10GEC THE 10 GIGABIT ETHERNET CONSORTIUM

PCS Test Suite V1.1

Technical Document

Last Updated: August 13, 2003 11:01am

10 Gigabit Ethernet Consortium
InterOperability Laboratory
Research Computing Center
University of New Hampshire

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

MODIFICATION RECORD

- 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 Lynskey	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 conforms to the IEEE 802.3 standard, nor are they purely interoperability tests. 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

Scope: The following tests cover PCS operations specific to 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 802.3.

Test 48.1.1 – 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 36.2.4.9, 48.2.4.2.1, 48.2.5, 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 or 53.

Last Modification: June 30, 2003

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

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. Instruct the testing station to send a minimum IFG followed by a valid frame to the DUT.

7. Repeat steps 1 through 6, increasing the number of ||K|| in step 3 until the DUT receives the frame sent in step 5. Set this number to *m*.

Part B

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 *m* obtained in Part A, instruct the testing station to send ||K||, ||R||, and then *n* columns of ||K|| to the DUT, where *n* is (*m* – 2). For example, if 4 columns were needed in Part A, then initially set *n* 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 a valid frame to the DUT.
6. Instruct the testing station to send a minimum IFG followed by a valid frame to the DUT.
7. Repeat steps 1 through 6, increasing the value of *n* in step 3 until the DUT receives the frame sent in step 5.
8. Repeat steps 1 through 7, 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 C

1. Repeat *Part B* substituting the following patterns for step 3. Using the number of *m* obtained in *Part A*, instruct the testing station to send ||K||, ||Invalid||, *n* columns of /K/ to the DUT, setting the initial value of *n* to (*m* – 1). For example, if 4 columns were needed in *Part A* then initially set *n* to 3.

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 D

1. Repeat *Part B* substituting the following patterns for step 3. Using the value of *m* obtained in *Part A*, instruct the testing station to send ||K||, ||K||, ||R||, *n* columns of /K/ to the DUT, setting the initial value of *n* to (*m* – 3). For example, if 4 columns were needed in *Part A* then initially set *n* to 1.

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 E

1. Repeat *Part B* substituting the following patterns for step 3. Using the value of *m* obtained in Part A, instruct the testing station to send ||K||, ||K||, ||Invalid||, *k* columns of /K/ to the DUT, setting the initial value of *n* to (*m* – 3). For example, if 4 columns were needed in *Part A* then initially set *n* to 1.

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 F

1. Repeat *Part B* substituting the following patterns for step 3. Using the value of *m* obtained in Part A, instruct the testing station to send ||K||, ||K||, ||K||, ||R||, *n* columns of /K/ to the DUT, setting the initial value of *n* to (*m* – 4). For example, if 4 columns were needed in *Part A* then initially set *n* to 0.

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 G

1. Repeat *Part B* substituting the following patterns for step 3. Using the value of *m* obtained in Part A, instruct the testing station to send ||K||, ||K||, ||K||, ||Invalid||, *k* columns of /K/ to the DUT, setting the initial value of *n* to (*m* – 4). For example, if 4 columns were needed in Part A, then initially set *k* to 0.

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 the frame after receiving 4 ||K||.
b. The DUT should receive the frame after receiving a total of *n*+2 ||K|| on each lane.
c. The DUT should receive the frame only after each lane receives *n*+1 consecutive /K/ code-groups without any intervening invalid code-groups.
d. The DUT should receive the frame after receiving a total of *n*+3 ||K|| on each lane.
e. The DUT should receive the frame only after each lane receives *n*+3 consecutive /K/ code-groups without any intervening invalid code-groups.
f. The DUT should receive the frame after receiving a total of *n*+4 ||K|| on each lane.
g. In Part G, the DUT should receive the frame after each lane receives *n*+4 consecutive /K/ code-groups without any intervening invalid code-groups.

Possible Problems: None

Test 48.1.2 – 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: IEEE Std. 802.3ae-2003 – 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 or 53.

Last Modification: June 30, 2003

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 a valid frame to the DUT.
3. Instruct the testing station to transmit 4 ||K|| followed by 1 ||Invalid|| 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. Instruct the testing station to send a minimum IFG followed by a valid frame to the DUT.
7. Repeat steps 1 through 5, increasing the number of ||Invalid|| in step 2 until the DUT no longer receives the frame.

Part B

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.

