IEEE STANDARDS ASSOCIATION

IEEE

IEEE Standard for High-Speed Test Access Port and On-Chip Distribution Architecture

IEEE Computer Society

Sponsored by the
Test Technology Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1149.10™-2017

IEEE Standard for High-Speed Test Access Port and On-Chip Distribution Architecture

Sponsor

Test Technology Standards Committee
of the
IEEE Computer Society

Approved 18 May 2017

IEEE-SA Standards Board

Abstract: Circuitry that may be built into an integrated circuit to assist in the test, maintenance, and support of assembled printed circuit boards, assembled multi-die packages, and the test of die internal circuits is defined in this standard. The circuitry includes a high-speed TAP (HSTAP) with a packet encoder/decoder and distribution architecture through which instructions and test data are communicated. The standard leverages the languages of IEEE Std 1149.1™ to describe and operate the on-chip circuits.

Keywords: 3D-IC, Boundary-Scan Description Language, BSDL, debug, High Speed JTAG, I2C, IEEE 1149.1™, PDL, IEEE 1149.10™, integrated circuit, JTAG, wafer, Procedural Description Language, SERDES, SPI, system level test

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2017 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 28 July 2017. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF:	ISBN 978-1-5044-3995-4	STD22564
Print:	ISBN 978-1-5044-3996-1	STDPD22564

IEEE prohibits discrimination, harassment, and bullying.
For more information, visit http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html.
No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimers.html.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Xplore at http://ieeexplore.ieee.org/ or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at http://standards.ieee.org.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: http://standards.ieee.org/findstds/errata/index.html. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this IEEE standard was completed, the P1149.10 Working Group had the following membership:

C. J. Clark, *Chair and Editor*
Bill Tuthill, *Secretary*

Gobinathan Athimolom
Tapan Chakraborty
Jonathon E. Colburn
Heiko Ehrenberg
Josh Ferry
Bob Gottlieb
Gurgen Harutyunyan
Marc Hutner
Dharma Konda
Adam W. Ley
Mike Ricchetti
Craig Stephan
Stephen Sunter
Brian Turmelle

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Saman Adham
Gobinathan Athimolom
Bill Brown
William Bush
Juan Carreon
Tapan Chakraborty
Keith Chow
C. J. Clark
Jonathon E. Colburn
Glenn Colon-Bonet
Adam Cron
Jason Doege
Heiko Ehrenberg
William Eklow
Josh Ferry
Joel Goergen
Bob Gottlieb
Randall Groves
Peter Harrod
Gurgen Harutyunyan
Werner Hoelzl
Marc Hutner
Piotr Karocki
Dharma Konda
Philippe Lebourg
Adam W. Ley
Jeffrey Moore
Charles Ngethe
Jim O'Reilly
Mike Ricchetti
Craig Stephan
Walter Struppler
Stephen Sunter
David Thompson
Bill Tuthill
Louis Ungar
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 18 May 2017, it had the following membership:

Jean-Philippe Faure, *Chair*
Gary Hoffman, *Vice Chair*
John D. Kulick, *Past Chair*
Konstantinos Karachalios, *Secretary*

Chuck Adams
Masayuki Ariyoshi
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Michael Janezic
Thomas Koshy
Joseph L. Koepfinger*
Kevin Lu
Daleep Mohla
Damir Novosel
Ronald C. Petersen
Annette D. Reilly
Robby Robson
Dorothy Stanley
Adrian Stephens
Mehmet Ulema
Phil Wennblom
Howard Wolfman
Yu Yuan

*Member Emeritus

Copyright © 2017 IEEE. All rights reserved.

Introduction

This introduction is not part of IEEE Std 1149.10™-2017, IEEE Standard for High-Speed Test Access Port and On-Chip Distribution Architecture.

With the approval of IEEE Std 1149.1™-2013, the industry now has a standardized approach to hierarchical design-for-test. IEEE Std 1149.1-2013 provides access to IP blocks via IEEE 1500 wrapper serial ports. IEEE 1149.1 package files and PDL standardized how to describe IP block operation destined for SoC integration. Some Working Group members started to discuss the bandwidth limitations of the IEEE 1149.1 TAP, the limitations of single scan-in/scan-out for test time, the diminishing single-ended I/O count for parallel scan due to die-stacking, and the desire for leveraging PDL to be re-used beyond the TAP. A group of interested parties was formed in August of 2013. C. J. Clark presented the fundamentals on the HSTAP, PEDDA, and Packet format at the first meeting. Industry-based SERDES packets were discarded because they required storing packet information in memory and generally required more on-chip resources. The IEEE 1149.10 architecture needed to be simple and easy to add on to the mission mode design. A PAR was approved in October 2013. A publicly available history of the Working Group's attendance, motions, minutes, and presentations can be found at http://grouper.ieee.org/groups/1149/10/.

Copyright © 2017 IEEE. All rights reserved.

Contents

1. Overview 11
1.1 Scope 11
1.2 Need 11
1.3 Document outline 12
1.4 Specifications 12
1.5 Descriptions 12
1.6 Text conventions 13
1.7 Logic diagram conventions 13

2. Normative references 14

3. Definitions, abbreviations, acronyms, and special terms 14
3.1 Definitions 14
3.2 Abbreviations and acronyms 16
3.3 Numbers 18

4. High-speed test access port (HSTAP) 18
4.1 HSTAP 18

5. Packet encoder/decoder and distribution architecture 21
5.1 PEDDA 21

6. Packet definitions 27
6.1 Packet overview 27
6.2 The CONFIG packet 30
6.3 The TARGET packet 32
6.4 The RESET packet 33
6.5 The RAW packet 35
6.6 The CH-SELECT packet 36
6.7 The SCAN packet 38
6.8 The BOND packet 46
6.9 The CONFIGR packet 47
6.10 The TARGETR packet 48
6.11 The RESETR packet 49
6.12 The RAWR packet 50
6.13 The CH-SELECTR packet 51
6.14 The SCANR packet 52
6.15 The BONDR packet 56

7. BSDL definitions 57
7.1 BSDL overview 57
7.2 Conformance attribute 59
7.3 HSTAP attribute 59
7.4 Packet_Map attribute 67
7.5 Control_Chars attribute 71
7.6 Scan_Channel_Association attribute 73
7.7 BSDL package for high speed JTAG 76

8. Channel bonding 77
8.1 Optimizing bandwidth 77

Copyright © 2017 IEEE. All rights reserved.

9. PDL 82
9.1 PDL Overview 82
9.2 iConfig command 83
9.3 iTarget command 84
9.4 iReset10 command 85
9.5 iRaw command 86
9.6 iBond command 87
9.7 Standardized PDL procedures 88

10. Compliance verification 89
10.1 Overview 89

Annex A (informative) Bibliography 94

List of Figures

Figure 1—Logic symbology used in this standard 14
Figure 2—State diagram for IEEE 1149.10 entry 19
Figure 3—Example daisy chain of HSTAPs 20
Figure 4—System that benefits from the compliance-enable character 21
Figure 5—Example IEEE 1149.10 PEDDA block diagram 22
Figure 6—Example implementation of TRST10* and Reset10* 25
Figure 7—Example IEEE 1149.1 TDR for asserting Enable_1149_10 26
Figure 8—Example circuit for asserting Enable_1149_10 via a compliance-enable character........ 26
Figure 9—Example use of B-S and Init-Data Register 27
Figure 10—General form of a IEEE 1149.10 packet 29
Figure 11—Format of CONFIG packet........ 30
Figure 12—Transmit order for CONFIG packet using LSB_First based HSTAP 32
Figure 13—Format of TARGET packet........ 32
Figure 14—Format of a RESET packet........ 34
Figure 15—Bit transmit order for LSB_First based HSTAP........ 35
Figure 16—Format of a RAW packet 36
Figure 17—CH-SELECT packet format 37
Figure 18—Example scan-channel selection detail........ 38
Figure 19—Generic format of a SCAN packet 39
Figure 20—Data in scan-channel 3 through 0 after scan........ 40
Figure 21—Formatted payload of packet........ 40
Figure 22—Payload with 3 scan-channels 41
Figure 23—Four example scan-channels, with final data contents 42
Figure 24—CH-SELECT, Capture-DR, and Shift-DR portion of a two channel 32-bit scan 42
Figure 25—Example Packet_Map........ 43
Figure 26—Packets needed to shift 33rd bit of Channel 3 43
Figure 27—Final SCAN packet to assert Update-DR........ 44
Figure 28—Format of a BOND packet 47
Figure 29—Example BOND packet for four lanes 47
Figure 30—Format of the CONFIGR packet 48
Figure 31—Format of TARGETR packet........ 49
Figure 32—Format of the RESETR packet........ 50
Figure 33—Format of RAWR packet........ 51
Figure 34—Format of CH-SELECTR packet 52
Figure 35—CH-SELECTR Packet example 52
Figure 36—Generic SCANR packet format........ 53
Figure 37—Four example scan-channels with data captured 54

Copyright © 2017 IEEE. All rights reserved.

Figure 38—First SCAN packet sent 54
Figure 39—SCANR response packet example 55
Figure 40—Example SCAN packet for Channel 3 55
Figure 41—SCANR response to return last bit of channel 3 56
Figure 42—Format of BONDR packet 56
Figure 43—BONDR response packet 56
Figure 44—Example timing for LATENCY_1149_10 62
Figure 45—HSTAP SERDES requiring TX_CLOCK_1149_10 input 66
Figure 46—SPI Bus as HSTAP 67
Figure 47—Four-bit PISO/SIPO on 2 scan channels 70
Figure 48—Use of ERROR_CHAR 73
Figure 49—Four lane HSTAP 79
Figure 50—Striping of data alternative view 80

List of Tables

Table 1—Inbound packets 28
Table 2—Outbound packets 28
Table 3—CONFIG packet transmit order 31
Table 4—TARGET packet transmit order 33
Table 5—RESET packet data and transmit order 35
Table 6—The subset of 64B/66B blocks used when encoding IEEE 1149.10 packets 45
Table 7—Example 64B/66B encoding of IEEE 1149.10 packets 46
Table 8—BSDL syntax from IEEE Std 1149.1 57
Table 9—BSDL syntax for IEEE 1149.10 without IEEE Std 1149.1 compliance 58
Table 10—HSTAP BSDL parameters 60
Table 11—Packet_Map parameters 68
Table 12—Control_Chars definitions 71
Table 13—Scan_Channel_Association description 74

Copyright © 2017 IEEE. All rights reserved.

IEEE Standard for High-Speed Test Access Port and On-Chip Distribution Architecture

1. Overview

1.1 Scope

This standard defines a high speed test access port for delivery of test data, a packet format for describing the test payload, and a distribution architecture for converting the test data to/from on-chip test structures.

The standard re-uses existing high speed I/O (HSIO) known in the industry for the high speed test access port (HSTAP). The HSIO connects to an on-chip distribution architecture through a common interface. The scope includes the distribution architecture test logic and packet decoder logic. The objective of the distribution architecture and packet decoder is that it can be readily re-used with different integrated circuits (ICs) that host different HSIO technology, such that the standard addresses as large a part of the industry as possible.

The scope includes IEEE 1149.1 Boundary-Scan Description Language (BSDL) and Procedural Description Language (PDL) documentation, which can be used for configuring a mission mode HSIO to a test mode compatible with the HSTAP. The same BSDL and PDL can then be used to deliver high-speed data to the on-chip test structures.

1.2 Need

Test time has always been an important metric for system on a chip (SoC). The original IEEE 1149.1 test access port is fine for simple board interconnect tests, but as on-chip operations via the IEEE 1149.1 test access port (TAP) have increased, the use of the IEEE 1149.1 TAP becomes inefficient for board test and on-board field programmable gate array (FPGA) configuration. Large FPGAs take tens of minutes to configure through the IEEE 1149.1 TAP. The IEEE 1149.1 TAP has always been too slow for production SoC test. Wide test access mechanisms (TAMs) are used to increase test throughput during production IC test at the cost of requiring more tester resources. Wide TAMs are also not useful for test re-use at the board/system level because many of the I/O of the TAM are not accessible. Pin limitations also exist where the pins required for the IEEE 1149.1 TAP cannot be supported by a small package or die. A high-speed test access port and packet encoder/decoder and distribution architecture (PEDDA) is needed by the industry to standardize a faster test data delivery mechanism for IC automatic test equipment (ATE), but also be re-usable at board and system level test. Today, in 2017, to get 10 Gbit/s data transfer on a die

Copyright © 2017 IEEE. All rights reserved.

requires one hundred touch-downs for sending data in at 100 Mbit/s and one hundred touch-downs for receiving data at 100 Mbit/s with one hundred scan chains that meet the timing for a 100 MHz clock rate. IEEE Std 1149.10-2017™ offers an alternative to deliver the same test data bandwidth with just a differential receiver and transmitter: four pins, a system clock, and power. By making the scan-channels “virtual,” tradeoffs can be made during design for test regarding scan rates, number of concurrent active scan channels, and the amount of test bandwidth desired.

Mission mode pins exist for SERDES, serial parallel interface (SPI), I^2C (I2C), and double data rate (DDR), which can be re-allocated for test purposes saving on dedicated test pins needed in the SoC to support IEEE 1149.1. IEEE Std 1149.10-2017 introduces the re-use of mission mode pins to facilitate either high-bandwidth test or low resource based test via two new objects: the HSTAP and the PEDDA. The HSTAP can layer on top of the mission mode pins (e.g., re-use the pins of SERDES, SPI, I2C, etc.) and deliver data to the PEDDA, which can access on-chip scan channels (test data registers and wrapper serial ports) to communicate data for test, debug, or FPGA configuration.

1.3 Document outline

Circuit designs such as those defined by this standard are more easily understood if their specifications are accompanied by general descriptive material that places the details of the various parts of each design in perspective and provides examples of implementation. Clause 1, therefore, contains an overview of this standard to the testing of an electronic circuit, the need, and document conventions. Clause 2 provides a list of normative references and Clause 3 provides definitions, acronyms, and abbreviations. Subsequent clauses of this document contain the specifications for particular features of this standard.

1.4 Specifications

Material titled “Specifications” contain the rules, recommendations, and permissions that define this standard:

— Rules specify the mandatory aspects of this standard. Rules contain the word shall.

— Recommendations indicate preferred practice for designs that seek to conform to this standard. Recommendations contain the word should.

— Permissions show how optional features may be introduced into a design that seeks to conform to this standard. These features will extend the application of the test circuitry defined by this standard. Permissions contain the word may.

1.5 Descriptions

CAUTION

The descriptive material contained in this standard is for illustrative purposes only and does not define a preferred implementation. Examples are provided throughout this standard to illustrate possible circuit implementations. Where discrepancies between examples and specifications may occur, the specifications always take precedence. Readers should exercise caution when using these examples in their specific applications. In particular, it is emphasized that the examples are designed to communicate effectively the meaning of this standard. As such, they are logically correct; however, as always, a particular implementation may not operate properly with respect to timing and other parametric characteristics.

Copyright © 2017 IEEE. All rights reserved.

Material not contained in "Specifications" is descriptive material that illustrates the need for the features being specified or their application. This material includes schematics that illustrate a possible implementation of the specifications in this standard. The descriptive material also discusses design decisions made during the development of this standard.

1.6 Text conventions

The following conventions are used in this standard:

a) The rules, recommendations, and permissions in "Specifications" are contained in a single alphabetically indexed list. References to each rule, recommendation, or permission are shown in the form:

```
                  15.1.1 c) 2)
                     |   |  |
     Sub-clause number   |  |
                     Index  |
                       Option (if any)
```

b) Instruction and state names defined in this standard are shown in italic type in the text.

c) Names of states and signals that control the test data registers defined by this standard contain the characters DR, while those that control the instruction register contain the characters IR.

d) Names for signals that are active in their low state have an asterisk as the final character, e.g., TRST*.

e) A positive logic convention is used; i.e., a logic 1 signal is conveyed as the more positive of the two voltages used for logic signals.

1.7 Logic diagram conventions

During the different iterations of this standard, logic diagram figures have been added in various styles with differing logic symbols. Figure 1 shows the symbologies currently used for the common combinational logic elements. Symbols for storage elements (generally edge-sensitive flip-flops) are reasonably consistent, although pin names on the element may vary from figure to figure.

Copyright © 2017 IEEE. All rights reserved.

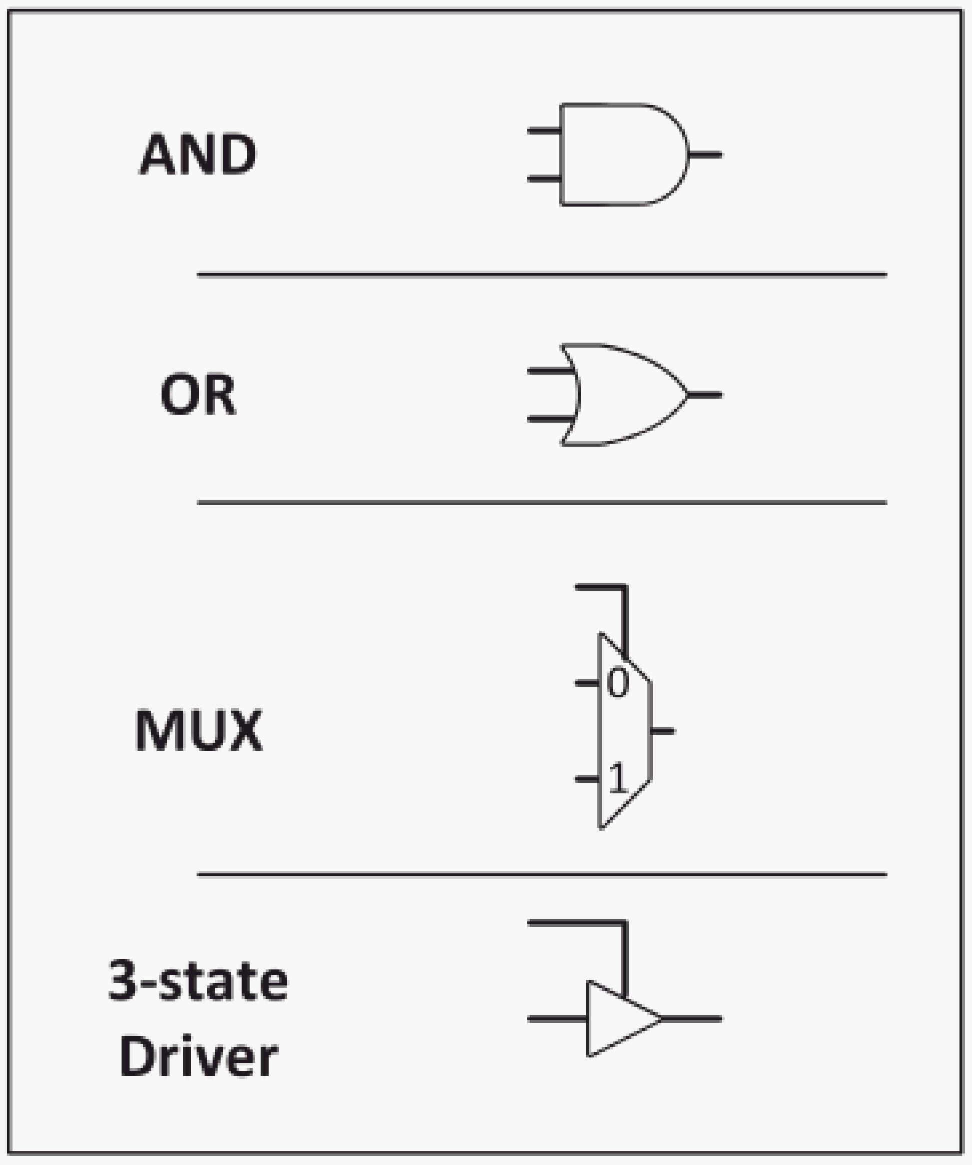

Figure 1—Logic symbology used in this standard

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 802.3™, IEEE Standard for Ethernet.[1, 2]

IEEE Std 1149.1™, IEEE Standard for Test Access Port and Boundary-Scan Architecture.

3. Definitions, abbreviations, acronyms, and special terms

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.[3]

channel: A scan-channel.

channel bonding: The use of multiple HSTAPs that work together in order to increase bandwidth.

[1] The IEEE standards or products referred to in Clause 2 are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.
[2] IEEE publications are available from the Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).
[3] *IEEE Standards Dictionary Online* is available at: http://dictionary.ieee.org

Copyright © 2017 IEEE. All rights reserved.

control character: Any of the special symbols used in serial communication encodings. 8B/10B, for instance, defines twelve control symbols. All of the control characters in this standard are shown and defined as unencoded hexadecimal values of eight bits in width.

CRC32: Cyclical Redundancy Check. All references to CRC32 in this document refer to the 32-bit computation as documented in IEEE Std 802.3.[4]

Enable_1149_10: A signal that is an input to the PEDDA when at least one scan-channel is shared with an IEEE 1149.1 TAP. When it is asserted, the PEDDA provides access to one or more scan-channels from a HSTAP. When it is de-asserted, the PEDDA provides access to one or more scan-channels from the IEEE 1149.1 TAP. An IEEE 1149.1 PDL iProc with the same name documents one method to assert the signal, other methods are left to the designer.

high-speed test access port (HSTAP): The HSTAP is the input/output pins and circuitry that deliver data to a PEDDA. It is the test access port of this standard and is not a high-speed version of the IEEE 1149.1 TAP. The HSTAP may include common serial encoding/decoding methods such as 8B/10B, 128B/130B, or others.

IEEE 1149.10 interface: A high-speed test access port (HSTAP) and packet encoder/decoder and distribution architecture (PEDDA).

instruction register: Refers to the IEEE 1149.1 instruction register. Bit "I" of the ICSU defined section in the scan packet indicates if there is an operation from a HSTAP and PEDDA to an IEEE 1149.1 instruction register. All other "instruction" registers, such as the IEEE 1500 wrapper instruction register, are treated as IEEE 1149.1 test data registers.

lane: A serial data transmission path with one transmit and receive pair. The terminology commonly is used with differential signaling, however, for this standard, a single-ended transmit and receive pair also make up a lane. A HSTAP may have one or more lanes.

multi-lane: A mode of operation where multiple lanes are used.

pack: A verb "to place" all the bits for enabled scan-channels contiguous with other enabled scan-channels.

packet frame: A four-byte portion of a packet that starts at any integer multiple of four bytes.

port: As used in the acronym, HSTAP, port refers to an input/output interface (e.g., Ethernet port). Port does not mean pin as it would in an IEEE 1149.1 context.

scan-channel: Any serially clocked input or output (or both) to or from the PEDDA. A scan-channel may be a scan chain or it may feed the input to a de-compressor, which then fans out to multiple scan chains. A scan-channel may be clocked by a scan clock or the IEEE 1149.1 TCK.

striping: The round-robin assignment of packet frames across a given set of HSTAPs during channel-bonding. One packet frame of a given packet is transmitted and received at each HSTAP sequentially.

TAP interface: The interface on the PEDDA that accepts the signals from an IEEE 1149.1 TAP. The TAP interface allows IEEE 1149.1 TDRs and IEEE 1500 wrapper serial ports to have shared access between the IEEE 1149.1 TAP and an IEEE 1149.10 HSTAP.

[4] Information on references can be found in Clause 2.

Copyright © 2017 IEEE. All rights reserved.

3.2 Abbreviations and acronyms

ASIC	application-specific integrated circuit
ATE	automatic test equipment
ATPG	automatic test-pattern generation
BIST	built-in self-test
BNF	Backus-Naur form
BSDL	Boundary-Scan Description Language
CMOS	complementary metal-oxide semiconductor
DC	direct current
DDR	double data rate
ECID	electronic chip identification
FIFO	first in, first out
HSTAP	high speed test access port
I2C	I^2C
IC	integrated circuit
ICSU	instruction, capture, shift and update
I/O	input or output
IP	intellectual property (commonly used to refer to a reusable design element)
LSB	least significant bit
LVDS	low-voltage differential signaling
MSB	most significant bit
PDL	Procedural Description Language or a file containing PDL statements
PEDDA	packet encoder/decoder and distribution architecture
PISO	parallel in, serial out
PLL	phase-locked loop
POR	power-on reset
PRBS	pseudo-random binary sequence
RAM	random access memory

Copyright © 2017 IEEE. All rights reserved.

SERDES	serializer/deserializer
SIPO	serial in, parallel out
SOC	system-on-a-chip
SPI	serial parallel interface
TAP	test access port
TCK	test clock input
Tcl	Tool Command Language or a file containing Tcl statements
TDI	test data input
TDO	test data output
TDR	test data register
TMP	test mode persistence
TMS	test mode select
TRST*	test reset
TTL	transistor-transistor logic
UUT	unit under test
VHSIC	very high-speed integrated circuit
WBR	wrapper boundary register
WBY	wrapper bypass
WDR	wrapper data register
WIR	wrapper instruction register
WRSTN	wrapper reset
WSC	wrapper serial control
WSI	wrapper scan in
WSO	wrapper scan out
WSP	wrapper serial port

Copyright © 2017 IEEE. All rights reserved.

3.3 Numbers

Values that are expressed in binary notation will be indicated by a contiguous string of characters starting with 0b or 0B followed by one of the set [01xX], followed by zero or more of the characters in the set [01xX_], and containing no spaces or format effectors.

Values that are expressed in hexadecimal notation will be indicated by a contiguous string of characters starting with 0x or 0X, followed by one of the set [0-9a-fA-FxX], followed by zero or more of the characters in the set [0-9a-fA-FxX_], and containing no spaces or format effectors.

Values expressed in decimal notation shall be an unsigned contiguous string of characters of the set [0-9]; multi-character values shall not start with 0, shall contain no spaces or format effectors, shall have a value less than 232 – 1, and shall always match any binary field large enough to hold the most significant 1 bit of the binary equivalent value of the decimal pattern.

4. High-speed test access port (HSTAP)

4.1 HSTAP

This subclause defines the rules for creating a HSTAP. The HSTAP interfaces between external ATE and the PEDDA.

4.1.1 Description

The HSTAP circuit is the primary interface of the PEDDA. The HSTAP terminology is used to differentiate the IEEE 1149.10 HSTAP from the IEEE 1149.1 TAP that has single-ended signals TCK, TMS, TDI, TDO, and TRST*. The designer is provided with flexibility in designing the HSTAP as long as the HSTAP can be documented with the HSTAP BSDL attribute and all rules in this clause are followed. A HSTAP must be ready to operate after power-on or after compliance is enabled. The HSTAP can take advantage of re-using mission mode interfaces (e.g., SERDES) only when the HSTAP is ready to respond to IEEE 1149.10 packets after compliance is enabled. There are two methods to enable compliance: use the IEEE 1149.10 compliance-enable character, or provide an IEEE 1149.1 TAP interface to configure mission mode signals into the HSTAP for IEEE 1149.10-based testing.

Copyright © 2017 IEEE. All rights reserved.

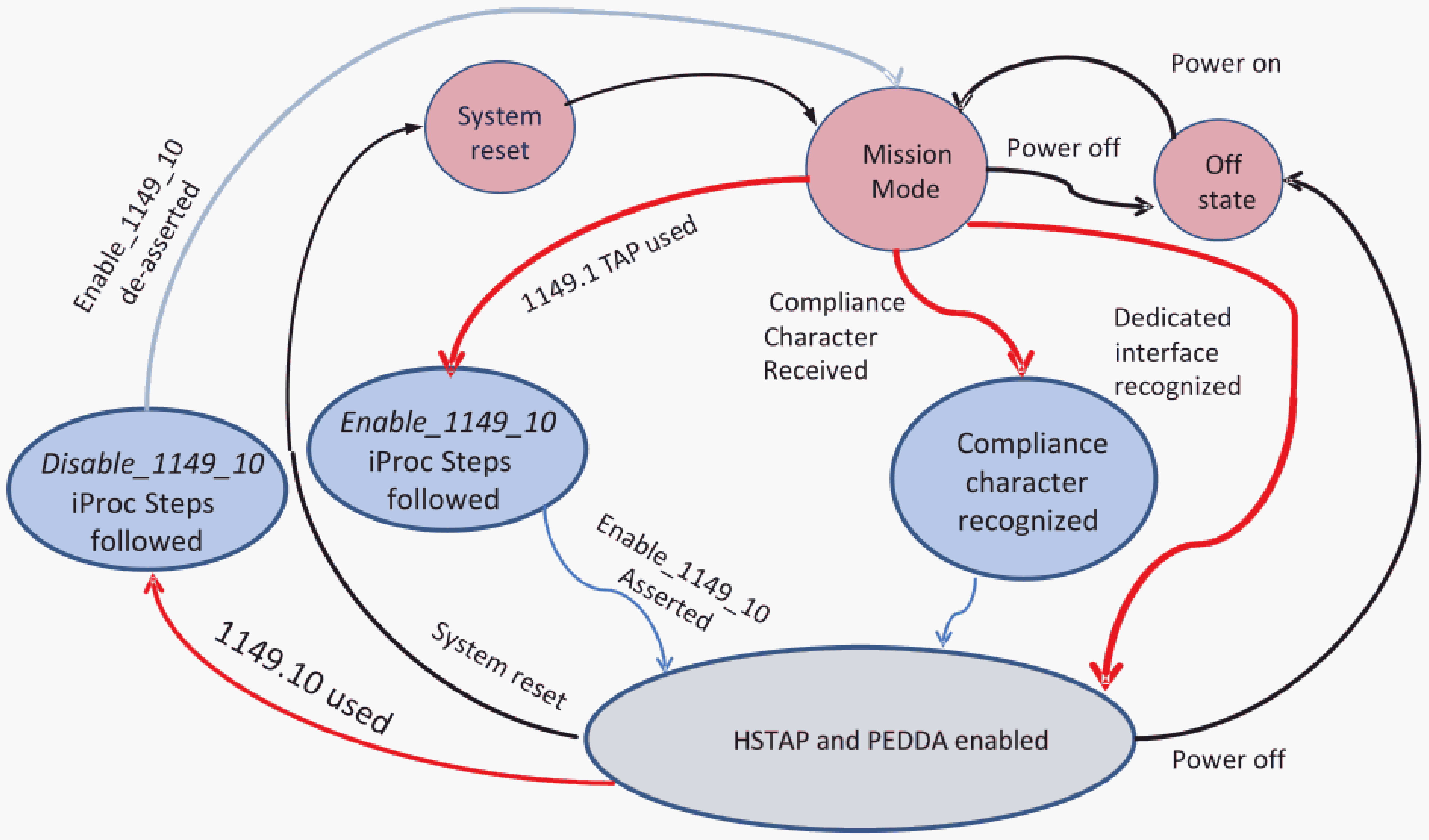
System reset
Mission Mode
Power on
Power off
Off state
Enable_1149_10 de-asserted
1149.1 TAP used
Compliance Character Received
Dedicated interface recognized
Disable_1149_10 iProc Steps followed
Enable_1149_10 iProc Steps followed
Compliance character recognized
Enable_1149_10 Asserted
System reset
1149.10 used
HSTAP and PEDDA enabled
Power off

Figure 2—State diagram for IEEE 1149.10 entry

Figure 2 shows the state diagram for enabling the HSTAP and PEDDA. This state diagram illustrates rule 4.1.2 c). After power-on, the device is in mission mode. A HSTAP can be dedicated in its function to support IEEE 1149.10 and not shared with a mission mode function. In that case, the dedicated HSTAP interface is enabled at power-on, as indicated by the path in Figure 2 from the mission mode state directly to the HSTAP and PEDDA enabled state. A dedicated HSTAP and PEDDA are enabled as part of the power-on sequence and do not need compliance enablement. If the HSTAP is shared with a mission mode I/O interface then one of the two compliance enablement methods are needed to enable the HSTAP and PEDDA. A system reset sets the state of a shared HSTAP back to mission mode. The system reset can be performed via system reset pins or through other means such as the IEEE 1149.1 IC_RESET instruction if it is provided. When an IEEE 1149.1 TAP is provided to enable the HSTAP and PEDDA, a PDL iProc *Enable_1149_10* must be provided to document the steps to enable compliance. An associated PDL iProc *Disable_1149_10* can be used by the IEEE 1149.10 interface to set test data register bits to re-enable IEEE 1149.1 TAP operation and mission mode.

4.1.2 Specifications

Rules

a) An IEEE Std 1149.10-2017 compliant design shall have one or more HSTAPs.
b) Each HSTAP shall be documented using the HSTAP BSDL attribute.
c) Each HSTAP shall power-up in the enabled state (i.e., ready to respond to a CONFIG packet and all subsequent IEEE 1149.10-defined packets) unless:
   i. An IEEE 1149.1 TAP interface is designed to enable the IEEE Std 1149.10-2017 compliant operation and the presence of the TAP is documented in BSDL

   and/or

   ii. A compliance-enable character received at the HSTAP interface enables IEEE Std 1149.10-2017 compliance and is documented in the CONTROL_CHARS BSDL attribute.

Copyright © 2017 IEEE. All rights reserved.

d) If the HSTAP does not power-up in the enabled state, and an IEEE 1149.1 TAP is provided to enable compliance with this standard, two PDL procedures, *Enable_1149_10* and *Disable_1149_10*, shall be documented.

e) When compliance is enabled as provided for in rule c) of 4.1.2, the HSTAP shall remain in compliance mode unless

   i. Power to the HSTAP is removed

   or

   ii. A system reset is performed

   or

   iii. The steps documented in the *Disable_1149_10* procedure are performed.

f) If required, the *Enable_1149_10* PDL procedure shall document the IEEE 1149.1 operations needed such that the HSTAP is ready to respond to a CONFIG packet and all subsequent IEEE 1149.10 defined packets.

g) A compliance-enable character received at an HSTAP interface shall enable IEEE Std 1149.10-2017 compliance and be forwarded to the HSTAP transmitter to enable connected HSTAPs to enter IEEE Std 1149.10-2017 compliance.

Recommendations

h) When an HSTAP is not compliant at power-up, it is recommended that the compliance-enable character be used to establish IEEE Std 1149.10-2017 compliance.

4.1.3 Examples

An HSTAP can be daisy-chained to another HSTAP (see Figure 3). The design of the PEDDA and packet format specification enables each HSTAP and PEDDA combination in a daisy-chain to receive data uniquely destined for that PEDDA. (See Clause 5 and Clause 6 for more information on the TARGET_ID and its use.)

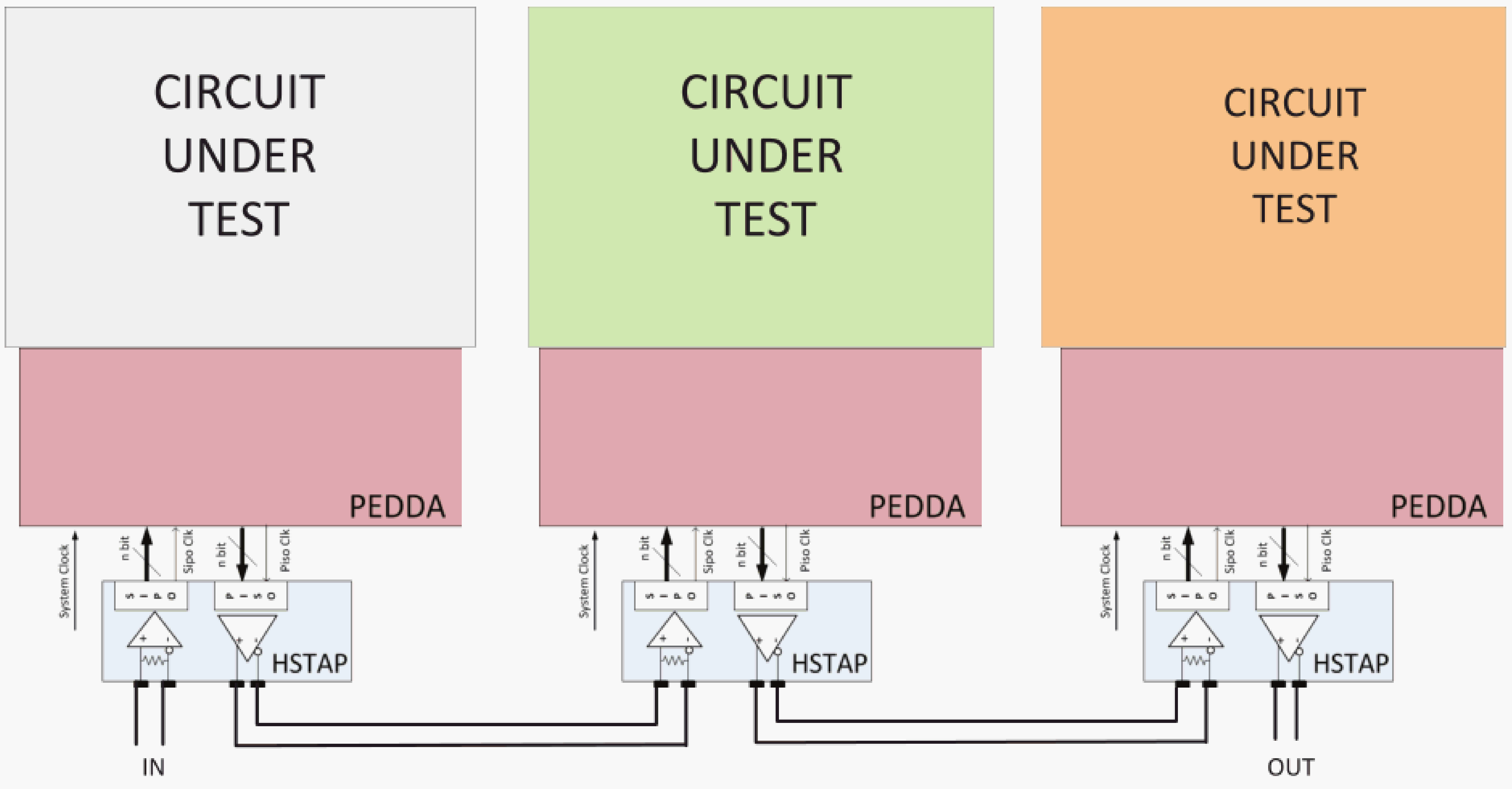

Figure 3—Example daisy chain of HSTAPs

Copyright © 2017 IEEE. All rights reserved.

