IEEE Std 1394a-2000
(Amendment to
IEEE Std 1394-1995)

IEEE Standard for a High Performance Serial Bus—Amendment 1

Sponsor

Microprocessors and Microcomputers Standards Committee
of the
IEEE Computer Society

Approved 30 March 2000

IEEE-SA Standards Board

Abstract: Amended information for a high-speed Serial Bus that integrates well with most IEEE standard 32-bit and 64-bit parallel buses is specified. This amendment is intended to extend the usefulness of a low-cost interconnect between external peripherals, as described in IEEE Std 1394-1995. This amendment to IEEE Std 1394-1995 follows the ISO/IEC 13213:1994 Command and Status Register (CSR) Architecture.

Keywords: bus, computers, high-speed Serial Bus, interconnect

---

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

Print: ISBN 0-7381-1958-X SH94821
PDF: ISBN 0-7381-1959-8 SS94821

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

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. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, 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.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received 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 interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: 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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Introduction

(This introduction is not part of IEEE Std 1394a-2000, IEEE Standard for a High Performance Serial Bus—Amendment 1.)

Standards development is an ongoing process and is, perhaps, never complete. In 1994, the working group responsible for IEEE Std 1394-1995, IEEE Standard for a High Performance Serial Bus, reluctantly elected to close the door to new material. Although many enhancements were well understood in principle (some were even making their way into contemporary silicon designs), significant work remained to document the details. Consensus emerged to publish the completed work and later to prepare an amendment. This document is that amendment—it extends and corrects facilities of Serial Bus.

In January 1996, an informal study group was convened by Gerald Marazas, Chair of the IEEE P1394 Working Group. The meeting was held in Dallas, TX, at the same time as a quarterly meeting of the nascent 1394 Trade Association. The topic was unfinished business in Serial Bus; brainstorming quickly identified six major areas of interest. Some of the areas readily resolved into clusters of related activity, which became other Serial Bus standards projects still active at the time of writing—IEEE P1394.1, Draft Standard for High Performance Serial Bus Bridges and IEEE P1394b, Draft Standard for a High Performance Serial Bus—Amendment 2. The topics that were deemed essentially complete (e.g., the alternate 4-pin cable and connector, the PHY arbitration enhancements, and miscellaneous corrections to the 1995 standard) were gathered together under the banner of IEEE P1394a. During the next month, the IEEE P1394a Study Group met to select a chair and draft a Project Authorization Request (PAR). The first official meeting of IEEE P1394a took place in October 1996; the working group continued to meet monthly until its last meeting in February 1998.

The working group organized the new effort as a amendment rather than a new Serial Bus standard intended to replace IEEE Std 1394-1995 entirely. This decision was based upon the belief that the changes in IEEE P1394a were localized to a few areas and that we would be able to complete our work rapidly if we did not have to reissue the entire standard. In retrospect this was an awkward choice. The reader who wishes to be informed of the current Serial Bus standard is forced to consult both the original standard and this amendment. The working group hopes that in the process of international standardization that it is possible to editorially combine the two documents into a single volume.

IEEE P1394a, Draft 2.0 failed the sponsor ballot conducted by the IEEE and generated a large number of comments. In an effort to resolve these comments and pave the way for a successful recirculation ballot, the Ballot Response Committee (BRC) was convened by the IEEE P1394a Chair, Peter Johansson. The BRC first met in the fall of 1998 and continued meeting into 1999 to complete a revision of the draft standard, which was resubmitted to the balloting pool for approval.

IEEE P1394a, Draft 3.0 passed its recirculation and the editor prepared IEEE P1394a, Draft 4.0 for submission to the IEEE-SA Standards Board for approval. The IEEE Standards Review Committee (RevCom) recommended disapproval; as a consequence, the draft was recirculated for its third ballot. The IEEE P1394a Ballot Response Committee believes that there are no substantive changes to the draft between all of these revisions.

IEEE P1394a, Draft 5.0 passed the third and final ballot and was approved as a standard at the 30 March 2000 meeting of the IEEE-SA Standards Board.

Patent notice

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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. A patent holder has filed a statement of assurance that it will grant a license under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Participants

The following is a list of voting members of the IEEE P1394a Working Group:

Peter Johansson, *Chair and Editor*

Prashant Kanhere, *Secretary*

Richard Baker
Steven R. Bard
Max Bassler
Joe Bennett
Vilas Bhade
Brad Bickford
Charles Brill
Mike Brown
David Brunker
Richard Churchill
Dan Colegrove
Claude A. Cruz
Bill Duckwall
Firooz Farhoomand
Steve Finch
Bill Frank
John Fuller
Eric Hannah
Yasumasa Hasegawa
Jerry Hauck
John Hill
Jack Hollins
David Johnson
Tom Jones
Marcus Kellerman
Al Kelley
Mark Knecht
David LaFollette
Steven Larky
Thang Le
Robert G. Liu
Hirokazu Mamezaki
Takashi Matsui
Keiji Miura
Bill Northey
Farrell Ostler
James Piccione
Bill Prouty
David Scott
John Smolka
John Ta
Ju-ching Tang
Motoyasu Tsunoda
Renard Ulrey
Sushant Verman
Colin Whitby-Strevens
Paul Wiener
David Wooten
Roy Yasoshima
Phil Young
Michael Zarreii
Peng Zhang

The following is a list of other major participants in the IEEE P1394a Working Group:

Kazuyuki Abe
Eric Anderson
Oleg Awsienko
Jim Busse
Ed Butler
Carissa Cheung
Alistair Coles
David Doman
Jim Doyle
Sagar Edara
Mike Eneboe
Lou Fasano
Taka Fujimori
James Gay
Sreekanth Godey
John Grant
Shinichi Hatae
Keith Heilmann
Burke Henehan
Joe Herbst
David Instone
Diana Klashman
Farrukh Latif
Paul S. Levy
Cyrus Momeni
Neil Morrow
Ganesh Murthy
Karl Nakamura
Takayuki Nyu
Kugao Ouchi
Bill Russell
Bradley Saunders
Hisato Shima
James Skidmore
Michael Sorna
Peter Teng
C. Brendan Traw
Tom Trodden
Kent Waterson
Lee Wilson
Calto Wong
Takao Yasuda
Patrick Yu

The following lists members of the Ballot Response Committee:

Steven Bard
Max Bassler
Sagar Edara
Firooz Farhoomand
Lou Fasano
John Fuller
Larry Getzin
Yasumasa Hasegawa
Jerry Hauck
David James
Peter Johansson
Hirokazu Mamezaki
David Scott
Michael Shinkarovsky
James Skidmore
Peter Teng
Tom Trodden
Renard Ulrey
Colin Whitby-Strevens
David R. Wooten

The following members of the balloting committee voted on this standard:

William B. Adams
Barbara P. Aichinger
Malcolm J. Airst
Harry A. Andreas
Keith D. Anthony
Steven R. Bard
Max Bassler
Vilas Bhade
Janos Biri
David Brearley
Charles Brill
Hakon Ording Bugge
Robert S. Crowder
Claude A. Cruz
Dante Del Corso
Stephen L. Diamond
Wayne P. Fischer
Martin Freeman
John Nels Fuller
Paul J. Fulton
Julio Gonzalez-Sanz
David B. Gustavson
Eric Hannah
Jerry Hauck
Donald N. Heirman
Burke Henehan
Jack Hollins
Venkat Iyer
Peter Johansson
Stephen Kempainen
Thomas M. Kurihara
David LaFollette
Tuvia Lamdan
Lawrence Lamers
Steven Larky
Gerald E. Laws
John V. Levy
Paul U. Lind
Robert G. Liu
Gerald A. Marazas
Joseph R. Marshall
Helen McGreal
Gene E. Milligan
Kiyoshi Miura
Klaus-Dieter Mueller
J. D. Nicoud
Daniel C. O'Connor
Roy Oishi
Florin Oprescu
Michael Orlovsky
Roman Orzol
Farrell Ostler
Granville Ott
Dan Paley
Elwood T. Parsons
Roy Reed
Gary S. Robinson
James W. Romlein
Bradley N. Saunders
Donald Senzig
Robert K. Southard
Larry Stein
Robert G. Stewart
Fred J. Strauss
Nobuaki Sugiura
Michael D. Teener
Michael G. Thompson
David W. Thompson
Brendan Traw
Paul Walker
Michael Wang
Colin Whitby-Strevens
Dave Wickliff
Ronald T. Wolfe
James Wolffe
David R. Wooten
Roy Yasoshima
Patrick W. Yu
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 30 March 2000, it had the following membership:

Donald N. Heirman, *Chair*
James T. Carlo, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal	James H. Gurney	James W. Moore
Mark D. Bowman	Richard J. Holleman	Robert F. Munzner
Gary R. Engmann	Lowell G. Johnson	Ronald C. Petersen
Harold E. Epstein	Robert J. Kennelly	Gerald H. Peterson
H. Landis Floyd	Joseph L. Koepfinger*	John B. Posey
Jay Forster*	Peter H. Lips	Gary S. Robinson
Howard M. Frazier	L. Bruce McClung	Akio Tojo
Ruben D. Garzon	Daleep C. Mohla	Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Donald R. Volzka, *TAB Representative*

Yvette Ho Sang
IEEE Standards Project Editor

National Electrical Code and NEC are both registered trademarks of the National Fire Protection Association, Inc.

Contents

IEEE Standard for a High Performance Serial Bus—Amendment 1

NOTE—The editing instructions contained in this amendment define how to merge the material contained herein into the existing base standard to form the comprehensive standard.

The editing instructions are shown in ***bold italic.*** Four editing instructions are used: change, delete, insert, and replace. ***Change*** is used to make small corrections in existing text or tables. The editing instruction specifies the location of the change and describes what is being changed by using ~~strikethrough~~ (to remove old material) and underscore (to add new material). ***Delete*** removes existing material. ***Insert*** adds new material without disturbing the existing material. Insertions may require renumbering. If so, renumbering instructions are given in the editing instruction. ***Replace*** is used to make large changes in existing text, subclauses, tables, or figures by removing existing material and replacing it with new material. Editorial notes will not be carried over into future editions because the changes will be incorporated into the base standard.

0. Introduction

0.1 Scope

IEEE Std 1394a-2000 is a full-use standard whose scope is to amend IEEE Std 1394-1995 by defining or clarifying features and mechanisms that facilitate management of Serial Bus resources, at reconfiguration or during normal operation, and by defining alternate cables and connectors that may be needed for specialized applications.

The following are included in IEEE Std 1394a-2000:

a) Cables and connectors for a 4-pin variant (from the 6-pin already standardized).
b) Standardization of the physical layer (PHY)/link interface, previously published as an informative annex (Annex J) in IEEE Std 1394-1995.
c) Performance enhancements to the PHY layer that are interoperable with the existing standard, e.g., a method to shorten the arbitration delay when the last observed Serial Bus activity is an acknowledge packet.
d) A redefinition of the isochronous data packet, transaction code A_{16}, to permit its use in either the asynchronous or isochronous periods.
e) More stringent requirements on the power to be supplied by a cable power source and a clarification of electrical isolation requirements.
f) Significant corrigenda and clarifications to the existing standard that do not clearly fall within any of the topics described above.