3. Instruct the testing station to transmit 4 columns of ||K|| followed by this test pattern to the DUT:

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/

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. Instruct the testing station to send a minimum IFG followed by a valid frame to the DUT.
7. Repeat steps 1 through 6 substituting the following patterns for step 3:

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/

Part C

1. Repeat *Part B* substituting the following patterns for step 3.

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

Part D

1. Repeat *Part B* substituting the following patterns for step 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/
/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/
/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/	/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/	/Invalid/	/R/
/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/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/R/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/
/R/	/R/	/R/	/Invalid/

Part E

1. Repeat *Part B* substituting the following patterns for step 3.

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/

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 F

1. Repeat *Part B* substituting the following patterns for step 3.

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/
/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/
/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/
/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/
/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/
/R/	/R/	/R/	/Invalid/

Part G

1. Repeat *Part B* substituting the following patterns for step 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/
/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/
/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/
/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/	/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/	/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/	/Invalid/

Observable results:

a. The DUT should receive the middle frame in the sequence when 3 or less invalid code-group columns are received.
b. The DUT should never receive the middle frame in the sequence.
c. The DUT should never receive the middle frame in the sequence.
d. The DUT should never receive the middle frame in the sequence.
e. The DUT should never receive the middle frame in the sequence.
f. The DUT should never receive the middle frame in the sequence.
g. The DUT should never receive the middle frame in the sequence.

Possible Problems: None

Test 48.1.3 – Maintain Synchronization

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

References: IEEE Std. 802.3ae-2003 – 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 or 53.

Last Modification: June 30, 2003

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 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.
3. Instruct the testing station to transmit the following test pattern to the DUT:

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

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. Instruct the testing station to send a minimum IFG followed by a valid frame to the DUT.
7. Repeat steps 1 through 6 with the following patterns in step 3:

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

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

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

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

Part B

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

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

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

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

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

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

Part C

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

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

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

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

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

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

Part D

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

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

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

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

Part E

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

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

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

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

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

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

Part F

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

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

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

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

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

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

Part G

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

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

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

Part H

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

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

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

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

Part I

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

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

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

Part J

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

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

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

Observable results:

- For all test cases a-j, the DUT should receive all frames.

Possible Problems: None

GROUP 2: Alignment

Scope: The following tests cover PCS operations specific to 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 802.3.

Test 48.2.1 – Acquire alignment

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

References: 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 or 53.

Last Modification: June 30, 2003

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

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 16-32 columns of ||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 part 2 until the DUT receives the frame. Set this number to *m*.

Part B

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 *m* obtained in Part A, instruct the testing station to send ||A||, ||R||, *n* columns of ||A|| to the DUT, where *n* is (*m* – 2). For example, if 4 columns were needed in Part A, then initially set *n* to 2, as shown in the following table. Following every column of ||A||, the testing station will transmit an additional 16-32 columns of ||R||.

Lane 0	Lane 1	Lane 2	Lane 3
/A/	/A/	/A/	/A/
/R/	/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 until the DUT receives the frame.
5. Repeat steps 1 through 4, substituting the following for the pattern in step 3:

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

Part C

1. Repeat *Part B* substituting the following patterns for step 3. Instruct the testing station to send ||A||, ||A||, ||R||, *n* columns of /A/ to the DUT, where *n* is (*m* – 3). For example, if 4 columns were needed in Part A, then initially set *n* to 1. Following every column of ||A||, the testing station will transmit an additional 16-32 columns of ||R||.

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

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

Part D

1. Repeat *Part B* substituting the following patterns for step 3. Instruct the testing station to send ||K||, ||K||, ||K||, ||R||, *n* columns of /A/ to the DUT, where *n* is (*m* – 4). For example, if 4 columns were needed in Part A, then initially set *n* to 0. Following every column of ||A||, the testing station will transmit an additional 16-32 columns of ||R||.

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

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

Part E

1. Repeat *Parts A – D* causing the DUT to enter the SYNC_ACQUIRED_1 state and the LOSS_OF_ALIGNMENT state by sending continuous misaligned ||A|| along with ||K|| and ||R||.

Observable results:

a. The DUT should receive the frames after receiving 4 consecutive aligned ||A||.
b. The DUT should receive the frames after receiving 4 consecutive aligned ||A||.
c. The DUT should receive the frames after receiving 4 consecutive aligned ||A||.
d. The DUT should receive the frames 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 48.2.2 – 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: 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 or 53.

Last Modification: August 7, 2003

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 16-32 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

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:

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

4. Instruct the testing station to send a valid frame to the DUT.
5. Instruct the testing station to transmit a combination of ||A||, ||K|| 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|| and ||K|| in the gaps between ||A||.