IEEE 1149.10 provides a robust capability for monitoring, measuring, configuring, and testing integrated circuits. The compliance-enable character allows the signals used to be switched from mission mode to IEEE Std 1149.10-2017 compliance. This can be done in-system from a single access point such as a CPU. The other compliance methods require access to the IEEE 1149.1 TAP. This may require physical access outside of the IEEE 1149.10 interface to gain compliance, which may not be achievable. Figure 4 illustrates a system that can benefit from the compliance-enable character. When re-testing of the application-specific integrated circuit (ASIC) in-system is desired, the CPU can send the compliance-enable character over the mission mode point-to-point SERDES links to enable the ASIC to receive IEEE 1149.10 packet information. This provides a capability of in-system test or re-programming of the ASIC over mission mode connections using IEEE 1149.10.

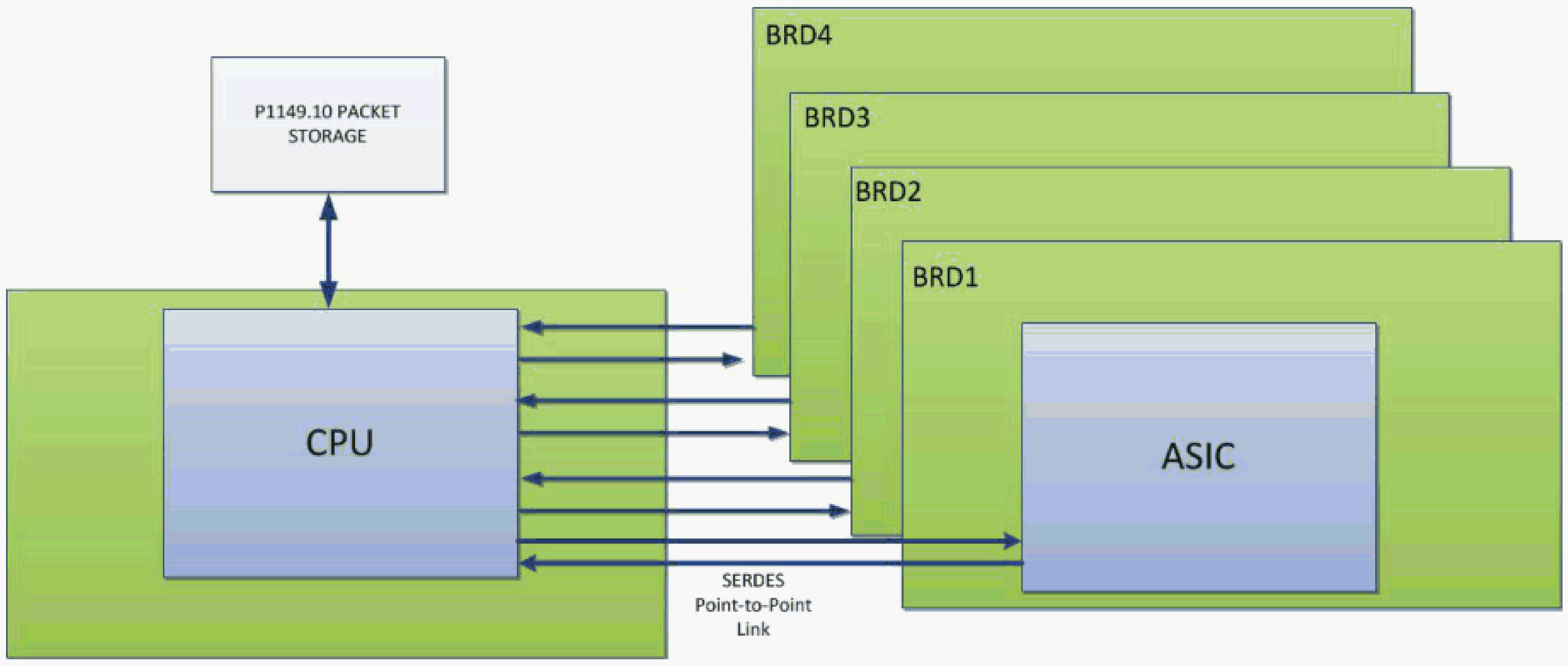

Figure 4—System that benefits from the compliance-enable character

5. Packet encoder/decoder and distribution architecture

5.1 PEDDA

The purpose of the PEDDA is to decode data arriving from the HSTAP and distribute it to the scan channels, while simultaneously collecting responses from the scan channels, encoding the data in packets, and providing them to the HSTAP.

5.1.1 Description

The PEDDA includes three interfaces: an HSTAP interface, a scan-channel interface, and an optional IEEE 1149.1 TAP Inferface. An example block diagram is shown in Figure 5.

Copyright © 2017 IEEE. All rights reserved.

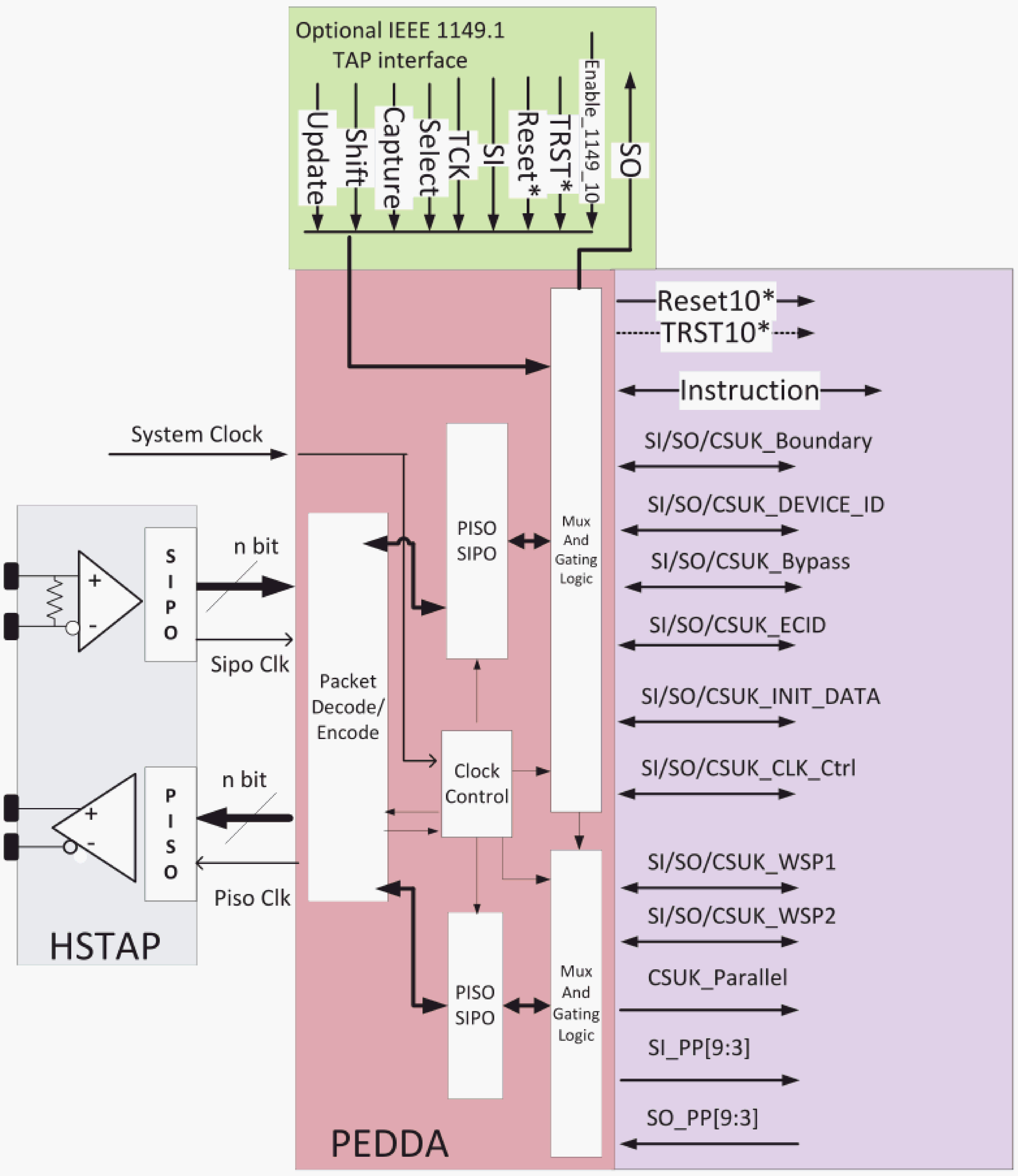

Figure 5—Example IEEE 1149.10 PEDDA block diagram

Note in the example that the scan-channel interface includes IEEE 1149.1-accessible registers, IEEE 1500 wrapper serial ports, and a set of parallel scan IEEE 1500 registers. CSUK is an acronym for Capture, Shift, Update, and Clock, the signals that typically control IEEE 1149.1 and IEEE 1500 registers.

When a PEDDA design includes an IEEE 1149.1 TAP interface, it includes inputs from the TAP and one output. *SO* is scan output, the output of the PEDDA that returns to the TAP. *SI* is the scan input, which delivers data from the TAP to the TDRs interfaced to the PEDDA. The *TCK* is the test clock signal defined in IEEE 1149.1. *Reset** is asserted when the IEEE 1149.1 TAP is in the *Test-Logic-Reset* state as defined in IEEE 1149.1. IEEE Std 1149.1 also defines *CH-Reset**, which is a gated version of *Reset** and cannot be asserted when the TMP Controller is in the *Persistence_on* state. When the Enable_1149_10 signal is de-asserted, the *Reset10** signal is asserted when the TAP interface *Reset** signal is asserted. *Reset10** is distributed in the IC the same way as *Reset** or *CH-Reset** would be in an IEEE 1149.1-only design.

Two methods of interfacing to the PEDDA from an IEEE 1149.1 TAP are allowed. One method uses the *Select*, *Capture*, *Shift,* and *Update* outputs of Figure 6.5 in IEEE Std 1149.1-2013; the other method allows for pre-designed legacy TAPs, which may have separate instruction and data register operation signals.

In rule b) v. of 5.1.2, the *Select* is an output of the TAP controller that determines which way the instruction or data register mux should be set. The signal *Capture* is defined as the logical OR of the Capture_DR state and the Capture_IR state. The signal *Shift* is the logical OR of the Shift_DR and the Shift_IR state. The signal *Update* is the logical OR of the Update_DR state and the Update_IR state. In rule b) vi. of 5.1.2, the signals represent the states Capture_DR, Shift_DR, Update_DR, Capture_IR, Shift_IR, and Update_IR.

5.1.2 Specifications

Rules

a) The PEDDA shall contain at least one HSTAP (see Clause 4) interface and a scan-channel interface.

b) When this standard's rules require an IEEE 1149.1 TAP [see rule 4.1.2 c)], the PEDDA shall include a TAP interface with the following inputs:
   i. TCK
   ii. SI
   iii. Reset*
   iv. Enable_1149_10
   and either
   v. Select, Capture, Shift and Update
   or
   vi. Capture_DR, Shift_DR, Update_DR, Capture_IR, Shift_IR, and Update_IR
   and the following output:
   vii. SO

c) When an IEEE 1149.1 TAP is connected to the TAP interface and the Enable_1149_10 signal is de-asserted, the PEDDA shall not interfere with the operation and access via the TAP of IEEE 1149.1 test data registers and IEEE 1500 wrapper serial ports connected at the scan-channel interface.

   NOTE—When compliance to IEEE 1149.10 is not enabled, then the IEEE 1149.10 circuitry should not interfere with the operation of the scan-channels.[5]

d) When the Enable_1149_10 signal is asserted the following shall occur:
   i. SCAN packets arriving at the PEDDA HSTAP interface shall be decoded and delivered to the appropriate scan-channel interface(s).
   ii. Scan data from the scan-channel interface(s) shall be encoded within a SCANR packet and delivered to the HSTAP interface.
   iii. All other IEEE 1149.10 packets shall be decoded and operated on as defined in this standard (see Clause 6) and a corresponding response packet shall be returned to the HSTAP interface.
   iv. All CONTROL_CHARS shall be operated on as defined in this standard.
   v. When a TAP is connected to the TAP interface and when the TAP is in the *Run-Test-Idle* or *Test-Logic-Reset* state, the circuitry of the TAP shall not interfere with the operation of the PEDDA on IEEE 1149.1 test data registers or IEEE 1500 wrapper serial ports connected at the scan-channel interface.

e) The scan-channel interface shall contain a Reset10* output.

f) When a RESET packet is received and the TYPE field bit position RESET10 is a logic one,

[5] Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

Copyright © 2017 IEEE. All rights reserved.

   i. The scan-channel interface shall assert Reset10* for a period of time determined by the designer and then de-assert it.

   and

   ii. The de-assert shall occur prior to any capture, shift, or update event which occurs due to the receipt of a SCAN packet.

   NOTE—The length of time here cannot be unreasonable. It is possible for the transmitter to send a RESET packet immediately followed by a SCAN packet without IDLE states in between.

g) When a RESET packet is received and the TYPE field bit position TRST10 is a logic one, and the optional TRST10* output is provided [see permission 5.1.2 y)], the scan-channel interface shall assert the signal TRST10*.
h) When a RESET packet is received and the TYPE field bit position TRST10 is a logic zero, the scan-channel interface shall de-assert TRST10*.

   NOTE—Asserting the signal Reset10* or the signal TRST10* does not change HSTAP compliance. See rule 4.1.2 e). This implies that scan registers documented in the PDL iProc *Disable_1149_10* must be initialized to mission mode operation using POR or other initialization method.

i) The PEDDA shall include a 16-bit TARGET_ID register.
j) The PEDDA shall be designed such that at power-on-reset or when IEEE 1149.10 compliance is enabled, the PEDDA's TARGET_ID register contents shall be set to all zeros.
k) The PEDDA shall be designed such that when the PEDDA'S TARGET_ID is all zeros, all packets received except the CONFIG packet are forwarded for transmission.
l) The PEDDA shall be designed such that a received CONFIG packet shall change the TARGET_ID register to the value specified by the <TARGET_ID> of the CONFIG packet, if the current TARGET_ID is un-configured (all zeros).
m) The TARGET_ID register shall retain its contents unless one of the following occurs:
   i. The PEDDA circuit is powered-off.
   ii. A RESET packet is received with the TYPE field TARGET_ID bit is a logic one.
   iii. IEEE 1149.10 compliance is disabled via the TAP.
n) When a RESET packet is received and the TYPE field bit position TARGET_ID is a logic one, the scan-channel interface shall set the TARGET_ID register to all zeros.
o) The PEDDA shall be designed such that a received TARGET packet that has a TARGET_ID that matches the TARGET_ID of the PEDDA, all subsequent packets received until the next TARGET packet received shall be operated on by the PEDDA.
p) The PEDDA shall be designed such that when a TARGET packet is received that has a TARGET_ID that does not match the TARGET_ID of the PEDDA, all subsequent packets until the next TARGET packet received shall be forwarded for transmission.
q) The PEDDA shall be designed such that a received CONFIG packet with an incorrect CRC32 shall have no effect on the TARGET_ID register.
r) The PEDDA shall be designed such that a received TARGET packet with an incorrect CRC32 shall have no effect on the operation of the PEDDA other than to return an ERROR_CHAR.
s) The PEDDA shall be designed such that a received RAW packet with an incorrect CRC32 shall have no effect on the operation of the PEDDA other than to return an ERROR_CHAR.
t) The PEDDA shall be designed such that if it has access to an IEEE 1149.1 boundary-scan register, said boundary-scan register shall have boundary-scan cells associated with the HSTAP signals on an excludable segment.

   NOTE—If IEEE 1149.10 is used to take a device into *EXTEST*, *EXTEST_PULSE*, *EXTEST_TRAIN*, *SELECTIVE_TOGGLE*, the boundary-scan register needs to be segmented at the HSTAP such that the HSTAP signals remain in mission-mode. See IEEE Std 1149.1 for details on excludable segments. See Figure 9.

Copyright © 2017 IEEE. All rights reserved.

Recommendations

u) The Reset10* signal should be routed to registers according to IEEE 1149.1 rules for RESET* when the PEDDA scan-channel interface is connected to IEEE 1149.1 test data registers.

v) The TRST10* signal, when provided, should be routed to registers according to IEEE 1149.1 rules for TRST* when the PEDDA scan-channel interface is connected to IEEE 1149.1 test data registers.

NOTE—These two recommendations are necessary to follow when an associated IEEE 1149.1 TAP is present and IEEE 1149.1 compliance is desired in addition to IEEE 1149.10. These signals should not be connected to registers which set or initialize IEEE 1149.10 compliance. See rule 4.1.2 e).

w) The Reset10* signal should be routed to registers according to IEEE Std 1500 [B1] rules for WRSTN when the PEDDA scan-channel interface is connected to IEEE 1500 registers.

NOTE—This recommendation is necessary to maintain IEEE Std 1500 [B1] compliance when IEEE Std 1500 is used for test data registers, and IEEE Std 1500 compliance is desired in addition to IEEE Std 1149.10-2017 compliance.

x) When it is feasible, a device that can benefit from IEEE Std 1149.1 compliance should be both IEEE Std 1149.1 and IEEE Std 1149.10-2017 compliant.

NOTE—Be careful not to simply abandon IEEE Std 1149.1 compliance in favor of IEEE Std 1149.10-2017 compliance without giving it some thought. IEEE Std 1149.1 offers a detailed boundary-scan register description that facilitates PCB-level test. System designers and OEMs may still require use of that boundary-scan register and the low speed TAP interface even if IEEE Std 1149.10-2017 is implemented.

Permissions

y) When a TAP is connected to the TAP interface, the PEDDA may include a TRST* input.

z) A PEDDA may include a scan-channel interface TRST10* output.

aa) A PEDDA may include a TAP interface with a connected TAP even when a TAP interface is not required by Rule 4.1.2 c)i.
NOTE—A TAP can be used with a PEDDA even if the HSTAP does not require a TAP to enter compliance.

5.1.3 Examples

Figure 6 shows an example implementation that meets rules 5.1.2 f), g), and h). If the IEEE 1149.1 TAP is in *Test-Logic-Reset* then there is no effect on Reset10* or TRST10* when Enable_1149_10 is asserted.

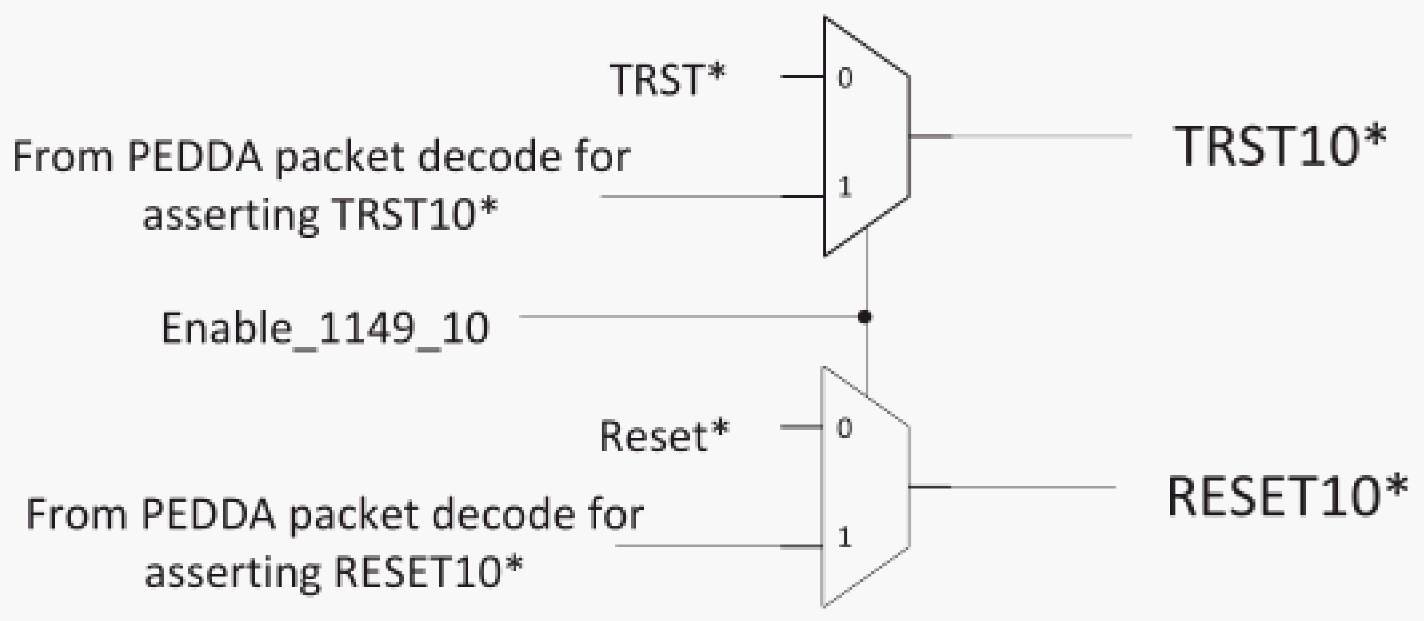
TRST*
0
1
From PEDDA packet decode for asserting TRST10*
TRST10*
Enable_1149_10
Reset*
0
1
From PEDDA packet decode for asserting RESET10*
RESET10*

Figure 6—Example implementation of TRST10* and Reset10*

Figure 7 shows an example TDR bit designed to assert Enable_1149_10 via the IEEE 1149.1 TAP. PORRESET is an IEEE 1149.1 description of a reset that occurs at power-on (or power-up). At power-on, a reset is performed to clear the update flop to zero and disable IEEE 1149.10 operation. This is just an example; other register bits may need to be set to configure the mission mode signals to operate as an HSTAP. See Clause 9 for more information on how PDL can operate this register.

Copyright © 2017 IEEE. All rights reserved.

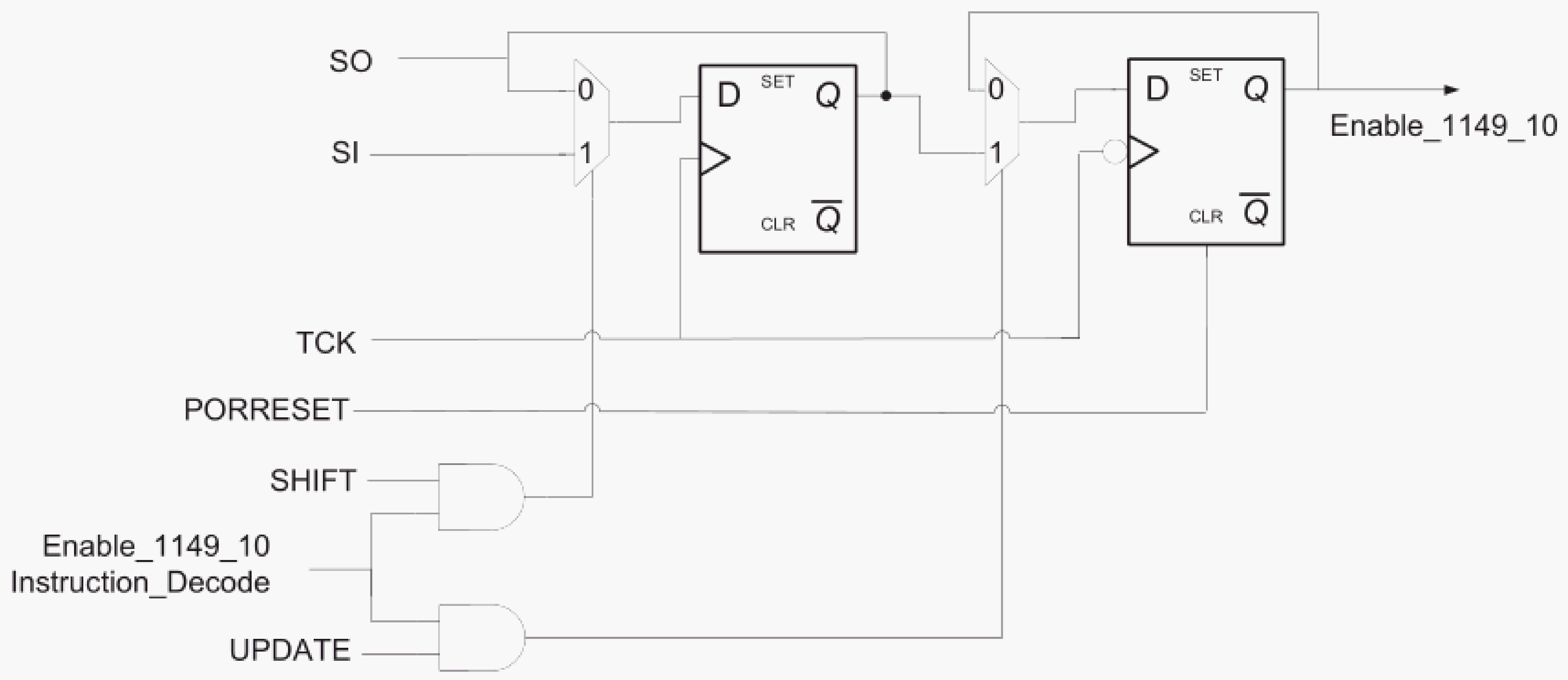

Figure 7—Example IEEE 1149.1 TDR for asserting Enable_1149_10

Figure 8 shows an example that uses both the compliance-enable character and an IEEE 1149.1 TDR to assert Enable_1149_10. The example shows a circuit that asserts Enable_1149_10 when either the compliance-enable character is received or via the IEEE 1149.1 TAP. An IEEE 1149.1 PULSE0 bit is used to de-assert Enable_1149_10 such that the rules in rule e) of 4.1.2 are met.

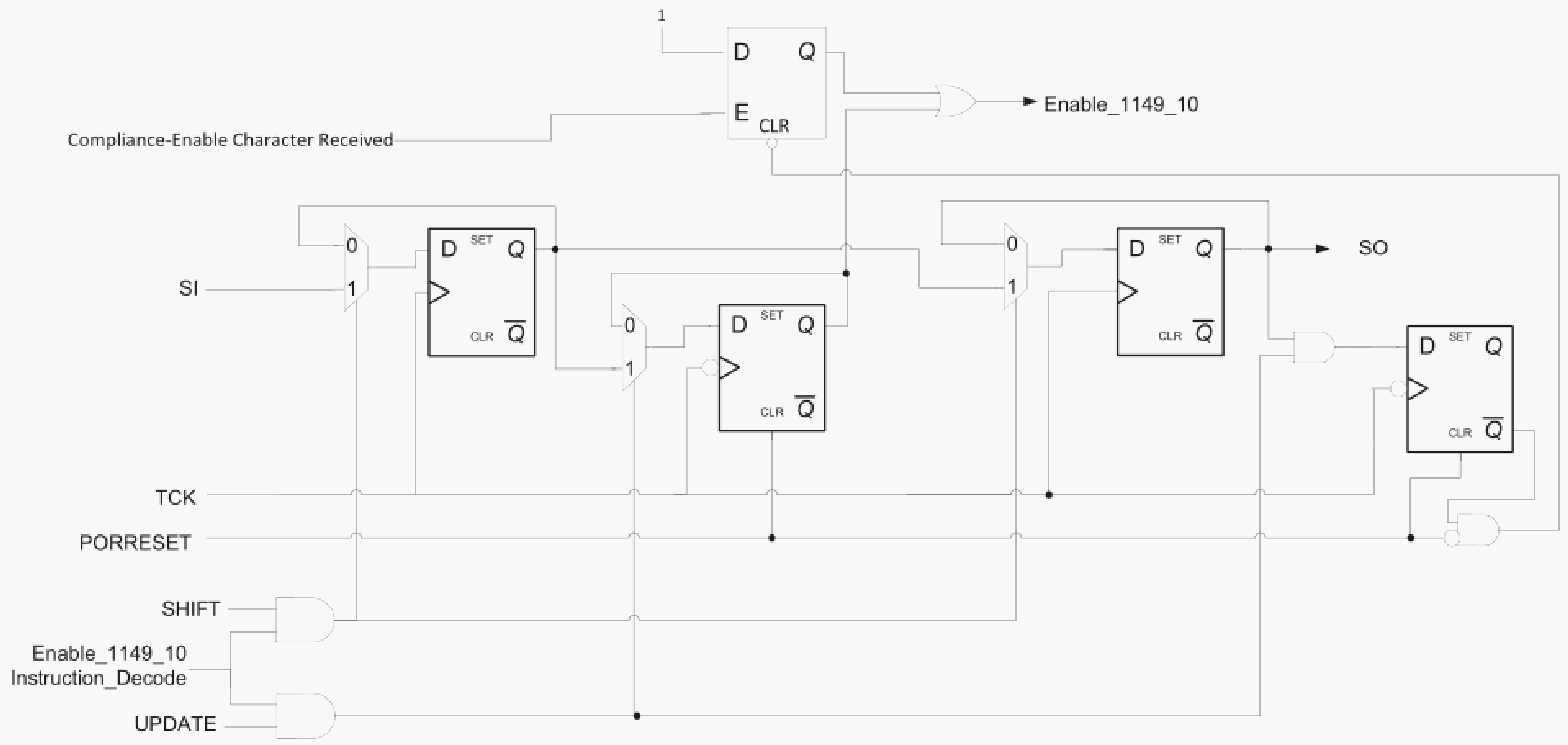

Figure 8—Example circuit for asserting Enable_1149_10 via a compliance-enable character

Figure 9 shows an example block diagram of how to segment TDRs for optimal use with IEEE 1149.10, and how to achieve compliance with rule t) of 5.1.2. The figure illustrates shared access via the TAP and HSTAP to the init-data register and the boundary-scan register. The access via the HSTAP is optimized by segmenting the two TDRs into multiple segments that can be scanned concurrently. Concurrent access enables better usage of the bandwidth provided by the HSTAP and shortens test time. The boundary-scan register and the init-data register are used for a variety of pin related tests. Leakage tests, Pull-up/Pull-down, V_{OH}/V_{OL}, and related pin tests can be tested faster when segmentation is present because the HSTAP/PEDDA can access each segment concurrently. This segmentation can also be used to achieve compliance with rule t) of 5.1.2. Without segmentation, tests using the boundary or init-data register would not be any faster than with an IEEE 1149.1 TAP for access. Figure 9 shows how to put the boundary-scan (b-s) cells associated with the HSTAP on a collapsible segment. This is necessary in order to leave the

Copyright © 2017 IEEE. All rights reserved.

HSTAP in mission mode when instructions such as EXTEST, EXTEST_PULSE, etc. are used; they will not put the HSTAP related b-s cells in test mode. Note that segments can have any number of pipeline registers (shown as the box "P" in the figure) in order to meet timing.

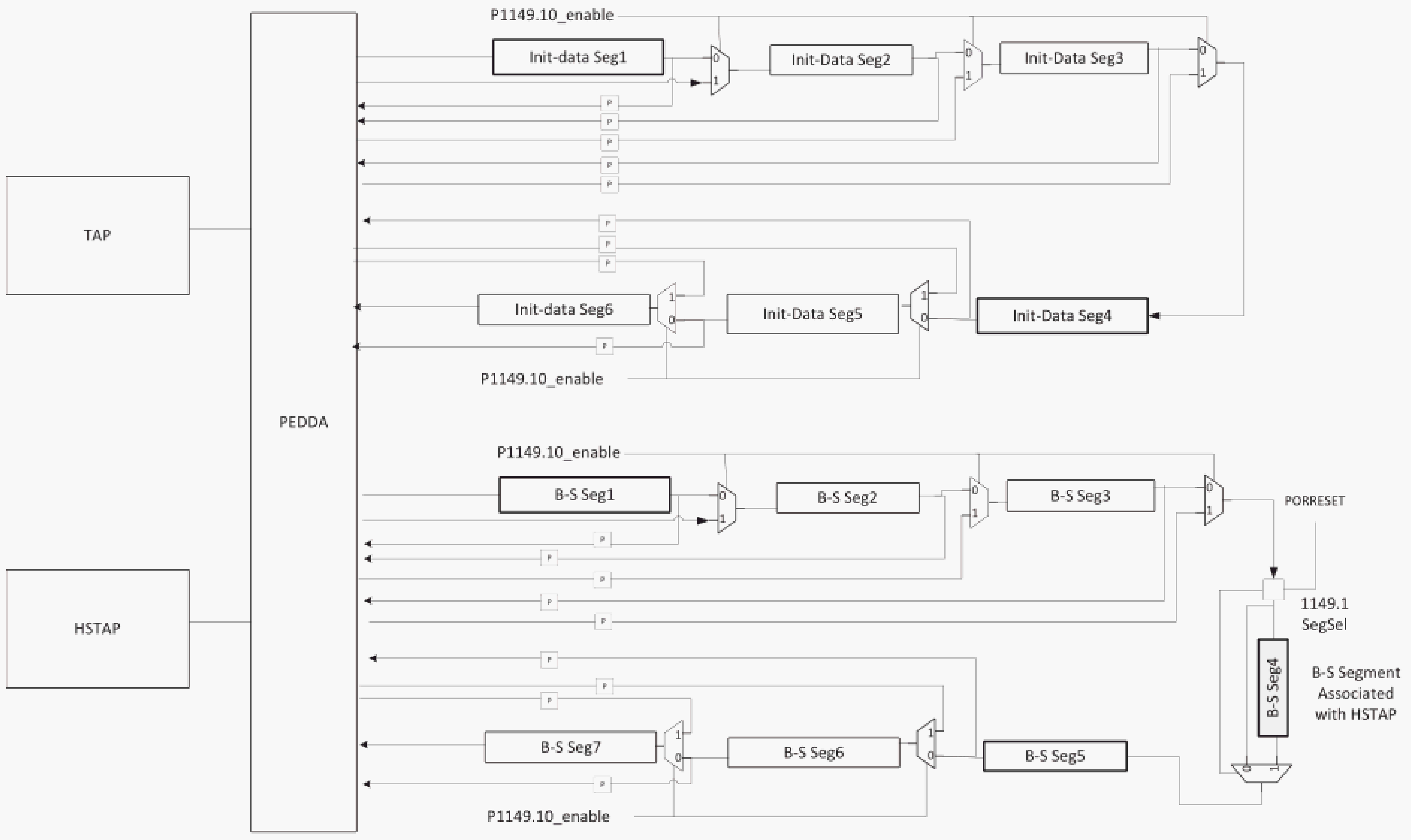

Figure 9—Example use of B-S and Init-Data Register

6. Packet definitions

6.1 Packet overview

This subclause provides the general overview of the inbound and outbound packets defined by this standard.

6.1.1 Description

This standard defines two packet types: inbound packets and outbound packets. The IEEE 1149.10 protocol also includes control characters that are used to define the start and end of a packet and any special communications such as error states or idle states. Each packet has a command that controls operation of the PEDDA. Inbound packets are listed in Table 1 and outbound packets are listed in Table 2. The full descriptions are available in the associated packet description in this clause.

Copyright © 2017 IEEE. All rights reserved.

