

Discussion: When align_status=FAIL, the PCS receive process is recognizing that valid data and idle are not yet ready to be received and passed up the stack. Instead, while in the LOCAL_FAULT_INDICATE state, the PCS receiver state machine should pass up local fault ordered_sets to its client. Once the DUT has entered either the DATA_MODE or IDLE_MODE states, it should decode and pass up everything that is being received. If the DUT receives local or remote sequence ordered_sets at this time, then this should be passed to the PCS client.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

1. Bring the DUT to a state in which it can receive and transmit frames.
2. Force the DUT into the LOCAL_FAULT_INDICATE state. One method to do this could be removing the connection between the receiver of the DUT and transmitter of the Link Partner.
3. Observe status and management indicators on the DUT.
4. Repeat steps 1-3 sending local fault sequence ordered_sets to the DUT, and then repeat sending remote fault sequence ordered_sets.

Observable results:

a. While in the LOCAL_FAULT_INDICATE state, the DUT should pass local fault sequence ordered_sets to the PCS client.
b. When receiving local fault sequence ordered_sets, the DUT should pass local fault sequence ordered_sets to the PCS client
c. When receiving remote fault sequence ordered_sets, the DUT should pass remote fault sequence ordered_sets to the PCS client

Possible Problems: None

Test 48.4.2 – check_end

Purpose: To verify that the DUT properly implements the check_end function.

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.3.2, 48.2.6.1.4, Figure 48-9.
[2] Annex B of this test suite
[3] http://www.ieee802.org/3/interp/interp-5-1103.pdf

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: May 2, 2004

Discussion: The check_end function is a prescient terminate function used by the PCS receive process to indicate whether or not a running disparity error was able to propagate through the frame and into idle code-groups in the column containing /T/ or in first full column of idle after ||T||. Due to the nature of the running disparity calculation, it is possible for some running disparity errors to propagate through a frame under certain conditions. All running disparity errors, however, will be caught in either the idle in ||T|| or in the column of idle after ||T||. If a running disparity error is found on any lane in these columns, the PCS will return the XGMII Error control character, thus forcing the frame to become invalid and discarded.

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A (running disparity error in column after ||T||)

1. Bring the DUT to a state in which it can receive and transmit frames.
2. Instruct the testing station to transmit a valid frame to the DUT followed by valid idle.
3. Instruct the testing station to transmit a frame to the DUT that has the /T/ occur on lane 3.
4. Instruct the testing station to have a running disparity error occur on lane 0 of the next column following the ||T||, but valid idle otherwise.
5. Instruct the testing station to transmit a valid frame to the DUT followed by valid idle.
6. Observe status and management indicators on the DUT.
7. Repeat steps 1 – 6 with the RD error located on lanes 1, 2, and 3 respectively.
8. Repeat steps 1 – 7 with the /T/ occurring on lanes 0, 1, and 2, respectively.
9. Repeat steps 1 – 8 replacing the RD error with a code-group other than /A/ or /K/.

Part B (running disparity error in||T||)

1. Bring the DUT to a state in which it can receive and transmit frames.
2. Instruct the testing station to transmit a valid frame to the DUT followed by valid idle.
3. Instruct the testing station to transmit a frame to the DUT that has the /T/ occur on lane 0.

4. Instruct the testing station to have a running disparity error occur on lane 1 of ||T||, but valid idle otherwise.
5. Instruct the testing station to transmit a valid frame to the DUT followed by valid idle.
6. Observe status and management indicators on the DUT.
7. Repeat steps 1 – 6 with the running disparity error located on lanes 2, and 3 respectively.
8. Repeat steps 1 – 7 replacing the RD error with a code-group other than /K/.

Observable results:

a. In Part A, the DUT should always receive the first and third frames. When the error occurs in a column that immediately follows a /D/ code-group, the second frame should always be discarded. When the error occurs in a column that immediately follows a /T/ or /I/ code-group, the second frame should not be discarded.
b. In Part B, the DUT should always receive the first and third frames but never receive the second frame.

This table shows, in greater detail, each test sequence and its observable result.