The preceding are arranged in no particular order.

0.2 Purpose

Experience with Serial Bus has revealed some areas in which additional features or improvements may result in better performance or usability. This amendment to IEEE Std 1394-1995 reflects the consideration of these features or improvements by a variety of users, and their refinement into generally useful facilities or features.

0.3 Document organization

IEEE Std 1394a-2000 contains this introduction, an overview, a list of definitions, an informative summary description, sections of technical specification, and application annexes. The new reader should read the informative summary in 3.9 and the clauses that precede it before the remainder of the document.

Most of the changes to IEEE Std 1394-1995, which begin with the editorial instructions, are preceded by background information that explains the scope and impact of the change. The background information should be read before implementing the change.

1. Overview

1.2 References

Insert the following in 1.2, in alphabetical order, and renumber footnotes:

ANSI/EIA 364-C-94, Electrical Connector/Socket Test Procedures Including Environmental Classifications.[1]

ANSI/EIA 364-06A-83 (Reaff 90), Contact Resistance Test Procedure for Electrical Connectors.
ANSI/EIA 364-09B-91, Durability Test Procedure for Electrical Connectors.
ANSI/EIA 364-13A-83 (Reaff 90), Mating and Unmating Forces Test Procedure for Electrical Connectors.
ANSI/EIA 364-17A-87, Temperature Life with or without Electrical Load Test Procedure for Electrical Connectors and Sockets.
ANSI/EIA 364-18A-84, Visual and Dimensional Inspection Procedure for Electrical Connectors.
ANSI/EIA 364-20A-83 (Reaff 90), Withstanding Voltage Test Procedure for Electrical Connectors, Sockets and Coaxial Contacts.
ANSI/EIA 364-21B-95, Insulation Resistance Test Procedure for Electrical Connectors.
ANSI/EIA 364-23A-85, Low Level Contact Resistance Test Procedure for Electrical Connectors.
ANSI/EIA 364-27B-96, Mechanical Shock (Specified Pulse) Test Procedure for Electrical Connectors.
ANSI/EIA 364-28C-97, Vibration Test Procedure for Electrical Connectors and Sockets.
ANSI/EIA 364-31A-83 (R90), Humidity for Electrical Connectors.
ANSI/EIA 364-32B-92, Thermal Shock Test Procedure for Electrical Connectors.
ANSI/EIA 364-41B-89, Cable Flexing Test Procedure for Electrical Connectors.
ANSI/EIA 364-46A-98, Microsecond Discontinuity Test Procedure for Electrical Connectors, Contacts and Sockets.
ANSI/EIA 364-65A-97, Mixed Flowing Gas.

IEC 60950 (1999-04), Safety of information technology equipment.[2]

IEC 61000-4-2 (1999-05) Edition 1.1, Electromagnetic compatibility (EMC)—Part 4-2: Testing and measurement techniques—Electrostatic discharge immunity test.

IEC 61883-1 (1998-02), Consumer audio/video equipment—Digital interface—Part 1: General.

[1]ANSI/EIA publications are available from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/).

[2]IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iec.ch/). IEC publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA. (http://www.ansi.org/)

Replace 1.5 with the following:

1.5 Service model

This standard uses a protocol model with multiple layers. Each layer provides services to the next higher layer and to Serial Bus management. These services are abstractions of a possible implementation; an actual implementation may be significantly different and still meet all the requirements. The method by which these services are communicated between the layers is not defined by this standard. The following four types of services are defined:

a) *Request service.* A request service is a communication from a layer to a lower or adjacent layer in order to request some action. A request may also communicate parameters that may or may not be associated with an action. A request may or may not be confirmed. A data transfer request usually triggers a corresponding indication on peer node(s). (Since broadcast addressing is supported on Serial Bus, it is possible for the request to trigger a corresponding indication on multiple nodes.)

b) *Indication service.* An indication service is a communication from a layer to a higher or adjacent layer in order to indicate a change of state or other event detected by the originating layer. An indication may also communicate parameters that are associated with the change of state or event. Indications are not necessarily triggered by requests; an indication may or may not be responded to by a response. A data transfer indication is originally caused by a corresponding request on a peer node.

c) *Response service.* A response service is a communication from a layer to a lower or adjacent layer in response to an indication; a response is always associated with an indication. A response may communicate parameters that indicate its type. A data transfer response usually triggers a corresponding confirmation on a peer node.

d) *Confirmation service.* A confirmation service is a communication from a layer to a higher or adjacent layer in order to confirm a request service; a confirmation is always associated with a request. A confirmation may communicate parameters that indicate the completion status of the request or that indicate other statuses. For data transfer requests, the confirmation may be caused by a corresponding response on a peer node.

If all four service types exist, they are related as shown by figure 1-2.

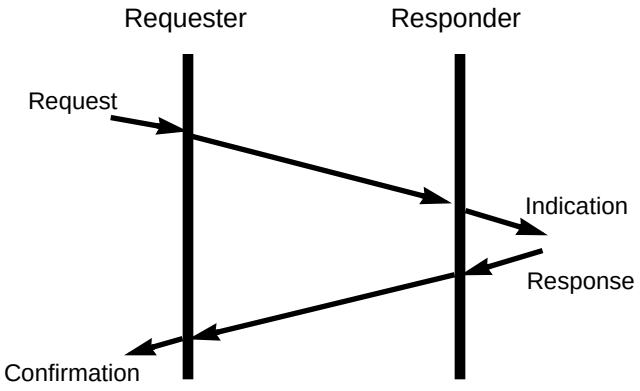


Figure 1-2—Service model

Replace 1.6 with the following:

1.6 Document notation

1.6.1 Mechanical notation

All mechanical drawings in this document use millimeters as the standard unit and follow ANSI Y14.2M-1992 [B1][3] and ANSI Y14.5M-1994 [B2] formats.

1.6.2 Signal naming

All electrical signals are shown in all uppercase characters and active-low signals have the suffix "*". For example, TPA and TPA* are the normal and inverted signals in a differential pair.

1.6.3 Size notation

This document avoids the terms *word, half-word,* and *double-word,* which have widely different definitions depending on the word size of the processor. In their place, processor-independent terms established by previous IEEE bus standards are used. These terms are illustrated in table 1-1.

Table 1-1 — Size notation examples

Size (in bits)	16-bit word notation	32-bit word notation	IEEE standard notation (used in this standard)
4	nibble	nibble	nibble
8	byte	byte	byte
16	word	half-word	doublet
32	long-word	word	quadlet
64	quad-word	double	octlet

Serial Bus uses big-endian ordering for byte addresses within a quadlet, and quadlet addresses within an octlet. For 32-bit quadlet registers, byte 0 is always the most significant byte of the register. For a 64-bit quadlet-register pair, the first quadlet is always the most significant. The field on the left (most significant) is transmitted first; within a field, the most significant bit (msb) (the leftmost bit) is also transmitted first. This ordering convention is illustrated in figure 1-3.

Although Serial Bus addresses are defined to be big-endian, their data values may also be processed by little-endian processors. To minimize the confusion between conflicting notations, the location and size of bit fields are usually specified by width, rather than their absolute positions, as is also illustrated in figure 1-3.

When specific bit fields must be used, the CSR Architecture convention of consistent big-endian numbering is used. Hence, the most significant bit of a quadlet ("msb" in figure 1-3) is labeled "quad_bit_example[0]," the most significant byte of a quadlet ("byte_0") is labeled "quad_byte_example[0:7]," and the most significant quadlet in an octlet ("quadlet_high") is labeled "dual_quadlet_example[0:31]."

The most significant bit shall be transmitted first for all fields and values defined by this standard, including the data values read or written to control and status registers (CSRs).

[3]The numbers in brackets correspond to those of the bibliography in Annex N.

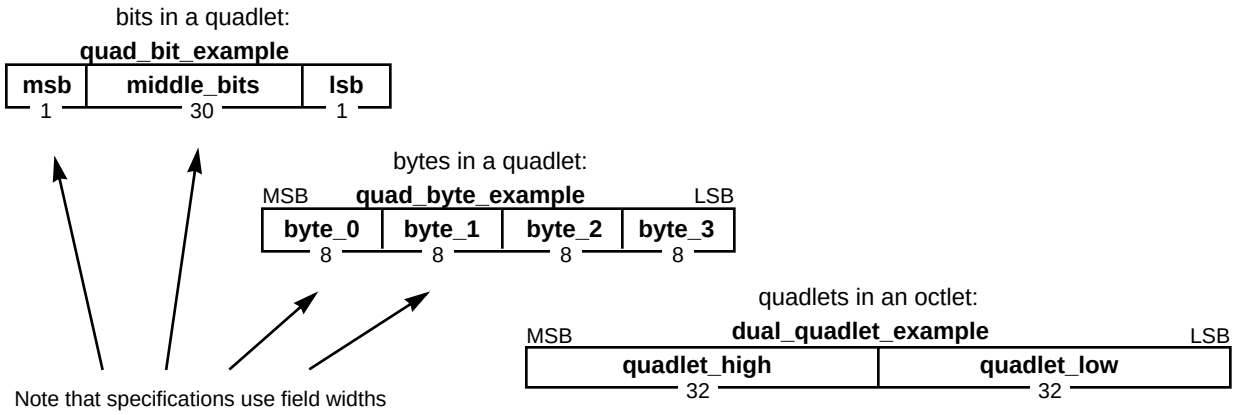


Figure 1-3 — Bit and byte ordering

1.6.4 Numerical values

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, the decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their standard 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9, A-F) digits followed by the subscript 16. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number “26” may also be represented as “$1A_{16}$” or “11010_2”. In C code examples, hexadecimal numbers have a “`0x`” prefix and binary numbers have a “`0b`” prefix, so the decimal number “26” would be represented by “`0x1A`” or “`0b11010`”.

1.6.5 Packet formats

Most Serial Bus packets consist of a sequence of quadlets. Packet formats are shown using the style given in figure 1-4.

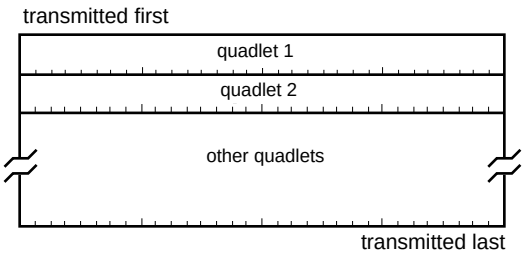


Figure 1-4 — Example packet format

Fields appear in packet formats with their correct position and widths. Bits in a packet are transmitted starting with the upper leftmost bit and finishing with the bottom rightmost bit. Given the rules in 1.6.3, this means that all fields defined in this standard are sent most significant bit first.

1.6.6 Register formats

All Serial Bus registers are documented in the style used by the CSR Architecture.

1.6.7 C code notation

The conditions and actions of the state machines are formally defined by C code. Although familiar to software engineers, C code operators are not necessarily obvious to all readers. The meanings of C code operators, arithmetic, relational logical and bitwise, both unary and binary, are summarized in table 1-2.