Table 1—Inbound packets

Command	Value	Description
CONFIG	0x01	Enable uninitialized IEEE 1149.10 interface to be enumerated.
TARGET	0x02	Specify the target IEEE 1149.10 interface for all subsequent packets.
RESET	0x03	Assert Reset10* or TRST10* (internally different signals) or clear the TARGET_ID.
RAW	0x04	Enable the interface in a RAW data mode (suitable for BER testing). Data is not processed by the packet processor and subsequently all RX data is sent to the TX.
CH-SELECT	0x05	Set the SCAN_GROUP and target channels on subsequent IR/DR scan packets.
SCAN	0x06	An interleaved packet for IR/DR scans.
BOND	0x07	Signal the HSTAP to go from single lane to multi-lane.
USER	0x20-0x3F	A user-defined packet used to transfer data not anticipated by this standard.
All outbound packets	See Table 2	All response packets are ignored by the input and forwarded unmodified to the TX.

Table 2—Outbound packets

Command	Value	Description
CONFIGR	0x81	A CONFIG packet response.
TARGETR	0x82	A TARGET packet response.
RESETR	0x83	A RESET packet response.
RAWR	0x84	A RAW packet response.
CH-SELECTR	0x85	A CH-SELECT packet response.
SCANR	0x86	An interleaved SCAN packet response.
BONDR	0x87	Response to BOND packet request.
USERR	0xA0-0xBF	A USER defined packet response.
All non-target inbound packets	See Table 1	All packets not destined for this target are forwarded unmodified.

Outbound response packets differ from the inbound packets by asserting bit 7 of the command byte high. An optional parity bit can be used on bit 6, hence that bit is reserved when that capability is enabled and documented in BSDL. Therefore, inbound commands 0x40-0x7F are also available when parity checking on the command is not used. The general format of both a send and a receive packet is shown in Figure 10. Packets begin with a Start-of-Packet (SOP) control character, followed by a command (CMD) byte, followed by the command-specific data (PAYLOAD), followed by the 32-bit CRC (CRC32), and terminated by four End-of-Packet (EOP) control characters. All packets have an integer multiple of four bytes up to the CRC32, followed by a CRC32 of four bytes and the four EOP bytes. A "packet frame" refers to any of the four bytes that start on an integer multiple of four. Other than the SOP character, control characters are sent in multiples of four before the next packet frame. A trade-off was made in requiring redundant control characters such as the four EOP characters in order to keep the packet aligned, which is thought to simplify the receiver decoding logic.

The packet definitions are shown in this standard without regard to the encoding/decoding that may be implemented. This standard follows the convention of other serial standards such that the data in the packet

Copyright © 2017 IEEE. All rights reserved.

is shown in human readable MSB to LSB, where the LSB is on the right without encoding. However, for many protocols, bits are transmitted with LSB first.

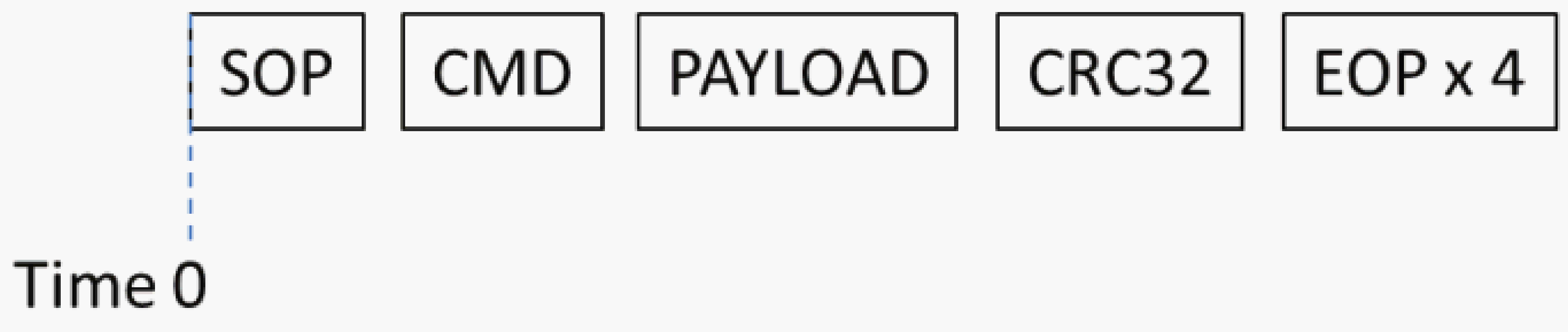

Figure 10—General form of a IEEE 1149.10 packet

6.1.2 Specifications

Rules

a) All packets shall start with an SOP character and end with four EOP characters.

NOTE—All packet descriptions and rules are provided prior to any encoding.

b) Following the SOP character, all packets shall have eight bits representing a command (CMD) and shall have a value as specified in Table 1 or Table 2.

NOTE—The values in the tables are shown without the use of parity [see permission 6.1.2 o)].

c) Following the command (CMD), all packets shall have a PAYLOAD, which represents the data for a particular packet as defined in this clause.
d) All packets shall have an SOP byte, CMD byte, and PAYLOAD bytes with a total sum of bytes that is an integer multiple of four.
e) All packets shall include a CRC32 (Cyclical Redundancy Check) as the last 32 bits of data prior to the four EOP characters.
f) The CRC32 shall be calculated on all of the data received between the SOP and the CRC32 32-bit value that is not a control character.

NOTE—The SOP and any IDLE, XON, or XOFF values are not included in the CRC32 calculation. See also 7.5 on control characters.

g) When a PEDDA receives a TARGET packet with a TARGET_ID not matching its own TARGET_ID, that TARGET packet and all subsequent packets shall be forwarded for transmission by the HSTAP.
h) All received packets that have the most significant bit of the CMD field set to a logic one shall be forwarded for transmitting by the HSTAP.

NOTE—All response packets that are received by a PEDDA are forwarded.

Permissions

i) An integer multiple of four IDLE characters may appear in a transmission before the start of any packet frame.
j) An ERROR character followed by three data bytes may appear in a transmission before the start of any packet frame. The three bytes that follow the ERROR character may contain any user defined data that is not a control character.

Copyright © 2017 IEEE. All rights reserved.

k) An integer multiple of four XOFF characters may appear in a transmission before the start of any packet frame.
l) An integer multiple of four XON characters may appear in a transmission before the start of any packet frame.
m) An integer multiple of four CLEAR characters may appear in a transmission before the start of any packet frame.
n) An integer multiple of four COMPLIANCE characters may appear in a transmission before an SOP and after the last EOP character.
o) Bit 6 (counting from Bit 0) of the CMD may be used to indicate even or odd parity of the eight-bit CMD value.

NOTE—See 7.4.2

6.2 The CONFIG packet

The CONFIG packet initializes an un-configured IEEE 1149.10 interface such that it is enumerated. A PEDDA powers up with a TARGET_ID of all zeros. The CONFIG packet is used to set the IEEE 1149.10 to an identification value greater than zero. After configuration, the TARGET packet can be used to select the IEEE 1149.10 interface and PEDDA.

6.2.1 Description

The CONFIG packet includes a command (CMD) value of 0x01 followed by a 16-bit TARGET_ID to identify the IEEE 1149.10 interface by number (see Figure 11). When a daisy-chain of IEEE 1149.10 interfaces exists, where the TX of one is connected to the RX of another, each interface would be assigned a unique TARGET_ID within the daisy-chain. The first interface in the chain, the HSTAP RX closest to the ATE, would be assigned a unique TARGET_ID first, and then, in turn, each interface along the chain would be assigned a different TARGET_ID.

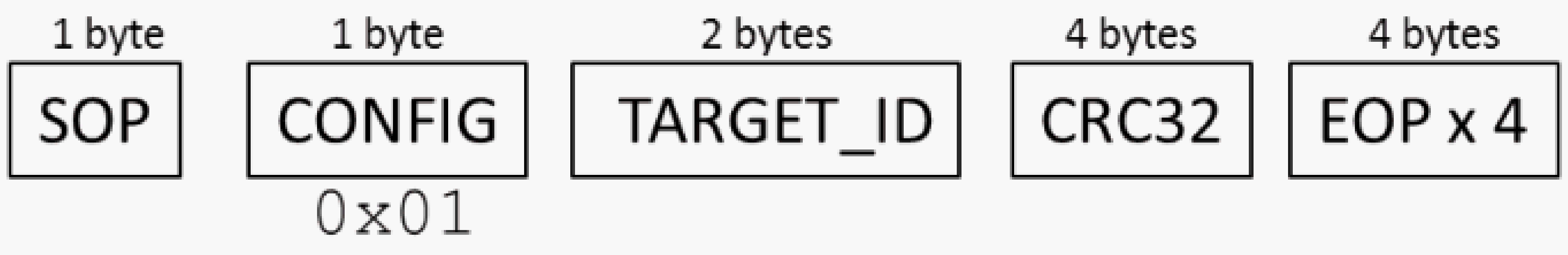

Figure 11—Format of CONFIG packet

The CONFIG packet is used early in the initialization process to set the TARGET_ID of each IEEE 1149.10 interface.

6.2.2 Specifications

Rules

a) The CONFIG command shall have a value of 0x01.
b) A TARGET_ID shall be 16 bits wide and have a value greater than zero.
c) The TARGET_ID shall directly follow the CONFIG command byte.

Examples

Given an attribute CONTROL_CHARS as follows (see 7.5 for more information on the attribute CONTROL_CHARS):

Copyright © 2017 IEEE. All rights reserved.

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR :     0xFB ), " &
"(EOP_CHAR  :    0xFD )," &
"(IDLE_CHAR :    0xBC),"&
"(ERROR_CHAR :   0xFE ),"&
"(XOFF_CHAR :    0x7C ),"&
"(XON_CHAR:      0x1C),"&
"(CLEAR_CHAR :   0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

The SOP is 0xFB and the EOP is 0xFD. The data transmitted from the ATE to the input of an IEEE 1149.10 compliant interface (assuming LSB is transmitted first and there is no encoding) is as follows:

0xFB followed by 0x01, followed by 0x0001 (sent as 0x01 then 0x00), followed by a CRC32 of 0xE7988264 for the three data bytes received, followed by four bytes of 0xFD used for the EOP. These are the raw unencoded data values; the bit lengths would be different depending on the encoding/decoding format (8B/10B, 64B/66B, etc.). Table 3 provides the packet transmit order in tabular form.

Table 3—CONFIG packet transmit order

Xmit	Data	Hex	Binary
1	SOP	0xFB	0b11111011
2	CMD	0x01	0b00000001
3	TARGET_ID Byte 0	0x01	0b00000001
4	TARGET_ID Byte 1	0x00	0b00000000
5	CRC32 Byte 0	0x64	0b01100100
6	CRC32 Byte 1	0x82	0b10000010
7	CRC32 Byte 2	0x98	0b10011000
8	CRC32 Byte 3	0xE7	0b11100111
9	EOP	0xFD	0b11111101
10	EOP	0xFD	0b11111101
11	EOP	0xFD	0b11111101
12	EOP	0xFD	0b11111101

The bits would be transmitted as shown in Figure 12. For ease of reading, the bits are shown without encoding and transmitted with LSB first.

Copyright © 2017 IEEE. All rights reserved.

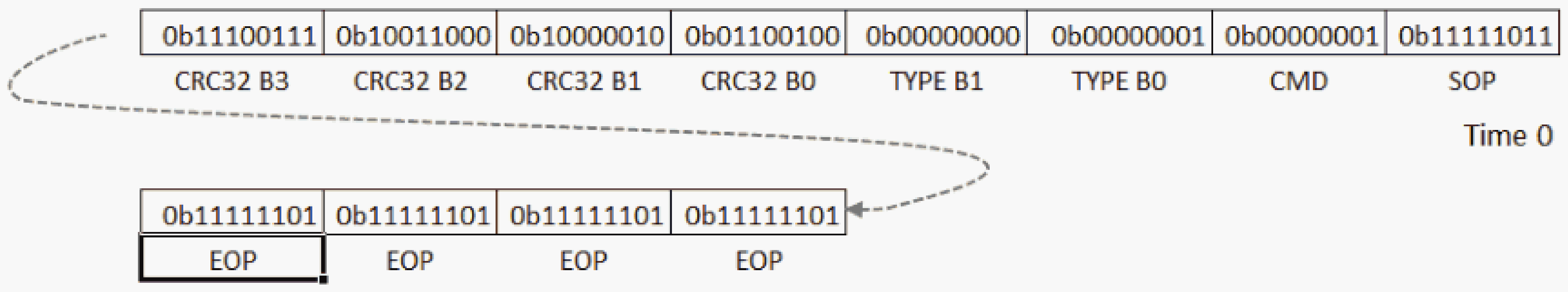

Figure 12—Transmit order for CONFIG packet using LSB_First based HSTAP

6.3 The TARGET packet

6.3.1 Description

The TARGET packet is used to communicate to an IEEE 1149.10 interface that the subsequent packets will be for the IEEE 1149.10 interface with the given TARGET_ID (see Figure 13).

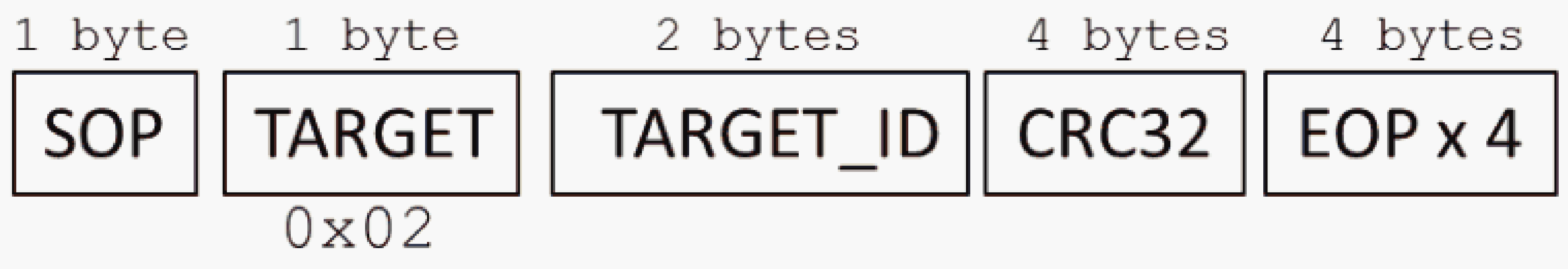

Figure 13—Format of TARGET packet

6.3.2 Specifications

Rules

a) The TARGET command shall have a value of 0x02.
b) A TARGET_ID shall be 16 bits wide and have a value greater than zero.
c) The TARGET_ID shall directly follow the TARGET command byte.

Examples

Given an attribute CONTROL_CHARS as follows:

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR:    0xFB ), " &
"(EOP_CHAR  :   0xFD ),"  &
"(IDLE_CHAR :   0xBC),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0x7C ),"&
"(XON_CHAR:     0x1C),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

Copyright © 2017 IEEE. All rights reserved.

The SOP is 0xFB and the EOP is 0xFD. The data transmitted from ATE to the input of an IEEE 1149.10 compliant interface is as follows:

0xFB followed by 0x02, followed by 0x0001, followed by a CRC32 of 0xE5DE3C3D calculated on the 0x02, 0x01, and 0x00 received in that order, followed by four bytes of 0xFD as the EOP. The data is shown in Table 4 without encoding for ease of reading.

Table 4—TARGET packet transmit order

Xmit	Data	Hex	Binary
1	SOP	0xFB	0b11111011
2	CMD	0x02	0b00000010
3	TARGET_ID Byte 0	0x01	0b00000001
4	TARGET_ID Byte 1	0x00	0b00000000
5	CRC32 Byte 0	0x3D	0b00111101
6	CRC32 Byte 1	0x3C	0b00111100
7	CRC32 Byte 2	0xDE	0b11011110
8	CRC32 Byte 3	0xE5	0b11100101
9	EOP	0xFD	0b11111101
10	EOP	0xFD	0b11111101
11	EOP	0xFD	0b11111101
12	EOP	0xFD	0b11111101

6.4 The RESET packet

6.4.1 Description

The RESET packet is used to indicate when resets should be asserted (see Figure 14). There are two reset signals (one mandatory and one optional) on the scan interface of the PEDDA. These are Reset10* and TRST10*. The 16-bit TYPE field of the packet indicates which type of reset to perform. The TYPE is one of three values: 0x01 for RESET10*, 0x02 for TRST10*, and 0x04 to reset the TARGET_ID to all zeros. Per the PEDDA rules, the reset's asserted state may have any duration required by the design itself within the limitations of the rules [see rule 5.1.2 f)]. The designer is responsible for de-asserting RESET10* after the appropriate duration. TRST10* remains asserted to keep compatibility with IEEE 1149.1 TRST* and the PDL command iTRST. A second RESET packet with TYPE set to 0x00 would be sent to de-assert TRST10*.

Copyright © 2017 IEEE. All rights reserved.

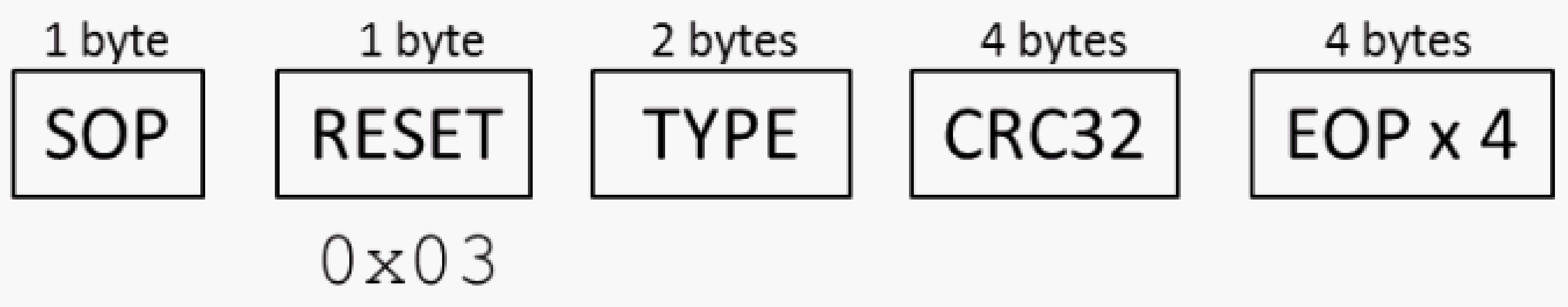

Figure 14—Format of a RESET packet

6.4.2 Specifications

Rules

a) The RESET command shall have a value of 0x03.
b) A 16-bit TYPE field shall follow the RESET command.
c) Bit 0 of the TYPE field shall represent the Reset10* signal and this bit position shall be known as the RESET10 bit position.

NOTE—See rule f) in 5.1.2. The PEDDA asserts Reset10* for a time determined by the designer, i.e., it is momentary and returns to its de-asserted state without another packet needing to be sent.

d) Bit 1 of the TYPE field shall represent the next state of the TRST10* signal and this bit position shall be known as the TRST10 bit position.
e) Bit 2 of the TYPE field shall be known as the TARGET_ID bit position and is used for clearing the TARGET_ID to all-zeros.

Permissions

f) The TYPE field may contain any value.

6.4.3 Examples

Given the following CONTROL_CHARS attribute:

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR :    0x7C ), " &
"(EOP_CHAR  :   0x1C )," &
"(IDLE_CHAR :   0xBC),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0xFB ),"&
"(XON_CHAR:     0xFD),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

Reset10* can be asserted via the packet data shown in Table 5. The bytes to transmit would be 0x7C followed by 0x03, followed by 0x0001, followed by a CRC32 of 0xE41C560A, followed by the EOP of 0x1C.

Copyright © 2017 IEEE. All rights reserved.

Table 5—RESET packet data and transmit order

Xmit	Data	Hex	Binary
1	SOP	0x7C	0b01111100
2	CMD	0x03	0b00000011
3	TYPE B0	0x01	0b00000001
4	TYPE B1	0x00	0b00000000
5	CRC32 Byte 0	0x0A	0b00001010
6	CRC32 Byte 1	0x56	0b01010110
7	CRC32 Byte 2	0x1C	0b00011100
8	CRC32 Byte 3	0xE4	0b11100100
9	EOP	0x1C	0b00011100
10	EOP	0x1C	0b00011100
11	EOP	0x1C	0b00011100
12	EOP	0x1C	0b00011100

The bits would be transmitted as shown in Figure 15. For ease of reading, the bits are shown without encoding and transmitted with LSB first.

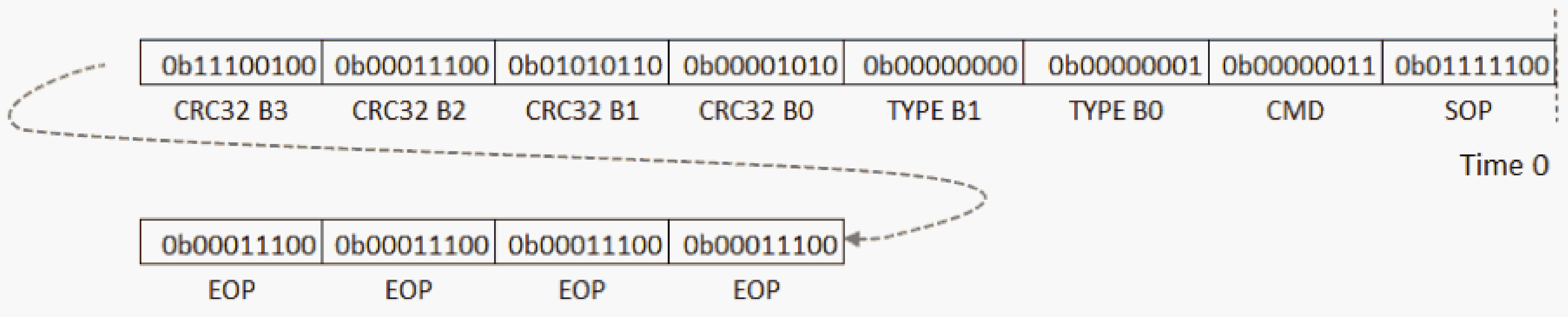

Figure 15—Bit transmit order for LSB_First based HSTAP

6.5 The RAW packet

6.5.1 Description

The RAW packet is used to set the IEEE 1149.10 interface into a loopback mode, where data arriving at the receiver is sent to the transmit pins (see Figure 16). One use model for the RAW packet would be to use this command to put the IEEE 1149.10 interface into a “far end” loopback mode. A PRBS generator could then be used to send random data to the IEEE 1149.10 receiver and responses observed at the IEEE 1149.10 transmitter. The response data could be compared or compacted to an expected value as part of a signal integrity test. The PEDDA exits RAW packet mode when the IEEE 1149.10 interface receives a power-on reset signal. If a TAP is present then it is recommended to provide IEEE 1149.1 PDL code and a TDR to enable the return of the HSTAP to either mission mode or IEEE 1149.10 packet processing. The

Copyright © 2017 IEEE. All rights reserved.

RAW packet should not be used at the system level when a daisy-chain of HSTAPs exists. ATE must have direct access to the TX and RX of an HSTAP to use the RAW packet.

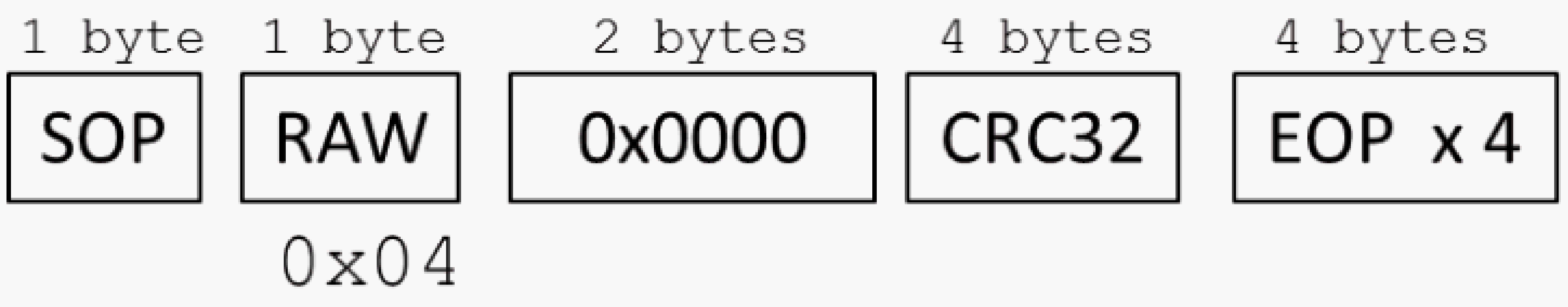

Figure 16—Format of a RAW packet

6.5.2 Specifications

Rules

a) The RAW command shall have a value of 0x04.
b) A 16-bit value of 0x0000 shall follow the CMD byte.
c) If the optional IEEE 1149.1 TAP is provided, and the design of user-provided test registers can clear the effects of the RAW command, an IEEE 1149.1 PDL procedure *Disable_Rawmode* shall be provided such that when the steps in the iProc *Disable_Rawmode* are performed, the interface returns to normal IEEE 1149.10 packet processing.

   NOTE—The TARGET_ID of the PEDDA remains unchanged. See rule m) in 5.1.2.

d) When a RAW packet is received then all subsequent data arriving to the HSTAP shall be transmitted on the HSTAP transmitter pins.

   NOTE—The RAW command puts the HSTAP interface into a permanent loopback mode, hence, this mode is not used when the ATE does not have direct access to the transmitter side of the HSTAP.

Recommendations

e) If an IEEE 1149.1 TAP is provided, a user-defined TDR that can return the IEEE 1149.10 interface back to normal IEEE 1149.10 packet processing should be provided.

6.6 The CH-SELECT packet

6.6.1 Description

Figure 17 shows the format of the CH-SELECT packet. The CH-SELECT packet specifies the SCAN_GROUP and the number of Channel-Select words that follow in the scan packet. A word is 16 bits in this context. Each scan-channel is enabled for Capture, Shift, or Update in a subsequent SCAN packet by having a corresponding bit set in the Channel-Select word.

Copyright © 2017 IEEE. All rights reserved.

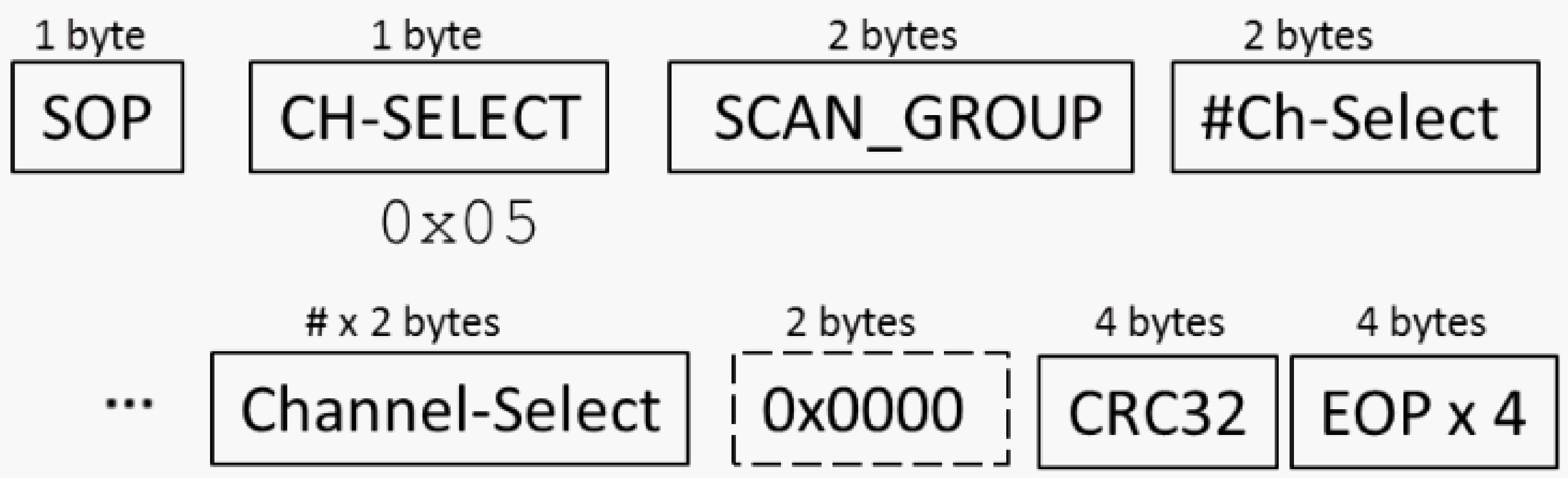

Figure 17—CH-SELECT packet format

If #Ch-Select is 0x0001, then there is one Channel-Select word of 16 bits wide that can communicate to a maximum of 16 channels (15 through 0). Because #Ch-Select is 16 bits wide then there are:

$$2^{**}16 \times 16 = 1{,}048{,}576 \text{ scan-channels per SCAN packet.}$$

The maximum number of scan-channels that can be communicated in a single interleaved SCAN packet is 1,048,576. The SCAN_GROUP indicates in which group the scan channels exist. With 2**16 possible SCAN_GROUP values then there is a maximum of:

$$2^{**}16 \times 1{,}048{,}576 = 68{,}719{,}476{,}736 \text{ possible scan-channels per PEDDA.}$$

At the time of this writing, neither the scan-channels per packet nor the scan-channels per PEDDA are expected to be reached in a real world design due to routing constraints. As more scan-channels are communicated to in a given SCAN packet, the bandwidth provided by the HSTAP is divided up across those scan-channels. The maximum theoretical limit a 50 Gbit/s interface would be able to deliver is 5000 scan-channels at 10 MHz.

When #Ch-Select is large, the size of the subsequent SCAN packet increases. One way to keep the SCAN packets small is to limit the number in the #Ch-Select word to a small number like four (4 × 16 = 64) and have 64 scan-channels per SCAN packet, and then use the SCAN_GROUP to target one of the four groups of scan-channels in each SCAN packet transmission.

6.6.2 Specifications

Rules

a) The CH-SELECT command shall have a value of 0x05.
b) A 16-bit SCAN_GROUP shall follow the CH-SELECT command byte.
c) The SCAN_GROUP value shall be followed by a two-byte #Ch-Select value representing the number of Channel-Select words provided in this packet.
d) The #Ch-Select value shall be non-zero.
e) Following the #Ch-Select value, there shall be a number of Channel-Select words; the number being equivalent to the contents of the #Ch-Select value and each of these Channel-Select words shall be 16 bits long.
f) The bits of the Channel-Select words in the packet shall represent the channel selects counted from LSB to MSB that are necessary for enabling a scan channel that is present in the PAYLOAD portion of a subsequent SCAN packet.

NOTE—This is a description of packet bit ordering and not representative of what may be transmitted at the physical layer. Bits are counted in the word starting with Channel zero for the LSB.

Copyright © 2017 IEEE. All rights reserved.

g) A logical one in a Channel-Select word bit position shall indicate the corresponding select for the scan channel shall be asserted.

h) Following the Channel-Select words there shall be a number of bytes with value of 0x00 equal to:
2 * ((#Ch-Select -1) mod 2)

Permissions

i) Positions in the Channel-Select words may have no meaning at all and may serve as "don't care" bits when it is necessary to align bits.

6.6.3 Examples

In this example, the PEDDA design connects to thirty-two (32) scan-channels. It is desired for a certain scan operation to enable Scan-Chanel 0 and Scan Channel 30. Figure 18 shows the detail of the #Ch-Select and Channel-Select words and how they would be transmitted serially if there was no encoding and the bits were transmitted LSB first. The #Ch-Select word specifies the number of Channel-Select words that follow the #Ch-Select word. In the example the #Ch-Select is two so the PEDDA receiver will expect two 16-bit words following the #Ch-Select. The order is shown with the most significant bit, indicating the largest enumerated scan channel transmitted in the second Channel-Select word.

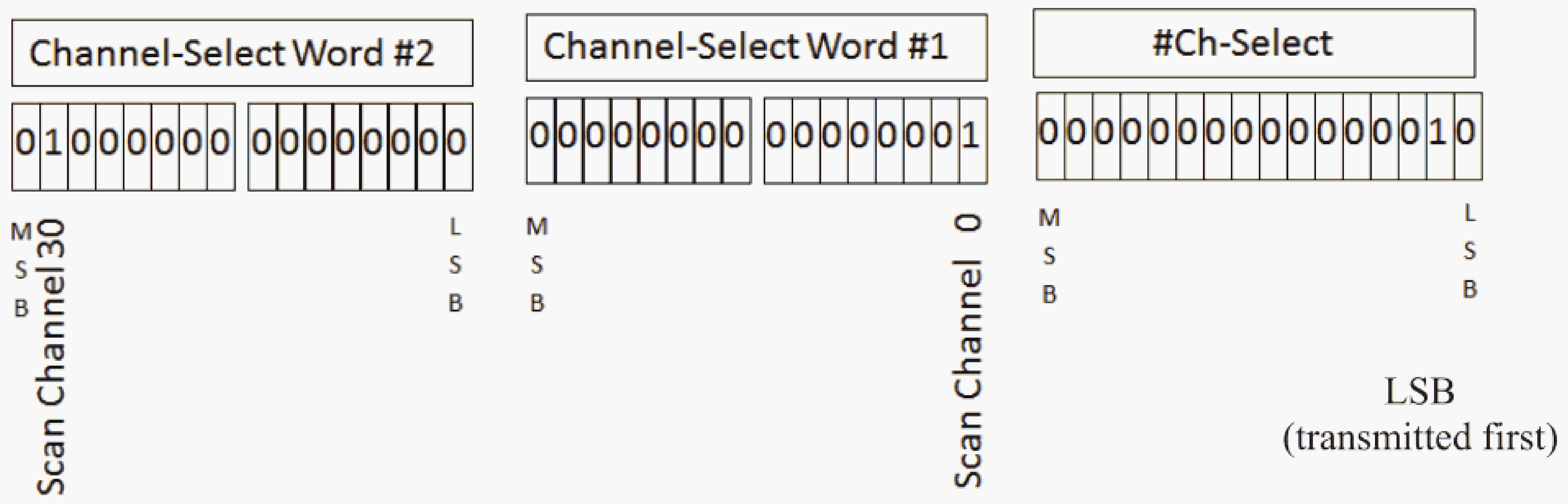

Figure 18—Example scan-channel selection detail

6.7 The SCAN packet

6.7.1 Description

The SCAN packet is used to communicate data to one or more scan channels. The generic format of the SCAN packet is shown in Figure 19. The SCAN command of 0x06 is provided after the SOP. Following the SCAN command is an 8-bit identification (ID). This ID is present to facilitate tracking packets and their responses, because they would have the same ID. The maximum ID is 0xFF. Typically the ATE will start applying SCAN packets starting with an ID of zero and incrementing each subsequent SCAN packet by one. There are no specific rules about the ID as the PEDDA does not interpret the ID. The PEDDA uses the ID to assign the same value to the SCANR response packet.

Following the ID byte there is a 4-bit value of 0b0000 such that the ICSU nibble and 4-bit value together are a full byte in width. The ICSU is a nibble representing the bits for enabling Instruction, Capture, Shift, and Update functions. When the MSB of the ICSU is asserted, the PEDDA will enable scan operations to

Copyright © 2017 IEEE. All rights reserved.

the instruction register, if present. The Capture, Shift, and Update bits, when asserted, determine if those scan functions are performed on the channels selected by the Channel-Select words from the CH-SELECT packet.

The ICSU is followed by a 32-bit value representing the #Payload-Frames in the PAYLOAD. The Cycle-Count follows and is another 32-bit value that indicates the number of scan clocks used to shift the scan-channels selected in the packet. Neither the capture nor the update events are counted in the Cycle-Count field.

The PAYLOAD is uniquely defined for each SCAN operation. The data in the payload for more than one scan-channel is interleaved based on the INTERLEAVE_SIZE value from the PACKET_MAP BSDL attribute. The PAYLOAD is followed by a CRC32, which is calculated on all of the non-control characters after the SOP.