Number	Test Sequence	Valid Checkend?
1	/ TKKK / RDfmmm KKKK /	Yes
2	/ TKKK / RDmfmm KKKK /	Yes
3	/ TKKK / RDmmfm KKKK /	Yes
4	/ TKKK / RDmmmf KKKK /	Yes
5	/ DTKK / RDfmmm KKKK /	No
6	/ DTKK / RDmfmm KKKK /	Yes
7	/ DTKK / RDmmfm KKKK /	Yes
8	/ DTKK / RDmmmf KKKK /	Yes
9	/ DDTK / RDfmmm KKKK /	No
10	/ DDTK / RDmfmm KKKK /	No
11	/ DDTK / RDmmfm KKKK /	Yes
12	/ DDTK / RDmmmf KKKK /	Yes
13	/ DDDT / RDfmmm KKKK /	No
14	/ DDDT / RDmfmm KKKK /	No
15	/ DDDT / RDmmfm KKKK /	No
16	/ DDDT / RDmmmf KKKK /	Yes
17	/ TKKK / KKKR /	Yes
18	/ TKKK / KKRK /	Yes
19	/ TKKK / KRKK /	Yes
20	/ TKKK / RKKK /	Yes
21	/ DTKK / KKKR /	Yes

Number	Test Sequence	Valid Checkend?
22	/ DTKK / KKRK /	Yes
23	/ DTKK / KRKK /	Yes
24	/ DTKK / RKKK /	No
25	/ DDTK / KKKR /	Yes
26	/ DDTK / KKRK /	Yes
27	/ DDTK / KRKK /	No
28	/ DDTK / RKKK /	No
29	/ DDDT / KKKR /	Yes
30	/ DDDT / KKRK /	No
31	/ DDDT / KRKK /	No
32	/ DDDT / RKKK /	No
33	/ RDmfmm TKKK /	No
34	/ RDmmfm TKKK /	No
35	/ RDmmmf TKKK /	No
36	/ TKKR /	No
37	/ TKRK /	No
38	/ TRKK /	No

Interpretation Note: "/ TKKK / RD fmmm KKKK / " signifies two columns, the first with /T/ on lane 0, the second all /K/s where the Running Disparity (RD) is flipped (f) on lane 0 (and thus an RD error), and maintained (m) on lanes 1,2,3 (and thus not an error)

Possible Problems: None

Test 48.4.3 – Tolerance to ||A|| spacing

Purpose: To determine if the DUT is sensitive to ||A|| spacing

References:

[1] IEEE Std. 802.3ae-2002 subclauses 48.2.4.2, 48.2.4.2.1, 48.2.4.2.2, 48.2.4.2.3, Figure 48-6.

Resource Requirements: A testing station capable of encoding (decoding) 8 bit octets to (from) 10-bit code groups as specified in clause 48 and sending (receiving) these code groups using the signaling method described in clause 47, 53, or 54.

Last Modification: July 29, 2004

Discussion: There is a very explicit rule that needs to be followed with regards to ||A|| spacing. A device that is transmitting Idle must see to it that ||A|| is sent after *r* non-||A|| columns, where *r* is uniformly distributed integer between 16 and 31, inclusive. This establishes a minimum and maximum ||A|| spacing in the absence of frames. There are no explicit rules that deal with sensitivity to the reception of different ||A|| spacing. This test seeks to verify whether the DUT is sensitive to a minimum ||A|| spacing and a maximum ||A|| spacing. For the maximum ||A|| spacing, the DUT must be able to receive two maximum sized frames, separated by 10 bytes of idle before it sees an ||A||. This corresponds to a total of 799 columns, as shown below. Each maximum sized frame of 1522 bytes takes up 380 columns, which is preceded by 2 columns of preamble. Using the assumption that the previous frame transmitted an ||A|| as its first full column of idle, this frame will transmit a ||K|| followed by an ||R||, followed by the beginning of the second frame.

||A||	||R|| and |K||	Pre.	Frame	||T||	||K||	||R||	Pre.	Frame	||T||	||A||
	31	2	380	1	1	1	2	380	1	

Test Setup: Connect the device under test (DUT) to the testing station (transmit to receive, receive to transmit) with the appropriate medium (i.e. multi-mode fiber, single mode fiber, SMA cables, backplane).