6. Instruct the testing station to send a valid frame to the DUT.
7. If the DUT does not lose alignment, then repeat steps 1 through 6, increasing the number of misaligned ||A|| columns in step 3.
8. Repeat steps 1 through 7 with the following patterns in step 3:

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

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

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

Part B

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

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

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

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

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

Part C

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

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

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

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

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

Part D

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

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

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

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

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

Part E

1. Repeat *Parts A – D* inserting 16 – 31 columns of non /A/ codes between each of the /A/ codes in each test pattern.

Observable results:

a. The DUT should always receive the frames in steps 2 and 6, but not in step 4.
b. The DUT should always receive the frames in steps 2 and 6, but not in step 4.
c. The DUT should always receive the frames in steps 2 and 6, but not in step 4.
d. The DUT should always receive the frames in steps 2 and 6, but not in step 4.

Possible Problems: None

Test 48.2.3 – Maintain alignment

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

References: 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 or 53.

Last Modification: August 7, 2003

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 16-32 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:

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:

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

4. Instruct the testing station to transmit a valid frame to the DUT.
5. Instruct the testing station to transmit valid idle followed by a valid frame to the DUT.
6. Repeat steps 1 through 4 with the following patterns in step 3:

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

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

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

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

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

7. Repeat steps 1-6, shifting the test pattern by one column to the left each time, testing columns 2, 1, and 0, respectively.

Observable results:

- In all cases, the DUT should receive all frames.

Possible Problems: None

Test 48.2.4 – 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: 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 or 53.

Last Modification: February 28, 2003

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. 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. 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:

- The DUT should be tolerant of at least 41 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

Scope: The following tests cover PCS operations specific to transmission.

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 802.3.

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: IEEE Std. 802.3ae-2002 – subclauses Subclauses 36.2.4.4: Running disparity rules, 36.2.4.5: Generating code-groups, Table 36-1: Valid data code-groups, and Table 36-2:Valid special code-groups, 48.2.3, Figure 48-3, 48.2.4.

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 53, or a testing station capable of encoding (decoding) 64-bit words to (from) 66-bit words as specified in clause 49 and optionally clause 50, and sending (receiving) these code groups using the signaling method described in clause 52.

Last Modification: February 28, 2003

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, /K/, /R/, /A/, /S/, and /T/ 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:

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 a packet containing both forms of every valid data code-group listed in Table 36-1.
3. Observe all transmissions from the DUT.

Observable Results:

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

Possible Problems: None

Test 48.3.2 – Idle Sequencing

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

References: 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 or 53.

Last Modification: February 28, 2003

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.

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:

a. 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.
b. 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.
c. In Part A, the second column of idle after the ||T|| should always be ||R||.
d. 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).
e. In Part B, the distribution of ||K|| and ||R|| should follow one of the two defined polynomials.

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: 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 or 53.

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:

- 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: 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 or 53.

Last Modification: February 28, 2003

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 RS 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

Scope: The following tests cover PCS operations specific to reception.

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 802.3.

Test 48.4.1 – Fault Reception

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

References: 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 or 53.

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: IEEE Std. 802.3ae-2002 – subclauses 48.2.4.3.2, 48.2.6.1.4, 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 or 53.

Last Modification: August 7, 2003

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 may be discarded.
b. In Part B, the DUT should always receive the first and third frames but never receive the second frame.

Possible Problems: None

GROUP 5: Management

Scope: The following tests cover PCS operations specific to management.

Overview: These tests are designed to verify that the device under test provides access to the appropriate status and configuration registers defined in Clause 48 of IEEE 802.3.

Note: As of August 2003, these tests are still under development.

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.