Table 1-2 — C code operators summary

Operator	Description
+, -, *, and /	Arithmetic operators for addition, subtraction, multiplication, and integer division
%	Modulus; x % y produces the remainder when x is divided by y
>, >=, <, and <=	Relational operators for greater than, greater than or equal, less than, and less than or equal
== and !=	Relational operators for equal and not equal; the assignment operator, =, should not be confused with ==
++	Increment; i++ increments the value of the operand after it is used in the expression while ++i increments it before it is used in the expression
--	Decrement; post-decrement, i--, and pre-decrement, --i, are permitted.
&&	Logical AND
||	Logical OR
!	Unary negation; converts a nonzero operand into 0 and a zero operand into 1
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise exclusive OR
<<	Left shift; x << 2 shifts the value of x left by two bit positions and fills the vacated positions with zero
>>	Right shift; vacated bit positions are filled with zero or one according to the data type of the operand, but in this standard, are always filled with zero
~	One's complement (unary)

A common construction in C code is conditional evaluation, in the form `(expr) ? expr1 : expr2.` This indicates that if the logical expression `expr` evaluates to a nonzero value, then `expr1` is evaluated, otherwise `expr2` is evaluated. For example, `x = (q > 5) ? x + 1 : 14` first evaluates `q > 5.` If nonzero (TRUE), `x` is incremented, otherwise `x` is assigned the value 14.

The descriptions above are casual; if in doubt, the reader is encouraged to consult ISO/IEC 9899:1990.

The C code examples assume the data types listed in table 1-3 are defined.

Table 1-3 — Additional C data types

Data type	Description
timer	A real number, in units of seconds, that autonomously increments at a defined rate
Boolean	A single bit, where 0 encodes FALSE and 1 encodes TRUE

All C code is to be interpreted as if it could be executed instantaneously, but time may elapse when conditional expressions are evaluated as part of iterated C code. Time elapses unconditionally only when the following function is called:

```
void wait_time(float time); // Wait for time, in seconds, to elapse
```

1.6.8 State machine notation

All state machines in this standard use the style shown in figure 1-5.

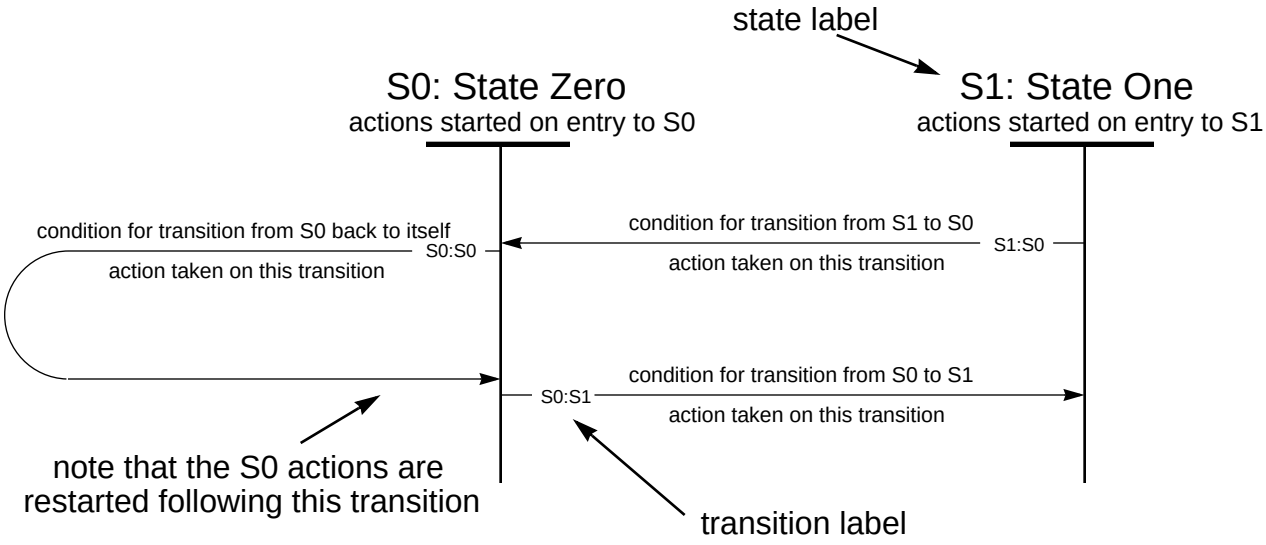


Figure 1-5 — State machine example

These state machines make the following three assumptions:

— Time elapses only within discrete states.
— State transitions are logically instantaneous, so the only actions taken during a transition are setting flags and variables, and sending signals. These actions complete before the next state is entered.
— Every time a state is entered, the actions of that state are started. Note that this means that a transition that points back to the same state causes the actions to be repeated from the beginning. All the actions started upon entry complete before any tests are made to exit the state.

1.6.9 CSR, ROM, and field notation

This standard describes CSRs and fields within them. To distinguish register and field names from node states or descriptive text, the register name is always capitalized. For example, the notation STATE_CLEAR.lost is used to describe the lost bit within the STATE_CLEAR register.

All CSRs are quadlets and are quadlet aligned. The address of a register is specified as the byte offset from the beginning of the initial register space and is always a multiple of four. When a range of register addresses is described, the ending address is the address of the last register.

This document describes a number of configuration ROM entries and fields within these entries. To distinguish ROM entry and field names from node states or descriptive text, the first character of the entry name is always capitalized. Thus, the notation Bus_Info_Block.cmc is used to describe the cmc bit within the Bus_Info_Block entry.

Entries within temporary data structures, such as packets, timers, and counters, are shown in lowercase (following normal C language conventions) and are formatted in a fixed-space typeface. Examples are `arb_timer` and `connected[i]`.

NOTE—Within the C code, the character formatting is not used, but the capitalization rules are followed.

1.6.10 Register specification format

This document defines the format and function of Serial Bus-specific CSRs. Registers may be read only, write only, or both readable and writable. The same distinctions may apply to any field within a register. A CSR specification includes the format (the sizes and names of bit field locations), the initial value of the register, the value returned when the register is read, and the effect(s) when the register is written. An example register is illustrated in figure 1-6.

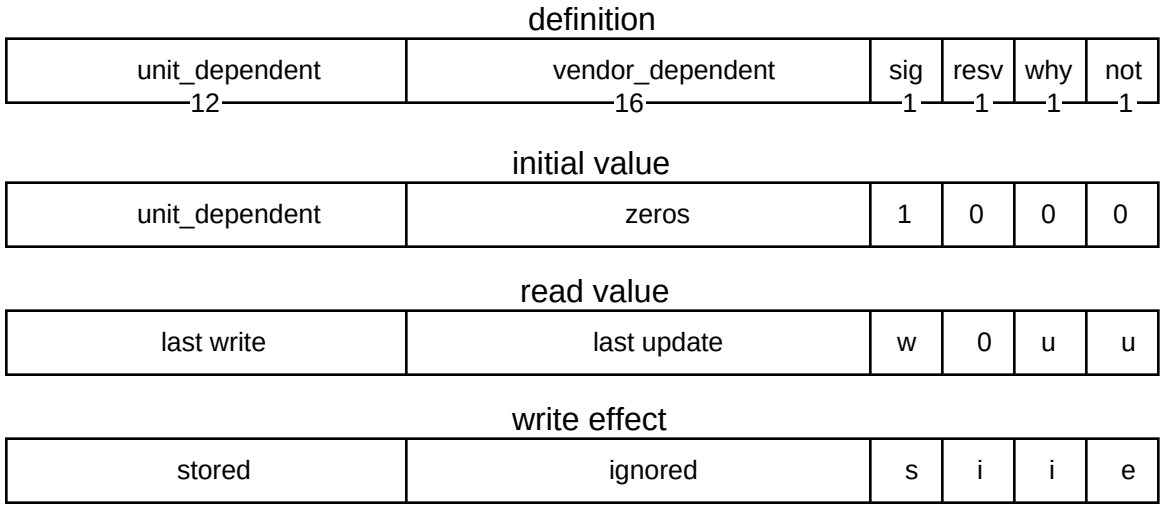


Figure 1-6 — CSR format specification (example)

The register definition lists the names of register fields. These names are descriptive, but the fields are defined in the text; their function should not be inferred solely from their names. However, the register definition fields in figure 1-6 have the meanings specified by table 1-5.

Table 1-5 — Register definition fields

Name	Abbreviation	Definition
unit dependent	unit_depend	The meaning of this field shall be defined by the unit architecture(s) of the node.
vendor dependent	vendor_depend	The meaning of this field shall be defined by the vendor of the node. Within a unit architecture, the unit-dependent fields may be defined to be vendor dependent.

A node's CSRs shall be initialized when power is restored (power reset), and may be initialized when a bus reset occurs or a quadlet is written to the node's RESET_START register (command reset). If a CSR's bus reset or command reset values differ from its initial values, they shall be explicitly specified.

The read value fields in figure 1-6 have the meanings specified by table 1-6.

Table 1-6 — Read value fields

Name	Abbreviation	Definition
last write	w	The value of the data field shall be the value that was previously written to the same register address.
last update	u	The value of the data field shall be the last value that was updated by node hardware.

The write effect fields in figure 1-6 have the meanings specified by table 1-7.

Table 1-7 — Write effect fields

Name	Abbreviation	Definition
stored	s	The value of the written data field shall be immediately visible to reads of the same register.
ignored	i	The value of the written data field shall be ignored; it shall have no effect on the state of the node.
effect	e	The value of the written data field shall have an effect on the state of the node, but may not be immediately visible to reads of the same register.

1.6.11 Reserved CSR fields

Reserved fields within a CSR conform to the requirements of the conformance glossary in this standard (see 2.1). Within a CSR, a field that conforms to this standard is labeled *reserved* (sometimes abbreviated as lowercase *r* or *resv*). Reserved fields behave as specified by figure 1-7; they shall be zero and any attempt to write to them shall be ignored.

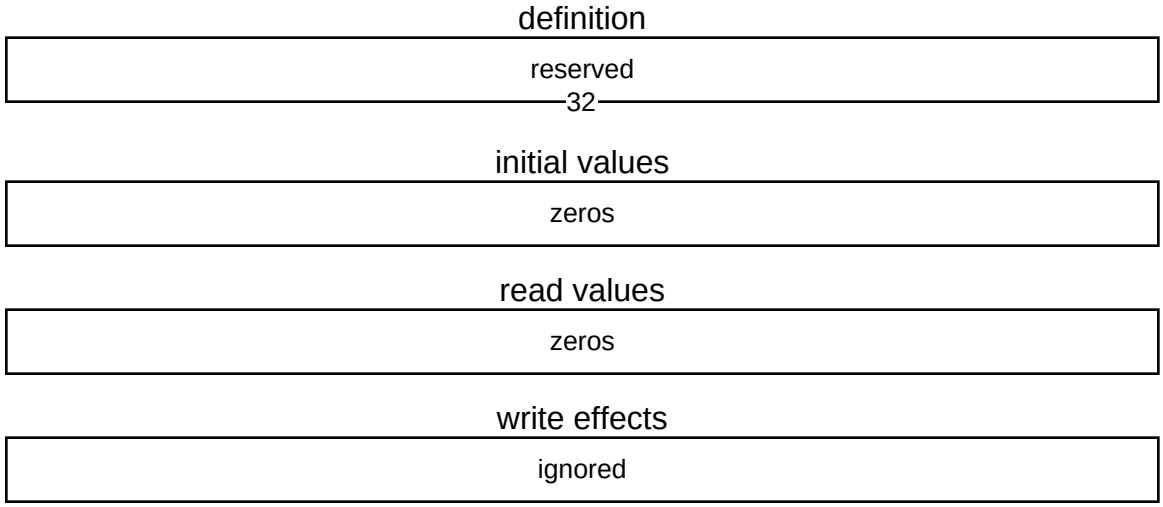


Figure 1-7 — Reserved CSR field behavior

This is straightforward as it applies to read and write requests. The same rules apply to lock requests, but the behaviors are less obvious; see 7.3.4.3 for details.