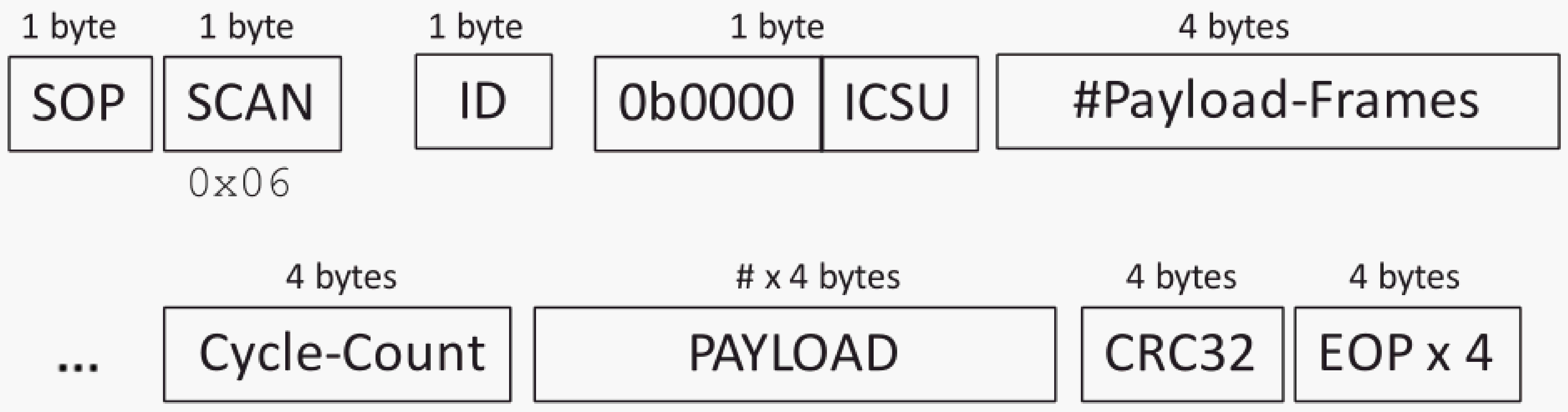

Figure 19—Generic format of a SCAN packet

The PAYLOAD of a SCAN packet consists of interleaved scan data to be shifted in. Interleave is specified by the <Interleave_Size> in the BSDL PACKET_MAP attribute. Details of the PACKET_MAP attribute are found in 7.4. The interleave size determines the number of contiguous bits sent to each channel selected by the Channel-Select word when the SCAN packet is transmitted. In the example attribute below, <Interleave_Size> is set to 1.

```
attribute Packet_Map of MyChip : entity IS
"(PEDDA_NAME     : PEDDA1)," &
"(TX_Order       : LSB_FIRST),"&
"(CMD_Parity     : NONE_1149_10)," &
"(SCAN_DATA_SIZE      : 4), " &
"(PACK                 : FALSE), " &
"(Interleave_Size: 1), " &
"(Scan_Group : 1)";
```

In order for the reader to follow the scan-in data for each of four example scan channels, the binary data is depicted as alphabetical characters in Figure 20. It is desired to shift in AEI into channel 0, BFJ in channel 1, etc. The final data in each scan-channel is shown in Figure 20. Bit I is in the register that is closest to the scan out.

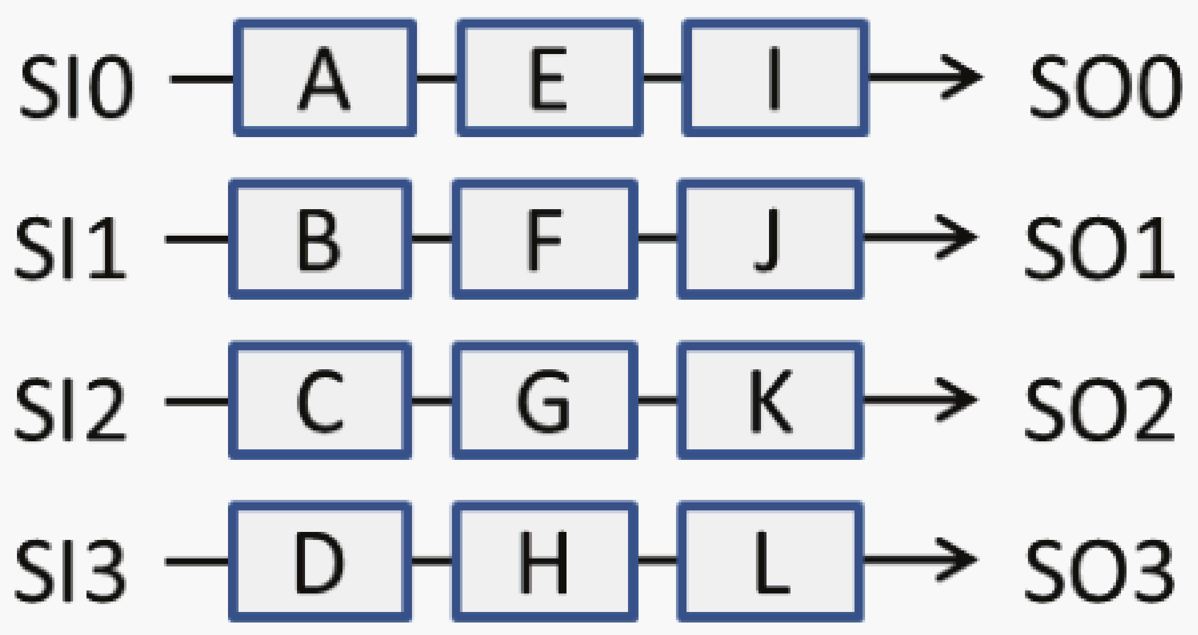

Figure 20—Data in scan-channel 3 through 0 after scan

Figure 21 shows how the first data word is formed when all four channels are enabled in the packet. Each position represents one bit of a round-robin of each scan channel starting at Channel 0 followed by each channel in ascending order, padding as necessary until the SCAN_DATA_SIZE (4 in this case) is reached. The Bit0 position is commonly transmitted first, so the bits are shown in that order (assuming there is no 8B/10B, 64B/66B, etc. encoding). Because there are just three bits to scan into each scan channel, the 2nd data byte must be padded to keep the data as two full bytes. Then two more bytes must be added to keep the scan payload a multiple of packet frames.

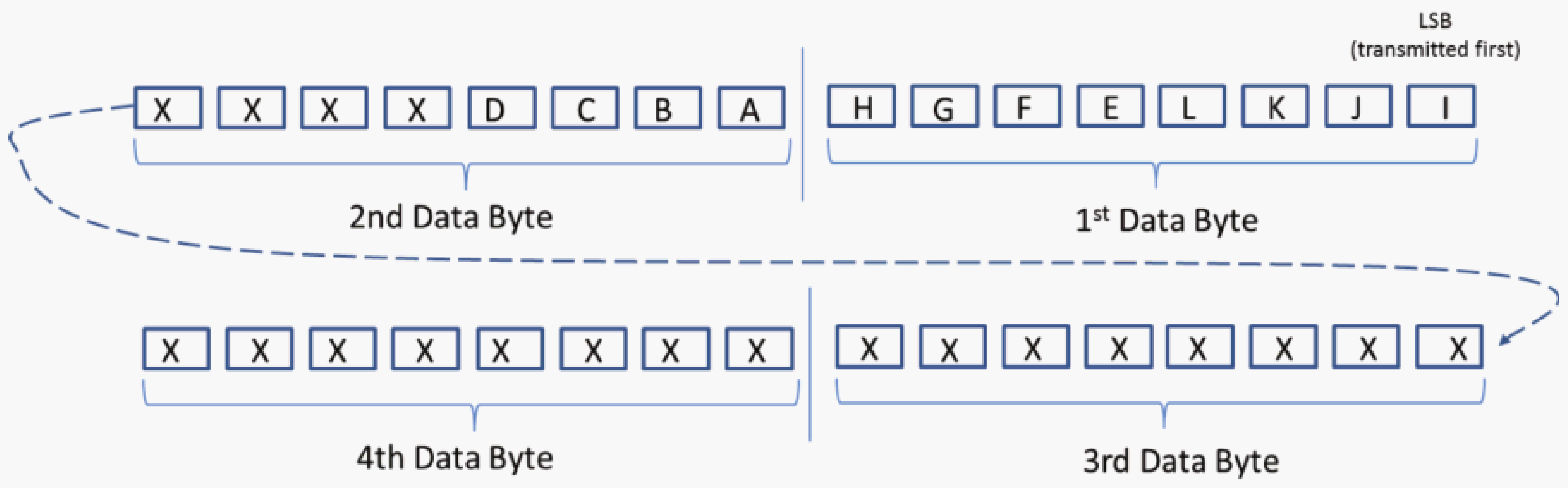

Figure 21—Formatted payload of packet

It may be desirable to implement a PEDDA that keeps the round-robin scan-channel sequence on nibble or byte boundaries. The bandwidth efficiency lost by transmitting bit positions with no meaningful data may be a reasonable tradeoff with PEDDA design simplification. This is typical when the INTERLEAVE_SIZE is an odd number, because SCAN data is expressed in packet frames. The PACKET_MAP BSDL attribute presented earlier indicated that no packing of the scan data should be done (PACK is set to FALSE). This construct provides documentation for an automated software program to create a SCAN packet such that the bit positions for each scan-channel in the SCAN_GROUP remain consistent and independently enabled. Figure 22 shows an example where only scan-channel 0, 2, and 3 are enabled. The position for scan-channel 1 is preserved, as the documentation indicates that there are four scan-channels in the group and the bits are not to be packed.

Copyright © 2017 IEEE. All rights reserved.

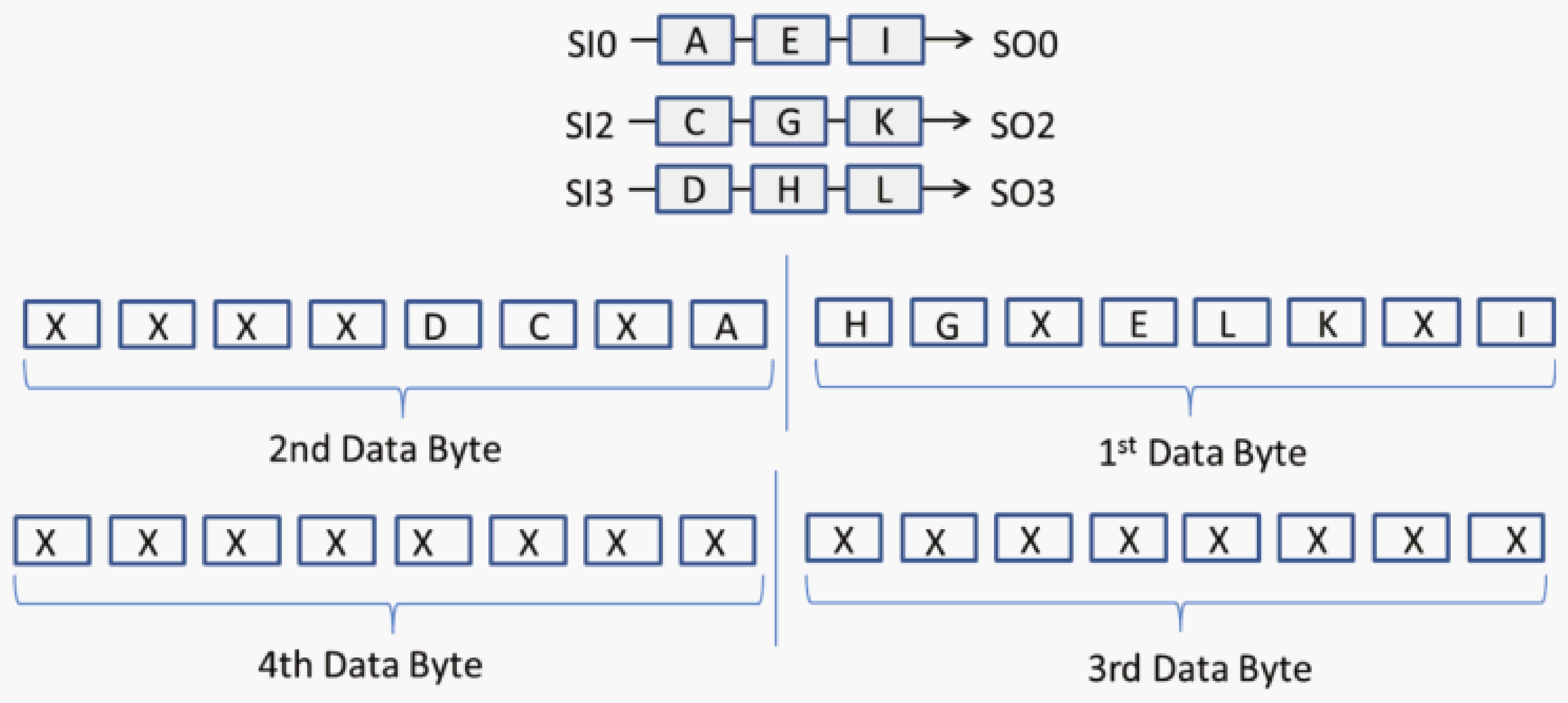

Figure 22—Payload with 3 scan-channels

6.7.2 Specifications

Rules (SCAN packet)

a) The SCAN command shall have a value of 0x06.
b) The SCAN command byte shall be followed by an ID byte.
c) The ID byte shall be followed by four bits of 0b0000 and four bits representing the ICSU of the packet.
d) The ICSU of a SCAN packet shall represent the intent to perform an Instruction, Capture, Shift, and Update operation on the scan-channels selected via a logic 1 in the respective bit position.

NOTE—An operation is not performed when the respective bit is zero.

e) There shall be a 32-bit #Payload-Frames value that:
i) represents the number of packet frames found in the PAYLOAD portion of the SCAN packet and
ii) follows the ICSU.
f) There shall be a 32-bit Cycle-Count value following the #Payload-Frames, which represents the number of bits to shift on the scan-channels selected.
g) There shall be PAYLOAD data with a size in packet frames equivalent to the value of #Payload–Frames immediately following the Cycle-Count.
h) When the MSB of the ICSU is logic 1, only one bit of the Channel-Select words shall be logic 1.

NOTE—When an instruction register is the target of a scan operation, it is done as a SCAN packet separate from SCAN packets that deliver data.

Rules (SCAN payload)

j) The Interleave size shall be the number of contiguous bits associated with a scan-channel in the PAYLOAD section of a SCAN packet and shall be greater than zero.
k) The data size of the PAYLOAD portion shall be an integer multiple of packet frames.

6.7.3 Examples

This example shows the concept of shifting in (and out) data on two scan-channels of different lengths at the same time when over-shifting of the shorter length scan-channel is not desired. Figure 23 shows an example set of scan-channels labeled Channel 0 through 3. Channel 0 is 32 bits long and Channel 3 is 33

Copyright © 2017 IEEE. All rights reserved.

bits long. The data that is desired to be present in the scan-channels is represented by the alphabet characters at the SI and SO. For brevity, not all of the positions are shown as filled. It is desired to scan both channel 0 and channel 3 at the same time.

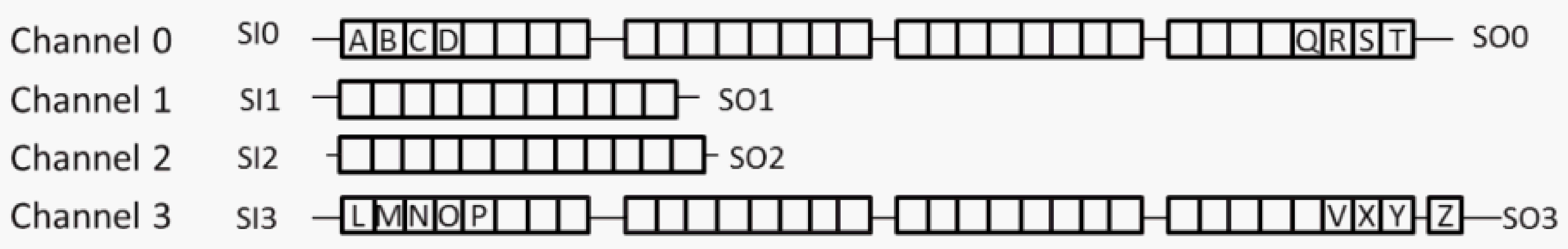

Figure 23—Four example scan-channels, with final data contents

Figure 24 shows the first packet as a CH-SELECT command packet. The Channel-Select is set to 0x0009 indicating both Channel 3 and Channel 0 will be used in the subsequent SCAN packet. The format of the SCAN packet to scan in (and scan out) data on Channel 0 and Channel 3 simultaneously is also shown in Figure 24. The start is indicated by the SOP character followed by the SCAN command of 0x06. This is followed by one 8-bit ID value set at 0x01, four bits of padding of 0b0000, and the ICSU set to 0b0110. The value of the ICSU indicates that both a Capture-DR operation and a Shift-DR operation are to be performed.

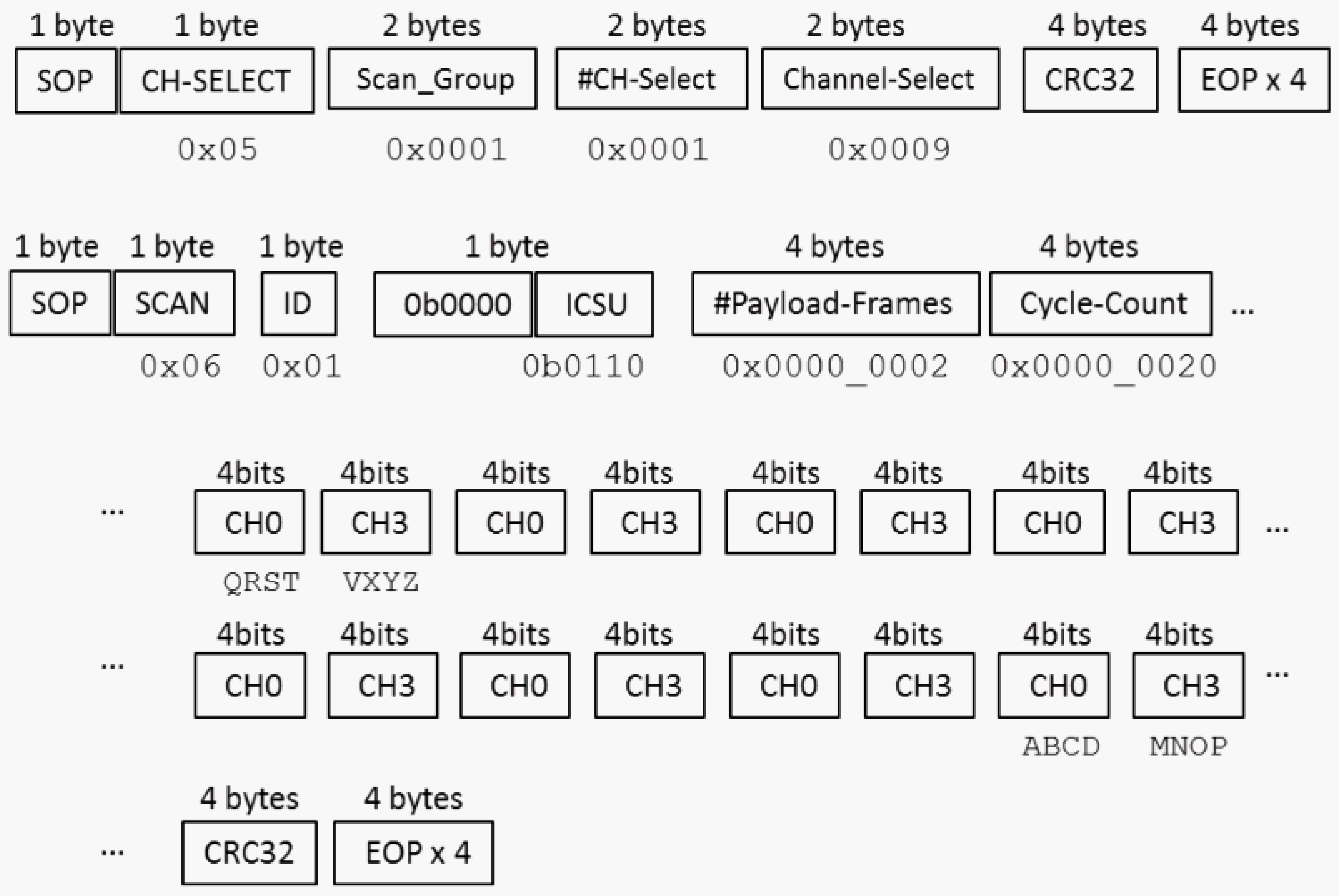

Figure 24—CH-SELECT, Capture-DR, and Shift-DR portion of a two channel 32-bit scan

The number of packet frames (#Payload-Frames) is a 32-bit value equal to 0x0000_0002 indicating eight data bytes follow the cycle-count (the portion of the packet that is variable length). The cycle count is a 32-bit value equal to 0x0000_0020 or decimal 32. This indicates that the number of scan clocks required for this SCAN packet is thirty-two. Figure 25 shows an example Packet_Map of the interface in BSDL. For a complete description of the Packet_Map, see 7.4. This Packet_Map specifies an Interleave_Size of four, which means that the scan data for each channel is broken up into 4-bit values.

```
attribute Packet_Map of MyChip : entity IS
"(PEDDA_NAME:              PEDDA1)," &
"(TX_Order        : LSB_FIRST),"&
"(CMD_Parity      : NONE_1149_10)," &
"(SCAN_DATA_SIZE       : 8), " &
"(PACK            : TRUE), " &
"(Interleave_Size: 4), " &
"(Scan_Group : 1)";
```

Figure 25—Example Packet_Map

After the Cycle-Count, the scan data follows with four bits of each channel interleaved for a total of eight nibbles for each channel, 16 nibbles in total. The first four bits for Channel 0 are sent first, the bits represented at positions QRST followed by a nibble for Channel 3 from the bit positions VXYZ. The bits to shift are interleaved, ending with bits for Channel 0 ABCD shifted as the next to last nibble and Channel 3 bits from position MNOP shifted last. The interleaved scan-in data is followed by a CRC32 of all of the bytes from the SCAN command until the CRC32, and then followed by four EOP symbols.

Because only 32-bits of shift occurred on Channel 3, it requires another bit of shift to shift in the bit at the "L" position. Figure 26 shows the two packets required. The CH-SELECT command uses the same SCAN_GROUP, 0x01, and the #CH-Select word is the same 0x01. The Channel-Select becomes 0x0008, indicating that just Channel 3 is active for the subsequent SCAN packet. The SCAN command is again 0x06 followed by an incremented ID of two (0b0010). ICSU is 0b0010, representing that only a shift function is occurring. The size of the payload is 1 packet frame (#Payload-Frames) because just one byte of data is required after the Cycle-Count. The Cycle-Count is 1 as there is just one bit to shift. The SCAN_DATA_SIZE specified in the Packet_Map indicates that it is eight bits.

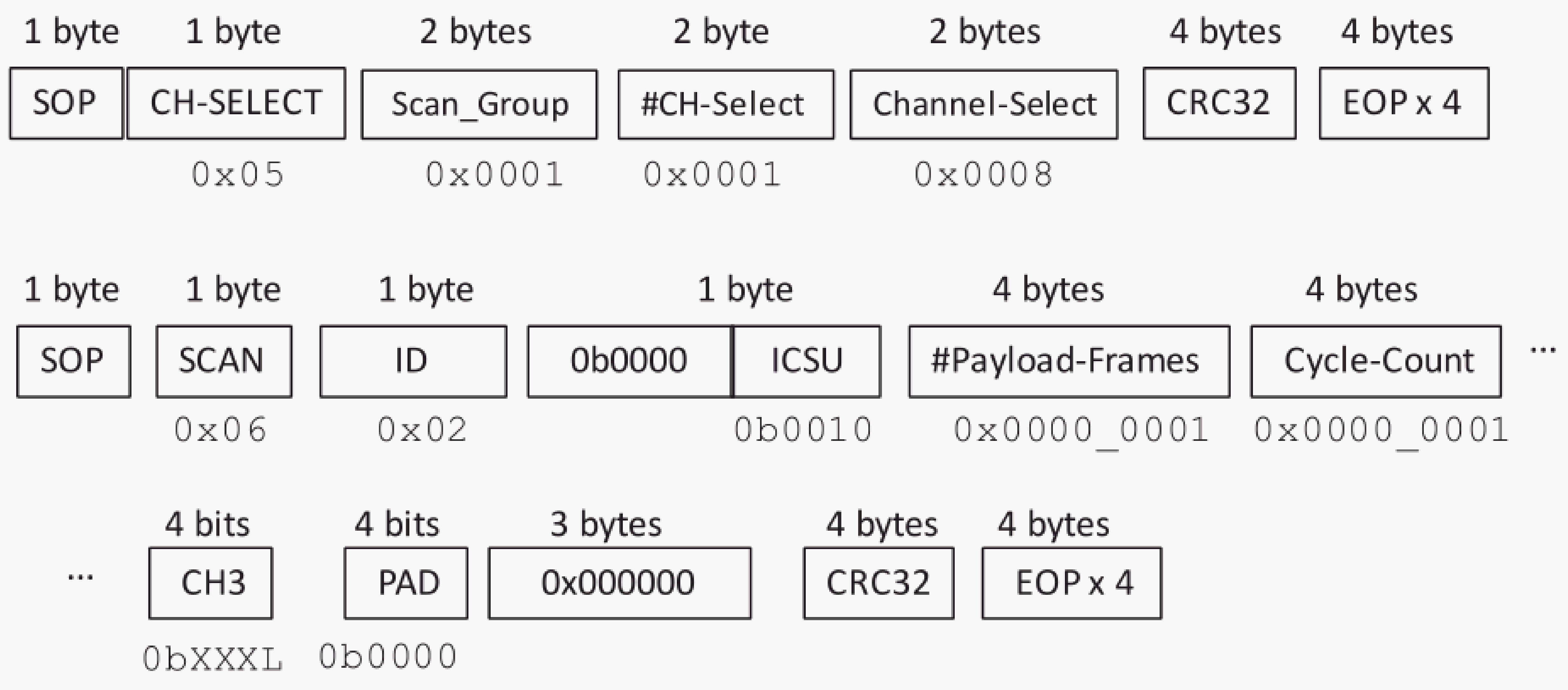

Figure 26—Packets needed to shift 33rd bit of Channel 3

Channel data immediately follows the Cycle-Count, followed by any padding data needed to meet the SCAN_DATA_SIZE. The bit in the position L is shown as the least significant bit of the Channel 3 nibble. The interleave is four bits, but there is just one Channel to shift so there are four bits of padding, four “don't

Copyright © 2017 IEEE. All rights reserved.

care" bits (shown as 0b000), which are sent to provide eight bits of data. To format the PAYLOAD to a multiple of a packet frame, three more pad bytes are added to complete the frame. The data is then followed by a 32-bit CRC32 word, followed by four EOP characters. If the scan channels do not have update registers, then the sequence is complete. However, if any of the register cells have an update stage, it will be necessary to also assert Update-DR for the channels.

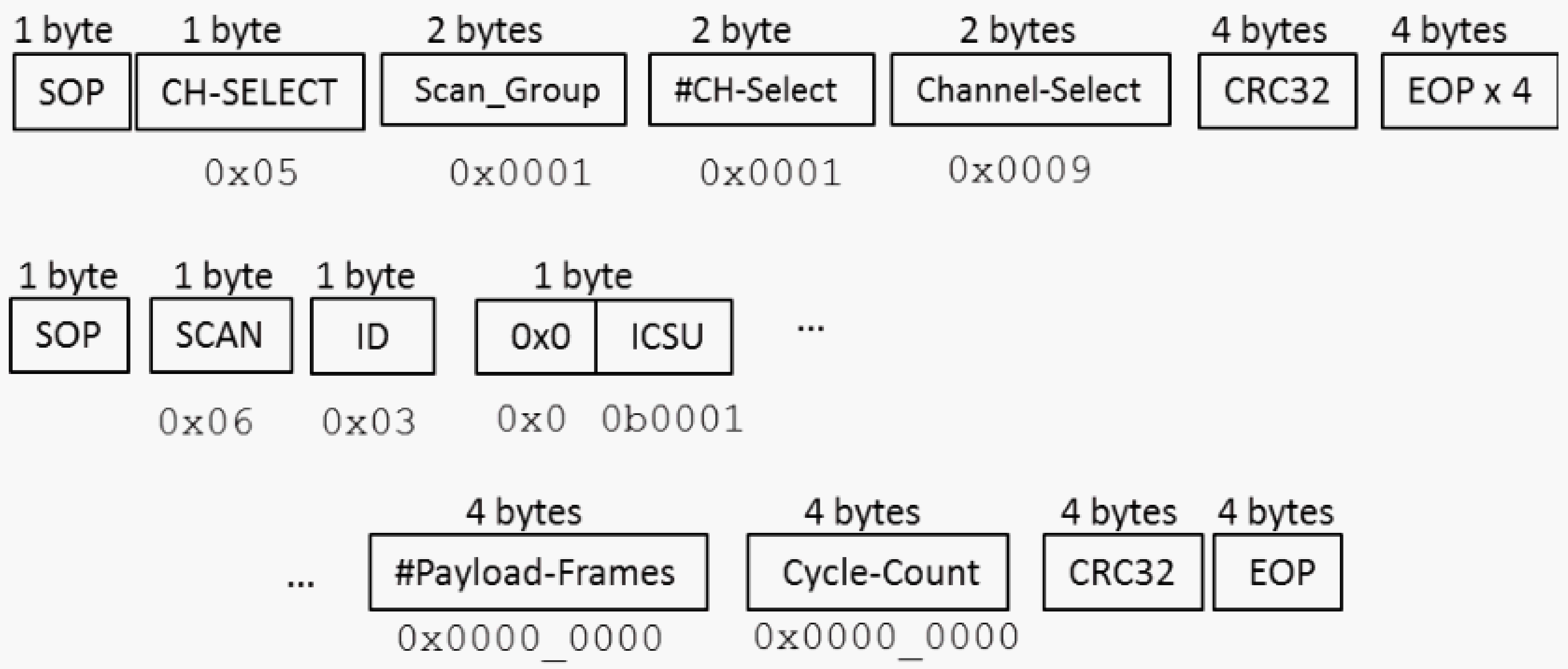

Figure 27 —Final SCAN packet to assert Update-DR

Figure 27 shows the CH-SELECT and SCAN packet to assert Update-DR on both channels. The CH-SELECT command packet uses a Scan_Group of 0x01, a #CH-Select of 0x01, and a Channel-Select word of 0x0009 (0b1001), indicating both Channel 0 and Channel 3 are going to be activated. The SCAN command (0x06) is followed by an incremented ID of three (0x03). This is followed by a nibble of 0b0000 and an ICSU of 0b0001, indicating that just an Update-DR is to be performed. The number of #Payload-Frames is zero because there is no new data and the Cycle-Count is also zero. This is followed by a CRC32 of the data from the SCAN command to the CRC32 followed by four EOP. After these six packets, the data will exist in Channels 0 and 3 as shown in Figure 23. Please note that when it is possible to over shift Channel 0, it is perfectly acceptable to format the data for Channel 0 by padding the first bit. This is just an example that shows a more complex shifting method.

6.7.4 Example using 64B/66B encoding

8B/10B is a byte-by-byte based line encoding developed in the early 1980s. To encode an IEEE 1149.10 packet, each 8-bit byte value is substituted into a 10-bit value. More recent encoding methods for SERDES use packet based encoding. That is, the fundamental architecture of a packet, having a "start", some data bytes or some control bytes, followed by an "End" or "Terminate", is built into the architecture of the encoding standard. The encoding is performed across sixty-four or more bits at a time using a set of pattern lookups, which are then scrambled to ensure enough randomness for clock recovery at the receiver. Table 6 shows the subset of 64B/66B encoding from IEEE Std 802.3 that is applicable when encoding IEEE 1149.10 packets. For clarity, Table 6 shows "Z" in the pattern specification. The "Z" shown in the pattern specification is an 8-bit control character and the "C" in the "Bits" specification is the 7-bit transmitted version. (IEEE Std 802.3 uses "C" as a designation for both the control characters to encode and the encoded control characters.)

IEEE 1149.10 does not use ordered sets as described in 64B/66B, hence those patterns would not be used when encoding IEEE 1149.10 packets. Due to the nature of the four-byte frames of IEEE 1149.10, the EOP (end of packet) can occur in just two positions, in the first byte or the fifth byte. The two matching patterns are shown in lines five and six. 64B/66B uses the "end of packet" notation "T" for terminate. Data is

Copyright © 2017 IEEE. All rights reserved.

transmitted LSB first, such that the data value of 0x1E as shown is transmitted as 01111000. While SOP and EOP are defined as control characters in this standard, they are not used in the Z_0 or Z_4 positions, for compatibility with 64B/66B encoders/decoders. It should be noted by the reader that based on the table shown, SOP is never actually transmitted, hence, its value as defined in BSDL is not critical to the implementation of the HSTAP or PEDDA. Similarly, lines 5 and 6 show the only possible format for an EOP. Hence, the decoder can rely on the D_0 value of 0x87 or 0xCC to know an EOP was received and its position. Encodings other than 8B/10B known to this standard all use a repetition of the 64B/66B table shown for the length the encoding standard specifies. For example, in128B/130B, the pattern to match is simply repeated for another sixty-four bits. Hence, D_0 through D_7 would simply repeat for bits 64:127. A line encoding standard may differ, however, by the number of sync bits as defined by that encoding.

Table 6—The subset of 64B/66B blocks used when encoding IEEE 1149.10 packets

	Pattern	Sync	Bits (0:63)								
1	$D_0D_1D_2D_3/D_4D_5D_6D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
2	$Z_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	10	0x1E "01111000"	C_0 0 - 6	C_1 0 - 6	C_2 0 - 6	C_3 0 - 6	C_4 0 - 6	C_5 0 - 6	C_6 0 - 6	C_7 0 - 6
3	$Z_0Z_1Z_2Z_3/S_4D_5D_6D_7$	10	0x33	C_0 0 - 6	C_1 0 - 6	C_2 0 - 6	C_3 0 - 6	0b0000	D_5	D_6	D_7
4	$S_0D_1D_2D_3/D_4D_5D_6D_7$	10	0x78	D_1	D_2	D_3	D_4	D_5		D_6	D_7
5	$T_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	10	0x87	0b0000000	C_1 0 - 6	C_2 0 - 6	C_3 0 - 6	C_4 0 - 6	C_5 0 - 6	C_6 0 - 6	C_7 0 - 6
6	$D_0D_1D_2D_3/T_4Z_5Z_6Z_7$	10	0xCC	D_0	D_1	D_2	D_3	0b000	C_5 0 - 6	C_6 0 - 6	C_7 0 - 6

Table 7 shows example 64B/66B encoding of four packets: a CONFIG packet, followed by a TARGET packet, followed by a CH-SELECT packet, and then followed by a SCAN packet. The SCAN packet matches the SCAN packet of Figure 24. The control characters used are defined below in the CONTROL_CHARS BSDL attribute. For ease of reading, the data in the table is shown unscrambled and normally would be scrambled according to the 64B/66B polynomial.