Procedure:

Part A (min ||A|| spacing)

1. Bring the DUT to a state in which it has achieved synchronization but not alignment.
2. Transmit *align_count* ||A|| to the DUT, each spaced by 15 columns of ||K|| and ||R||.
3. Transmit a valid frame to the DUT.
4. Observe status and management indicators on the DUT.
5. Repeat steps 1-4 decreasing the number of non ||A|| columns in step 2 until the DUT no longer receives the frame.

Part B (max ||A|| spacing)

1. Bring the DUT to a state in which it has achieved synchronization but not alignment.
2. Transmit *align_count* ||A|| to the DUT, each spaced by 799 columns of ||K|| and ||R||.
3. Transmit a valid frame to the DUT.

4. Observe status and management indicators on the DUT.
5. Repeat steps 1-4 increasing the number of non ||A|| columns in step 2 until the DUT no longer receives the frame.

Part C (maintain link with no ||A||)

1. Bring the DUT to a state in which it has achieved synchronization and alignment.
2. Modify the idle sequence being sent to the DUT, without bringing the link down, such that the DUT will be receiving valid ||R|| and ||K|| but not ||A||.
3. Transmit a valid frame to the DUT.
4. Observe status and management indicators on the DUT.

Observable results:

a. The min ||A|| acceptable spacing should be between 0 and 16.
b. The max ||A|| acceptable spacing should be greater than 799 columns.
c. The DUT should maintain the link even if it is receiving no ||A||.

Possible Problems: None

Appendix A: A_CNT Simulation

The IEEE Std. 802.3ae-2002 (standard for 10 Gigabit Ethernet) defines a mechanism for scrambling the idle stream when using a 10-gigabit attachment unit interface (XAUI). Specifically, the Standard specifies that the spacing of the special characters ||A|| must be uniformly and randomly distributed between 16 and 31 columns, inclusive, and that the generation of the polynomial used to create this distribution must be one of two specified polynomials: X^7+X^6+1, or X^7+X^3+1. Additionally, it is specified that when not sending ||A||, that ||K|| and ||R|| should be sent with a random uniform distribution using the same polynomials. This simulation shows that using the specified polynomials in the manner defined by the standard does not give a uniform distribution of ||A|| spacing and does not give a uniform number ||K|| and ||R|| columns.

In general, one of the purposes for scrambling is to reduce electromagnetic interference (EMI) while sending idle, which is sent in the absence of data. This randomized stream produces no discrete spectrum, thus reducing the interference that could arise from transmitting a less random pattern. The benefits of scrambling the idle stream are not in question, but rather the manner and means by which the Standard says the scrambling shall take place is contradictory and can be confusing, especially from the point of view of how to test against the Standard.