Change the head for clause 2 as follows:

2. Definitions ~~and abbreviations~~

Insert the following after the clause 2 title:

For the purposes of this standard, the following definitions, terms, and notational conventions apply. The IEEE Dictionary of Electrical and Electronics Terms [B3] should be consulted for terms not defined in this clause.

Replace 2.1 with the following:

2.1 Conformance

The following keywords are used to differentiate between different levels of requirements and optionality in this standard.

2.1.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented.

2.1.2 ignored: A keyword that describes bits, bytes, quadlets, octlets, or fields whose values are not checked by the recipient.

2.1.3 may: A keyword that indicates flexibility of choice with no implied preference.

2.1.4 reserved: A keyword used to describe objects—bits, bytes, quadlets, octlets, and fields—or the code values assigned to these objects; the object or the code value is set aside for future standardization by the IEEE. A reserved object shall be zeroed by its originator or, upon development of a future IEEE standard, set to a value specified by such a standard. The recipient of a reserved object shall not check its value. The recipient of an object whose code values are defined by this standard shall check its value and reject reserved code values.

2.1.5 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products conforming to this standard.

2.1.6 should: A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “is recommended.”

Change the head for 2.2 as follows:

2.2 Technical ~~glossary~~

Replace the following in 2.2, in alphabetical order. Replace the “x” in 2.2.x with the appropriate numbers:

2.2.x acknowledge packet: An 8-bit packet that may be transmitted in response to the receipt of a primary packet. The most and least significant nibbles are the one’s complement of each other. *Syn:* **acknowledge.**

2.2.x arbitration: The process by which nodes compete for control of the bus. Upon completion of arbitration, the winning node is able to transmit a packet or initiate a short bus reset.

2.2.x arbitration reset gap: The minimum period of idle bus (longer than a normal subaction gap) that separates fairness intervals.

2.2.x asynchronous packet: A primary packet transmitted in accordance with asynchronous arbitration rules (outside of the isochronous period).

2.2.x base rate: The lowest data rate used by Serial Bus in a backplane or cable environment. In multiple speed environments, all nodes are able to receive and transmit at the base rate. The base rate for the cable environment is 98.304 MHz ± 100 ppm.

2.2.x bus manager: The node that provides power management, sets the gap count in the cable environment, and publishes the topology of the bus and the maximum speed for data transmission between any two nodes on the bus. The bus manager node may also be the isochronous resource manager node.

2.2.x concatenated transaction: A split transaction comprised of concatenated subactions.

2.2.x cycle master: The node that generates the periodic cycle start packet 8000 times a second.

2.2.x doublet: Two bytes, or 16 bits, of data.

2.2.x fairness interval: A time period delimited by arbitration reset gaps. Within a fairness interval, the total number of asynchronous packets that may be transmitted by a node is limited. Each node's limit may be explicitly established by the bus manager or it may be implicit.

2.2.x gap: A period of idle bus.

2.2.x initial register space: A 2 kilobyte portion of initial node space with a base address of FFFF F000 0000_{16}. This address space is reserved for resources accessible immediately after a bus reset. Core registers defined by ISO/IEC 13213:1994 are located within initial register space, as are Serial Bus-dependent registers defined by this standard.

2.2.x isochronous: Uniform in time (i.e., having equal duration) and recurring at regular intervals.

2.2.x isochronous gap: For an isochronous subaction, the period of idle bus that precedes arbitration.

2.2.x isochronous resource manager: A node that implements the BUS_MANAGER_ID, BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE, and BROADCAST_CHANNEL registers (some of which permit the cooperative allocation of isochronous resources). Subsequent to each bus reset, one isochronous resource manager is selected from all nodes capable of this function.

2.2.x isochronous subaction: Within the isochronous period, either a concatenated packet or a packet and the gap that preceded it.

2.2.x link layer: The Serial Bus protocol layer that provides confirmed and unconfirmed transmission or reception of primary packets.

2.2.x listener: An application at a node that receives a stream packet.

2.2.x node: A Serial Bus device that may be addressed independently of other nodes. A minimal node consists of only a physical layer (PHY) without an enabled link. If the link and other layers are present and enabled they are considered part of the node.

2.2.x octlet: Eight bytes, or 64 bits, of data.

2.2.x packet: A sequence of bits transmitted on Serial Bus and delimited by DATA_PREFIX and DATA_END.

2.2.x payload: The portion of a primary packet that contains data defined by an application.

2.2.x PHY packet: A 64-bit packet where the most significant 32 bits are the one's complement of the least significant 32 bits.

2.2.x physical layer (PHY): The Serial Bus protocol layer that translates the logical symbols used by the link layer into electrical signals on Serial Bus media. The physical layer is self-initializing. Physical layer arbitration guarantees that only one node at a time is sending data. The mechanical interface is defined as part of the physical layer. There are different physical layers for the backplane and for the cable environment.

2.2.x port: The part of the physical layer (PHY) that allows connection to one other node.

2.2.x primary packet: Any packet that is not an acknowledge or a physical layer (PHY) packet. A primary packet is an integral number of quadlets and contains a transaction code in the first quadlet.

2.2.x quadlet: Four bytes, or 32 bits, of data.

2.2.x request: A primary packet (with optional data) sent by one node's link (the requester) to another node's link (the responder).

2.2.x response: A primary packet (with optional data) sent in response to a request subaction.

2.2.x self-ID packet: A physical layer (PHY) packet transmitted by a cable PHY during the self-ID phase or in response to a PHY ping packet.

2.2.x speed code: The code used to indicate bit rates for Serial Bus.

2.2.x split transaction: A transaction where unrelated subactions may take place on the bus between its request and response subactions.

2.2.x subaction: A complete link layer operation: optional arbitration, packet transmission, and optional acknowledgment.

2.2.x subaction gap: For an asynchronous subaction, the period of idle bus that precedes arbitration.

2.2.x talker: An application at a node that transmits a stream packet.

2.2.x transaction: A request and the optional, corresponding response.

2.2.x transaction layer: The Serial Bus protocol layer that defines a request-response protocol for read, write, and lock operations.

2.2.x unit architecture: The specification document that describes the interface to, and the behaviors of, a unit implemented within a node.

Insert the following in 2.2, in alphabetical order. Replace the "x" in 2.2.x with the appropriate numbers:

2.2.x acronym: A contrived reduction of nomenclature yielding mnemonics (ACRONYM).

2.2.x active port: A connected, enabled port that observes bias and is capable of detecting all Serial Bus signal states and participating in the reset, tree identify, self-identify, and normal arbitration phases.

2.2.x arbitration signaling: A protocol for the exchange of bidirectional, unclocked signals between nodes during arbitration.

2.2.x boundary node: A node with two or more ports, at least one of which is active and another suspended.

2.2.x channel: A relationship between a group of nodes, talkers, and listeners. The group is identified by a number between 0 and 63. Channel numbers are allocated cooperatively through isochronous resource management facilities.

2.2.x disabled port: A port configured to neither transmit, receive, or repeat Serial Bus signals. A disabled port shall be reported as disconnected in a physical layer's (PHY's) self-ID packet(s).

2.2.x disconnected port: A port whose connection detect circuitry detects no peer physical layer (PHY) at the other end of a cable. It is not important whether the peer PHY is powered or the peer port is enabled.

2.2.x initial node space: The 256 terabytes of Serial Bus address space that is available to each node. Addresses within initial node space are 48 bits and are based at zero. The initial node space includes initial memory space, private space, initial register space, and initial units space.

NOTE—See ISO/IEC 13213:1994 for more information on address spaces.

2.2.x initial units space: A portion of initial node space with a base address of FFFF F000 0800_{16}. This places initial units space adjacent to and above initial register space. The CSR's and other facilities defined by unit architectures are expected to lie within this space.

2.2.x isochronous period: A period that begins after a cycle start packet is sent and ends when a subaction gap is detected. During an isochronous period, only isochronous subactions may occur. An isochronous period begins, on average, every 125 µs.

2.2.x isolated node: A node without active ports; the node's ports may be disabled, disconnected, or suspended in any combination.

2.2.x kilobyte: A quantity of data equal to 2^{10} bytes, or 1024 bytes.

2.2.x nibble: Four bits of data.

2.2.x null packet: A packet in which no clocked data is transmitted on Serial Bus between DATA_PREFIX and DATA_END.

2.2.x originating port: A transmitting port on a physical layer (PHY), which has no active receiving port. The source of the transmitted packet is either the PHY's local link or the PHY itself.

2.2.x ping: A term used to describe the transmission of a physical layer (PHY) packet to a particular node in order to time the response packet(s) provoked.

2.2.x repeating port: A transmitting port on a physical layer (PHY) that is repeating a packet from the PHY's receiving port.

2.2.x resuming port: A previously suspended port that has observed bias or has been instructed to generate bias. In either case, the resuming port engages in a protocol with its connected peer physical layer (PHY) in order to reestablish normal operations and become active.

2.2.x suspended domain: One or more suspended nodes linked by suspended connection(s). Two nodes are part of the same suspended domain if there is a physical connection between them and all ports on the path are suspended. A boundary node is adjacent to one or more suspended domain(s) but not part of the suspended domain(s).

2.2.x suspended node: An isolated node with at least one port that is suspended.

2.2.x suspended port: A connected port not operational for normal Serial Bus arbitration, but otherwise capable of detecting both a physical cable disconnection and received bias.

2.2.x suspend initiator: An active port that transmits the TX_SUSPEND signal and engages in a protocol with its connected peer physical layer (PHY) to suspend the connection.

2.2.x suspend target: An active port that observes the RX_SUSPEND signal. A suspend target requests all of the physical layer's (PHY's) other active ports to become suspend initiators while the suspend target engages in a protocol with its connected peer PHY to suspend the connection.

2.2.x terabyte: A quantity of data equal to 2^{40}, or 1099511627776, bytes.

2.2.x transmitting port: Any port transmitting clocked data or an arbitration state. A transmitting port is further characterized as either originating or repeating.