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR :    0xFB ), " &
"(EOP_CHAR  :   0xFD )," &
"(IDLE_CHAR :   0x07),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0x7C ),"&
"(XON_CHAR:     0x1C),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xF7)";
```

Line 1 of Table 7 shows the encoding for the CONFIG packet using 64B/66B. All IEEE 1149.10 packets start with an SOP and, hence, they contain both data and control characters. Because there is data and control in this transmission, 64B/66B requires the sync value to be “10”. The code 0x78 is selected because the first eight bytes in the CONFIG packet map to the $S_0D_1D_2D_3/D_4D_5D_6D_7$ (see Table 6) pattern specified in 64B/66B. It directly corresponds to the SOP in the first position followed by seven data bytes. The transmission then is two bits “10”, followed by 0x78, followed by the seven bytes shown in line 1. Because all of the data required for the CONFIG packet is sent on line 1, the remaining bytes of the CONFIG packet are the four EOP characters. Anytime four EOP characters are found at the start of a 64B/66B block, it matches the pattern in Table 6 line 5. T_0 in 64B/66B terminate packet is equivalent to EOP. Line 2 of Table 7 encodes the four EOP characters required in the packet format as four EOPs followed by four IDLE characters. Line 2 uses code 0x87, followed a 0b0000000, followed by eight 7-bit control character values. The 64B/66B block needs to be padded out with IDLE characters as it is the only encoding pattern that could be used to match this IEEE 1149.10 packet frame.

Copyright © 2017 IEEE. All rights reserved.

In lines 3 and 4, the TARGET packet is similarly encoded like the CONFIG packet. The CH-SELECT packet is encoded at lines 5 and 6. Line 6 uses the 0xCC code from Table 6 for this CH-SELECT packet. The IEEE 1149.10 CH-SELECT packet ends as four data bytes followed by an EOP. This matches the $D_0D_1D_2D_3/T_4Z_5Z_6Z_7$ pattern in 64B/66B. The first EOP is equivalent to the terminate character, T_4; the remaining three EOPs matching the three "Z" control characters specified in the encoding. The SCAN packet is transmitted on lines 7-10. Lines 8 and 9 use the sync value of "01" because they match the $D_0D_1D_2D_3/D_4D_5D_6D_7$ pattern in Table 6. These two 64B/66B blocks are 64-bits each of pure data without control characters.

Table 7—Example 64B/66B encoding of IEEE 1149.10 packets[6]

	Sync	Bits (0:63)								
1	10	0x78	0x01 CONFIG	0x01 TID	0x00 TID	0x64 CRC	0x82 CRC	0x98 CRC	0xE7 CRC	
2	10	0x87	0b0000000	0x7D EOP	0x7D EOP	0x7D EOP	0x07 IDLE	0x07 IDLE	0x07 IDLE	0x07 IDLE
3	10	0x78	0x02 TARGET	0x01 TID	0x00 TID	0x3D CRC	0x3C CRC	0xDE CRC	0xE5 CRC	
4	10	0x87	0b0000000	0x7D EOP	0x7D EOP	0x7D EOP	0x07 IDLE	0x07 IDLE	0x07 IDLE	0x07 IDLE
5	10	0x78	0x05 CH- SELECT	0x01 SG0	0x00 SG1	0x01 #CHS0	0x00 #CHS1	0x09 CH-S	0x00 CH-S	
6	10	0xCC	0x50 CRC0	0xFF CRC1	0x76 CRC2	0x6D CRC3	0b000	0x7D EOP	0x7D EOP	0x7D EOP
7	10	0x78	0x06 SCAN	0x01 ID	0x06 ICSU	0x02 #PF0	0x00 #PF1	0x00 #PF2	0x00 #PF3	
8	01	0x20 Cycle-Count0		0x00 CC1	0x00 CC2	0x00 CC3	0x01 Data	0x02 Data	0x03 Data	0x04 Data
9	01	0x05 Data		0x06 Data	0x07 Data	0x08 Data	0xA2 CRC0	0x6A CRC1	0xCB CRC2	0xCA CRC3
10	10	0x87	0b0000000	0x7D EOP	0x7D EOP	0x7D EOP	0x07 IDLE	0x07 IDLE	0x07 IDLE	0x07 IDLE

Acronym Legend	
TID: TARGET_ID	CC2 : Cycle Count Byte 2
SG0: SCAN GROUP Byte 0	CC3 : Cycle Count Byte 3
SG1: SCAN GROUP Byte 16	Data : Scan Data to be shifted in
#CHS0 : #Channel_Select Byte 0	#PF0 : #Payload Frames Byte 0
#CHS1 : #Channel_Select Byte 1	#PF1 : #Payload Frames Byte 1
CH-S: Channel Select values	#PF2 : #Payload Frames Byte 2
CC1: Cycle Count Byte 1	#PF3 : #Payload Frames Byte 3

6.8 The BOND packet

6.8.1 Description

The BOND packet is used to enter a multi-lane channel bonded mode. Bonding of multiple HSTAPs enables higher bandwidth to and from the target than a single lane (see Figure 28). BOND packets are sent

[6] 64B/66B scrambles the sixty-four bits, hence the table shown is not the binary stream that is transmitted.

Copyright © 2017 IEEE. All rights reserved.

over the first lane and signal to the interface the number of lanes that are going to be used during channel bonding.

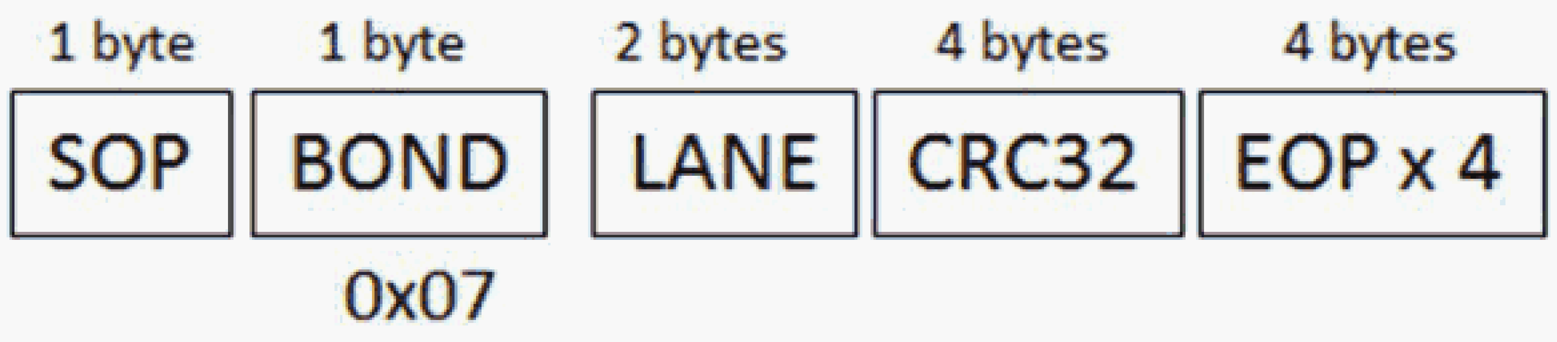

Figure 28—Format of a BOND packet

6.8.2 Specifications

Rules

a) The BOND command shall have a value of 0x07.
b) A 16-bit LANE value shall follow the command byte.
c) The LANE shall indicate a value greater than one.
d) After receipt of the BOND command, the operation of the HSTAP and PEDDA shall be as described in 8.1.2.

6.8.3 Examples

Figure 29 shows an example packet format for bonding four lanes. The SOP and EOP are not specified in the example.

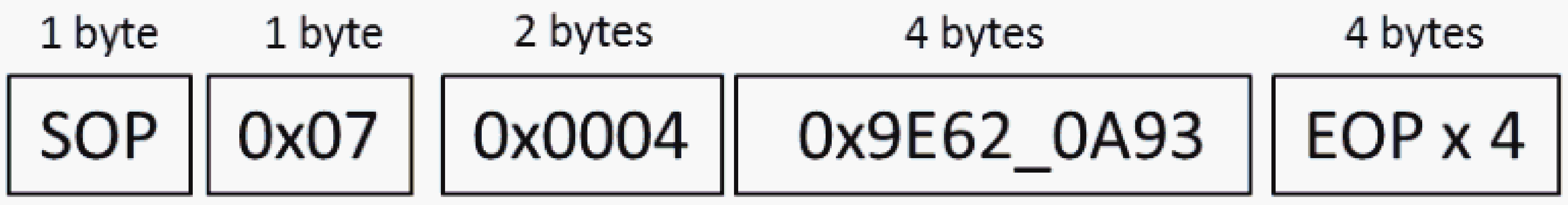

Figure 29—Example BOND packet for four lanes

Please see Clause 8 on Channel bonding for more details on how channel bonding works.

6.9 The CONFIGR packet

The CONFIGR packet is the response to the initialization CONFIG packet. The TARGET_ID value in the response is always greater than zero. The CONFIGR response confirms to the ATE that the CONFIG packet was received and which TARGET_ID was specified.

6.9.1 Description

The CONFIGR packet uses a command (CMD) value of 0x81 followed by a TARGET_ID to identify the IEEE 1149.10 interface by the number that was assigned with the CONFIG packet (see Figure 30). When a daisy-chain of IEEE 1149.10 interfaces exists, each interface would be assigned a unique TARGET_ID within the daisy-chain. The first interface in the chain would be assigned a TARGET_ID first, and then, in turn, each interface along the chain would be assigned a different TARGET_ID.

Copyright © 2017 IEEE. All rights reserved.

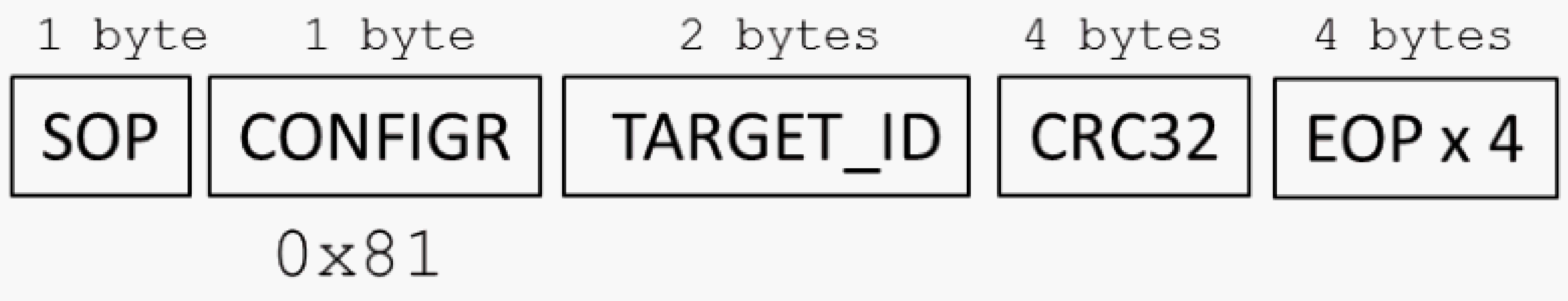

Figure 30—Format of the CONFIGR packet

The CONFIGR packet is a response to the initialization process to set the TARGET_ID of each IEEE 1149.10 interface.

6.9.2 Specifications

Rules

a) The CONFIGR command shall have a value of 0x81.
b) A 16-bit TARGET_ID response shall follow the CONFIGR command in the packet.
c) The TARGET_ID response of this packet shall be 16 bits and have a value greater than zero.
d) The TARGET_ID value in the CONFIGR response shall contain the value of the TARGET_ID register.

6.9.3 Examples

Given an attribute CONTROL_CHARS as follows:

```
attribute CONTROL_CHARS  of MyChip : entity IS
   "(SOP_CHAR:    0xFB ), " &
   "(EOP_CHAR  :   0xFD )," &
   "(IDLE_CHAR :   0xBC),"&
   "(ERROR_CHAR :  0xFE ),"&
   "(XOFF_CHAR :   0x7C ),"&
   "(XON_CHAR:     0x1C),"&
   "(CLEAR_CHAR :  0x5C),"&
   "(COMPLIANCE_CHAR: 0xDC)";
```

The SOP is 0xFB and the EOP is 0xFD. The data transmitted to the ATE is as follows (assuming there is no encoding):

0xFB followed by 0x81, followed by 0x0001 (sent with LSB first as 0x01 then 0x00), followed by a CRC32 of 0x06AD_99E4 calculated on the 0x81, 0x01, and 0x00, followed by four closing EOP with a value of 0xFDFD_FDFD.

6.10 The TARGETR packet

6.10.1 Description

The TARGETR packet is used to respond to an ATE when a TARGET packet is received (see Figure 31). The TARGETR response confirms that the IEEE 1149.10 design received the TARGET command.

Copyright © 2017 IEEE. All rights reserved.

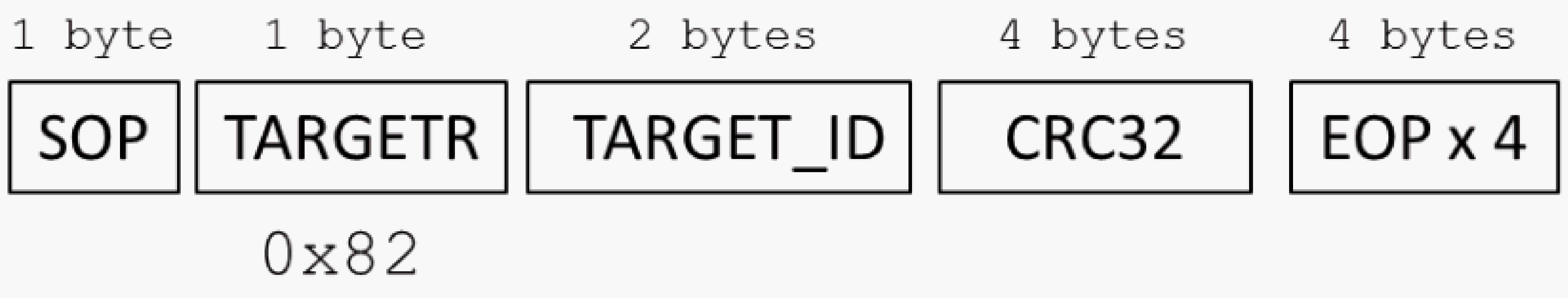

Figure 31—Format of TARGETR packet

6.10.2 Specifications

Rules

a) The TARGETR command shall have a value of 0x82.
b) A 16-bit TARGET_ID response shall follow the TARGETR value in the packet.
c) The 16-bit TARGET_ID shall have a value greater than zero.
d) The TARGET_ID value in the response shall contain the value of the TARGET_ID register.

6.10.3 Examples

Given an attribute CONTROL_CHARS as follows:

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR:    0xFB ), " &
"(EOP_CHAR  :   0xFD )," &
"(IDLE_CHAR :   0xBC),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0x7C ),"&
"(XON_CHAR:     0x1C),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

The SOP is 0xFB and the EOP is 0xFD. The data transmitted to the ATE is as follows (assuming there is no encoding):

0xFB followed by 0x82, followed by 0x0001 (sent with LSB first as 0x01 then 0x00), followed by a CRC32 of 0x04EB_27BD, followed by four closing EOP with a value of 0xFDFD_FDFD.

6.11 The RESETR packet

6.11.1 Description

The RESETR packet is used to validate that the RESET packet has been received. The RESETR packet consists of the RESETR command of 0x83, a TYPE field of 16 bits, followed by a CRC32 and four EOP (see Figure 32).

Copyright © 2017 IEEE. All rights reserved.

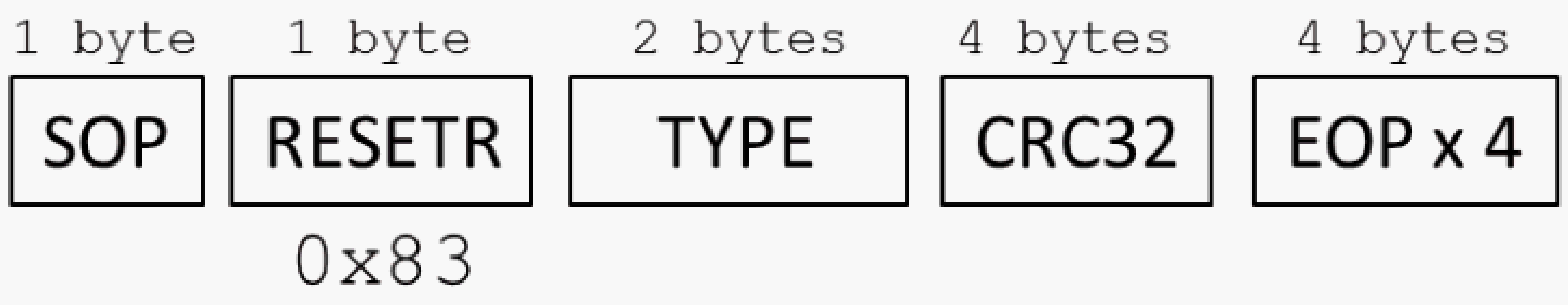

Figure 32—Format of the RESETR packet

6.11.2 Specifications

Rules

a) The RESETR command shall have a value of 0x83.
b) A 16-bit TYPE value shall follow the RESETR command representing the TYPE value received in the last RESET packet.

NOTE—The TYPE value is the same value as what was received in the last RESET packet.

6.11.3 Examples

Given the following CONTROL_CHARS attribute:

```
attribute CONTROL_CHARS of MyChip : entity IS
"(SOP_CHAR :    0x7C ), " &
"(EOP_CHAR  :   0x1C )," &
"(IDLE_CHAR :   0xBC),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0xFB ),"&
"(XON_CHAR:     0xFD),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

If a TYPE field was 0x0001, the bytes transmitted to the ATE would be 0x7C, followed by 0x83, followed by 0x0001, followed by a CRC32 of 0x0529_4D8A, followed by four EOP of 0x1C1C_1C1C.

6.12 The RAWR packet

6.12.1 Description

The RAWR packet is used to validate that the RAW packet has been received. The RAWR packet consists of the RAWR command of 0x84, followed by a 16-bit padding field of all zeros, followed by a CRC32 and four EOP (see Figure 33).

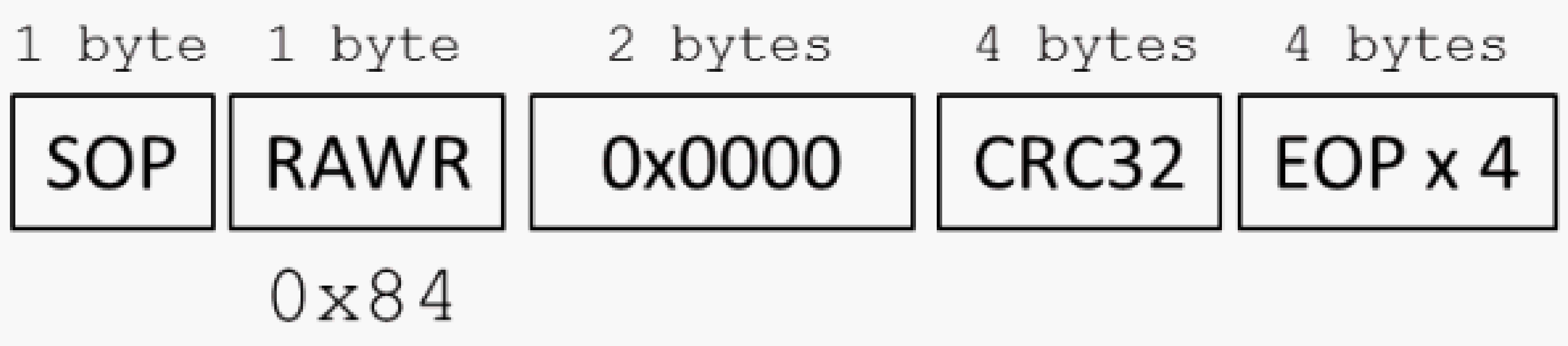

Figure 33—Format of RAWR packet

6.12.2 Specifications

Rules

a) The RAWR command shall have a value of 0x84.
b) A 16-bit padding field of all zeros shall follow the RAWR command.

6.12.3 Examples

Given the following CONTROL_CHARS attribute:

```
attribute CONTROL_CHARS  of MyChip : entity IS
"(SOP_CHAR :    0x7C ), " &
"(EOP_CHAR  :   0x1C )," &
"(IDLE_CHAR :   0xBC),"&
"(ERROR_CHAR :  0xFE ),"&
"(XOFF_CHAR :   0xFB ),"&
"(XON_CHAR:     0xFD),"&
"(CLEAR_CHAR :  0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

The bytes transmitted to the ATE would be 0x7C, followed by 0x84, followed by 16 bits of 0x0000, followed by a CRC32 of 0x197D_6A4E, followed by four EOP with a value of 0x1C1C_1C1C.

6.13 The CH-SELECTR packet

6.13.1 Description

The response to a CH-SELECT packet is a CH-SELECTR packet. The format for the CH-SELECTR packet is shown in Figure 34. The CH-SELECTR command is followed by two bytes of the SCAN_GROUP being used by the PEDDA. This is followed by two bytes of #Ch-Select indicating how many words of Channel-Selects are present. Channel-Selects are followed by padding when necessary to create an integer number of packet frames prior to the CRC32.

Copyright © 2017 IEEE. All rights reserved.

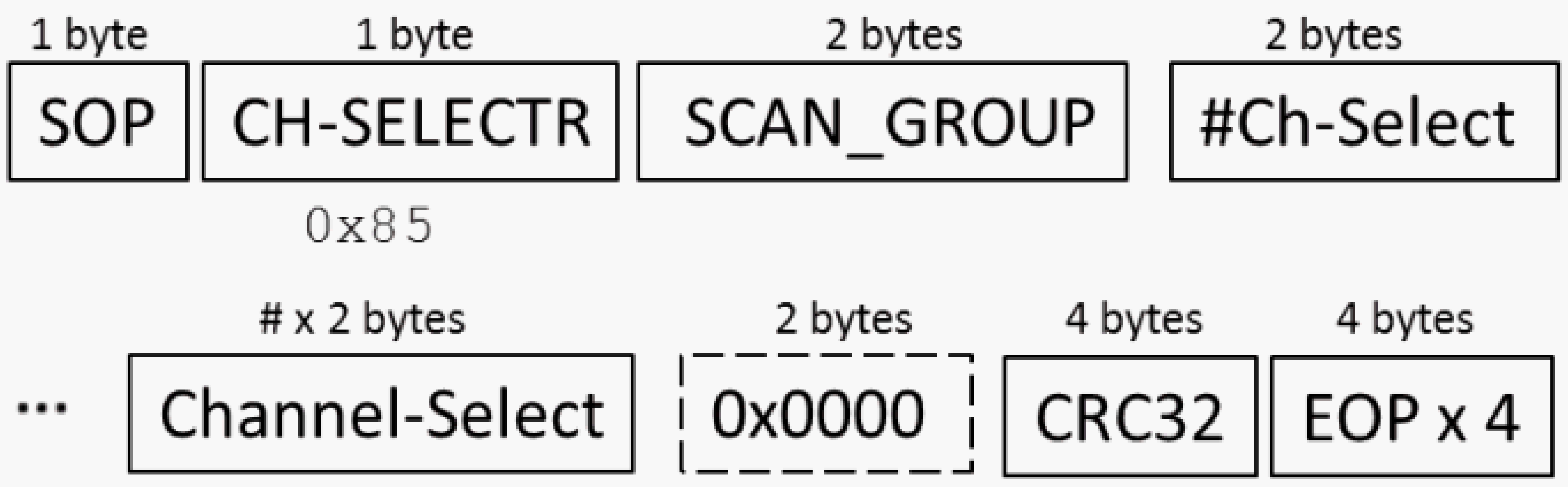

Figure 34—Format of CH-SELECTR packet

Specifications

a) The CH-SELECTR command shall have a value of 0x85.
b) A 16-bit SCAN_GROUP value shall follow the CH-SELECTR command value.
c) The SCAN_GROUP value shall be followed by a 2-byte #Ch-Select value representing the number of Channel-Select words provided in this packet.
d) Following the #Ch-Select value, there shall be a number of Channel-Select words equal to the #Ch-Select value and each Channel-Select word shall be 16 bits.
e) Each bit of the Channel-Select words in the packet shall represent the scan-channel number counted from LSB to MSB.

NOTE—This is a description of packet bit ordering and not representative of what may be transmitted at the physical layer. Bits are counted in the word starting on the right with Channel zero for the LSB.

f) Following the Channel-Select words, there shall be the number of bytes with value of 0x00 equal to:

$$2 * ((\#\text{Ch-Select} -1) \bmod 2)$$

6.13.2 Examples

The response to the CH-SELECT packet sent in Figure 24 is shown as the CH-SELECTR response packet in Figure 35.

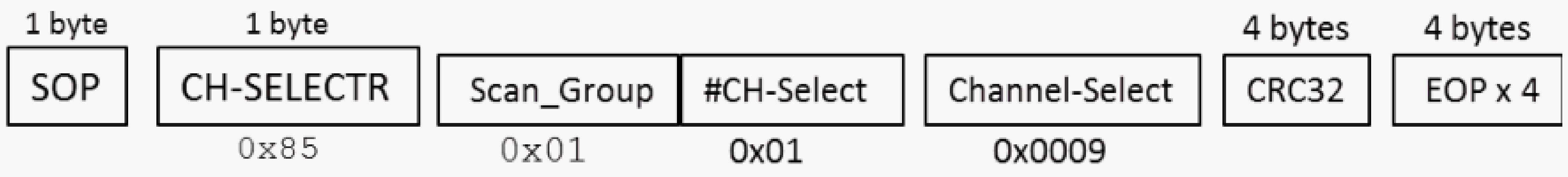

Figure 35—CH-SELECTR Packet example

6.14 The SCANR packet

6.14.1 Description

The SCANR packet is the response to the SCAN packet. Figure 36 shows the generic format of the SCANR packet. This format follows the SCAN packet in 6.6.

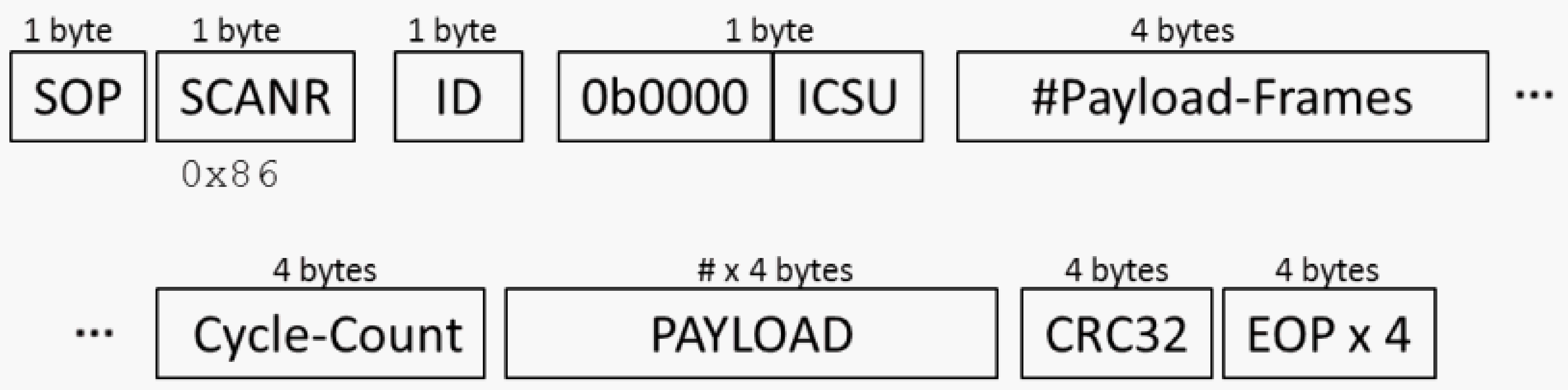

Figure 36—Generic SCANR packet format

6.14.2 Specifications

Rules

a) The SCANR command shall have a value of 0x86.
b) The SCANR command byte shall be followed by an 8-bit ID equivalent to the ID of the last received SCAN packet.
c) The 8-bit ID shall be followed by a 4-bit padding of value 0b0000.
d) The four bits of padding shall be followed by a 4-bit ICSU.
e) The ICSU of the SCANR packet shall be equivalent to the ICSU of the last received SCAN packet.
f) Following the ICSU, there shall be a 32-bit #Payload-Frames value, which represents the number of packet frames found in the PAYLOAD portion of the SCANR packet.
g) There shall be a 32-bit Cycle-Count value following the #Payload-Frames, which represents the number of bits that were shifted on the selected scan-channels.
h) There shall be PAYLOAD data with a size in 4-byte frames equivalent to the value of #Payload-Frames immediately following the Cycle-Count.
i) The PAYLOAD data shall be the formatted scan out data of the scan-channels.

NOTE—See rule d) ii of 5.1.2.

Rules (SCANR PAYLOAD)

j) The Interleave size shall be the number of contiguous bits associated with a scan-channel in the PAYLOAD section of a SCANR packet and shall be greater than zero.
k) The data size of the PAYLOAD portion shall be an integer multiple of four-byte packet frames.

6.14.3 Examples

Subclause 6.7.3 includes an example using four scan-channels with two scan-channels targeted in the SCAN packet. The four scan-channels are repeated here in Figure 37 showing the content of the scan-channels as a result of entering the CAPTURE-DR state. The figure is used to illustrate how the SCANR packet returns data from these scan-channels. Letters and numerals represent the bits in the position of two of the four scan-channels. A numeral used here is just a symbol like a letter symbol for the binary digit position. Scan-channel 0 is 32 bits long and scan-channel 3 is 33 bits long.

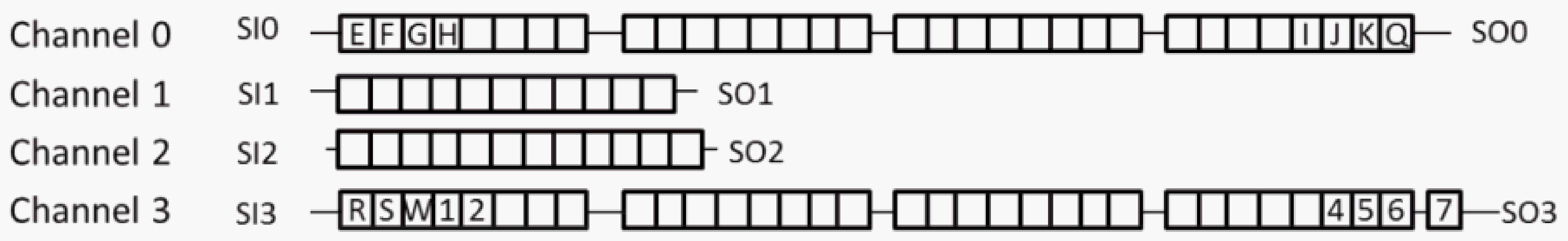

Figure 37—Four example scan-channels with data captured

In 6.7.3, the first CH-SELECT and SCAN packet sent is illustrated below in Figure 38.

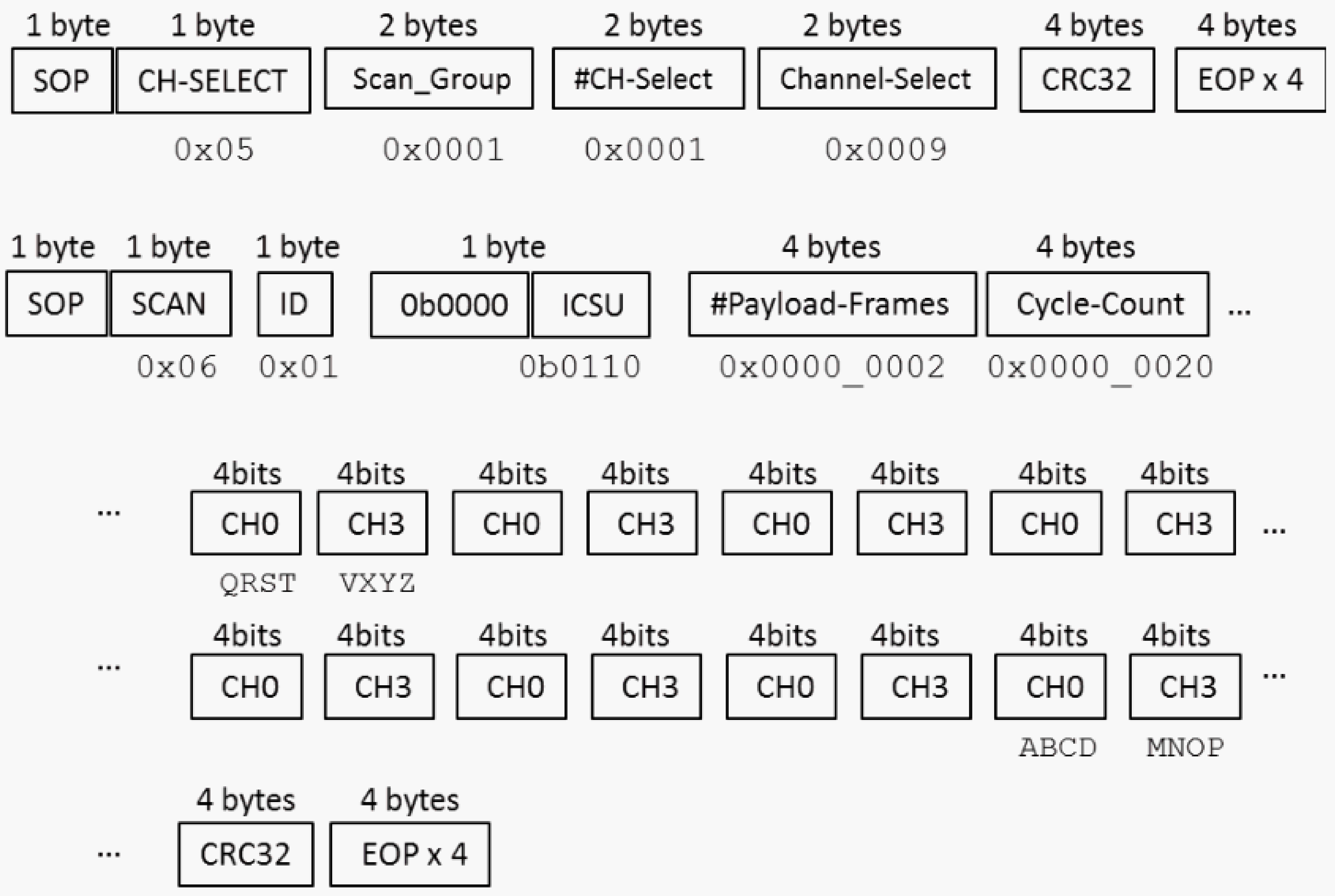

Figure 38—First SCAN packet sent

Copyright © 2017 IEEE. All rights reserved.

The response to the SCAN packet is the SCANR packet of Figure 39.

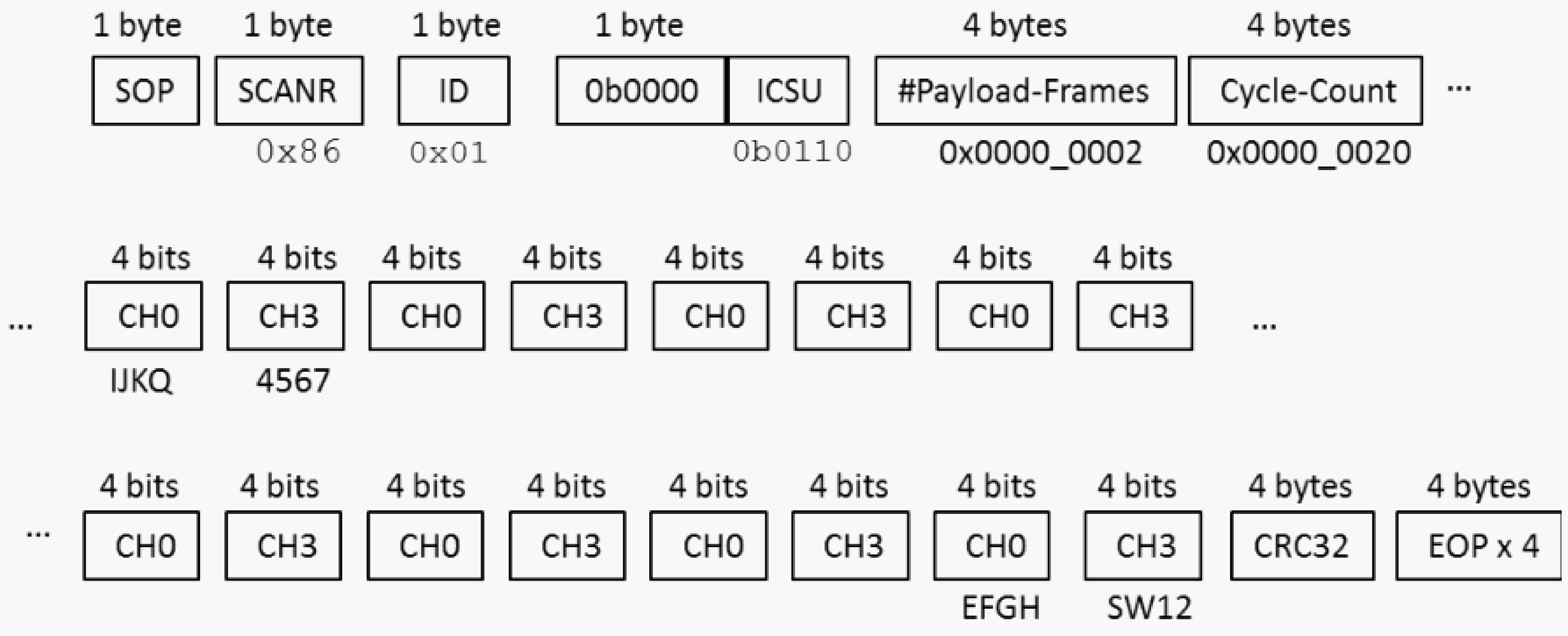

Figure 39—SCANR response packet example

The first nibble sent after the Cycle-Count consists of the bits IJKQ from Channel 0 followed by the 4567 nibble of Channel 3. This continues with the nibble interleaved between Channel 0 and Channel 3. The last two nibbles are EFGH for Channel 0 and SW12 for Channel 3. Note that the "R" captured in the first bit of Channel 3 is not sent on this SCANR as the length shifted was just 32-bits (0x20). The last bit, R, is shifted out on the next SCANR packet in response to the SCAN packet of one bit for Channel 3. The packet from 6.7.3 is repeated here in Figure 40. The "L" bit is shifted in via the SCAN packet and a padding nibble (shown as 0b0000) is sent to maintain the 8-bit SCAN_DATA_SIZE. "X" values shown on SCAN packets are "don't care" values. The PEDDA does nothing with the PAD and CH3 bits in the "X" positions as they are unused. A binary 1 or 0 may be used in any of those positions, only affecting the calculated CRC32 value. The "L" bit is the last bit to shift in, but the SCAN_DATA_SIZE requires a full eight bits to be transmitted.

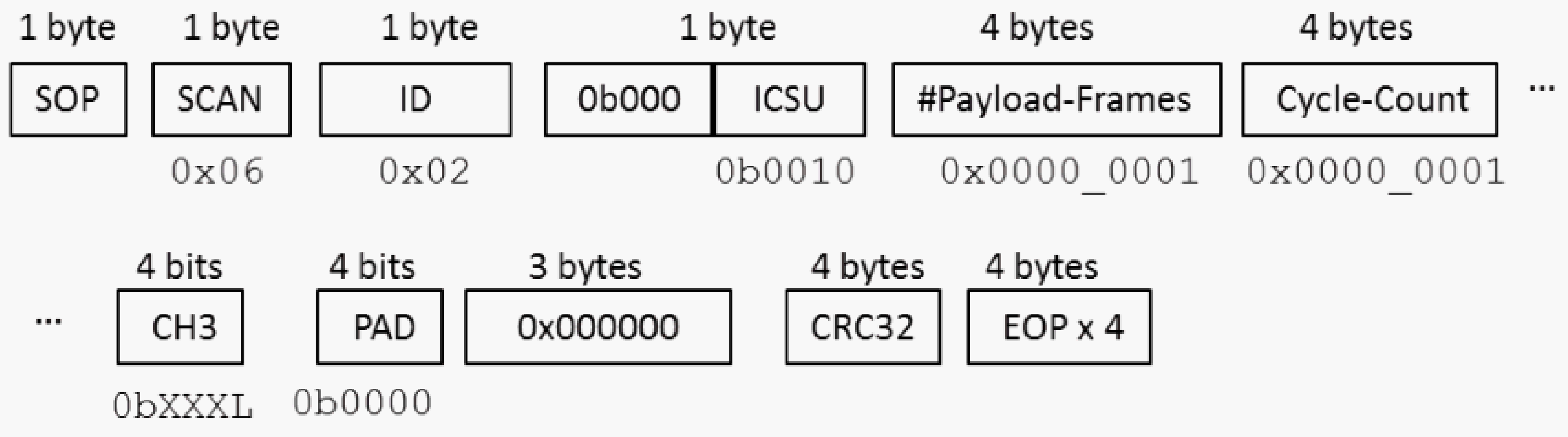

Figure 40—Example SCAN packet for Channel 3

The response with the captured "R" value from scan-channel 3 is shown in Figure 41. The nibble is required to be padded to keep the SCAN_DATA_SIZE at eight bits. "X" values shown in the SCANR packets are "don't care" values from the perspective of the ATE. The nibble at CH3 and PAD will return actual 1s and 0s to the ATE, but because R is the last bit that has real response data, the other bit positions are “don't care”. Ones or zeros may be filled in the SCANR packet but they have no effect on the response other than affecting the CRC32 calculated value.

Copyright © 2017 IEEE. All rights reserved.

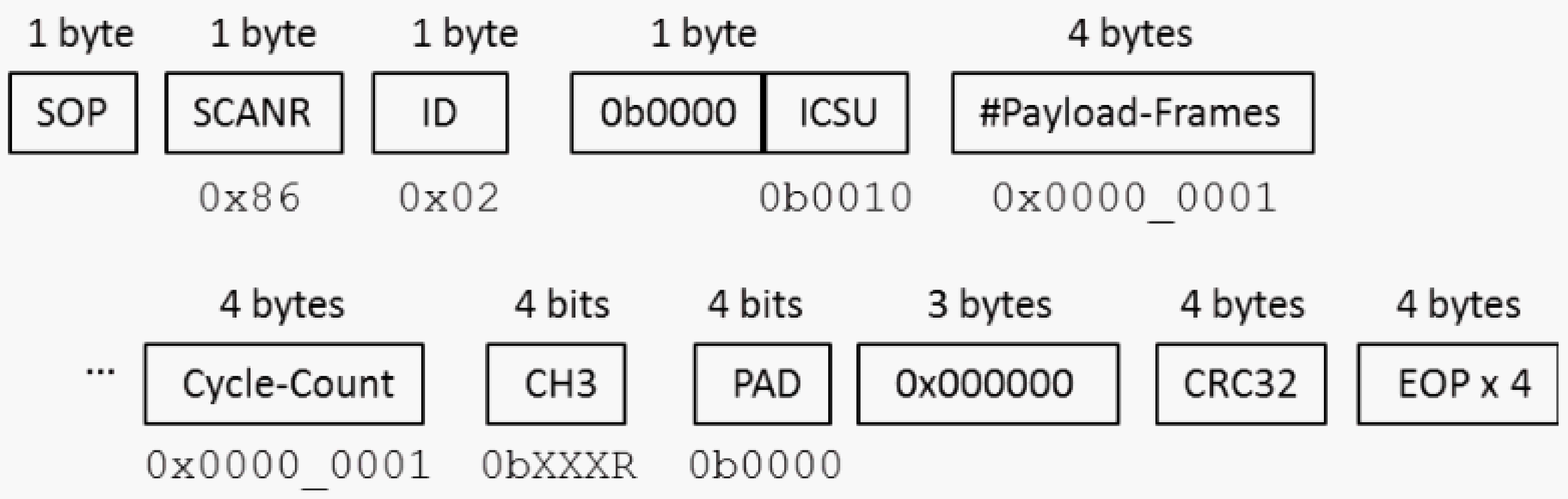

Figure 41—SCANR response to return last bit of channel 3

6.15 The BONDR packet

6.15.1 Description

The BONDR packet is used to validate that the BOND packet has been received. The BONDR packet consists of the BONDR command of 0x87, followed by a 16-bit LANE field, followed by a CRC32 and four EOPs (see Figure 42).

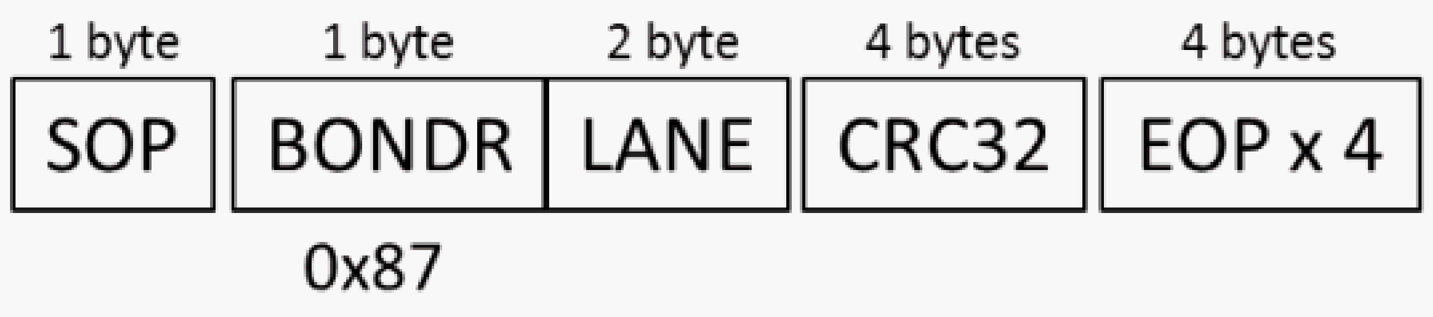

Figure 42—Format of BONDR packet

6.15.2 Specifications

Rules

a) The BONDR command shall have a value of 0x87.
b) A 16-bit LANE value shall follow the BONDR command.
c) A 32-bit CRC32 value shall follow the LANE value.
d) The LANE value shall be equivalent to the LANE value received in the last BOND command packet unless channel-bonding is not supported and in which case the LANE value transmitted in a BONDR packet shall always be zero.

Examples

The response to a BOND packet bonding four lanes is shown in Figure 43 below. The SOP and EOP values are not shown.

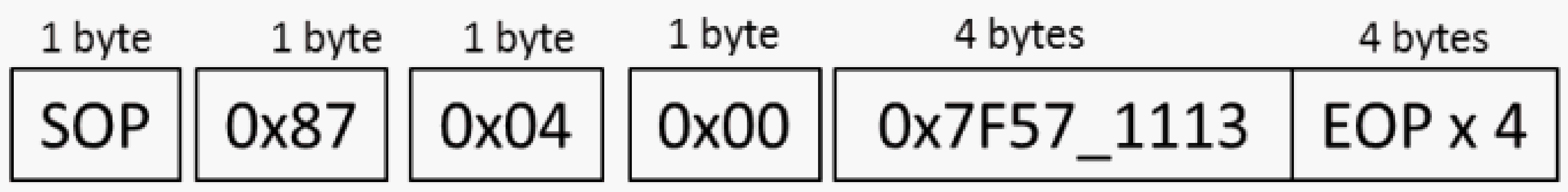

Figure 43—BONDR response packet

Copyright © 2017 IEEE. All rights reserved.

7. BSDL definitions

7.1 BSDL overview

This standard uses IEEE 1149.1 BSDL for its documentation language. Additional BSDL attributes are provided for documenting the features that are unique to IEEE Std 1149.10-2017. These BSDL attributes are considered "BSDL Extensions". The file format for IEEE 1149.10 BSDL is as specified in B.8.1.1 of IEEE Std 1149.1-2013 except where noted in 7.1.1 and 7.3.2.

7.1.1 Description

Except where noted, the conventions of IEEE Std 1149.1-2013 BSDL are used for IEEE Std 1149.10-2017. BSDL syntax common to IEEE Std 1149.1-2013 is shown for reader convenience. Tokens with spaces in the name from IEEE 1149.1 are preserved for compatibility, however, all IEEE 1149.10 grammar tokens use underline as a word connector, which is compatible with BNF and compilers such as lex and yacc.

Table 8—BSDL syntax from IEEE Std 1149.1

B.8.1	Overall syntax of the entity description
B.8.1.1	Specifications Syntax
<BSDL description>::= entity <component name> is	
<generic parameter>	(see B.8.2)
<logical port description>	(see B.8.3)
<standard use statement>	(see B.8.4)
{<use statement>}	(see B.8.5)
<component conformance statement>	(see B.8.6)
<device package pin mappings>	(see B.8.7)
[<grouped port identification>]	(see B.8.8)
<scan port identification>	(see B.8.9)
[<compliance-enable description>]	(see B.8.10)
<instruction register description>	(see B.8.11)
[<optional register description>]	(see B.8.12)
[<register access description>]	(see B.8.13)
<boundary-scan register description	(see B.8.14)
[<runbist description>]	(see B.8.15)
[<intest description>]	(see B.8.16)
[<system clock description>]	(see B.8.17)
{<register mnemonics description>}	(see B.8.18)
{<register fields description>}	(see B.8.19)
{<register assembly description>}	(see B.8.21)
{<register constraints description>}	(see B.8.22)
{<register association description>}	(see B.8.23)
{<power port association description>}	(see B.8.23)
{<BSDL extensions>}	(see B.8.24)
[<design warning>]	(see B.8.25)
end <component name> <semicolon>	

The BSDL syntax from IEEE Std 1149.1 is included in Table 8 for reference. The full BSDL grammar is used when the IEEE 1149.1 TAP is present. When a TAP is not used to enable compliance, then a reduced form of the BSDL grammar is specified, as shown in Table 9. The attributes defined in this standard are provided in the <BSDL extensions> portion of the BSDL. The token 1149_1_TAP_IS_NOT_PRESENT is described by the grammar element <1149_1_TAP_IS_NOT_PRESENT>. This token is introduced in this standard to indicate in the BSDL when the attributes for describing a TAP are not required.

Copyright © 2017 IEEE. All rights reserved.

Table 9—BSDL syntax for IEEE 1149.10 without IEEE Std 1149.1 compliance

<BSDL description>::= entity <component name> is	
<generic parameter>	(see IEEE Std 1149.1-2013 B.8.2)
<logical port description>	(see IEEE Std 1149.1-2013 B.8.3)
<standard use statement>	(see IEEE Std 1149.1-2013 B.8.4)
{<use statement>}	(see IEEE Std 1149.1-2013 B.8.5)
<1149_1_TAP_IS_NOT_PRESENT >	
<device package pin mappings>	(see IEEE Std 1149.1-2013 B.8.7)
[<grouped port identification>]	(see IEEE Std 1149.1-2013 B.8.8)
[<system clock description>]	(see IEEE Std 1149.1-2013 B.8.17)
{<register mnemonics description>}	(see IEEE Std 1149.1-2013 B.8.18)
{<register fields description>}	(see IEEE Std 1149.1-2013 B.8.19)
{<register assembly description>}	(see IEEE Std 1149.1-2013 B.8.21)
{<register constraints description>}	(see IEEE Std 1149.1-2013 B.8.22)
{<register association description>}	(see IEEE Std 1149.1-2013 B.8.23)
{<power port association description>}	(see IEEE Std 1149.1-2013 B.8.23)
{<BSDL extensions>}	(see IEEE Std 1149.1-2013 B.8.24)
[<design warning>]	(see IEEE Std 1149.1-2013 B.8.25)
end <component name> <semicolon>	

BSDL grammar tokens common to IEEE 1149.1 and IEEE 1149.10 are as follows. The meaning of the grammar tokens is identical.