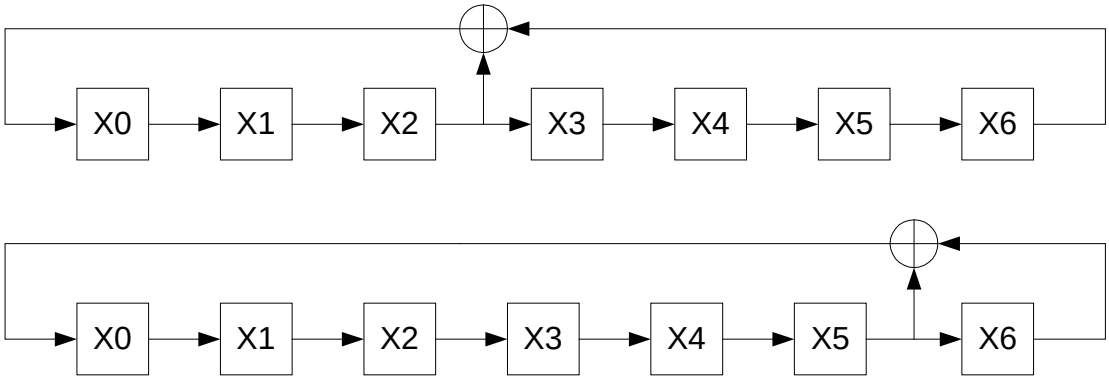


Figure 1. Idle Scrambler Implementations

Figure 1 shows a pictorial representation of the two defined scrambler polynomials, where the top scrambler is X^7+X^3+1 and the bottom scrambler is X^7+X^6+1. Note that there are 127 possible settings of this scrambler. The case in which all bits are set to zero is not a valid case since the state of the scrambler would never change. In both cases, the output of the scrambler, representing the code_sel variable and the four bits used to feed the A_CNT block are undefined. This simply means that any bit and any four different bits can be used as the output and A_CNT feed, respectively. During idle, when an ||A|| is not being sent and the output of the scrambler is a 1, an ||R|| should be sent, and when the output is a 0, a ||K|| will be sent. Whenever the 5-bit A_CNT counter decrements to zero, an ||A|| will be sent and then a new four bits will be shifted into this counter from the random integer generator. This will set the ||A|| spacing to some random integer between 16 and 31 columns, inclusive.

Uniform ||K|| and ||R|| Spacing

When one of the pseudo-random number generators is sampled after each clock, then a 127-bit repeating pseudo-random number will be produced. Since the pattern will repeat after 127 clock cycles, then it is clear that there will be 64 bits taking on a value of one, and 63 bits taking on a value of zero after 127 clocks. This pattern will then repeat. The tables shown below depict the repeating pattern of /K/ and /R/ code-groups without the introduction of any /A/ code-groups, which would overwrite the /K/ or /R/. The DUT should transmit one of these two distributions of /K/ and /R/ code-groups. For these tables, each column is read from top to bottom and then left to right. It should also be noted that observation of the idle transmitted from the DUT may begin at any point in these tables, and thus, the observed idle should be shifted to align with the tables.

X^7+X^6+1			
K28.0	K28.0	K28.5	K28.0
K28.0	K28.0	K28.5	K28.5
K28.0	K28.0	K28.5	K28.5
K28.0	K28.5	K28.0	K28.5
K28.0	K28.5	K28.0	K28.0
K28.0	K28.0	K28.0	K28.0
K28.0	K28.5	K28.5	K28.5
K28.5	K28.5	K28.5	K28.0
K28.5	K28.5	K28.5	K28.5
K28.5	K28.0	K28.0	K28.5
K28.5	K28.5	K28.5	K28.0
K28.5	K28.0	K28.5	K28.5
K28.5	K28.0	K28.0	K28.0
K28.0	K28.5	K28.5	K28.0
K28.5	K28.5	K28.5	K28.0
K28.5	K28.0	K28.0	K28.5
K28.5	K28.0	K28.0	K28.0
K28.5	K28.0	K28.5	K28.0
K28.5	K28.5	K28.0	K28.0
K28.0	K28.0	K28.0	K28.5
K28.0	K28.5	K28.5	K28.5
K28.5	K28.0	K28.0	K28.0
K28.5	K28.5	K28.5	K28.0
K28.5	K28.0	K28.5	K28.0
K28.5	K28.5	K28.0	K28.5
K28.5	K28.0	K28.0	K28.5
K28.0	K28.0	K28.5	K28.0
K28.5	K28.0	K28.0	K28.5
K28.0	K28.0	K28.0	K28.0
K28.5	K28.0	K28.0	K28.5
K28.5	K28.5	K28.0	K28.0
K28.5	K28.0	K28.5	K28.5
K28.0	K28.5	K28.0	