2.2.x unit: A component of a Serial Bus node that provides processing, memory, input/output (I/O), or some other functionality. Once the node is initialized, the unit provides a Command and Status Register (CSR) interface. A node may have multiple units, which normally operate independently of each other.

Change the following in 2.2. Replace the "x" in 2.2.x with the appropriate numbers after alphabetization:

2.2.x cycle start packet: A primary packet sent by the cycle master that indicates the start of an isochronous ~~cycle~~ period.

2.2.x ~~node_ID~~ node ID: ~~This is a unique 16-bit number, which distinguishes the node from other nodes in the system.~~ A 16-bit number that uniquely differentiates a node from all other nodes within a group of interconnected buses. The ten most significant bits of ~~node_ID~~ node ID are the same for all nodes on the same bus~~; this is,~~ i.e., the ~~bus_ID~~ bus ID. The six least significant bits of ~~node_ID~~ node ID are unique for each node on the same bus; this is called the ~~physical_ID~~ physical ID. The physical ID is assigned as a consequence of bus initialization.

2.2.x ~~physical_ID~~ physical ID: The six least significant bits of the ~~node_ID~~ node ID. ~~This number is unique on a particular bus and is chosen by the physical layer during initialization.~~ On a particular bus, each node's physical ID is unique.

3. Summary description

Insert the following subclause after 3.8:

3.9 New features of IEEE Std 1394a-2000

The changes to IEEE Std 1394-1995 contained in IEEE Std 1394a-2000 are related to each other only in that they are incremental enhancements or extension to the existing standard. This clause provides an informative description of some of these new facilities; the clause is intended as a foundation for the reader's better understanding of the normative specifications that follow.

3.9.1 Connection debounce

In the cable environment, Serial Bus configures itself and assigns a unique 6-bit physical ID to each of the 63 devices that may be interconnected within a single arbitration domain. The bus initialization and configuration process is triggered by a change in connection status at any PHY port—one or more devices have been added or removed and the connection topology has changed.

IEEE Std 1394-1995 specifies that a change in detected bias voltage causes an instantaneous connection status change and commences bus initialization. Unfortunately, this idealized model fails to account for two aspects of the physical world:

— The connection process is asymmetric. When two nodes are connected, it is extraordinary for bias to be detected by both at the same time. The node that detects bias first immediately commences a bus reset but is unable to complete bus initialization until the other node detects bias and also commences a bus reset. During this interval, which may be on the order of tens of milliseconds, the first node is unable to send or receive packets. If that node is connected to others, that entire Serial Bus is unable to operate normally; and
— The connection process is not smooth. As the plug and connector scrape together, electrical contact is made and broken many times. Bus initialization requires no more than approximately 200 µs, but the completion of the insertion proceeds at a human pace. What appears to the user as one new connection may generate a storm of connections and disconnections.

All of this occurs quickly by human standards but the disruption caused to normal bus operations is particularly serious for isochronous data.

The problem of contact scrape is fairly simple to resolve—implement a connection time-out before new connections are confirmed. Disconnections may be detected immediately. There is no need to measure the connection time-out with great precision. An *n*-stage counter could derive a clock tick from the PHY's 24.576 MHz clock on the order of 5 ms; an overall time-out in the vicinity of 340 ms is adequate to debounce the contact scrape.

The problem of connection asymmetry is not solved by the connection timer. The first step in its solution is to require that all arbitration line states (except BUS_RESET) be ignored during the connection time-out interval. If BUS_RESET is observed, skip the remaining connection time-out interval and commence a bus reset. This works well when the connection is between two IEEE Std 1394a-2000 (new) PHYs or between an isolated node with a new PHY connected to an IEEE Std 1394-1995 (old) PHY of an operational bus.

In the case where an isolated node with an old PHY is connected to a new PHY of an operational bus, the old PHY generates a storm of bus resets which are propagated by the new PHY. This may be avoided if the new PHY implements an additional "reset detect" time-out of approximately 80 ms before it responds to BUS_RESET.

3.9.2 Cable arbitration enhancements

At the time IEEE Std 1394-1995 was in the final steps toward becoming a standard, a number of valuable performance enhancements had been identified. Their overall effect is to reclaim Serial Bus bandwidth used unnecessarily in arbitration and make it available for data transmission. This both increases the throughput of the bus as a whole and reduces the latency of individual transactions.

3.9.2.1 Arbitrated (short) bus reset

The addition of connection debounce, described in 3.9.1, does much to reduce the time the bus is unusable during bus initialization. However, even under ideal circumstances, e.g., a bus reset initiated by software, that involve no physical connection changes, bus initialization as specified by IEEE Std 1394-1995 is still more time consuming than it needs to be. Bus initialization is composed of three phases: reset, tree identify, and self-identify. The last phase, self-identify, requires approximately 1 µs per node or about 70 µs worst case when there are 63 nodes. Tree identify is also quite rapid and takes less than 10 µs. The longest phase is bus reset and it lasts about 167 µs while the BUS_RESET signal is propagated. The total duration of bus initialization is longer than the nominal isochronous cycle time, 125 µs, and may disrupt two isochronous periods. This compels device designers to add additional buffer depth to preserve the smooth flow of isochronous data from the perspective of their application. If sufficient time could be trimmed from bus initialization, the necessity for larger buffers could be reduced.

The reason for the long duration of BUS_RESET is that a transmitting node is unable to detect this arbitration line state. It is only after packet transmission is complete that the node observes the reset. Hence, BUS_RESET must be asserted longer than the longest possible packet transmission. This guarantees the success of bus reset regardless of the bus activity in progress.

Suppose a node arbitrates for and is granted control of Serial Bus, then all the other nodes are in the receive state. If BUS_RESET is transmitted, all the nodes will detect it. In this case, the bus reset duration can be shortened to approximately 1.3 µs. The worst case bus initialization time is improved from roughly 250 µs to 80 µs for a bus fully populated by 63 devices.[4] Typical bus initialization times would be shorter, e.g., approximately 20 µs for a bus with 16 devices.

The concept is simple, but requires analysis for the following particular cases:

a) *Parent port disconnection.* A prerequisite for arbitrated (short) bus reset is the ability to arbitrate. Since there is no longer a connection to the root, the long bus reset defined by IEEE Std 1394-1995 is all that is available.

b) *Child port disconnection.* The node requests the bus and, if granted control of the bus, initiates a short reset. If the arbitration request ultimately fails because of an arbitration state time-out, a long bus reset is initiated.

c) *New connection (root node or a node with a parent port).* After the connection time-out, the node requests the bus and, if granted control of the bus, initiates a short reset. If the arbitration request ultimately fails because of an arbitration state time-out, a long bus reset is initiated.

d) *New connection (isolated node).* The isolated node should defer bus reset in anticipation that the other node, after its connection time-out, successfully arbitrates and initiates a short reset. If the isolated node fails to observe BUS_RESET within 1 s, it initiates a long bus reset.

When two buses are connected, it is extremely unlikely that timings align to produce a short bus reset. Both nodes that sense the new connection are behaving as in item c), and one likely wins arbitration before the other. When the slower node observes BUS_RESET, it initiates a long reset. Even in the event that the slower node fails to sense the reset, the other node's bus initialization process times-out and a long bus reset results.

3.9.2.2 Ack-accelerated arbitration

The timing strategy for asynchronous arbitration specified by IEEE Std 1394-1995 is straightforward—arbitration cannot start until the bus is idle for a subaction gap time. For a bus with few hops between nodes, the subaction gap might be little more than 1 µs, while for the worst case topology, it could be slightly in excess of 13 µs. This design was adopted when simplicity, proof of concept, and time to market were of greater importance to Serial Bus than the extraction of the last possible bit of bandwidth. However, the penalty of wasted bus idle time becomes increasingly onerous as transmission rates and the number of connected nodes increase.

The duration of a subaction gap is determined by bus topology; it is necessary for it to be greater than the worst case round-trip propagation time between any two nodes on the bus. This ensures that, after the transmission of an asynchronous primary packet, no node starts arbitration before the acknowledge packet has been transmitted and received. Unfortunately this makes no distinction between acknowledge packets and primary packets—an acknowledge packet never follows another acknowledge packet. Arbitration can start immediately after an acknowledge packet is observed.

NOTE—In a sense this is not a new form of arbitration but exactly the form of arbitration used for isochronous packets. Since there are no acknowledge packets during the isochronous period, isochronous arbitration starts as soon as an idle gap is detected after a packet.

[4]The calculation is based on the assumption that worst case PHY repeater delay and cable delay together are less than 500 ns. This would permit up to 100 m of cable.

3.9.2.3 Fly-by concatenation

IEEE Std 1394-1995 defines one arbitration enhancement—concatenated subactions. This is a method of completing a split transaction without relinquishing the bus in between the acknowledge packet and the response packet. From the standpoint of the link that receives both the acknowledge packet and the response packet, it is impossible to distinguish between concatenated subactions and closely spaced subactions separated by an intervening arbitration. This provides an opportunity to reclaim more Serial Bus bandwidth for data transmission.

Suppose a node has a packet ready for transmission when an unrelated packet addressed to the same node is received and acknowledged. Are there any consequences if the node concatenated the packet awaiting transmission to the acknowledge packet? This sequence of received packet, acknowledge packet, and transmitted packet is indistinguishable from the concatenated subactions permitted by IEEE Std 1394-1995, except for the content of the transmitted packet. Provided that fair arbitration is observed and some timing and topology issues are considered, Serial Bus operates as before.

One consideration is that fly-by concatenation may be used only when the candidate packet is received on a child port or is originated by the node's link. When a packet is received on a parent port it is likely that other nodes are simultaneously receiving the packet. If more than one node attempted fly-by concatenation, their transmitted packets would collide. When a packet is originated by the node's link or received on a child port, packet reception is unique—it is not possible for any other node to receive the same packet on a child port at the same time.

When a packet is received on a child port or is originated by the node's link, there are two opportunities for fly-by concatenation

a) After an acknowledge packet, an unrelated asynchronous primary packet may be concatenated, or

b) After a cycle start packet or isochronous packet, an isochronous packet may be concatenated.

Fly-by concatenation is illustrated by figure 3-32. The transmitted packet appears concatenated on the repeating ports, but as a separate packet transmitted by the child port. The figure shows a two-port PHY with a child port at the left. At the outset, a packet is about to be received by the child port. Moving down figure 3-32, the succeeding snapshots of Serial Bus state for both ports of the PHY are shown. As the received packet is repeated by the other port, it is seen to emerge, first the data prefix (now signaled as TX_DATA_PREFIX by the repeating port), then the clocked data of the packet. The packet retransmission is nearly complete as trailing TX_DATA_PREFIX is signaled by the repeating port; the use of data prefix instead of data end on retransmission signals concatenation. In the last snapshots, the concatenated packet is seen to emerge from both ports of the PHY; on the original receiving port the new packet appears as a single, unconcatenated packet.