```
<asterisk>
<target>
<hex pattern>
<pattern>
<port id>
<representative port>
<mnemonic identifier>
<real>
<reg or seg name>
<quote>
<range>
<TRUE>
<FALSE>
```

IEEE 1149.10 BSDL attributes are defined using the extension capability of BSDL. The IEEE 1149.10 attributes are defined in the IEEE 1149.10 package file described in 7.7. The grammar and syntax for IEEE 1149.10 extension attributes is provided below. The attributes may be listed more than once and in any order. Unlike IEEE Std 1149.1 where keywords are global across the entire BSDL, keywords provided in this standard are valid only in the attribute in which they are described.

7.1.2 Specifications

Syntax

```
<HSJTAG_Extension_List> ::= <HSJTAG_Extension > | <HSJTAG_Extension_List>
<HSJTAG_Extension> ::= <Conformance_1149_10_Description>
                       | <HSTAP_Description> | <Packet_Map_Description>
                       | <Control_Character_Description> | <Scan_Channel_Description>
```

Copyright © 2017 IEEE. All rights reserved.

Rules

a) Except as noted in this document, IEEE 1149.10 BSDL follows the grammar, syntax, and rules of IEEE Std 1149.1-2013, Annex B.

b) When an IEEE 1149.1 TAP is not required to enable compliance with this standard, the overall BSDL syntax shall be as described in Table 9.

NOTE—This is one area where a current IEEE 1149.1 BSDL parser would need to be modified such that <scan port identification> is not required during parsing when the "USE IEEE_1149_10_201x.all" statement is present in the BSDL file.

c) To minimize ambiguity between the BNF notation and the special characters required in an input stream, the following syntax tokens will be used in place of special characters allowed in the parsed input stream:
   i. <left_paren> and <right_paren> shall be the parenthesis characters "(“ and “)", respectively.
   ii. <semicolon> shall be the character “;”.
   iii. <colon> shall be the character “:”.
   iv. <comma> shall be the character “,”

d) The <Conformance_1149_10_Description>, <HSTAP_Description>, <Packet_Map_Description>, <Control_Character_Description> and <Scan_Channel_Description> shall each appear at least once in a <HSJTAG_Extension_List>.

7.2 Conformance attribute

7.2.1 Description

The CONFORMANCE_1149_10 attribute is a BSDL extension to the IEEE Std 1149.1 BSDL language. It provides a standardized description for communicating the year of this standard with which the implementation conforms. Its use is mandatory.

7.2.2 Specifications

<Conformance_1149_10_Description> :: = **attribute Conformance_1149_10**
of <target> **is** <HSJTAG_conformance_string> <semicolon>

<HSJTAG_conformance_string> ::= <quote> <HSJTAG_conformance_identification> <quote>
<HSJTAG_conformance_identification> ::= **STD_1149_10_2017**

7.2.3 Examples

Attribute Conformance_1149_10 of top is "STD_1149_10_2017";

7.3 HSTAP attribute

7.3.1 Description

The HSTAP attribute is a BSDL extension to the IEEE Std 1149.1 BSDL language. It provides a standardized description for communicating the parameters of an HSTAP such that ATE can connect and communicate to it unambiguously (see Table 10).

Copyright © 2017 IEEE. All rights reserved.

Table 10—HSTAP BSDL parameters

Parameter	Definition	Possible Value(s)
HSTAP_NUM	A lane number assignment. This value is used for striping when more than one lane is present.	An integer value
PEDDA_NAME	The Packet_Map associated with this HSTAP. PEDDA_NAMEs that are the same indicate HSTAP that can be channel bonded.	A VHDL identifier that matches at least one identifier found in the PACKET_MAP attribute.
TX_1149_10	The TX is the single-ended or differential pair of signals that the HSTAP uses to transmit data to an ATE. A minimum and maximum peak-to-peak differential or single-ended voltage is specified.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. <min voltage> and <max voltage> are specified as comma separated parameters.
RX_1149_10	The RX is the single-ended or differential pair of signals that the HSTAP uses to receive data from an ATE. A minimum and maximum peak-to-peak differential or single-ended voltage is specified.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. <min voltage> and <max voltage> are specified as comma separated parameters.
ENCODING_1149_10	Specifies that no serial encoding and decoding is present or that one of the known serial encoding methods is used. A user-defined value is allowed for future use; however, user-defined values would not be handled automatically by ATE.	NONE_1149_10, 8B_10B, 64B_66B, 64B_67B, 128B_130B, 128B_132B, 256B_257B or User defined value.
DATARATE_1149_10	Specifies a minimum or maximum transmit and receive data rate for the HSTAP or specifies the data rate is dependent on a multiple of a provided system clock (SYSCLOCK_1149_10), transmit (TXCLOCK_1149_10), reference clock (REFCLOCK_1149_10), or receive clock (RXCLOCK_1149_10).	Two frequencies as a minimum and maximum real values or a real value as a multiple of a system clock or transmit clock.
COMPLIANCE_VIA_TAP	IEEE 1149.10 compliance is enabled via the IEEE 1149.1 TAP. When True, the Enable_P1149_10 and Disable_P1149_10 PDL iProcs need to be provided per Rule 4.1.2 d).	True/False
IDLE_CHAR_REQUIRED	IDLE characters are required to keep the interface clock/PLL operational. HSTAP interfaces based on SPI, for instance, do not require IDLES to be sent during non-data times.	True/False
MAX_FRAMES_1149_10	Specifies the maximum number of frames that can be received on a half-duplex HSTAP before the HSTAP needs to turn-around and transmit frames back due to FIFO size limits.	A positive integer. A zero indicates no limit or full duplex operation.
LATENCY_1149_10	Specifies a minimum and maximum delay present in the design of the HSTAP and PEDDA. The latency is the delay from the receipt of a character at the RX of an HSTAP to the TX response. The latency description allows the ATE to better determine a window of time to expect a response.	A minimum and maximum delay specified as a real number in seconds or a multiple of one of the HSTAP clock periods.

(Table continues)

Copyright © 2017 IEEE. All rights reserved.

Table 10—HSTAP BSDL parameters *(continued)*

Parameter	Definition	Possible Value(s)
TXCLOCK_1149_10	An optional single-ended or differential clock source for either a common clock required for the HSTAP RX or as the TX clock transmitter or a reference clock of the HSTAP. The voltage and frequency of the clock is provided as required attributes.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. A minimum and maximum voltage is provided as a real and a minimum and maximum frequency is provided as real numbers.
RXCLOCK_1149_10	An optional single-ended or differential clock source for either a common clock required for the HSTAP receiver or a reference clock of the HSTAP. The voltage and frequency of the clock is provided as required attributes.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. A minimum and maximum voltage is provided as a real and a minimum and maximum frequency is provided as real numbers.
REFCLOCK_1149_10	An optional single-ended or differential clock source for a reference clock for a HSTAP. The voltage and frequency of the clock is provided as required attributes.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. A minimum and maximum voltage is provided as a real and a minimum and maximum frequency is provided as real numbers.
SYSCLOCK_1149_10	An optional single-ended or differential clock source necessary for operation of the HSTAP and PEDDA logic. The voltage and frequency of the clock is provided as required attributes.	<port id> or <representative port> as defined in the BSDL port and port_grouping attribute. A minimum and maximum voltage is provided as a real and a minimum and maximum frequency is provided as real numbers.
SELECT_1149_10	An optional single-ended or differential enable source that is needed for some common clock serial transmission methods in order to signal the end of a transmitted word.	<representative port> or <port id>. A minimum and maximum voltage is provided as well as the number of clocks in word size duration to assert the signal via the keyword NUM_CLOCKS. A logic value is provided for the state to assert the signal.
BOND_1149_10	This is used when the HSTAP supports channel-bonding. The value specifies the range of bytes that can be transmitted before a new BOND_CHAR must be sent to all channels that are bonded.	<min_byte_count> and <max_byte_count>.

Figure 44 (A) shows in more detail the specifics for calculating latency for the LATENCY_1149_10 parameter. In the example, 8B/10B encoding is used so each character is transmitted as ten bits. In (A), a Start-of-Packet (SOP) is received and the design of the HSTAP and PEDDA is such that TX differential pair (TXP and TXN) continues to complete the transmission of an 8B/10B IDLE character. The transmitter is still not ready and it transmits another IDLE character first followed by the SOP. If the data rate was

Copyright © 2017 IEEE. All rights reserved.

3.125E9 bits per second, then the LATENCY_1149_10 as shown in (A) would be 3.84E-9. In Figure 44 (B) a different design is present that uses an internal clock that is a multiple of the SYSCLOCK_1149_10 specified. While the internal clock frequency is unknown, it creates a much smaller LATENCY_1149_10 of 6.4E-10.

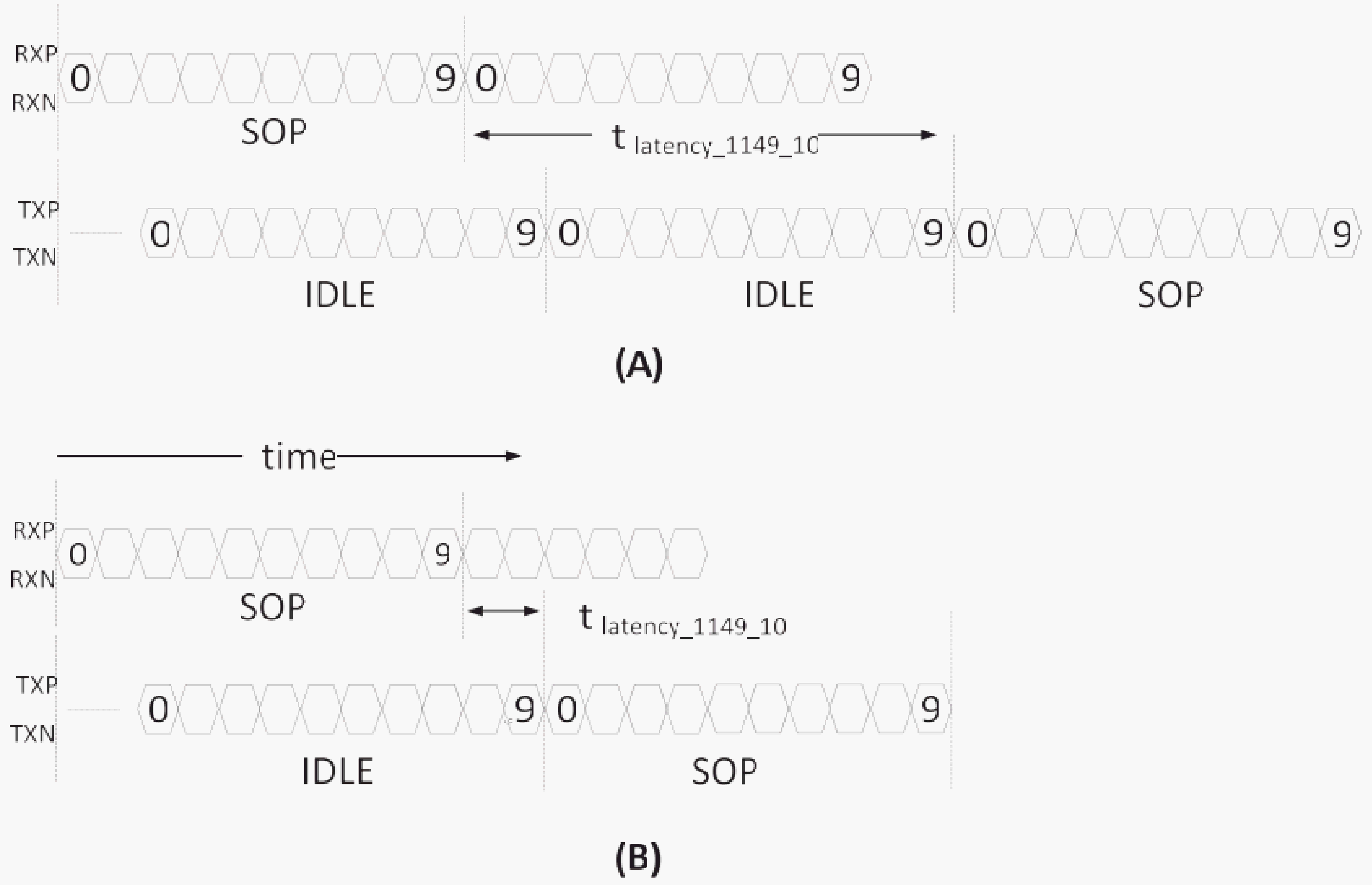

Figure 44—Example timing for LATENCY_1149_10

7.3.2 Specifications

Syntax

```
<HSTAP_Description> ::= attribute HSTAP of <target> IS
                              <quote><phy_string_list ><quote>  <semicolon>


<phy_string_list> ::= <phy_string> | <phy_string_list> <comma> <phy_string>

<phy_string> ::= <left_paren> <phy_list_with_clocks > <right_paren>
                   | <left_paren> <phy_list> <right_paren>

<phy_list> ::=  <left_paren>
          HSTAP_NUM <colon> <integer> <comma>
                PEDDA_NAME <colon> <identifier> <comma>
          TX_1149_10  <tx>  <colon>  <min_voltage><comma><max_voltage> <comma>
           RX_1149_10  <rx>  <colon>  <min_voltage><comma><max_voltage> <comma>
           ENCODING_1149_10 <colon> <encoding><comma>
           DATARATE_1149_10 <colon> <minmax_or_multiple> <comma>
           COMPLIANCE_VIA_TAP <colon> <Truefalse><comma>
           IDLE_CHAR_REQUIRED <colon> <Truefalse><comma>
           MAX_FRAMES_1149_10 <colon> <integer><comma>
           <latency_and_align>
```

Copyright © 2017 IEEE. All rights reserved.

```
<latency_and_align> ::=  <latency_string> <right_paren>
                        | <latency_string> <comma> <align_string> <right_paren>

<latency_string> :: = LATENCY_1149_10    <colon> <time_or_multiple>

<align_string> ::= BOND_1149_10 <colon> <min_byte_count> <comma> <max_byte_count>


<phy_list_with_clocks > ::= <phy_list> <comma> <clocks_string>

<clocks_string> ::=  <tx_clock_string> | <rx_clock_string> | <refclock_string> |
                        <sysclock_string> | <clocks_string> <comma>


< tx_clock_string> ::= <left_paren> <tx_clock_list> <comma> <select_list> <right_paren>
                    |  <left_paren> < tx_clock_list> <right_paren>

<tx_clock_list> ::=  TXCLOCK_1149_10 <tx_clock> <colon>  <min_voltage> <comma>
                    <max_voltage>  <comma><min_rate> <comma> <max_rate>

<select_list> ::=  SELECT_1149_10 <enable> <colon> <min_voltage><comma><max_voltage>
                      <comma> NUM_CLOCKS <colon> <integer> <comma>
                        ASSERT_VALUE <colon>  <logic_assert>

< sysclock_string> ::= <left_paren>< sysclock_list><right_paren>

<sysclock_list> ::=  SYSCLOCK_1149_10  <sysclock> <colon>
<min_voltage><comma><max_voltage>
                      <comma><min_rate> <comma> <max_rate>


< rx_clock_string> ::= <left_paren>< rx_clock_list> comma> <select_list> <right_paren>
                    |  <left_paren> < rx_clock_list> <right_paren>


<rx_clock_list> ::=  RXCLOCK_1149_10  <rxclock> <colon>
<min_voltage><comma><max_voltage>
                      <comma><min_rate> <comma> <max_rate>

< ref_clock_string> ::= <left_paren>< refclock_list><right_paren>

<refclock_list> ::=  REFCLOCK_1149_10  <refclock> <colon>
<min_voltage><comma><max_voltage>
                      <comma><min_rate> <comma> <max_rate>

<minmax_or_multiple> ::= <min_rate> <comma> <max_rate> | <multiple_expression>

<time_or_multiple> ::= <min_time> <comma> <max_time> | <multiple_expression>

<multiple_expression> ::= <sys_expression> | <tx_expression> | <rx_expression> |
<ref_expression>

<sys_expression> :: = <multiple> <asterisk> SYSCLOCK_1149_10
<tx_expression> ::= <multiple> <asterisk> TXCLOCK_1149_10
```

Copyright © 2017 IEEE. All rights reserved.

<rx_expression> ::= <multiple> <asterisk> **RXCLOCK_1149_10**
<ref_expression> ::= <multiple> <asterisk> **REFCLOCK_1149_10**

<multiple> ::= <real>

<tx> ::= <representative port> | <port ID>

<rx> ::= <representative port> | <port ID>

<sysclock> ::= <representative port> | <port ID>

<rxclock> ::= <representative port> | <port ID>

<refclock> ::= <representative port> | <port ID>

<min_voltage> ::= <real>
<max_voltage> ::= <real>

<encoding> ::= **NONE_1149_10** | **8B_10B** | **64B_66B** | **64B_67B** |
128B_130B | **128B_132B** | **256B_257B** | <Mnemonic identifier>

<min_rate>::= <real>

<max_rate>::= <real>
<min_time> ::= <real>
<max_time> ::= <real>

<tx_clock> ::= <representative port> | <port ID>

<enable > ::= <representative port> | <port ID>

<logic_assert> ::= 1 | 0

<min_byte_count> ::= <real>

<max_byte_count> ::= <real>

<Truefalse> ::= TRUE | FALSE

Rules

a) When a <representative port> is specified, the <min_voltage> shall represent the minimum differential voltage required by the receiver to operate.
b) When a <representative port> is specified, the <max_voltage> shall represent the maximum differential voltage that the receiver can tolerate.
c) When an HSTAP uses a common clock, a <tx_clock> value shall be provided.
d) When a <tx_clock> value is provided, the <max rate> shall indicate the maximum frequency of the provided <tx_clock> and the <min rate> shall indicate the minimum frequency of the provided clock.
e) The <max_time> value shall represent the maximum delay associated with the response of the TX to received data at the RX.
f) The <min_time> value shall represent the minimum delay associated with the response of the TX to received data at the RX.
g) When a <sys_expression> is specified, at least one <sysclock _list> shall be specified.
h) When a <tx_expression> is specified, at least one <tx_clock _list> shall be specified.
i) When a <rx_expression> is specified, at least one <rx_clock _list> shall be specified.

Copyright © 2017 IEEE. All rights reserved.

j) When a <ref_expression> is specified, at least one <ref_clock _list> shall be specified.
k) A <max_voltage> shall be greater than or equal to the <min_voltage>.
l) A <max_time> shall be greater than or equal to the <min_time>.
m) When a <phy_string_list> contains more than one <phy_string>, the CONTROL_CHAR attribute shall define a BOND_CHAR definition.
n) The <integer> value associated with a HSTAP_NUM shall be unique within a given HSTAP attribute.
o) There shall be at least one HSTAP_NUM with an <integer> value of zero.
p) An HSTAP_NUM with an <integer> value of zero shall document the first lane in a multi-lane HSTAP description.
q) The <identifier> associated with the PEDDA_NAME in a <phy_list> shall be specified in at least one PACKET_MAP attribute.
r) The <max_byte_count> shall document the maximum number of bytes received on a single lane during channel bonding before the next BOND_CHAR.
s) The <min_byte_count> shall document the smallest number of bytes that may be received before the next BOND_CHAR.
t) The <max_byte_count> shall be at least four and an integer multiple of four.
u) The <min_byte_count> shall be at least four and an integer multiple of four.
v) The <multiple> expression in a <time_or_multiple> shall be calculated as the <multiple> times the period of the clock specified in one of the <sys_expression>, <tx_expression>, <rx_expression>, or <ref_expression> expressions.
w) The value of COMPLIANCE_VIA_TAP shall be FALSE when the 1149_1_TAP_IS_NOT_PRESENT keyword is present in the BSDL.

   NOTE—Compliance via the TAP is not possible when the keyword 1149_1_TAP_IS_NOT_PRESENT is used.

x) <min_voltage> and <max_voltage> shall be a voltage in V represented by a real number.
y) <min_rate > and <max_rate > shall be a frequency in Hz represented by a real number.
z) <min_rate > shall be less than or equal to <max_rate >.
aa) <min_time > and <max_time > shall be a time in s represented by a real number.

7.3.3 Examples

This first example shows how the pins of a SATA interface are re-used for the IEEE 1149.10 HSTAP. In this example, two differential pairs exist in a device BSDL port grouping statement as follows:

```
attribute PORT_GROUPING of top  : entity is
"Differential_Voltage ( " &
"(SATA_TXP,  SATA_TXN),"&
"(SATA_RXP,  SATA_RXN)) ";
```

SATA_TXP is the IC's representative port of a differential pair. SATA_TXN is defined in the BSDL port grouping attribute such that the tester knows which two pins are parts of the IC's transmitter. SATA_RXP is the representative port of a differential pair. SATA_RXN is a signal that is part of the BSDL port grouping for this IC such that the tester can determine the differential pair to receive data from the ATE. The second two parameters indicate 500 mV and 800 mV as the minimum and maximum voltage applied to the interface. 8B/10B encoding is specified and the rate of the interface is 3.125 Gbits/s.

```
attribute HSTAP of EXA : entity IS
"(" &
"(HSTAP_NUM      : 0, " &
" PEDDA_NAME            : PEDDA1, "&
                   -- tx+ port min/max (with SATA_TXP)
" TX_1149_10 SATA_TXP : 500.0E-3, 800.0E-3,"&
                         -- rx+ port min/max (with SATA_RXN)
" RX_1149_10 SATA_RXP : 500.0E-3,  800.0E-3," &
" ENCODING_1149_10 : 8B_10B, "&          --  Encoding
" DATARATE_1149_10 : 3.124E9, 3.126E9,"& --  min and max bit
   rate
" COMPLIANCE_VIA_TAP :  TRUE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&
--latency from RX to TX
" LATENCY_1149_10   : 600.0E-12, 1100.0E-12 )" &
")";
```

An HSTAP may require a transmitter clock to allow data to flow from the parallel input to the transmitter serial driver. Figure 45 shows an example where the TX side of the HSTAP requires a TX clock.