X^7+X^3+1			
K28.0	K28.0	K28.5	K28.0
K28.0	K28.5	K28.0	K28.5
K28.0	K28.0	K28.0	K28.5
K28.0	K28.5	K28.5	K28.5
K28.0	K28.0	K28.5	K28.5
K28.0	K28.5	K28.5	K28.5
K28.0	K28.5	K28.5	K28.5
K28.5	K28.5	K28.5	K28.0
K28.5	K28.5	K28.0	K28.5
K28.5	K28.0	K28.0	K28.5
K28.0	K28.5	K28.5	K28.0
K28.0	K28.0	K28.0	K28.5
K28.0	K28.0	K28.0	K28.5
K28.5	K28.5	K28.5	K28.0
K28.0	K28.0	K28.0	K28.0
K28.0	K28.0	K28.5	K28.5
K28.5	K28.0	K28.0	K28.0
K28.5	K28.0	K28.0	K28.5
K28.5	K28.5	K28.0	K28.5
K28.0	K28.5	K28.5	K28.0
K28.5	K28.0	K28.0	K28.0
K28.0	K28.0	K28.5	K28.0
K28.5	K28.0	K28.5	K28.0
K28.5	K28.5	K28.5	K28.5
K28.0	K28.5	K28.0	K28.0
K28.5	K28.0	K28.0	K28.0
K28.0	K28.5	K28.5	K28.0
K28.0	K28.0	K28.5	K28.5
K28.5	K28.0	K28.5	K28.5
K28.0	K28.0	K28.5	K28.5
K28.0	K28.0	K28.5	K28.5
K28.5	K28.5	K28.5	

Uniform ||A|| Spacing

A bit that chooses between ||K|| and ||R|| is produced during every clock cycle. The counter that determines when it is time to send an ||A|| is only evaluated and reset every 16-31 clock cycles. When the A_CNT variable reaches zero, an ||A|| is sent and a new value is loaded into the counter. The four bits loaded into the counter can take on any value from zero through fifteen, and these are combined with an MSB that is always high, thus initializing the counter to 16-31. This counter will then start to count down, decrementing once for every clock cycle. While this counter is decrementing, the pseudo-random generator is continuing to shift through its 127-bit cycle, and ||R|| and ||K|| codes are being transmitted. Thus, the bits being fed to the A counter are not being fed continuously, but only after the expiration of the current A counter.

If a separate pseudo-random generator existed that only provided a new value upon the expiration of A_CNT, then the next sequence would get shifted into the counter and no possible combinations would be missed. If the pseudo-random generator generated its first four output states as W_1, W_2, W_3, and W_4 (where each W_n is 4-bits in length and takes on a decimal value between 0 and 15, inclusive) then the A counter would be initialized with W_1+16. After A_CNT expired, which would be W_1+16 clock cycles, the new value of the counter would be set to W_2+16. This same process would then continue through all 127 different combinations: W_4...W_{127}. However, this is not what happens within a device with a single random number generator.