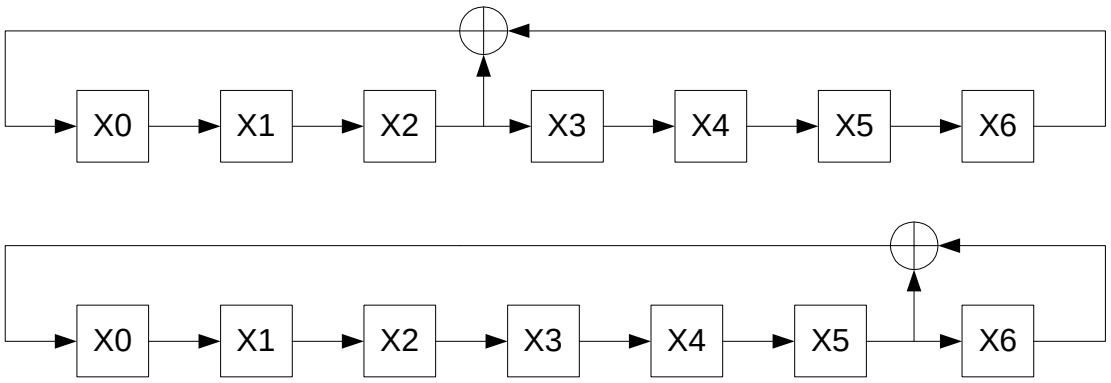


Figure 1. Idle Scrambler Implementations

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

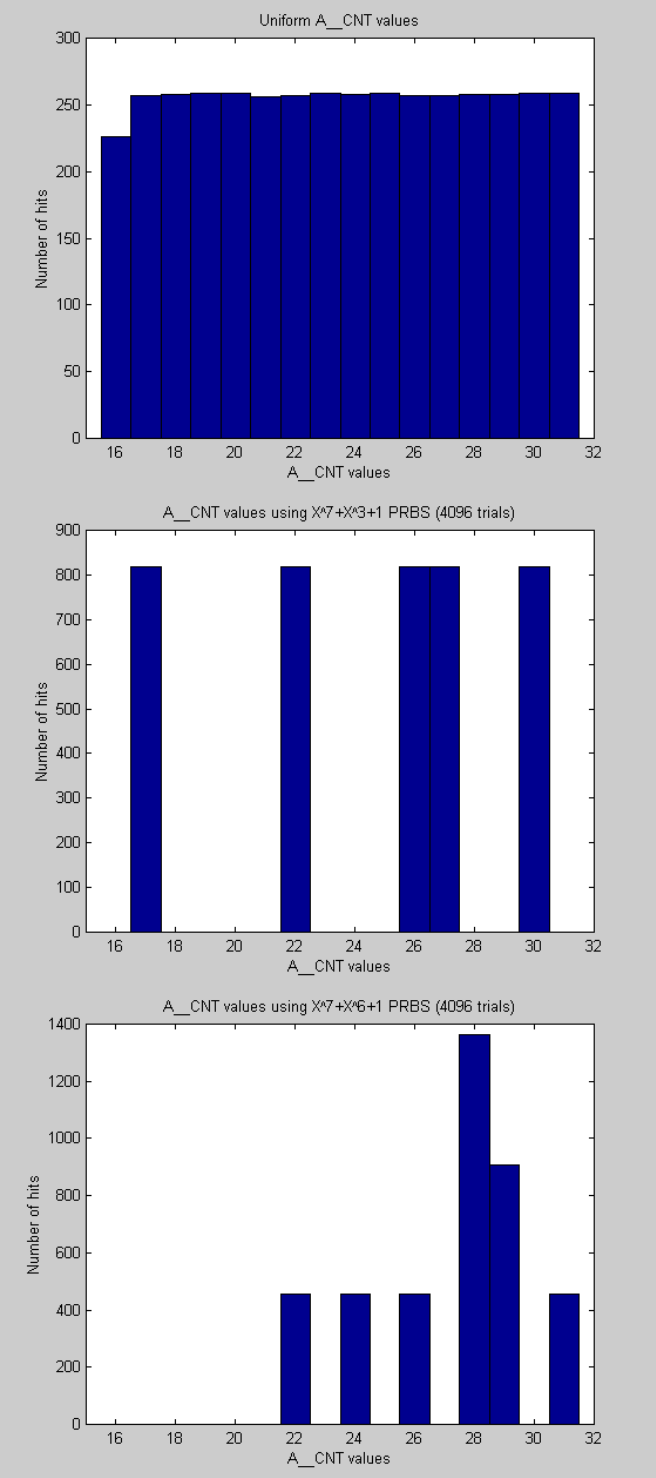
If there is a single pseudo-random generator, the A_CNT will only get initialized after the previous A_CNT has expired. Since the pseudo-random generator is continuously running to pick between ||R|| and ||K||, the A_CNT will not be initialized with state W_{X+1} if the current state is W_X, as was shown 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.

If the ||A|| spacing of the DUT matches any of these three histograms, then observable result *d* of test 48.3.2 will be listed as a PASS. Should the ||A|| spacing not conform to one of these histograms, then the result will be listed as FAIL.

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. It is also generally agreed upon that some form of interpretation or maintenance request needs to be filed against the current text so that a future version of the Standard will contain, in clear and easy to read text, one and only one way that the check_end function can be interpreted and that such revision to the text will convey the original intent of the function.

Until such a revision of the Standard is issued, 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 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.

Until the text defining the check_end function is improved, the 10GEC will interpret the results of the check_end PCS test based on the multiple interpretations set forth in this document. Implementations that follow other interpretations will be treated as failing cases.