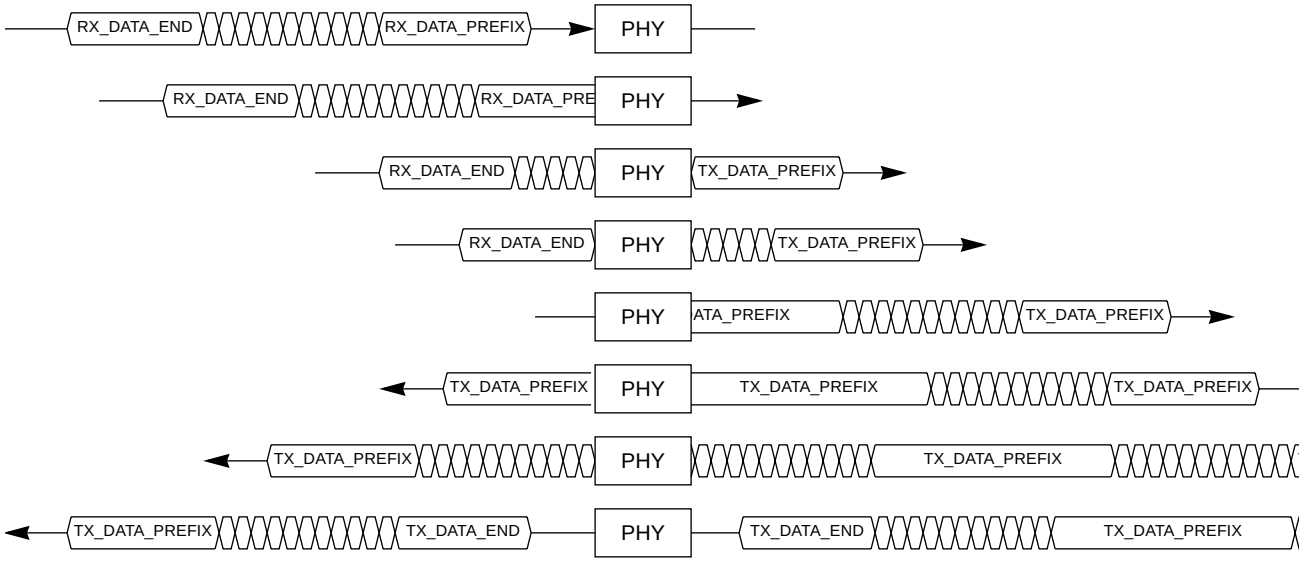


Figure 3-32 — Fly-by concatenation

3.9.2.4 Multi-speed-packet concatenation

IEEE Std 1394-1995 was published before 200 Mbit/s and faster PHYs were available. As a consequence, the details of operation at speeds other than S100 are sketchy and in some cases susceptible to multiple interpretations, each of which is arguably reasonable. As an example, IEEE Std 1394-1995 contains a contradictory requirement that PHYs signal speed only for the first packet of a multiple packet sequence but expect to observe a separate speed-signal for each received packet.

Apart from the interoperability problems, multi-speed-packet concatenation is an essential building block for the enhancements specified by IEEE Std 1394a-2000 because it permits the following:

— Concatenation of a read response of arbitrary speed after an acknowledge packet

— Concatenation of isochronous packets without regard to the speed of each

— Fly-by concatenation without regard to the speed of the concatenated packet

The deficiencies of IEEE Std 1394-1995, with respect to multi-speed-packet concatenation, are corrected in both the PHY/link interface specified in 4.2.1A and the PHY state machines and C language code in clause 5A. Complete interoperability with older PHYs cannot be guaranteed but is promoted by a requirement that S100 packets cannot be concatenated after faster packets.

3.9.2.5 Arbitration enhancements and cycle start

There are interoperability problems created by the enhancements described in 3.9.2.4 when utilized by nodes other than the cycle master (root). The problem arises when the cycle master needs to transmit a cycle start packet. Existing nodes give precedence to arbitration requests from child ports over their own requests. When none of the nodes implement arbitration enhancements, this is not a problem; a subaction gap appears at the end of any transaction in progress and the root, by virtue of its natural arbitration priority, wins arbitration and is able to send the cycle start packet. If some or all of the child nodes are employing arbitration enhancements, the root may be unable to win arbitration for a protracted period. IEEE Std 1394-1995 is designed to accommodate a delayed cycle start packet, but only up to the extent of the longest asynchronous primary packet. The delay caused by the root's children could violate isochronous timing assumptions and disrupt the flow of isochronous data.

The solution is to define new requests over the interface between the link and PHY so that the link may selectively disable and enable ack-accelerated and fly-by concatenation enhancements when a cycle start is due to be transmitted. All nodes except the cycle master are required to disable these accelerations from the time of their local cycle synchronization event until a cycle start packet is observed. Once the cycle start packet is observed, it is safe to reenable acceleration.

3.9.3 Performance optimization via PHY "pinging"

The simplest performance improvement that a bus manager can make is to tune the "gap count" to the topology of Serial Bus. The gap count determines the value of two fundamental time intervals—the subaction gap and the arbitration reset gap. The smaller the gap count, the less idle bus time consumed by these gaps.

IEEE Std 1394-1995 contains recommendations that permit the bus manager to reduce the gap count from the default $3F_{16}$ in effect after bus reset. The bus manager[5] calculates the worst case round-trip propagation delay between any two nodes on the bus. The calculation should be based upon the greatest hop count between any two nodes (as determined from the self-ID packets) and an assumption that cable lengths do not exceed 4.5 m. Some implementations do not analyze the self-ID packets to determine the greatest hop count, but assume a maximum hop count of 16. Approximate as they are, neither the cable length nor maximum hop count assumption is reliable, since they are not normative requirements of IEEE Std 1394-1995.

[5]In the absence of a bus manager, the isochronous resource manager is permitted to optimize the gap count.

IEEE Std 1394a-2000 provides an alternate method for the bus manager to optimize gap count—the PHY “ping” packet. This is a packet that is addressed to any one of the 63 local nodes on a bus. In response, the addressed PHY returns a set of self-ID packets. The originator of the ping packet may time the transaction and calculate a round-trip time between itself and the targeted PHY.

With these tools, the bus manager can readily calculate the round-trip time between any two leaf nodes, as explained in more detail in annex M.

NOTE—For the purpose of calculating the round-trip time, the contents of the packet returned by the PHY are unimportant. The return of the self-ID packet(s) permit some additional diagnostic utility in the bus manager. Alternatively, the bus manager could address some other packet, such as the remote access packets defined for the suspend/resume facility, and time the packet sent in response.

3.9.4 Priority arbitration

A fundamental part of IEEE Std 1394-1995 is the concept of *fairness*, which ensures that all nodes on the bus are each able to arbitrate within a bounded time period; no node may starve another node’s arbitration requests. Within any particular period (called a *fairness interval*) the order in which nodes are granted the bus is arbitrary, but each node is guaranteed fair access.

This mechanism is useful to promote equal access to the bus, but individual nodes may consume bus time awaiting their chance to arbitrate. These inefficiencies can be significant, particularly in cases where there are few nodes attached to the bus.

One opportunity to improve efficiency concerns response subactions. Since each response subaction is originally triggered by a request subaction, it is arguably true that fair arbitration for requests inherently limits the number of responses. Hence the new provision of IEEE Std 1394a-2000 that responses are always permitted to use priority arbitration.

In the case where there are fewer nodes than the maximum of 63, the number of fair arbitration opportunities may be divided amongst them. For example, the system disk for a computer might be granted permission to use additional priority arbitration requests within a fairness interval. This would reduce the latency and increase the throughput for the transfer of data to and from the disk.

Finally, for reasons of simplicity, some infrequent subactions, such as the transmission of a PHY packet, are granted exemption from the normal requirements of fair arbitration.

3.9.5 Port disable, suspend, and resume

The cable PHY arbitration state machines in IEEE Std 1394-1995 describe the behavior of connected, fully functional ports in the four bus phases: reset, tree identify, self-identify, and normal arbitration. The only other port state is disconnected; it requires little or no description. IEEE Std 1394a-2000 defines additional port states, disabled and suspended, and the protocols for orderly transition between port states.

Each PHY port may independently be in one of the following four states:

— *Disabled.* The port generates no signals and is not capable of detecting signals. From the standpoint of a connected peer PHY, there is no observable difference between a disabled port and an unpowered PHY.
— *Disconnected.* There is no physical cable connection between the port and a peer PHY.

— *Suspended.* No bias is generated by this port but a physical connection exists with a peer PHY (which may or may not be generating bias). The absence of bias generated by this port permits a connection detect circuit to monitor the physical connection.
— *Active.* A physical connection exists with a peer PHY and bias is both generated and observed by this port. The active state is called *connected* in IEEE Std 1394-1995, but IEEE Std 1394a-2000 redefines *connected* to mean only the detection of a peer PHY, powered or unpowered, at the other end of a cable.

The preceding four states are fundamental; additional, transitional states are defined in the state diagrams in order to accurately describe PHY behavior.

An important motivation for the definition of these new states is to permit power savings in PHY implementations. In all states except active, many PHY components may be left unpowered. When all the ports of a PHY are either disabled, disconnected, or suspended, there are opportunities for substantial savings.

3.9.5.1 Connection detect circuit

Additional circuitry is required to detect the presence or absence of a physical connection. The circuit produces meaningful output only while the port itself is not generating bias.

3.9.5.2 Suspended connection

A suspended connection exists between two connected peer PHYs if the port at each end of the connection is suspended. The suspended connection is established by a protocol between the two ports, one of which is the suspend initiator and the other is the suspend target.

A port becomes a suspend initiator because of the receipt of a remote command packet that sets the port's suspend variable to one. The remote command packet may have been transmitted by the PHY's local link or by some other node. In either case, the originating PHY arbitrated for the bus; consequently the suspend initiator has ownership of the bus to transmit a remote response packet. Subsequent to the transmission of the remote response packet, the suspend initiator asserts the TX_SUSPEND signal for a period equal to MIN_DATA_PREFIX.

A port becomes a suspend target when it observes RX_SUSPEND while in the *Idle* state. In response to RX_SUSPEND, the suspend target drives TpBias low and starts a timer.

In the meantime, the suspend initiator idles its transmitters after signaling TX_SUSPEND for the required time. The suspend initiator then starts a timer and monitors bias. If the suspend initiator detects that bias has been removed by the suspend target, the suspend initiator in turn drives TpBias low and continues to do so for a fixed time, after which the suspend initiator enters the suspended state. If the suspend initiator's timer expires and bias is still present, the suspend initiator enters the suspended state in a faulted condition.

While the suspend target drives TpBias low it also monitors bias. If bias is removed by the suspend initiator, the handshake is complete and the suspend target enters the suspended state. Otherwise, the suspend target waits until its timer expires and then suspends even though bias is still observed.

Entry into the suspended state necessitates activation of the port's connection detect circuitry, which is usable only if the port itself does not generate bias. Each port, suspend initiator, or target delays entry into the suspended state until the output of the connection detect circuit becomes stable and indicates a physical connection.