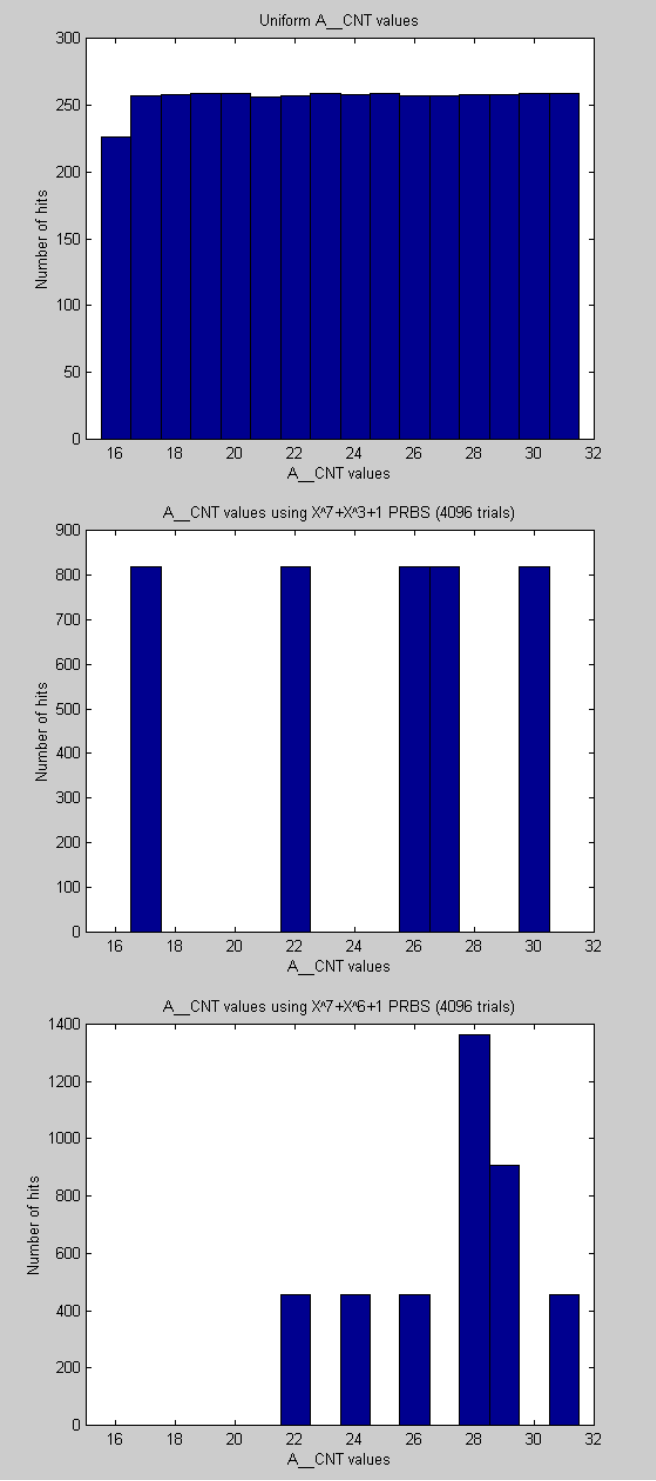
If there is a single pseudo-random generator, the A_CNT will only get initialized after the previous A_CNT has expired. Since the pseudo-random generator is continuously running to pick between ||R|| and ||K||, the A_CNT will not be initialized with state W_{X+1} if the current state is W_X, as was shown in the previous paragraph. If A_CNT is originally initialized with W_X, then the next state would be W_{WX+16}. By not sampling the pseudo-random generator after every clock cycle, not every possible value output by the pseudo-random generator will be observed. The plots on the next page show simulations of what the ||A|| spacing would actually look like when the pseudo-random generator is sampled in this manner. It should be noted that if different bits, or a different order of bits is taken for these four bits, then the histograms may look differently.

It should be noted that the authors of this test suite, to change the text of 48.2.4.2 so that any of these three histograms may become acceptable, have generated and submitted a maintenance request to IEEE 802.3. The complete text of this request can be found here:
http://www.ieee802.org/3/maint/requests/maint_1118.pdf

Here, we can see what is quite clearly a uniformly distributed random number from 16-31, inclusive. If 4096 trials were held, and the A_CNT value was uniformly distributed (with the generating polynomial only incrementing when A_CNT reaches zero), then each bin would contain approximately 256 hits. It is interesting to note that an A_CNT value of 16 will always be slightly less than the other values.

When the X^7+X^3+1 polynomial is used, the distribution of the A_CNT values appears to be uniform, but only among 5 bins. Nulls exist in all of the other bins after 4096 trials. The bins that do have hits appear to have approximately 820 hits in each bin.

When the X^7+X^6+1 polynomial is used, the distribution of the A_CNT values is also not uniform. Values appear in only 6 bins, and nulls are in the others after 4096 trials.



Appendix B: check_end interpretation

Through discussions with several interested parties, it has become apparent that there are multiple interpretations of how this function should be implemented. It is generally agreed upon that the original intent of the function is not clearly captured within the text of the Standard. In November 2003, interpretation request 5-11/03 was submitted, by the authors of this test suite, and discussed during the IEEE 802.3 plenary meeting. Complete documentation of the request and response may be found here: http://www.ieee802.org/3/interp/interp-5-1103.pdf. For the sake of convenience, parts of that document will be repeated in this annex.

The 10 Gigabit Consortium of the University of New Hampshire's InterOperability Laboratory, with the cooperation of members of the 10GEC, has settled on multiple interpretations of different aspects of the check_end function. When devices are tested using the 10GEC Clause 48 PCS Test Suite, a device that implements any of these interpretations will receive a passing or pass with comments result. A device that does not implement one of these interpretations will receive a failing result. It is also recognized that in the instances that have multiple interpretations, it is not possible to implement more than one of these interpretations.