```
attribute HSTAP of EXB : entity IS
"(" &
"(HSTAP_NUM         : 0, " &
" PEDDA_NAME              : PEDDA1, "&
" TX_1149_10 TXP    : 500.0E-3, 800.0E-3,"&  -- tx+ port
   min/max
" RX_1149_10 RXP    : 500.0E-3, 800.0E-3," & -- rx+ port
   min/max
" ENCODING_1149_10 : 8B_10B, "&          --  Encoding
" DATARATE_1149_10 : 2.5E1 * TXCLOCK_1149_10,"&
" COMPLIANCE_VIA_TAP :  TRUE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&
" LATENCY_1149_10   : 3.0E-9, 4.0E-9 )," & -- latency from RX
   to TX
"( TXCLOCK_1149_10 TRANSMITCLK: 2.5E0, 4.0E0, " &
"  1.25E8, 2.60E8 )" &
")";
```

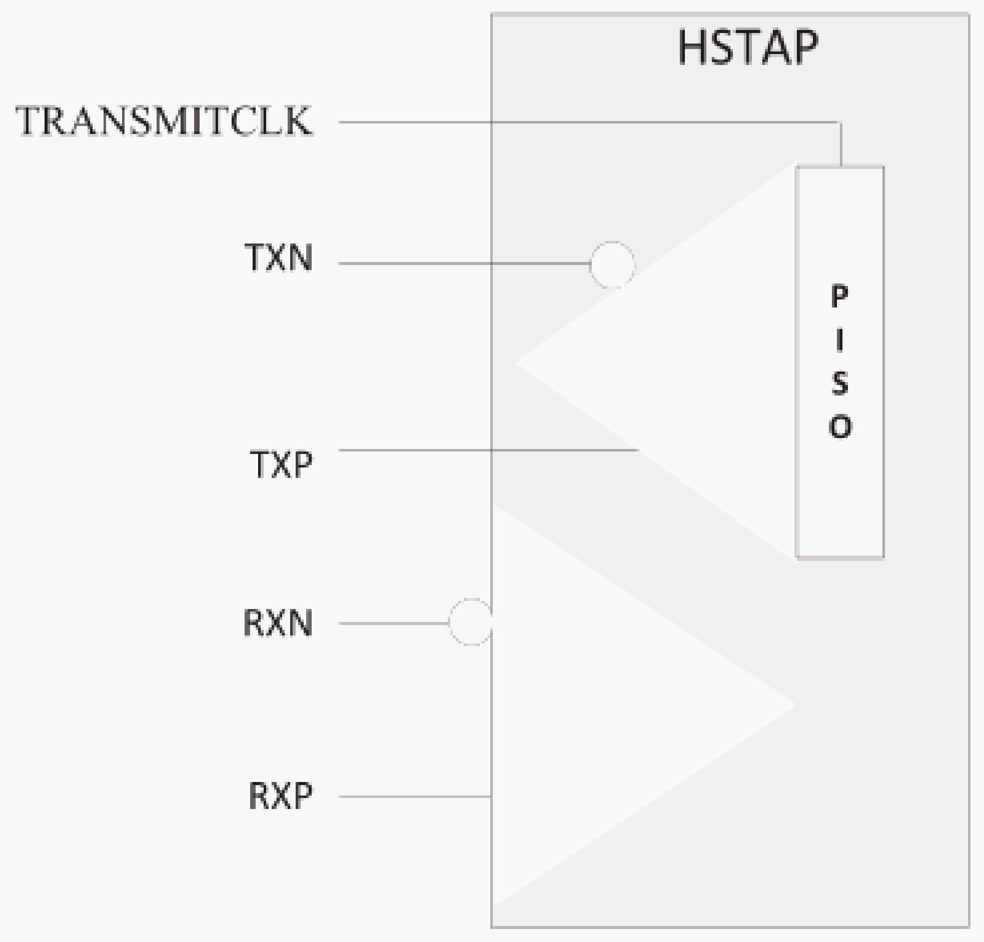

Figure 45—HSTAP SERDES requiring TX_CLOCK_1149_10 input

A serial peripheral interface (SPI) bus can be supported as an HSTAP. Figure 46 shows the timing of an example SPI interface with a 10-bit register. The TX_CLOCK_1149_10 is used to support transmission of data from the ATE to a SPI-based device. The SPI SCLK (serial clock) would be described with the TX_CLOCK_1149_10 parameter and the SSN (slave select active low) would be denoted as the SELECT_1149_10 port. The SELECT_1149_10 is used to indicate the SSN. The duration for SSN to be asserted is specified by the NUM_CLOCKS parameter. In the example, the SSN is asserted low (as specified by ASSERT_VALUE) for a time of 10 SCLKS.

Copyright © 2017 IEEE. All rights reserved.

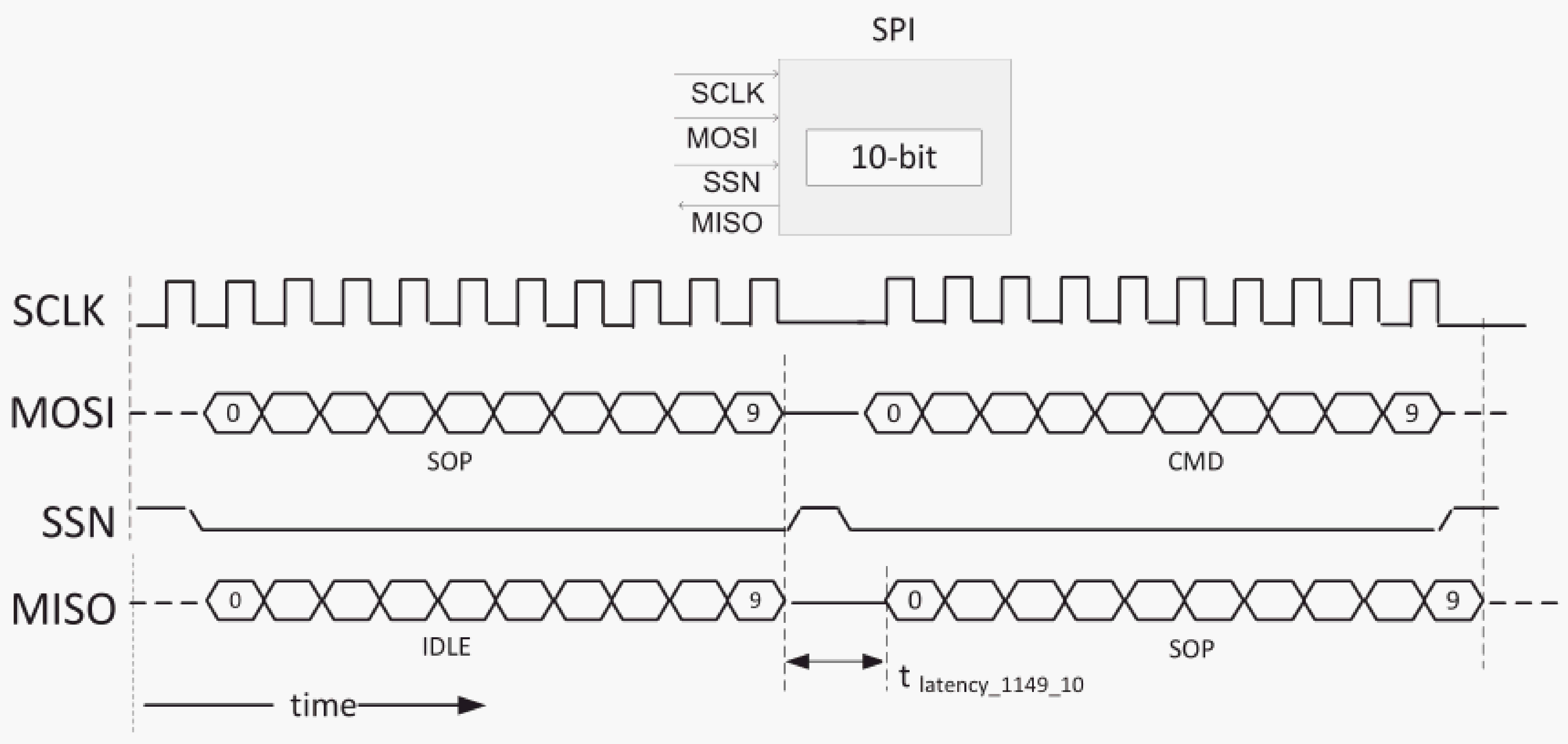

Figure 46—SPI Bus as HSTAP

```
attribute HSTAP of SPI: entity IS
"(" &
"(HSTAP_NUM : 0, " &
" PEDDA_NAME            : SPI_PEDDA, "&
" TX_1149_10 MISO: 3.0E0,  3.8E0,"&  -- tx port min/max
" RX_1149_10 MOSI: 3.0E0,  3.8E0," & -- rx port min/max
" ENCODING_1149_10 : NONE_1149_10, "&               --
   Encoding
" DATARATE_1149_10 : 1.0E6, 1.0E8,"& --  min and max bit rate
" COMPLIANCE_VIA_TAP :  FALSE,"&
" IDLE_CHAR_REQUIRED :  FALSE,"&
" LATENCY_1149_10  : 5.0E-9, 1.0E-8 )," & -- latency from RX
   to TX
"( TXCLOCK_1149_10 SCLK : 3.0E0, 3.8E0, 1.0E6, 1.0E8, " &
"  SELECT_1149_10  SSN  : 3.0E0, 3.8E0, NUM_CLOCKS: 10," &
"  ASSERT_VALUE : 0 )" &
" )";
```

7.4 Packet_Map attribute

7.4.1 Description

The BSDL attribute PACKET_MAP communicates the data size, the interleave size of the packet, and the mapping of the groups chosen by the designer of the PEDDA. The payload data of the SCAN and SCANR type packets contain data for one or more target scan channels. In order to understand how the data maps to the scan channels, the INTERLEAVE_SIZE needs to be documented. The <interleave_string> specifies the number of bits for each target scan channel before the next scan channel in the sequence. An IEEE 1149.10 packet decoder/encoder may have one or more <interleave_string> specifications, such that each scan channel group can be interleaved differently (see Table 11).

Copyright © 2017 IEEE. All rights reserved.

Table 11 —Packet_Map parameters

Parameter	Definition	Possible Value(s)
PEDDA_NAME	A descriptive name for this PEDDA architecture.	A VHDL identifier
SCAN_DATA_SIZE	The number of bits available to the round-robin assignment of unencoded scan data in the SCAN and SCANR packet. The round-robin handling of data for scan-channels in a given SCAN_GROUP must fit within the bit size of SCAN_DATA_SIZE. The SCAN_DATA_SIZE is chosen by the designer such that packet generation tools can best map SCAN/SCANR Payload data as efficiently as possible. SCAN_DATA_SIZE has a minimum value of INTERLEAVE_SIZE multiplied by the number of scan channels in the SCAN_GROUP when PACK is false. When PACK is true, SCAN_DATA_SIZE is INTERLEAVE_SIZE multiplied by the largest number of scan-channels in a SCAN_GROUP enabled simultaneously.	An integer greater than zero
TX_ORDER	Defines whether the transmission of data is performed using the MSB or LSB bit first. Most serial communications use LSB first, however SONET uses MSB first.	LSB_FIRST, MSB_FIRST
CMD_PARITY	Defines whether there is a parity calculation on the command word in the packet. If parity is used, ODD and EVEN parity is supported in the sixth bit of the CMD of the packet.	NONE_1149_10, EVEN_1149_10, ODD_1149_10
INTERLEAVE_SIZE	The size of the contiguous bits destined for a given channel in the SCAN packet and received from the SCANR packet. The interleave size will vary based on PEDDA design optimizations.	An integer greater than zero.
SCAN_GROUP	The assignment of scan groups that use the packet map attribute described.	A range or integer list of scan groups.
PACK	The interleaved data is packed without containing skipped scan-channels that are in the SCAN_GROUP but not enabled via the CH-SELECT packet. Such a capability may work best with INTERLEAVE_SIZE of nibble or larger width.	TRUE or FALSE

7.4.2 Specifications

Syntax

<Packet_Map_Description> ::= **attribute PACKET_MAP of** <target> **IS**
<quote> <Packet_Map_String>
<quote> <semicolon>

<Packet_Map_String> ::= <Name_String> <comma> <Order_String> <comma>
<CMD_Parity_String> <comma> <Interleave_Group>

<Name_String> ::= <left_paren> **PEDDA_NAME** <colon> <identifier><right_paren>

Copyright © 2017 IEEE. All rights reserved.

<Order_String> ::= <left_paren> **TX_ORDER** <colon> <order_type> <right_paren>

<order_type> ::= **LSB_FIRST** | **MSB_FIRST**

<CMD_Parity_String> ::= <left_paren> **CMD_PARITY** <colon> <parity_type> <right_paren>

<parity_type>   ::= **NONE_1149_10** | **EVEN_1149_10** | **ODD_1149_10**

<Interleave_Group> ::= <Interleave_Combo> | <Interleave_Group> <comma>
<Interleave_Combo>

<Interleave_Combo> ::= <Data_String> <comma> <Pack_String> <comma>
                        <Interleave_String> <comma>
                        <scan_group_string>

<Data_String> ::= <left_paren> **SCAN_DATA_SIZE** <colon> <integer><right_paren>

<Pack_String> ::= <left_paren> **PACK** <colon> <true_false> <right_paren>

<Interleave_String> ::= <left_paren> **INTERLEAVE_SIZE** <colon> <integer><right_paren>

<scan_group_string> ::= <left_paren> **SCAN_GROUP** <colon>
                         <group_list><right_paren>

< group_list> ::= <group_field> | <group_list> <comma> <group_field>

<group_field> ::= <range> | <integer>

<true_false> :: = TRUE | FALSE

Rules

a) The <parity_type> shall indicate the parity implemented on the CMD value at bit 6 (Counting from bit zero) and shall be one of the following:
   i. EVEN_1149_10 for even CMD parity (i.e., an even number of ones in the 8-bit CMD)
   ii. ODD_1149_10 for odd CMD parity

   or

   iii. NONE_1149_10 when parity is not used on the CMD value.
b) The <integer> value of a <Data_String> shall be greater than zero.
c) The <integer> value of a <Interleave_String> shall be greater than zero.
d) The <integer> value of a <Scan_Group_String> shall be less than 65,536.
e) The <identifier> of a <Name_String> shall be unique across all PACKET_MAP attributes in a given BSDL or BSDL package file.
f) When the <true_false> value of PACK is false, the <integer> value of <Data_String> shall be greater than or equal to the <integer> value of <Interleave_String> multiplied by the total number of scan-channels available in the associated <chain_list>.
g) When the <true_false> value of PACK is true, the <integer> value of <Data_String> shall be greater than or equal to the <integer> value of <Interleave_String>.

Copyright © 2017 IEEE. All rights reserved.

Recommendations

h) <SCAN_DATA_SIZE> should be an even multiple of 4-bit nibbles.

NOTE—SCAN and SCANR packet data is specified in 4-byte frames.

7.4.3 Examples

The size of the PISO on the scan-in and the size of the SIPO on the scan-out generally set the Interleave_Size. In this example, two scan channels are interfaced by two 4-bit PISOs and two 4-bit SIPOs. The payload data for any SCAN and SCANR packet is then interleaved between data going to S1 and data going to S2 (see Figure 47).

```
attribute Packet_Map of EXA: entity IS
"(PEDDA_NAME:             PEDDA1)," &
"(TX_Order:  LSB_FIRST),"&
"(CMD_Parity: NONE_1149_10)," &
"(Scan_Data_Size:  8 ), " &
"(PACK      :  TRUE), " &
"(Interleave_Size:  4), " &
"(Scan_Group : 1)";
```

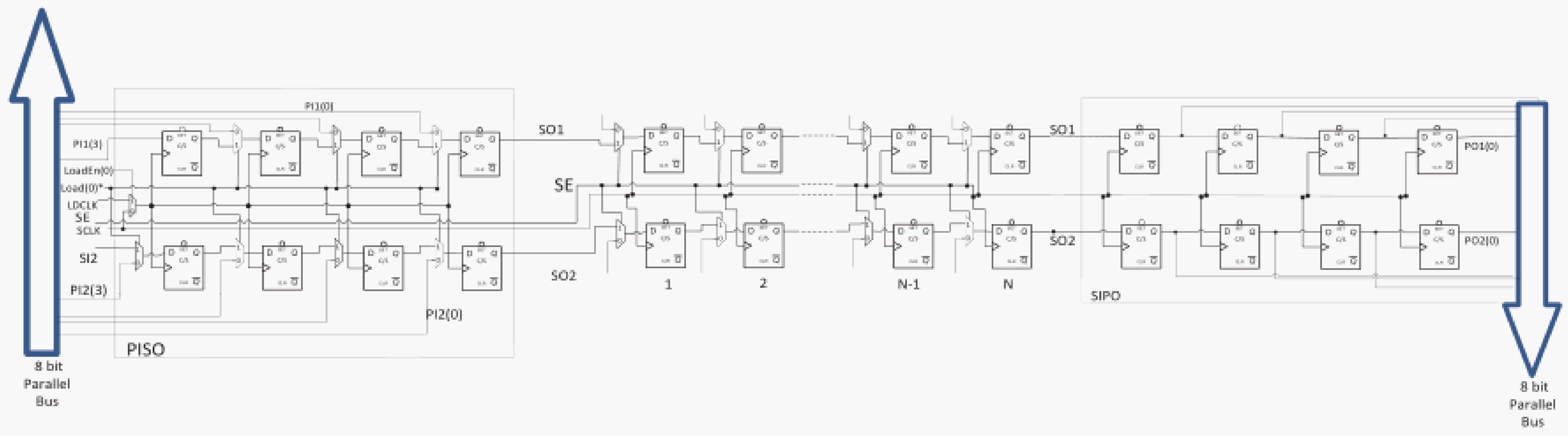

Figure 47—Four-bit PISO/SIPO on 2 scan channels

The attribute can be used to describe multiple smaller groupings. In the following attribute, there are two scan groups with different Interleave_Sizes. Scan_Group 1 is made up of scan channels 1 to 16 with an Interleave_Size of one. Scan_Group 2 is made up of scan channels 17 to 64 with an 8-bit Interleave_Size.

```
attribute Packet_Map of EXB : entity IS
"(PEDDA_NAME:             PEDDA1)," &
"(TX_Order:  LSB_FIRST),"&
"(CMD_Parity: NONE_1149_10)," &
"(Scan_Data_Size:  16 ), " &
"(PACK:        FALSE), " &
"(Interleave_Size:  1), " &
"(Scan_Group : 1)," &
"(Scan_Data_Size:  16 ), " &
"(PACK:        TRUE ), " &
"(Interleave_Size:  8 ), " &
"(Scan_Group : 2)";
```

The construct for the first Scan_Group indicates the PEDDA is designed such that it is expecting scan data formatted on 16-bit boundaries, no matter how few scan-channels are enabled during a SCAN packet for

Copyright © 2017 IEEE. All rights reserved.

Scan_Group 1. The second group has an INTERLEAVE_SIZE of eight, perhaps indicating that the scan-channels are higher-speed and have a small eight-bit PISO. The SCAN_DATA_SIZE of 16 indicates that the maximum number of scan-channels that the PEDDA is designed to decode at one time is two scan-channels.

The Packet_Map attribute of SPI illustrates:

```
attribute Packet_Map of SPI: entity IS
"(PEDDA_NAME:              SPI_PEDDA)," &
"(TX_Order:  LSB_FIRST),"&
"(CMD_Parity: NONE_1149_10)," &
"(Scan_Data_Size:  16 ), " &
"(PACK:        FALSE ), " &
"(Interleave_Size:  8 ), " &
"(Scan_Group : 1 to 2 )" &
" " )";
```

7.5 Control_Chars attribute

7.5.1 Description

The Control_Chars attribute is a BSDL extension that defines special symbols used in the serial communication to an IEEE 1149.10 HSTAP. Control characters are any of the special symbols used in serial communication encodings. 8B/10B, for instance, defines twelve control symbols. The control characters in the Control_Chars BSDL attribute are described in their un-encoded form. For instance, a control character when used with 8B/10B encoding is described with the un-encoded value and the encoding would convert the 8-bit value into a 10-bit value. The 10-bit value is determined from the 8B/10B table look up based on the running disparity.

There are nine predefined control characters used in communicating with an IEEE 1149.10 HSTAP. Table 12 provides the definitions of each one.

Table 12—Control_Chars definitions

Character	Definition
SOP_CHAR	Start-of-Packet. Signals to the receiver that a packet is starting.
EOP_CHAR	End-of-Packet. Signals to the receiver that a packet has ended.
IDLE_CHAR	The IDLE character is sent when no other data is being transmitted. The idle character allows the receiver to recover an embedded clock and keep the data aligned. IDLE characters can also be sent by a transmitter to compensate for slow scan-channels.
ERROR_CHAR	The ERROR character is transmitted when the IEEE 1149.10 interface detects a data transmission error. For instance, if the receiver calculates a CRC32 different than the value transmitted to it, then an ERROR character should be transmitted.
XOFF_CHAR	The XOFF character is used for flow control. The XOFF character is transmitted by an IEEE 1149.10 interface when no additional data can be processed by the interface.
XON_CHAR	When the IEEE 1149.10 interface is ready to receive more data, it transmits the XON character to the ATE.
CLEAR_CHAR	When a CLEAR character is received, an IEEE 1149.10 interface resets itself, clearing all pending data processing, data transmitting, and all internal data counters.
COMPLIANCE _CHAR	When a COMPLIANCE character is received by the mission mode circuitry, the circuitry enables compliance to this standard. There must be at least one unused control character in the mission mode encoding to provide for a COMPLIANCE_CHAR that does not trigger a mission mode function.
BOND_CHAR	When channel-bonding and striping is performed the ATE must transmit the BOND_CHAR periodically such that the internal SERDES gearbox for the channels can de-skew the incoming data and align the multiple octets being received. The BOND_CHAR is optional in the BSDL grammar.

Copyright © 2017 IEEE. All rights reserved.

7.5.2 Specifications

Syntax

<Control_Character_Description> ::= **attribute CONTROL_CHARS of** <target>
IS <quote> <control_string_list> <quote> <semicolon>

<control_string_list> ::= <control_string> | <control_string_list> <comma> <control_string>

<control_string> ::= <left_paren><control_char_all> <colon> <hex pattern> <right_paren>

<control_char_all> ::= <control_char_mandatory> | <control_char_optional>

<control_char_mandatory> ::= SOP_CHAR | EOP_CHAR | IDLE_CHAR | ERROR_CHAR |
XOFF_CHAR | XON_CHAR | CLEAR_CHAR

<control_char_optional> ::= COMPLIANCE_CHAR | BOND_CHAR

Rules

a) Each <control_char_mandatory> shall be defined within the attribute CONTROL_CHARS.
b) There shall be a unique one byte <hex pattern> for each <control_char_mandatory> and each <control_char_optional>.
c) A <control_char_all> shall be defined just once in a given CONTROL_CHARS attribute.
d) The COMPLIANCE_CHAR shall be defined when a character in the mission mode data stream is necessary to enable the HSTAP to operate in an IEEE 1149.10 compliant mode.
e) The BOND_CHAR shall be defined when a HSTAP <phy_string_list> contains more than one <phy_string>.

7.5.3 Examples

In this example, the control characters are defined for an interface. The characters are eight bits in length and typical of an example interface that uses 8B/10B.

Example A

```
attribute CONTROL_CHARS  of EXA : entity IS
"(SOP_CHAR :    0xFB ), " &  -- K27.7
"(EOP_CHAR  :   0xFD)," &   -- K29.7
"(IDLE_CHAR :   0xBC),"&          -- K28.5
"(ERROR_CHAR :  0xFE),"&          --K30.7
"(XOFF_CHAR :   0x7C),"&    -- K28.3
"(XON_CHAR:     0x1C),"&     -- K28.0
"(CLEAR_CHAR :  0x5C)";      -- K28.2
```

Example B

In this example the interface is similar to example A; however, the optional COMPLIANCE character is defined such that the mission mode interface can be switched to IEEE 1149.10 compliant mode through the reception of the COMPLIANCE control character. An example would be a CPU that communicates to an integrated circuit via SATA. In order to perform in-the-field test of the integrated circuit, the CPU sends the COMPLIANCE character 0xDC (and encodes it using the encoding specified). For instance, in 8B/10B, 0xDC is K28.6 and when it is encoded and running disparity is +1, the logical value that would be

Copyright © 2017 IEEE. All rights reserved.

transmitted over a SATA differential pair would be 0b1100001001. After the COMPLIANCE character is decoded, the mission mode SERDES switches the interface from SATA to IEEE 1149.10 compliant mode, enabling the HSTAP and PEDDA.

```
attribute CONTROL_CHARS  of EXB : entity IS
"(SOP_CHAR :     0xFB ), " &
"(EOP_CHAR  :    0xFD )," &
"(IDLE_CHAR :    0xBC),"&
"(ERROR_CHAR :   0xFE ),"&
"(XOFF_CHAR :    0x7C ),"&
"(XON_CHAR:      0x1C),"&
"(CLEAR_CHAR :   0x5C),"&
"(COMPLIANCE_CHAR: 0xDC)";
```

ERROR_CHAR example usage

This is example usage of the ERROR_CHAR. ATE can expect that the HSTAP and PEDDA will return an ERROR_CHAR when an error in the transmission or packet formation is received. Figure 48 shows a typical coupling of ATE to the HSTAP with a TX and RX differential pair. All IEEE 1149.10 packets include a CRC32 calculation that validates the integrity of the transmission. A portion of a transmitted packet is shown with the 4-byte CRC32 value shown in one block being transmitted to the HSTAP.

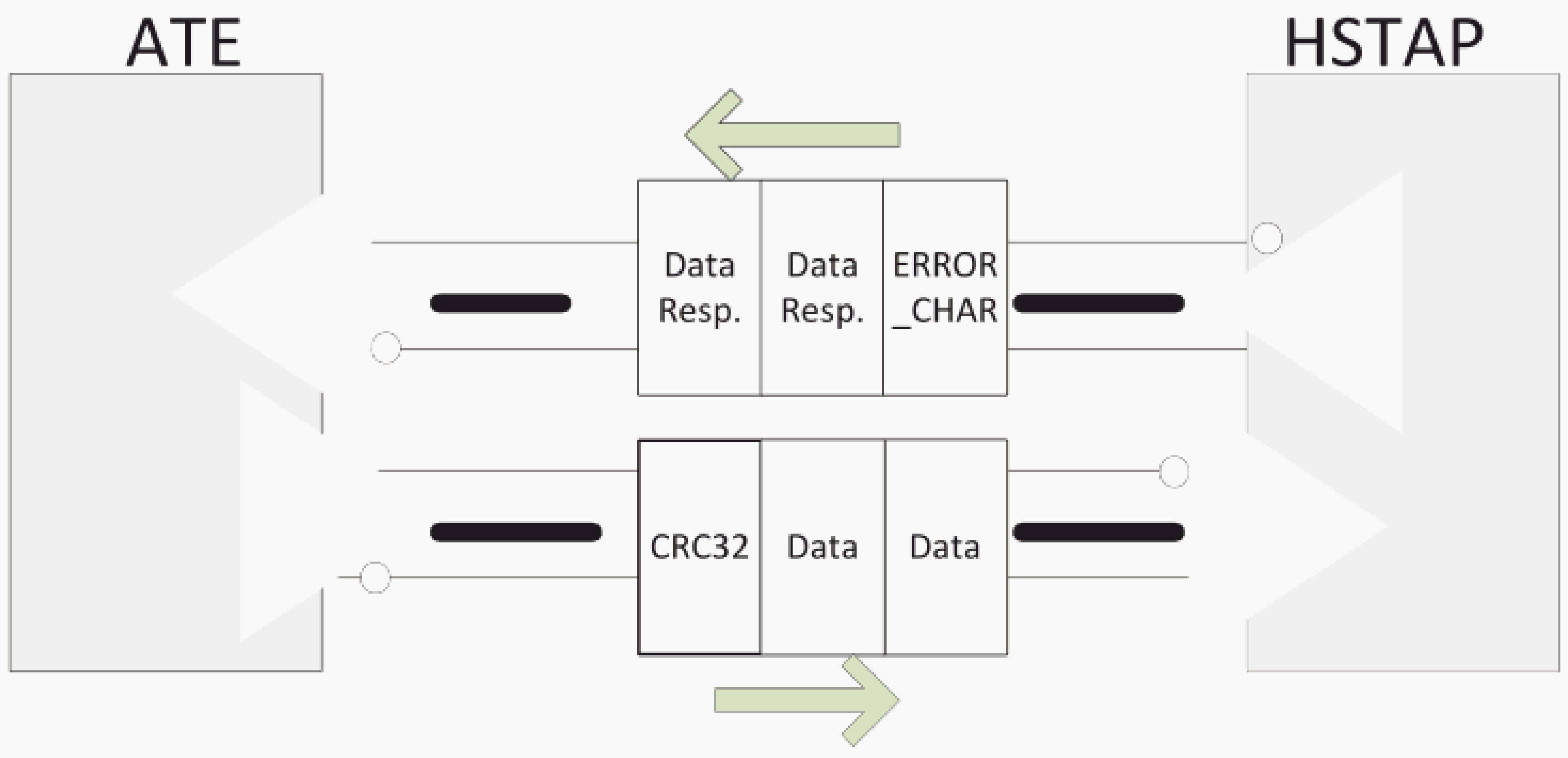

Figure 48—Use of ERROR_CHAR

The IEEE 1149.10 interface calculates a CRC32 for the data received and, if the data received is incorrect (due to transmission errors for instance), then the IEEE 1149.10 interface transmits the ERROR_CHAR and three padding bytes [see permission 6.1.2 j)] back to the ATE at the end of a packet frame. One of the advantages of the IEEE 1149.10 approach over a traditional scan test is that there is validation that the data applied to the circuit under test is error-free.

7.6 Scan_Channel_Association attribute

7.6.1 Description

Register descriptions need to be associated with a scan channel such that PDL code written can be translated into the appropriate IEEE 1149.10 packets (see Table 13).

Copyright © 2017 IEEE. All rights reserved.

Table 13—Scan_Channel_Association description

Parameter	Definition	Possible Value(s)
PEDDA_NAME	A descriptive name for this PEDDA architecture.	A VHDL identifier
SCAN_GROUP	Defines which group the specified channel belongs to.	An integer between zero and 65,535.
Association_string	Defines the scan channel as a <reg or seg name>, <register element> or keyword "Instruction" and assigns the scan channel number used by the PEDDA.	An IEEE 1149.1 <reg or seg name>, <register element>, or instruction register. The integer is the scan channel number used by the PEDDA.
Instruction	The keyword indicates when the scan-channel is associated with the IEEE 1149.1 instruction register.	"Instruction"
Max_Clock	The parameter indicates the maximum clock rate that a given scan-channel can achieve.	A real greater than zero.
CH_Select	Indicates the bit position in the CH-Select packet that a given Scan-Channel is enabled. A logic one in the bit position enables the scan-channel capture, shift, and update signals.	An integer number greater than zero.

7.6.2 Specifications

Syntax

<Scan_Channel_Description> ::= **attribute SCAN_CHANNEL_ASSOCIATION of** <target>
IS <quote> <association_string_list><quote> <semicolon>

<association_string_list> ::= <group_association_string> | <association_string_list> <comma> <group_association_string>

<group_association_string>::= <left paren>
PEDDA_NAME <colon>
<identifier> <comma>
SCAN_GROUP <colon> <integer> <comma>
<association_string_list>
<optional_overshift>

<association_string_list> ::= <association_string> |
<association_string_list> <comma> <association_string>

<association_string> ::= <left_paren> <channel_string> | <channel_string> <comma>
<max_clock_string> <right_paren>

<channel_string> ::= <channel> <colon> <integer> <comma> <ch_select_string>
<ch_select_string> :: = CH_SELECT <colon> <integer>
<max_clock_string>:: = MAX_CLOCK <colon> <real>

<channel> ::= <reg or seg name> | <register element> | **Instruction**

Copyright © 2017 IEEE. All rights reserved.

<optional_overshift> :: = <overshift_yes>| <right_paren>

<overshift_yes> :: = <comma> <no_overshift_string> <right_paren>

<no_overshift_string> ::= <left_paren> **NO_OVERSHIFT** <colon> <no_overshift_list> <right_paren>

<no_overshift_list> ::= <no_overshift_field> | <no_overshift_list> <comma> <no_overshift_field>

<no_overshift _field> ::= <range> | <integer>

Rules

a) The <integer> in a <group_association_string> shall be between zero and 65,535, inclusive.
b) The <integer> in an <channel_string> shall be between 0 and 1,048,575 inclusive.
c) The <identifier> in a <group_association_string> shall be unique for a given SCAN_GROUP <integer> value.
d) The <identifier> in a <group_association_string> shall be used at least once in a PACKET_MAP attribute.
e) The <real> in an <max_clock_string> shall be greater than zero.
f) The <channel> shall not be equivalent to "Instruction" when the 1149_1_TAP_IS_NOT_PRESENT keyword is present in the BSDL containing the SCAN_CHANNEL_ASSOCIATION attribute.
g) The <reg or seg name> in an <association_string> shall be defined in a REGISTER_FIELDS or REGISTER_ASSEMBLY attribute.
h) A <channel> shall be unique within an <association_string_list>.
i) The <integer> in a <channel_string> shall be unique within an <association_string_list>.

NOTE—Rules h) and i) ensure that there is just one channel described per bit position.

j) For each unique <integer> in a <ch_select_string> there shall be at least one <max_clock_string> specified.

NOTE—Rule j) ensures that at least one clock frequency is specified for any unique ch_select integer value.

Permissions

k) An <integer> in a <ch_select_string> may appear more than once within a <association_string_list>.
l) The values specified for any <integer> in a <ch_select_string> may be non-sequential.

7.6.3 Examples

In the following example the PEDDA is designed with two groups: zero and one. Group zero contains the standard instruction register, standard defined test data registers, and four user defined test data registers: User1, User2, User3, and CoreVoltage. CoreVoltage is a TDR associated with four core voltage monitors. Group one includes 16 scan channels that are used for IC test. The CoreVoltage TDR also appears as a scan-channel in Group 1 such that the voltage can be monitored within the IC test data stream.

```
attribute SCAN_CHANNEL_ASSOCIATION of EXA : entity IS

"(PEDDA_NAME : PEDDA1," &
" SCAN_GROUP : 0 ,"&
"(Instruction : 0, CH_SELECT: 0, MAX_CLOCK : 5.0e7), " &
"(Bypass  : 1, CH_SELECT: 1, MAX_CLOCK : 5.0e7), " &
```

Copyright © 2017 IEEE. All rights reserved.