Once a port is suspended it is capable of detecting either of the following two events:

a) Physical disconnection

b) The presence of bias

Physical disconnection causes the port to transition to the disconnected state. The effects of bias depend upon the fault state of the port. If the port completed the suspend initiator/target handshake normally and bias was removed, the presence of bias causes the port to resume normal operations. Otherwise, if the port entered the suspended state in a faulted condition (because bias was not removed by the connected peer PHY), the presence of bias has no effect until software clears the fault. Once the faulted condition is cleared, the port attempts to resume normal operations if bias is present.

3.9.5.3 Suspended domain

A suspended domain is a set of one or more suspended nodes that are physically connected through suspended nodes via suspended connections. A node without active ports and at least one suspended port is a suspended node. In order for two suspended nodes to be part of the same suspended domain, a path of suspended connections exists between them. It is possible for connected but disabled ports to split a group of physically connected suspended nodes into separate suspended domains.

A suspended domain propagates as the result of suspend target behavior. When a port becomes a suspend target (as a consequence of observing RX_SUSPEND), it not only suspends the connection with its peer suspend initiator but also requests that all other active ports on the same PHY become suspend initiators. That is, during the same time that the suspend target observes RX_SUSPEND, the formerly active ports transmit TX_SUSPEND.

Unless blocked, the TX_SUSPEND signal transmitted by a suspend initiator propagates until it reaches a leaf node. The propagation of the suspended domain may be blocked by a PHY compliant with IEEE Std 1394-1995 (but unmodified by IEEE Std 1394a-2000), a disabled port, or a suspended port. It is possible to control the extent of a suspended domain by disabling selected PHY ports prior to the activation of the suspend initiator.

NOTE—A Serial Bus configuration that includes nodes compliant with IEEE Std 1394-1995 (unmodified by IEEE Std 1394a-2000) as well as nodes capable of suspension requires careful analysis by a power manager or other application prior to the creation of a suspended domain. For example, if a remote PHY command packet is used to create a suspended domain and a legacy node lies on the path between the sender of the packet and the initiator of the suspended domain, the legacy node blocks the spread of the suspended domain from the suspend initiator toward the sender of the packet. In addition, the original sender of the packet subsequently has no means to cause the suspended domain to resume; it is unreachable because the legacy node perceives the connection to the suspended domain as disconnected.

3.9.5.4 Resumption

Any one of a number of events may cause a suspended port to attempt to resume normal operations

— Bias is detected and there is no fault condition.
— Except at a boundary node, another suspended (but unfaulted) port detects bias.
— A resume packet is received or transmitted by the PHY. The originator of the resume packet may be either the PHY's local link or another node.
— A remote command packet that sets the resume variable to one is addressed to the suspended port.
— At an isolated node, another disconnected (but enabled) port detects a new connection.

If a port is in the process of suspending, any of the previous events causes the port to resume after the completion of the suspend handshake.

Except for boundary nodes or the resumption of a single port by means of its resume variable, a resumption by one suspended port causes all of a PHY's other suspended ports to initiate resumption.

A suspended port initiates the resume process by generating TpBias and then waits to observe bias. Once bias is present at both ends of the cable, an active connection has been restored. Because topology changes may have occurred while connections were suspended, resumption of a connection always generates a bus reset. Heuristics are employed to minimize any proliferation of bus resets—delay a short time before initiating a bus reset, attempt to perform an arbitrated (short) bus reset if a boundary node, else delay a short while longer if not a boundary node—and in all cases defer to a detected bus reset.

3.9.5.5 Boundary nodes

Boundary nodes, on the border between an active Serial Bus and a suspended domain, behave differently than suspended nodes during resumption. The boundary node acts to minimize disruption of the active bus while the suspended domain resumes.

After a resume event on one of its suspended ports, a boundary node waits for three RESET_DETECT intervals before it initiates an arbitrated (short) bus reset on the active bus. The delay is intended to permit all nodes in the suspended domain to resume before the active bus is made aware of their presence. If the boundary node experiences a resume event on more than one of its suspended ports, the delay is measured from the most recent resume event.

If a boundary node detects BUS_RESET on any of its resumed ports during this interval, it initiates a long bus reset on the active bus. This is necessary because there is no time to arbitrate on the active bus and issue a short bus reset; bus reset is already underway on the resumed segment and must be propagated without delay.

The suspended nodes that have resumed wait seven RESET_DETECT intervals before they initiate bus reset. This longer interval permits the entire suspended domain to resume and then allows time for the boundary node(s) to arbitrate on the active bus.

When a suspended domain that adjoins more than one boundary node is resumed, one of the boundary nodes likely arbitrates on its active bus before the other(s) and transmits BUS_RESET into the resumed domain. When the bus reset is observed by the other boundary node(s), they respond by converting it to a long bus reset.

4. Cable PHY specification

4.2 Cable physical connection specification

Background

The facilities of Serial Bus, defined in IEEE Std 1394-1995, have found wide applicability in the consumer electronics industry for a new generation of digital products. For some of these product applications, the cable and 6-pin connectors specified by the standard are less than ideal:

— *Battery operated devices.* Because these devices draw no power from the cable, their design could be simplified and their cost reduced if electrical isolation were not required for the connector assembly. In addition, the cable's power conductors represent a potential source of analog noise—a significant concern for audio equipment.
— *Hand-held devices.* In contrast to the compactness of some consumer products, such as video camcorders, the cable and 6-pin connectors are relatively bulky. A more compact design is better suited to these products.

This subclause specifies alternative cables and connectors in addition to the cables and 6-pin connectors specified by IEEE Std 1394-1995. Except for the absence of power, these alternatives are intended to be interoperable with equipment that is compliant with either IEEE Std 1394-1995 or IEEE Std 1394a-2000. The remarks below apply to external (inter-crate) cabling, where extra care must be exercised for safety and EMC compliance. (Intra-crate connections are not standardized in this clause.)

Except as superseded by material in IEEE Std 1394a-2000, all subclauses in clause 4 of IEEE Std 1394-1995 apply to alternative cables and 4-pin connectors. With respect to these alternative cables and connectors only, the subclauses that follow apply. With respect to the cables with six conductors and 6-pin connectors at both ends, clause 4 of IEEE Std 1394-1995 is not affected in any way by IEEE Std 1394a-2000.

NOTE—This specification of alternative cables and connectors is intended to update and replace any and all earlier standards or draft standards that describe 4-pin variants of cable assemblies. Products in compliance with earlier standards (in particular, IEC 61883-1) may not operate at speeds faster than S100.

4.2.1 Media attachment

4.2.1.4 Signal propagation performance

4.2.1.4.8 Shield ac coupling

Background

Although electrical isolation is an important system design issue for many Serial Bus devices, it is not possible to specify uniform isolation requirements for all devices. Electrical isolation is not a normative requirement of this standard so 4.2.1.4.8 should be removed from normative text.

Delete 4.2.1.4.8 in its entirety.

Insert the following after 4.2.1:

4.2.1A Alternative cable media attachment specification

4.2.1A.1 Connectors

In typical applications, computer, consumer electronic, or peripheral equipment boxes shall present one or more connector sockets for attachment to other boxes via cables. The detachable ends of the cable shall be terminated with connector plugs. Subclause 4.2.1 specifies connectors that have six contacts; this subclause specifies alternative connectors that have four contacts.

All dimensions, tolerances, and descriptions of features that affect the intermateability of the 4-pin shielded connector plugs and sockets are specified within this subclause. Features of connector plugs and sockets that do not affect intermateability are not specified and may vary at the option of the manufacturer. Connector features that are not directly controlled within this subclause shall be indirectly controlled by performance requirements in 4.2.1A.3 and 4.2.1A.4.

The holes and patterns (footprint) for the mounting of some of the possible versions of connectors to the printed circuit board are recommended in 4.2.1A.1.8.

4.2.1A.1.1 Connector plug

The mating features of the connector plug are specified in figure 4-11A and figure 4-11B. The features assure the intermateability of the plug with the 4-pin sockets specified in 4.2.1A.1.

Figure 4-11A and figure 4-11B describe a plug intended to be used when only detent retention with the socket is required.