Clause 48.2.6.1.4 of IEEE Std. 802.3ae-2002 defines the check_end function. The definition of this function is given here:

> Prescient Terminate function used by the PCS Receive process to set the RXD<31:0> and RXC<3:0> signals to indicate Error if a running disparity error was propagated to any Idle code-groups in ||T||, or to the column following ||T||. The XGMII Error control character is returned in all lanes less than n in ||T||, where n identifies the specific Terminate ordered-set ||T n ||, for which a running disparity error or any code-groups other than /A/or /K/are recognized in the column following ||T||. The XGMII Error control character is also returned in all lanes greater than n in the column prior to ||T||, where n identifies the specific Terminate ordered-set ||T n ||, for which a running disparity error or any code group other than /K/is recognized in the corresponding lane of ||T||. For all other lanes the value set previously is retained.

If we break the complicated definition of check_end into smaller parts it's a little easier to understand.

"The XGMII Error control character is returned in all lanes less than n in ||T||, where n identifies the specific Terminate ordered-set ||Tn||, for which a running disparity error or any code-groups other than /A/ or /K/ are recognized in the column following ||T||."

Looking at this sentence, we know that we have four cases for ||Tn||:

||T0|| - terminate in lane 0
||T1|| - terminate in lane 1
||T2|| - terminate in lane 2
||T3|| - terminate in lane 3

For each given ||Tn|| you then can see what "all lanes less than n in ||T||" means:

||T0|| - there are no lanes less than n
||T1|| - lane 0
||T2|| - lane 0, lane 1
||T3|| - lane 0, lane 1, lane 2

Note that it never says that the error control character is returned in lane n, but it is only returned in lanes less than n when an RD error occurs in the column following ||T||. Let's look at the four cases individually for each lane:

||T0||

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
T	K	K	K	T	K	K	K	T	K	K	K	T	K	K	K
*	K	K	K	K	*	K	K	K	K	*	K	K	K	K	*

In this example, the terminate character is in lane 0, and the * represents a running disparity error (or code-group other than /A/ or /K/) in the column of idle following the ||T0||. Now, since there are no columns less than 0, the error control character will not be pushed back into the frame, and the frame should be accepted. Part of the reason is that the /T/ and /K/ will catch running disparity errors, so anything happening after a valid /T/ or /K/ means that the error occurred outside the frame and therefore the frame should not be touched.

Now let's examine the other possibilities, when the /T/ occurs in a lane other than 0:

||T1||

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
D	T	K	K	D	T	K	K	D	T	K	K	D	T	K	K
*	K	K	K	K	*	K	K	K	K	*	K	K	K	K	*

In this example, the terminate character is in lane 1. Since we have ||T1|| we can return an Error code in lane 0. Now here is where things get a little vague. There are multiple interpretations of what should happen. For each of the 4 examples above with ||T1|| this is what you would receive after going through the PCS:

Option 1

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
E	T	K	K	E	T	K	K	E	T	K	K	E	T	K	K
E	K	K	K	K	E	K	K	K	K	E	K	K	K	K	E

(E is error code)

Option 2

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E T K K   D T K K   D T K K   D T K K
E K K K   K E K K   K K E K   K K K E
```

In option 1, the E code is returned in lane 0 if a running disparity error is detected in any of the 4 lanes. In option 2, the E code is returned in lane 0 only when the running disparity error is detected in lane 0.

Similar options apply when the example is extended to the other ||Tn||, as shown here:

||T2||

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
D D T K   D D T K   D D T K   D D T K
* K K K   K * K K   K K * K   K K K *
```

Option 1

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E E T K   E E T K   E E T K   E E T K
E K K K   K E K K   K K E K   K K K E
```

Option 2

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E E T K   E E T K   D D T K   D D T K
E K K K   K E K K   K K E K   K K K E
```

Option 3

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E D T K   D E T K   D D T K   D D T K
E K K K   K E K K   K K E K   K K K E
```

Option 3 is similar to Option 2, but the E is only returned in the lane corresponding to the lane with the error, and not all lanes. Option 2 has the E returned in all lanes less than n. It should be noted that the result of both of these options are identical with respect to which frames are accepted and which frames are discarded.

||T3||