```
"(Boundary: 2, CH_SELECT: 2, MAX_CLOCK : 5.0e7), " &
"(Device_ID : 3, CH_SELECT: 3, MAX_CLOCK : 5.0e7), " &
"(Init_Data : 4, CH_SELECT: 4,MAX_CLOCK : 5.0e7), " &
"(ECID : 5, CH_SELECT: 5, MAX_CLOCK : 5.0e7), " &
"(Reset_Select : 6, CH_SELECT: 6, MAX_CLOCK : 5.0e7), " &
"(User1: 7, CH_SELECT: 7, MAX_CLOCK : 5.0e7), " &
"(CoreVoltage: 8, CH_SELECT: 8, MAX_CLOCK : 1.0e8), " &
"(User2: 9, CH_SELECT: 9, MAX_CLOCK : 5.0e7), " &
"(User3: 10, CH_SELECT: 10, MAX_CLOCK : 5.0e7)" &
")," &
"(PEDDA_NAME : PEDDA1," &
" SCAN_GROUP : 1 ,"&
"(CoreVoltage: 0, CH_SELECT: 8, MAX_CLOCK : 1.0e8), " &
"(Internal1: 1, CH_SELECT: 17, MAX_CLOCK : 1.0e8), " &
"(Internal2: 2, CH_SELECT: 17), " &
"(Internal3: 3, CH_SELECT: 17), " &
"(Internal4: 4, CH_SELECT: 17)," &
"(Internal5: 5, CH_SELECT: 17), " &
"(Internal6: 6, CH_SELECT: 17), " &
"(Internal7: 7, CH_SELECT: 17), " &
"(Internal8: 8, CH_SELECT: 17)," &
"(Internal9: 9, CH_SELECT: 17), " &
"(Internal10: 10, CH_SELECT: 17), " &
"(Internal11: 11, CH_SELECT: 17), " &
"(Internal12: 12, CH_SELECT: 17)," &
"(Internal13: 13, CH_SELECT: 17), " &
"(Internal14: 14, CH_SELECT: 17), " &
"(Internal15: 15, CH_SELECT: 17), " &
"(Internal16: 16, CH_SELECT: 17)," &
"(Response1 : 17, CH_SELECT: 17)," &
"(NO_OVERSHIFT: 1 to 17)" &
")";
```

7.7 BSDL package for high speed JTAG

7.7.1 Description

The information provided above showed how to document features of a device implementing IEEE 1149.10 high speed JTAG using the existing syntax of BSDL. BSDL has been defined (see IEEE Std 1149.1) to be extensible using a mechanism called a “BSDL Extension”. A BSDL Extension for describing additional features of High Speed JTAG is given here.

The extension mechanism chosen for describing devices is based on the definition of a very high-speed integrated circuit (VHSIC) High-Level Design Language (VHDL) package with the name “STD_1149_10_2017”, which contains the definitions of attributes that will be used to supply relevant data. Therefore, a compliant device BSDL will contain an additional “use” statement appearing just after the “standard use statement” as in this excerpt of a BSDL file:

```
…
use STD_1149_1_2013.all; -- Standard ‘use’ statement
use STD_1149_10_2017.all; -- BSDL Extension for High Speed JTAG
…
```

NOTE—Refer to IEEE Std 1149.1-2013, B.8.5, Use Statement, for precise information regarding references to additional packages.

Copyright © 2017 IEEE. All rights reserved.

Utilization of the extension mechanism of BSDL ensures that high speed JTAG information can be supplied to applications that are cognizant of this functionality without hindering other applications that may not be aware of this functionality. Non-cognizant applications will simply ignore the extension.

The VHDL package STD_1149_10_2017 contains the definition of additional attributes used to complete the description of the High Speed JTAG. The content of this VHDL package is given as the specification.

7.7.2 Specifications

Syntax

```
Package STD_1149_10_2017 is -- Attribute definitions for HSJTAG
use STD_1149_1_2013.all; -- Refer to BSDL definitions

attribute CONFORMANCE_1149_10 : BSDL_Extension;
attribute HSTAP : BSDL_Extension;
attribute PACKET_MAP : BSDL_Extension;
attribute CONTROL_CHARS : BSDL_Extension;
attribute SCAN_CHANNEL_ASSOCIATION : BSDL_Extension;

end STD_1149_10_2017;

Package Body STD_1149_10_2017 is
use STD_1149_1_2013.all; -- Refer to BSDL definitions

end STD_1149_10_2017;
```

This VHDL package is “read-only” and may be maintained within a given system in the same location as the standard package STD_1149_1_2013.

8. Channel bonding

8.1 Optimizing bandwidth

This subclause describes this standard’s approach to increasing bandwidth by using more than one TX/RX pair per HSTAP.

8.1.1 Description

Two approaches can be used to aggregate bandwidth for an integrated circuit or stacked die IC. One method would be to simply include multiple HSTAP and PEDDAs, each with a single TX/RX SERDES pair. Such an approach would allow physically locating the HSTAP and PEDDA in different quadrants of a die, and would be useful for cases where it is difficult to route all of the test traffic to a single location on the die or when there are different test partitions on the die and can be tested concurrently through these independent ports. This method; however, is problematic when the additional bandwidth is needed for a single test partition. A second approach to get additional bandwidth, described in this clause, is channel bonding, using more than one TX and RX SERDES pair per HSTAP. In this approach, the test content is shared (striped) across the TX/RX pairs into the single HSTAP. Striping can include any number of lanes, and the striping is done on a packet frame granularity. That is, a single packet is striped across the different lanes with each subsequent frame coming from the next lane in a rotating pattern. Channel bonding is optional. However, if it is implemented then it must be implemented according to the rules and permissions of this subclause.

Figure 49 shows four SERDES lanes connected from an ATE to the HSTAP. Lanes all have slight skew due to distance and differences in clocking. In order to accomplish this channel bonding, and eliminate this skew, four BOND_CHAR characters must be sent such that the HSTAP design can recognize this special character and synchronize the internal gearboxes of each lane. This synchronization must be done every so many characters depending on the SERDES and technology used. Software tools that format the packet frames have to take into account the HSTAP attribute BOND_1149_10 parameter's maximum byte specification and insert four BOND_CHAR symbols on each lane before the maximum value is reached by a receiver.

Copyright © 2017 IEEE. All rights reserved.

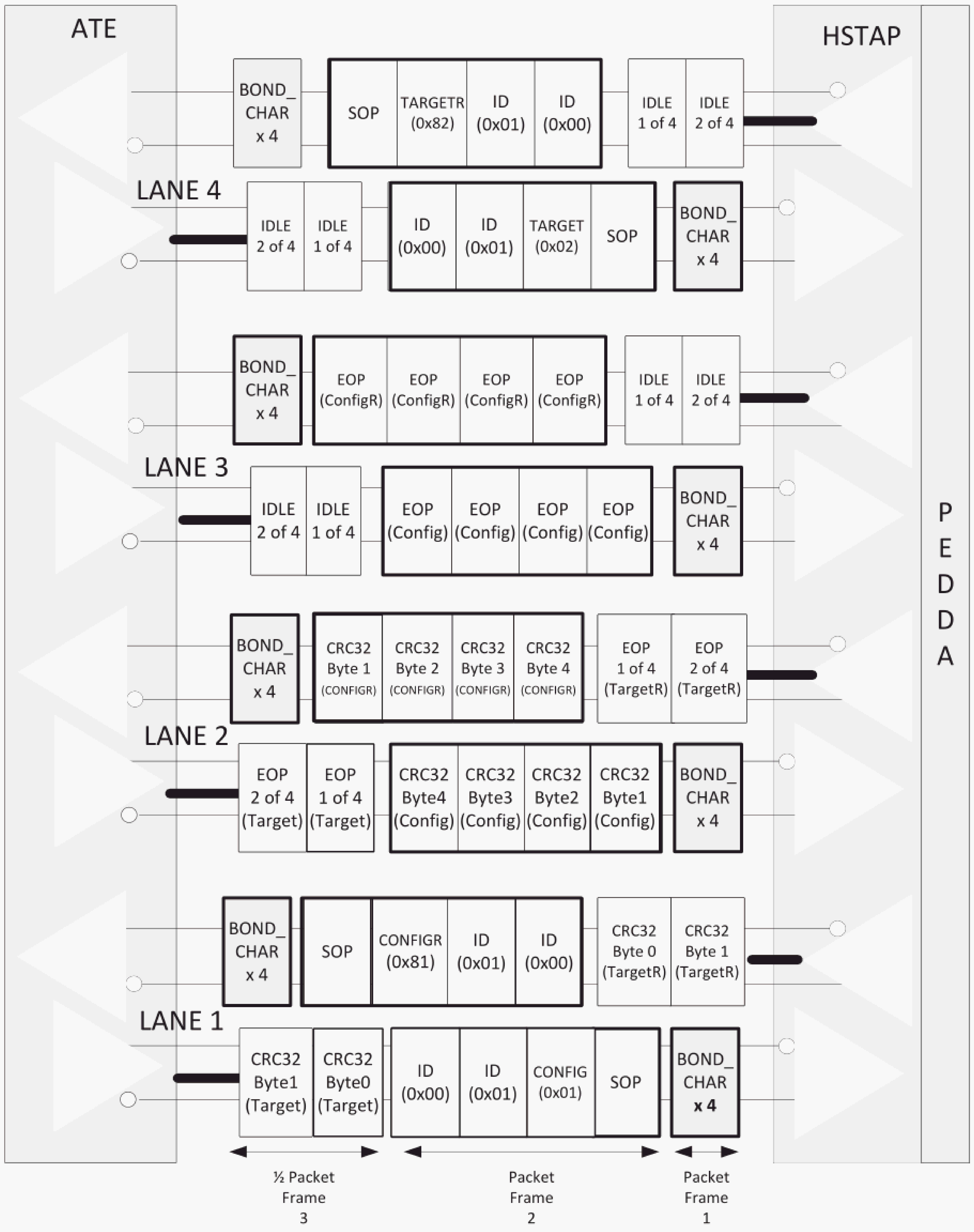

Figure 49—Four lane HSTAP

Figure 50 shows the transmitted striped date from the ATE but in a linear format. The four BOND characters are sent to lane1, lane2, lane3, and lane 4. This is followed by the first frame of a CONFIG

Copyright © 2017 IEEE. All rights reserved.

packet sent to lane1 and the CRC32 sent to lane2 followed by the four EOP sent to lane3. Then the ATE transmits the SOP and TARGET_ID on lane4, followed by the CRC32 for the TARGET_ID packet on lane1, followed by the four EOP on lane2, and IDLE characters on lane3 and lane4.

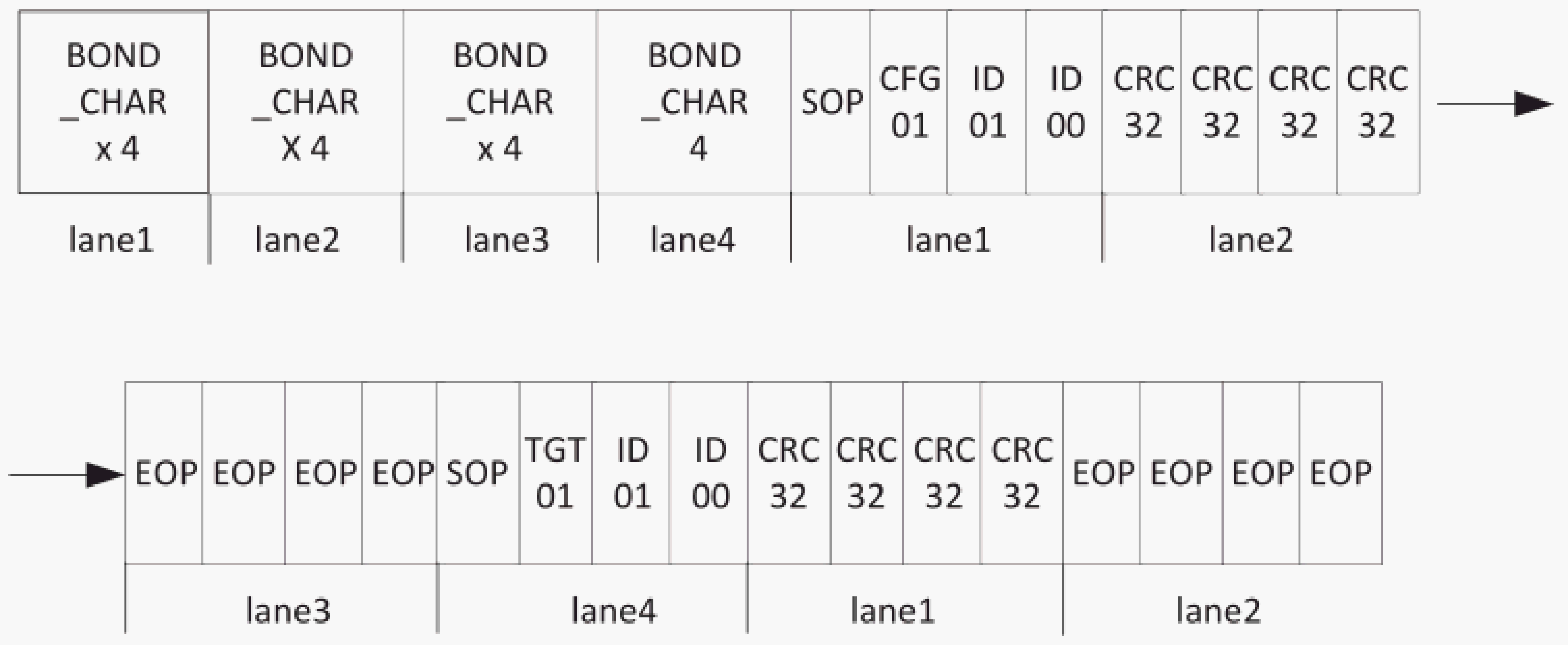

Figure 50—Striping of data alternative view

8.1.2 Specifications

Rules

a) An HSTAP designed to support channel-bonding shall send and receive packets as defined in Clause 6 using striping.
b) Any HSTAP that uses the same PEDDA as identified by the PEDDA_NAME attribute of 7.3.2 shall be available for channel-bonding.
c) Channel bonding shall be initialized after the receipt of the BOND packet and receipt of four sequential BOND_CHAR sent concurrently to each of the lanes participating.
d) The response from an HSTAP to the four BOND_CHAR characters shall be four BOND_CHAR characters if the lane is ready for channel bonding and the ERROR_CHAR followed by three data bytes otherwise.
e) The design of an HSTAP with channel bonding support shall be such that the HSTAP sends a packet frame of IDLE characters (4) and accepts the packet frame of IDLE characters when there is no data to be transmitted.

NOTE—Striping continues after the packet frame of four EOP such that each lane receives a packet frame of IDLE characters until the first lane is encountered in the round-robin sequence.

8.1.3 Examples

The following is an example of the HSTAP attribute when channel bonding is used. Four lanes are enumerated and the HSTAP_NUM with a value of zero becomes the first lane for the purposes of channel bonding.

```
attribute HSTAP of EXBOND : entity IS
"(" &
"(HSTAP_NUM : 0, " &
" PEDDA_NAME          : PEDDA1, "&
"  TX_1149_10  TXP(0)      :  500.0E-3,  800.0E-3,"&    --  tx+  port
min/max
"  RX_1149_10  RXP(0)      :  500.0E-3,  800.0E-3,"  &  --  rx+  port
min/max
" ENCODING_1149_10 : 8B_10B, "&                        --  Encoding
" DATARATE_1149_10 : 3.124E9, 3.126E9,"& --  min and max bit rate
" COMPLIANCE_VIA_TAP :  FALSE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&

" LATENCY_1149_10  : 3.0E-9, 4.0E-9, " & -- latency from RX to TX
" BOND_1149_10    : 1.0E05, 1.0E07  " &
")" &
")," &
"(" &
"(HSTAP_NUM : 1, " &
" PEDDA_NAME         : PEDDA1, "&
"  TX_1149_10  TXP(1)     :  500.0E-3,  800.0E-3,"&    --  tx+  port
min/max
"  RX_1149_10  RXP(1)     :  500.0E-3,  800.0E-3,"  &  --  rx+  port
min/max
" ENCODING_1149_10 : 8B_10B, "&                        --  Encoding
" DATARATE_1149_10 : 3.124E9, 3.126E9,"& --  min and max bit rate
" COMPLIANCE_VIA_TAP :  FALSE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&

" LATENCY_1149_10  : 3.0E-9, 4.0E-9," & -- latency from RX to TX
" BOND_1149_10    : 1.0E05, 1.0E07  " &
")" &
")," &
"(" &
"(HSTAP_NUM : 2, " &
" PEDDA_NAME         : PEDDA1, "&
"  TX_1149_10  TXP(2)     :  500.0E-3,  800.0E-3,"&    --  tx+  port
min/max
"  RX_1149_10  RXP(2)     :  500.0E-3,  800.0E-3,"  &  --  rx+  port
min/max
" ENCODING_1149_10 : 8B_10B, "&                        --  Encoding
" DATARATE_1149_10 : 3.124E9, 3.126E9,"& --  min and max bit rate
" COMPLIANCE_VIA_TAP :  FALSE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&
" LATENCY_1149_10  : 3.0E-9, 4.0E-9," & -- latency from RX to TX
" BOND_1149_10    : 1.0E05, 1.0E07  " &
")" &
")," &
"(" &
"(HSTAP_NUM : 3, " &
" PEDDA_NAME         : PEDDA1, "&
"  TX_1149_10  TXP(3)     :  500.0E-3,  800.0E-3,"&    --  tx+  port
min/max
"  RX_1149_10  RXP(3)     :  500.0E-3,  800.0E-3,"  &  --  rx+  port
min/max
" ENCODING_1149_10 : 8B_10B, "&                        --  Encoding
" DATARATE_1149_10 : 3.124E9, 3.126E9,"& --  min and max bit rate
" COMPLIANCE_VIA_TAP :  FALSE,"&
" IDLE_CHAR_REQUIRED :  TRUE,"&

" LATENCY_1149_10  : 3.0E-9, 4.0E-9," & -- latency from RX to TX
" BOND_1149_10    : 1.0E05, 1.0E07  " &
")," &
"( TXCLOCK_1149_10 TRANSMITCLK: 2.5E0, 4.0E0, " &
   "  1.25E6, 1.256E6 )" &
")" ;
```

Copyright © 2017 IEEE. All rights reserved.

9. PDL

9.1 PDL Overview

This subclause provides the overview of how IEEE 1149.1 PDL is used within this standard.

9.1.1 Description

IEEE 1149.10 PDL uses the grammar of IEEE 1149.1 PDL level 0 or level 1 as a baseline and defines additional commands specific to IEEE 1149.10.

9.1.2 Specifications

Rules

a) Except as defined here, IEEE Std 1149.10-2017 shall follow the rules of IEEE Std 1149.1-2013, Annex C.

9.1.3 Examples

In the example code below, the iProc reset_and_scan can be re-used for IEEE 1149.10. The iTMSReset and iTRST ON/OFF commands document the iRESET packet and the iApply can be converted to the CH-Select packet and SCAN packet.

```
# File IO.pdl
#Set PDL Level
iPDLLevel 0 -version STD_1149_1_2013
iProcGroup IO

iProc reset_and_scan -export -TMSreset -TRSTreset {
iTMSReset
iTRST ON
iTRST OFF
iWrite test_enable
iRead  ready
iApply
}
…
…
<EOF>

#File P1149_10_TOP.pdl
iSource IO.PDL
iPDLLevel 0 -version STD_1149_10_2017

#Configure and enumerate first two IEEE 1149.10 circuits
iConfig 0x0001
iConfig 0x0002

#Target circuit with Target_ID of 0x0001
iTarget 0x0001
```

Copyright © 2017 IEEE. All rights reserved.

```
# re-use the IEEE 1149.1 PDL as IEEE 1149.10 packets
iCall U1.IO(1).reset_and_scan

...

 <EOF>
```

9.2 iConfig command

9.2.1 Description

The iConfig PDL command is used to specify a Target_ID for an un-configured IEEE 1149.10 circuit. A tool that understands IEEE 1149.10 PDL would convert this command to the CONFIG packet and expect a CONFIGR packet in response. The -nofail parameter is used to specify that the CONFIGR packet contents of the CONFIGR response packet should be ignored. The -noResponse parameter is used to specify that no CONFIGR packet should be expected. The -nofail and -noResponse parameters are standardized parameters for user convenience for troublesome first silicon. A PEDDA is required to return a response for compliance.

9.2.2 Specifications

Syntax

<iConfig_cmd> ::= **iConfig** <Target_ID> | <Target_ID> <iConfig_param> <ct>

<Target_ID> ::= <bin_num> | <hex_num> | <dec_num>
<iConfig_param> ::= **-nofail | -noResponse**

Rules

a) Target_ID shall be a 16-bit value greater than zero.
b) When -nofail is specified, a CONFIGR response packet contents shall be ignored.
c) When -noResponse is specified, a CONFIGR response packet shall not be expected.

9.2.3 Examples

```
#Set PDL Level
iPDLLevel 0 -version STD_1149_10_2017

#Configure and enumerate first IEEE 1149.10 circuit
iConfig 0x0001

#Configure and enumerate the next IEEE 1149.10 circuit,
   without regard to the response packet
iConfig 0x0002 -nofail
```

Copyright © 2017 IEEE. All rights reserved.

9.3 iTarget command

9.3.1 Description

The iTarget PDL command is used to specify a given IEEE 1149.10 circuit by TARGET_ID. A tool that understands IEEE 1149.10 PDL would convert this command to the TARGET packet and expect a TARGETR packet in response. The -nofail parameter is used to specify that the TARGETR packet contents should be ignored. The -noResponse packet is used to specify that the TARGETR response should not be expected due to some defect or scenario preventing responses.

9.3.2 Specifications

Syntax

<iTarget_cmd> ::= **iTarget** <Target_ID> | <Target_ID> <iTarget_param> <ct>

<Target_ID> ::= <bin_num> | <hex_num> | <dec_num>
<iTarget_param> ::= **-nofail | -noResponse**

Rules

a) Target_ID shall be a 16-bit value greater than zero and equal to a previous iConfig Target_ID.
b) When -nofail is specified, a TARGETR response packet contents shall be ignored.
c) When -noResponse is specified, a TARGETR response packet shall not be expected.

9.3.3 Examples

```
#Set PDL Level
iPDLLevel 0 -version STD_1149_10_2017

#Configure and enumerate first two IEEE 1149.10 circuits
iConfig 0x0001
iConfig 0x0002

#Target circuit with Target_ID of 0x0001
iTarget 0x0001

# do some operations with target 0x0001
…

#Target second circuit with Target_ID of 0x0002,
#which de-selects the first device and ignore the contents
#of the TARGETR
iTarget 0x0002 -noFail

# ignore the lack of TARGETR response

iTarget 0x0002 -noResponse
```

Copyright © 2017 IEEE. All rights reserved.

9.4 iReset10 command

9.4.1 Description

The iReset10 PDL command specifies the RESET packet, which asserts/de-asserts various reset types of the IEEE 1149.10 PEDDA. The iReset10 has some overlap with the IEEE 1149.1 iReset and iTRST PDL commands in that the -Reset10_Assert parameter documents a packet that asserts the Reset* equivalent RESET10* signal and the -TRST10_On parameter asserts the TRST* internal signal. When re-using IEEE 1149.1 PDL code with a IEEE 1149.10 interface, the iReset and iTRST commands also generate the RESET10 packet.

A RESETR packet is expected as a response unless the -noResponse parameter is set. The contents of the RESETR packet is ignored when the -nofail parameter is set.

The signal RESET10* is momentarily asserted when the RESET packet TYPE field bit 0 is a 1 [see rule 6.4.2 c)] so there is no equivalent command necessary to de-assert it. The value is not persistent from one PDL command to the next. Similarly the -TargetRST_Assert parameter causes bit 2 of the TYPE field to be asserted to clear the TARGET_ID. It is automatically de-asserted and therefore also non-persistent from one PDL command to the next. The TRST10 is a toggle function due to legacy IEEE 1149.1 reasons. TRST10 remains persistent from one PDL command to the next and must be de-asserted with -TRST10_off.

The -Type parameter allows the user to specify user-defined bits of the TYPE field directly.

9.4.2 Specifications

Syntax

<iReset10_cmd> ::= **iReset10** <iResetSpec> | <iResetSpec> <iReset10_param> <ct>

<iResetSpec> ::= <Reset10_val> | <TRST10_val> |
<TargetReset_val> | <TypeSpec> | <iResetSpec>
<Reset10_val> ::= **-Reset10_Assert**
<TRST10_val> ::= **-TRST10_on** | **-TRST10_off**
<TargetReset_val > ::= **-TargetRST_Assert**
<TypeSpec > ::= **-Type** <bin_num> | <hex_num> | <dec_num>
<iReset10_param> ::= **-nofail | -noResponse**

Rules

a) The -Reset10_Assert parameter shall document a RESET packet with at least the RESET10 bit asserted.
b) The -TRST10_On parameter shall document a RESET packet with at least the TRST10 bit asserted.
c) The -TargetRST_Assert parameter shall document a RESET10 packet with at least the TARGET_ID bit asserted.
d) When -nofail is specified, a RESETR response packet is expected but the contents of the packet are ignored.
e) When -noResponse is specified, a RESETR response packet is not expected.

9.4.3 Examples

```
#Set PDL Level
```

Copyright © 2017 IEEE. All rights reserved.

```
iPDLLevel 0 -version STD_1149_10_2017

#Configure and enumerate first three IEEE 1149.10 circuits
iConfig 0x0001
iConfig 0x0002
iConfig 0x0003

#Target device 0x0003
iTarget 0x0003

#Reset device 0x0003 PEDDA and toggle TRST10
iReset10 –Reset10_Assert –TRST10_on
iReset10 -TRST10_off

#this is equivalent to IEEE 1149.1 commands
iTMSReset
iTRST ON
iTRST OFF



#this is equivalent to setting the value directly to all zeros
iReset10 -Type 0x0000

iReset10 -TRST_On
iReset10 -Reset10_Assert   # TRST is still asserted
iReset10 -TRST_Off

# clear Target_ID
iReset10 -TargetRST_Assert
#Re-Configure and target #3 enumerate
iConfig 0x0003

#Target device 0x0003
iTarget 0x0003
```

9.5 iRaw command

9.5.1 Description

The iRaw command documents the RAW packet and sets the PEDDA into loopback mode where data is received at the IEEE 1149.10 interface without any packet format and retransmitted. The raw data sent is encoded as specified in the HSTAP attribute (see 6.5.2 and 7.3).

9.5.2 Specifications

Syntax

<iRaw_cmd> ::= **iRaw** | **iRaw** <iRaw_param> <ct>

<iRaw_param> ::= **-nofail | -noResponse**

Copyright © 2017 IEEE. All rights reserved.

Rules

a) When -nofail is specified, a RAWR response packet is expected but the contents of the packet are ignored.
b) When -noResponse is specified, a RAWR response packet is not expected.

9.5.3 Examples

```
#Target PEDDA of device 0x0001
iTarget 0x0001

#Set device 0x0001 into raw mode
iRaw
#Execution of PDL will not function after this
#command until the HSTAP/PEDDA are re-configured
```

9.6 iBond command

9.6.1 Description

The iBond command is used to enter a multi-lane channel bonded mode. Software that understands IEEE 1149.10 PDL would generate a BOND packet when the iBond command is specified.

9.6.2 Specifications

Syntax

<iBond_cmd> ::= **iBond** <Lane> | <Lane> <iBond_param> <ct>

<Lane> ::= <bin_num> | <hex_num> | <dec_num>
<iBond_param> ::= **-nofail | -noResponse**

Rules

a) <Lane> shall indicate an even number of HSTAP lanes that will participate in channel bonding.
b) When -nofail is specified, a BONDR response packet is expected but the contents of the packet are ignored.
c) When -noResponse is specified, a BONDR response packet is not expected.

9.6.3 Examples

```
#Target PEDDA of device 0x0001
iTarget 0x0001

#Bond 2 lanes for use
iBond 0x2
```

Copyright © 2017 IEEE. All rights reserved.

9.7 Standardized PDL procedures

9.7.1 Description

When an IEEE 1149.1 TAP is required for enabling IEEE 1149.10 compliance then there are two PDL routines that are required such that a software tool can perform the necessary scan sequences to enable compliance or disable it. The two iProcs are *Enable_1149_10* and *Disable_1149_10*. IEEE 1149.10 also specifies a *Disable_Rawmode* procedure.

9.7.2 Specifications

Rules

a) The *Enable_1149_10* and *Disable_1149_10* PDL procedures shall be a level 0 or level 1 IEEE 1149.1 iProc.
b) The steps in the iProc *Enable_1149_10* shall document how to enter compliance by configuring the HSTAP and assert the signal Enable_1149_10.
c) The steps in the iProc *Disable_1149_10* shall document how to exit compliance and de-assert Enable_1149_10.

NOTE—The iProc *Disable_1149_10* is a valid IEEE 1149.1 PDL procedure, however, the access to the scan-channels described is via the P1149.10 HSTAP.

d) The *Disable_Rawmode* PDL procedure shall be a level 0 IEEE 1149.1 iProc.

9.7.3 Examples

Figure 7 shows an example TDR that is used to set the Enable_1149_10 signal. A snippet of BSDL for an IC that uses this TDR is as follows:

```
attribute REGISTER_MNEMONICS of MyIC : entity is
   "Compvalue (Enable_1149_10  (0B1) < Enable compliance >, "&
   "           Disable_1149_10 (0B0) < Disable compliance >) ";


 attribute REGISTER_FIELDS of MyIC : entity is
 "MyTDR[1] ( "&
 "(Compliance [1] IS (0) PORRESET(Compvalue(Disable_1149_10) )
   )"&
 "         )";
```

The *Enable_1149_10* and *Disable_1149_10* PDL procedure examples are as follows:

```
   iPDLLevel 0 -version STD_1149_1_2013
   iProcGroup MyIC

   iProc Enable_1149_10{ } {

   iWrite Compliance Enable_1149_10
   iApply

   }
```

Copyright © 2017 IEEE. All rights reserved.

```
   iProc Disable_1149_10{ } {

   iWrite Compliance Disable_1149_10
   iApply

   }
```

When a TAP is present then it is recommended to provide a disable feature to take the HSTAP interface out of RAW mode. The PDL *Disable_Rawmode* procedure clears the RAWMODE set by the RAW packet.

```
attribute REGISTER_MNEMONICS of MyRawModeIC: entity is
   "Rawvalue  (Set      (0B1) < Set Rawmode >, "&
   "           Clear    (0B0) < Clear Rawmode set by IEEE 1149.10
   >) ";


 attribute REGISTER_FIELDS of MyRawModeIC : entity is
 "MyTDR2[1]  ( "&
 "(Rawmode [1] IS  (0)  PORRESET(Rawvalue(Clear) )   )"&
 "          )";
```

The PDL associated with MyRawModeIC would be as follows:

```
   iPDLLevel 0 -version STD_1149_1_2013
   iProcGroup MyRawModeIC


   iProc Disable_Rawmode { } {

   iWrite Rawmode Clear
   iApply

   }
```

10. Compliance verification

10.1 Overview

This clause provides a basic description of how compliance to the standard can be verified.

10.1.1 Description

This description is not prescribing how to test an IC but an outline of the steps required to validate compliance, ideally pre-silicon. This section is informative, hence, there are no rules, recommendations, or permissions.

10.1.2 Initialization steps

Step 1. Read the associated BSDL file.

Step 2. Perform the power-up sequence to the HSTAP and PEDDA circuitry.

Copyright © 2017 IEEE. All rights reserved.

The following two steps are independent, as each provides a way for the HSTAP to be enabled for IEEE 1149.10 compliance.

Step 3. If an IEEE 1149.1 TAP is documented in BSDL and the COMPLIANCE_VIA_TAP parameter of the HSTAP attribute is true, perform the following steps:

a. For a given PEDDA_NAME, configure tester channels according to the specification in the HSTAP attribute for RX_1149_10 and TX_1149_10. Supply any clocks as specified in the <clocks_string> of the HSTAP attribute.
b. Enable IEEE 1149.1 compliance using the IEEE 1149.1 COMPLIANCE_ENABLE attribute, if any.
c. Follow the steps documented in the PDL procedure *Enable_1149_10* to enable IEEE 1149.10 compliance on the signals provided in the HSTAP attribute.
d. If the HSTAP parameter IDLE_CHAR_REQUIRED is TRUE, send IDLE characters as specified by the IDLE_CHAR in the CONTROL_CHARS attribute and continue sending unless sending a CONTROL_CHAR or the packets described in Clause 6.
e. Go to Step 5.

Step 4. For a given PEDDA_NAME, configure tester channels according to the specification in the HSTAP attribute for RX_1149_10 and TX_1149_10. Supply any clocks as specified in the <clocks_string> of the HSTAP attribute.

a. If the HSTAP parameter IDLE_CHAR_REQUIRED is TRUE, send IDLE characters as specified by the IDLE_CHAR in the CONTROL_CHARS attribute and continue sending unless sending a CONTROL_CHAR or the packets described in Clause 6.
b. If a COMPLIANCE_CHAR is specified in the CONTROL_CHARS attribute, transmit the COMPLIANCE_CHAR to the HSTAP.
c. Go to Step 5.

Step 5. Ready to receive CONFIG packet.

10.1.3 Verification of CONFIG, TARGET, and RESET

Step 1. Set a variable n to 0x0001.

Step 2. Send the CONFIG packet with a TARGET_ID of n using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10. Expect a response with the same TARGET_ID.

Step 3. Send the CONFIG packet with a TARGET_ID of n+1 using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10. Expect a CONFIG packet response with the TARGET_ID of n+1.

NOTE—Because the circuit under test is configured for TARGET_ID of n, CONFIG packets received when the circuit under test TARGET_ID is non-zero should be forwarded.

Step 4. Send the TARGET packet with a TARGET_ID of n using the encoding specified in the HSTAP attribute parameter called ENCODING_1149_10. Expect a TARGETR response with the same TARGET_ID.

Step 5. Send the RESET packet with the TYPE field set to 0x04. Expect a response RESETR packet with the same TYPE field. The TARGET_ID should now be set to 0x0000.

Step 6. Repeat Step 2 through Step 5 for all possible TARGET_IDs by incrementing n by one for each iteration.

Copyright © 2017 IEEE. All rights reserved.

10.1.4 Verification response packets are forwarded

Step 1. Send the CONFIG packet with a TARGET_ID of 0x0001 using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10. Expect a CONFIGR response with the same TARGET_ID.

Step 2. Send the TARGET packet with a TARGET_ID of 0x0001 using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10. Expect a TARGETR response with the same TARGET_ID.

Step 3. Send a TARGETR response packet with a TARGET_ID of 0x0002. Expect a TARGETR response packet with a TARGET_ID of 0x0002. Repeat for all response packets in Table 2.

NOTE—Because all packets are examined by the PEDDA, the HSTAP to PEDDA connections are well tested at the end of Step 3. 2^{16}-1 CRC32s have been received at the PEDDA and sent back out to the HSTAP. Further validation can be done for those seeking additional verification by sending random scan (PAYLOAD) data in a SCANR response packet as the input to the HSTAP thus further validating the HSTAP to PEDDA interface.

10.1.5 Verification of CRC32 check and ERROR_CHAR

Step 1. Send the RESET packet with the TYPE field set to 0x00 but with the CRC32 value calculated incorrectly. Expect a complete RESETR packet response with the same TYPE field followed by a frame with an ERROR_CHAR or an incomplete RESETR packet response ending in a frame with an ERROR_CHAR. Repeat with a different CRC32 value calculated incorrectly for all packets in Table 1.

10.1.6 Verification of IDLE frames

Step 1. Send the RESET packet with the TYPE field set to 0x00 as single frames interleaved with a frame of IDLE characters. Expect a RESETR packet response with the same type field.

10.1.7 Verification that a bad CONFIG packet does not alter the TARGET_ID

Step 1. Send the RESET packet with the TYPE field set to 0x04 as single frames interleaved with a frame of IDLE characters. Expect a RESETR packet response with the same type field.

Step 2. Send the CONFIG packet with a TARGET_ID of 0x0001 using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10 and an incorrectly calculated CRC32 value. Expect a CONFIGR response with the same TARGET_ID and at least one frame that contains the ERROR_CHAR.

10.1.8 Validate that a CONFIG packet with a bad CRC32 value does not change the TARGET_ID

Step 1. Send the CONFIG packet with a TARGET_ID of 0x0001 using the encoding specified in the HSTAP attribute parameter ENCODING_1149_10. Expect a CONFIGR response with the same TARGET_ID.

10.1.9 Validate CH-SELECT and SCAN packet for unique CH_SELECTs

Step 1. Determine the first scan-channel with 1) a unique CH_SELECT value in a SCAN_GROUP and 2) the scan-channel integer is not listed in the NO_OVERSHIFT attribute.

NOTE—These steps perform over-shifting of a scan-channel, hence, those scan-channels that do not allow over-shifting should be excluded from these steps. The steps assume that the scan-channels are IEEE 1149.1

Copyright © 2017 IEEE. All rights reserved.

or IEEE 1500 [B1] compliant and, therefore, there are an even number of inversions in the scan-channel, otherwise the expected sentinel value requires adjustment.

Step 2. Set a variable j to 0xAA

.

Step 3. Send the CH-SELECT packet for the scan-channel and the SCAN_GROUP number that the selected scan-channel is in.

Step 4. Expect a CH-SELECTR packet response with the same scan-channel selected and calculated CRC32.

Step 5. Send the SCAN packet for the scan-channel selected. Use a scan-in sentinel value of j as data to be shifted first asserting C and S in the ICSU field. Set the SCAN packet Cycle-Count to the length of the scan-channel plus eight.

Step 6. Expect a SCANR packet response with the j value as the first received value. Validate known scan-channel capture values that may be described in BSDL.

Step 7. Repeat Step 3 to Step 6 setting variable j to 0x55.

Step 8. Repeat Step 2 to Step 7 for the next scan-channel with 1) a unique CH_SELECT value in a SCAN_GROUP and 2) the scan-channel integer is not listed in the NO_OVERSHIFT attribute.

10.1.10 Validate CH-SELECT and SCAN packets that have a common CH_SELECT

Step 1. Determine the first SCAN_GROUP with scan-channels that share a common CH_SELECT value and at least one scan-channel integer in the SCAN_GROUP is not listed in the NO_OVERSHIFT attribute.

NOTE—These steps perform over-shifting of a group of scan-channels, hence, those scan-channels that do not allow over-shifting should be excluded from these steps.

Step 2. Set a variable j to 0xAA.

Step 3. Send the CH-SELECT packet for the scan-channels that are not listed in the NO_OVERSHIFT attribute.

Step 4. Expect a CH-SELECTR packet response with the same scan-channels selected and calculated CRC32.

Step 5. Set a variable len to the length of the shortest scan-channel in the currently selected SCAN_GROUP.

Step 6. Set a variable cyclecount to len plus eight. Use a scan-in sentinel value of j as data to be shifted in first in each scan-channel participating (each scan-channel asserted in the CH-SELECT packet) and create a SCAN packet asserting C in the ICSU field.

Step 7. Set the SCAN packet Cycle-Count to the variable cyclecount. Send the SCAN packet for the scan-channels selected asserting S in the ICSU field.

Step 8. Expect a SCANR packet response with the j value as the first received data on each scan-channel with a length of len. Validate known scan-channel capture values that may be described in BSDL.

Step 9. Set a variable newlen to the length of the next longest scan-channel in the SCAN_GROUP. Set the cyclecount variable to newlen minus len. Set the variable len to newlen. De-assert C in the ICSU field for subsequent SCAN packets.

Copyright © 2017 IEEE. All rights reserved.

Step 10. Repeat Step 7 to Step 9 until all scan-channels in the SCAN_GROUP that are not in the NO_OVERSHIFT list are validated.

Step 11. Repeat Step 5 to Step 10 setting variable j to 0x55.

10.1.11 Validate the RAW packet

Step 1. Send the RAW packet.

Step 2. Expect a RAWR packet response.

Step 3. Send the CONFIG packet with a TARGET_ID of 0x0001.

Step 4. Expect a CONFIG packet with the same TARGET_ID.

NOTE—Expect a CONFIG packet returned here and not the CONFIGR as the interface should remain in constant raw mode loopback.

Step 5. Perform a power-on reset of the HSTAP and PEDDA such that loopback mode can be exited.

10.1.12 User-defined verification

Step 1. Perform user-defined verification steps using PDL and/or automatic test-pattern generation (ATPG) patterns.

Copyright © 2017 IEEE. All rights reserved.

Annex A

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] IEEE Std 1500™-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits.[7], [8]

[7] The IEEE standards or products referred to in Annex A are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.
[8] IEEE publications are available from the Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).

Copyright © 2017 IEEE. All rights reserved.

IEEE STANDARDS ASSOCIATION

IEEE

Consensus

WE BUILD IT.

Connect with us on:

Facebook: https://www.facebook.com/ieeesa

Twitter: @ieeesa

LinkedIn: http://www.linkedin.com/groups/IEEESA-Official-IEEE-Standards-Association-1791118

IEEE-SA Standards Insight blog: http://standardsinsight.com

YouTube: IEEE-SA Channel

IEEE
standards.ieee.org
Phone: +1 732 981 0060   Fax: +1 732 562 1571
© IEEE