```
-----------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
D D D T   D D D T   D D D T   D D D T
* K K K   K * K K   K K * K   K K K *
```

Option 1

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E E E T   E E E T   E E E T   E E E T
E K K K   K E K K   K K E K   K K K E
```

Option 2

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E E E T   E E E T   E E E T   D D D T
E K K K   K E K K   K K E K   K K K E
```

Option 3

```
-----------------------------------------------------------------
0 1 2 3   0 1 2 3   0 1 2 3   0 1 2 3
D D D D   D D D D   D D D D   D D D D
E D D T   D E D T   D D E T   D D D T
E K K K   K E K K   K K E K   K K K E
```

* * * * *

The second half of the check_end definition is somewhat easier to understand. "The XGMII Error control character is also returned in all lanes greater than n in the column prior to ||T||, for which a running disparity error or any code group other than /K/ is recognized in the corresponding lane of ||T||."

Looking at this sentence, we know that we have four cases for ||Tn||:

||T0|| - terminate in lane 0
||T1|| - terminate in lane 1
||T2|| - terminate in lane 2
||T3|| - terminate in lane 3

For each given ||Tn|| "all lanes greater than n in the column prior to ||T||" means:

||T0|| - lane 1, lane 2, lane 3
||T1|| - lane 2, lane 3
||T2|| - lane 3
||T3|| - no lanes

Now let's look at similar examples as above:

||T0||

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3    0 1 2 3
D D D D    D D D D    D D D D
T * K K    T K * K    T K K *
K K K K    K K K K    K K K K
```

Option 1

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3    0 1 2 3
D E E E    D E E E    D E E E
T E K K    T K E K    T K K E
K K K K    K K K K    K K K K
```

Option 2

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3    0 1 2 3
D E D D    D D E D    D D D E
T E K K    T K E K    T K K E
K K K K    K K K K    K K K K
```

||T1||

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3
D D D D    D D D D
D T * K    D T K *
K K K K    K K K K
```

Option 1

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3
D D E E    D D E E
D T E K    D T K E
K K K K    K K K K
```

Option 2

```
------------------------------------------------------------------
0 1 2 3    0 1 2 3
D D E D    D D D E
D T E K    D T K E
K K K K    K K K K
```

||T2||

```
0 1 2 3
D D D D
D D T *
K K K K
```

Option 1

```
0 1 2 3
D D D E
D D T E
K K K K
```

* * * * *

For the first half of the check_end function, there are two interpretations over which frames get discarded, and two interpretations on which code-groups are changed to /E/. For the second half of the function, there are two interpretations on which code-groups are changed to /E/.

The original intent of the check_end function was that potentially valid frames not be invalidated by the PCS. When an error occurs in a column directly following a valid /T/, /K/, or /A/ code-group, then the error could not have propagated through data code-groups in the frame, as the /T/, /K/, and /A/ code-groups will effectively prevent errors such as running disparity errors from going through them. Since the error could not possibly have occurred within the data portion of the frame, there should be no need for the PCS to invalidate the frame. One possible interpretation of the current text is that such action should be taken by the PCS.

It is not clear what the original intent of the check_end function was with respect to replacing data code-groups with error code-groups. The replacement of a single data code-group with an error code-group is sufficient to force the frame to be discarded. Also, since the lanes are independent of each other, it is not possible for an error to propagate from one lane to another. Although the insertion of multiple error code-groups will have the same result as the insertion of a single error code-group, it is possible that certain error counters may be caused to increment needlessly.

According to the previously mentioned interpretation request, the unambiguous response is that the Standard clearly specifies that option 3 should be used for implementation of the first part of check_end. Since the observable behavior of a device is identical for option 2 and option 3, the 10GEC will offer a result of PASS for any device that implements option 3, and PASS With Comments, for any implementation of option 2. A device that implements option 1, or any other option will receive a result of FAIL. For the second part of check_end, option 2 will receive a result of PASS, and option 1 will receive a result of PASS with Comments (or PASS if there is no other option). Any other implementations will receive a result of FAIL.