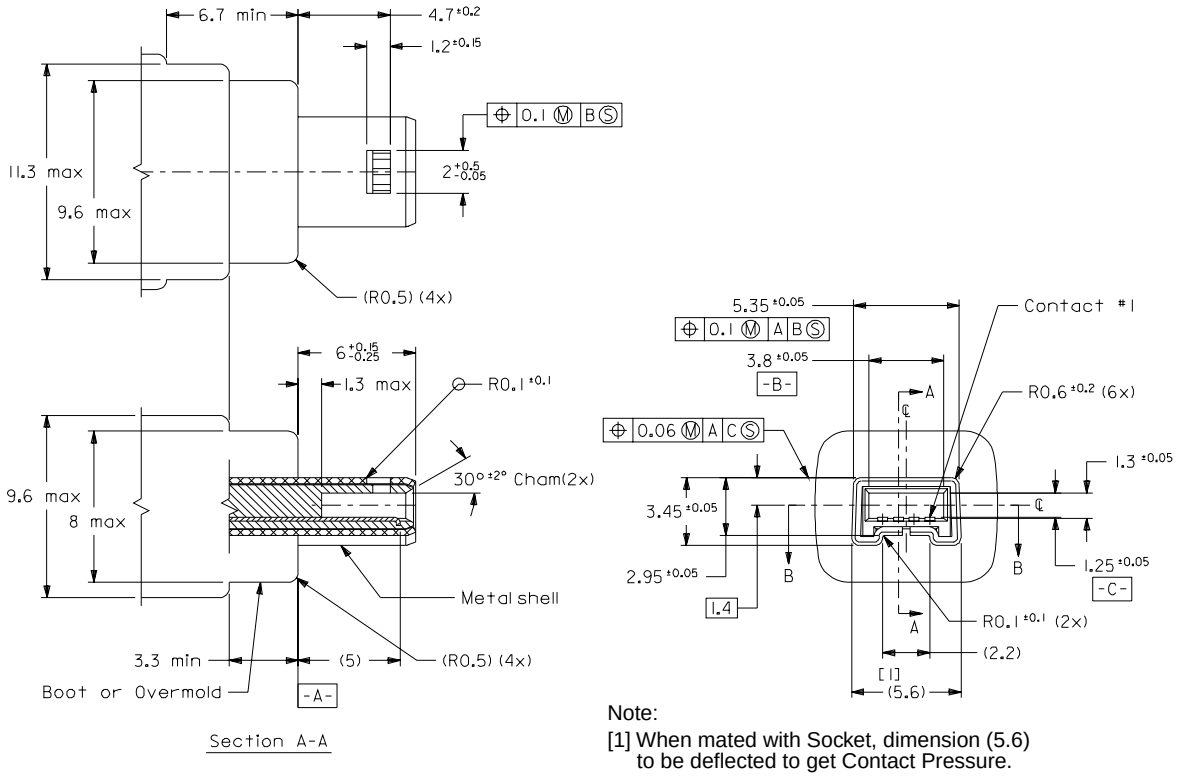


Figure 4-11A — Plug body

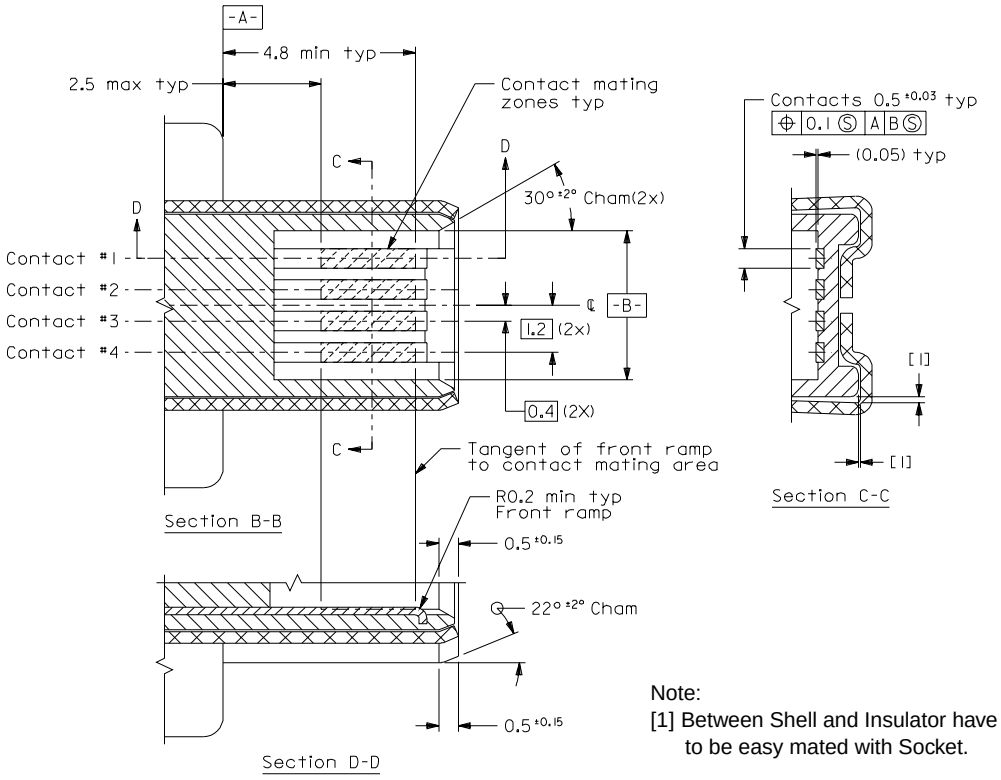


Figure 4-11B — Plug section details

4.2.1A.1.2 Connector plug terminations

The termination of the stranded wire to the plug contacts may be varied to suit the manufacturing process needs of the cable assembler.

For reference, the following methods are listed:

— Crimp
— Insulation displacement (IDC)
— Insulation piercing
— Welding and soldering

4.2.1A.1.3 Connector socket

The mating features of the connector socket are described in figure 4-11C through figure 4-11E. They assure the intermateability of the socket with the 4-pin plugs specified in 4.2.1A.1.

.

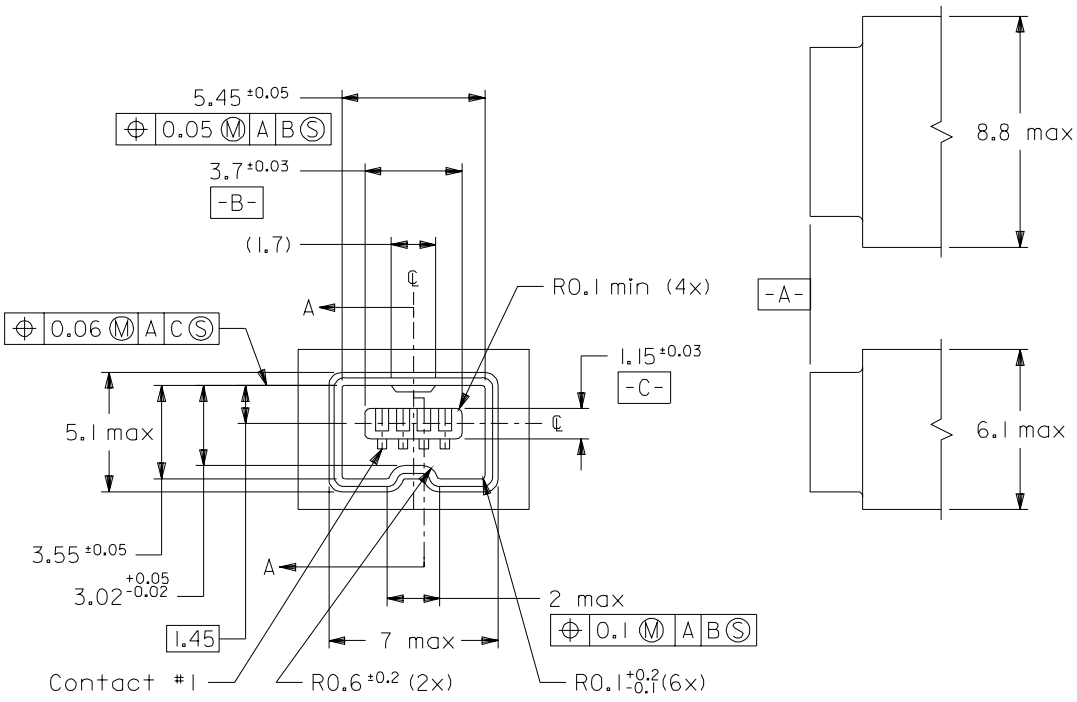


Figure 4-11C — Connector socket interface

The contacts are attached to the signals using the guidance in table 4-11A.

Table 4-11A — Connector socket signal assignment

Contact number	Signal name	Comment
1	TPB*	Strobe on receive, data on transmit (differential pair)
2	TPB	
3	TPA*	Data on receive, strobe on transmit (differential pair)
4	TPA	
Shell	VG	Necessary for ground reference

Figure 4-11D describes the relationship of the contacts and the shell. This includes the wiping portion of the contact and shell detent.

Figure 4-11E shows the mated cross-section of the plug and socket contacts.

When mounted on a printed circuit board, the socket should be at a fixed height, as illustrated by figure 4-11F.

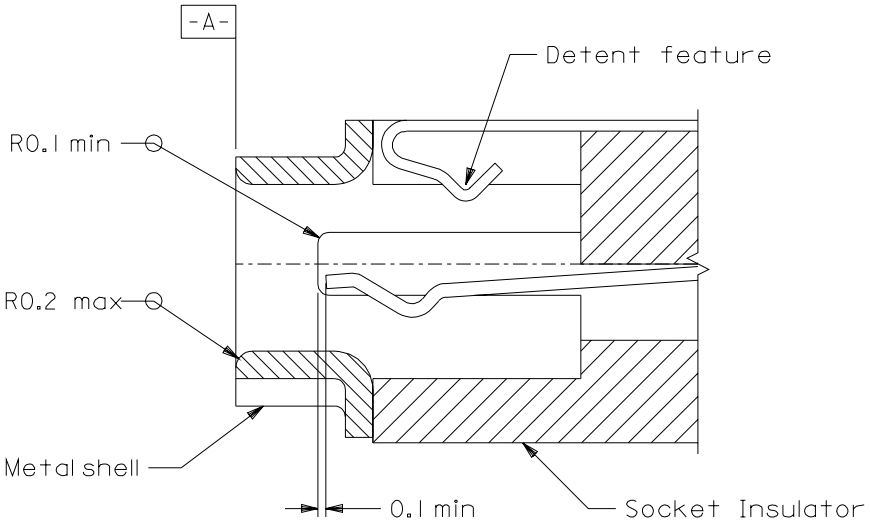


Figure 4-11D — Socket cross-section A–A

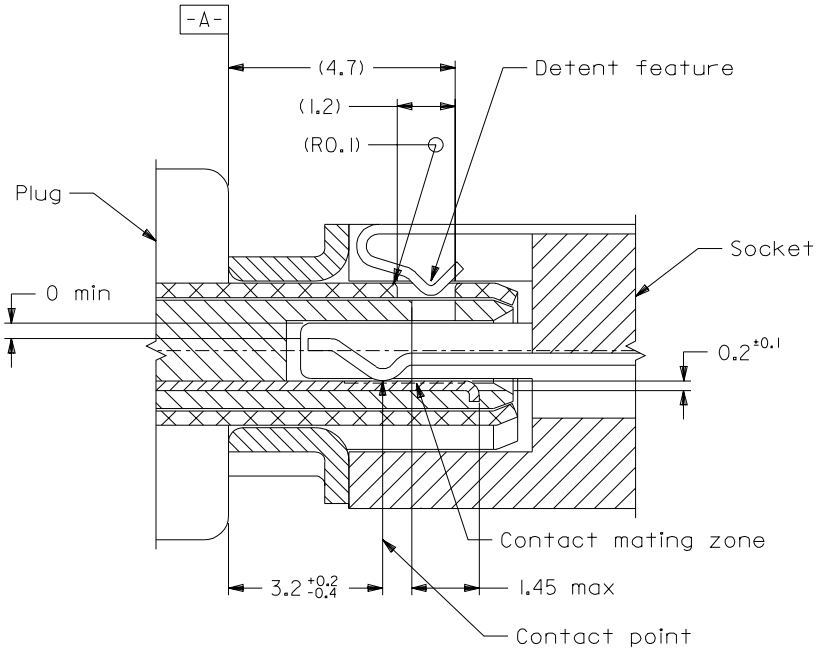


Figure 4-11E — Cross-section of plug and socket contacts

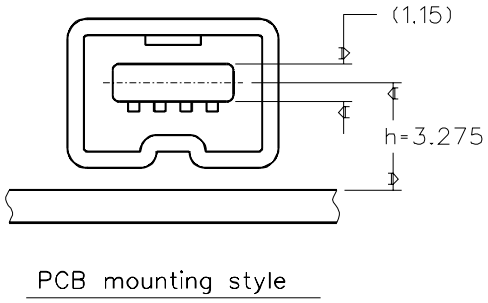


Figure 4-11F — Socket position when mounted on a printed circuit board

4.2.1A.1.4 Contact finish on plug and socket contacts

It is necessary to standardize the electroplated finish on the contacts to assure the compatibility of plugs and sockets from different sources. The following standardized electroplatings are compatible, and one shall be used on contacts.

a) 0.76 μm (30 μin), minimum, gold, over 1.27 μm (50 μin), minimum, nickel
b) 0.05 μm (2 μin), minimum, gold, over 0.76 μm (30 μin), minimum, palladium, over 1.27 μm (50 μin), minimum, nickel
c) 0.05 μm (2 μin), minimum, gold, over 0.76 μm (30 μin), minimum, palladium-nickel alloy (80% Pd-20% Ni), over 1.27 μm (50 μin), minimum, nickel

NOTES

1—Selective plating on contacts is acceptable. In that case, the above electroplating shall cover the complete area of contact, including the contact wipe area.

2—Either a copper or palladium strike is acceptable under the nickel electroplate.

4.2.1A.1.5 Termination finish on plug and contact socket terminals

It is acceptable to use an electroplate of tin-lead with a minimum thickness of 3.04 μm (120 μin) over 1.27 μm (50 μin), minimum, nickel. A copper strike is acceptable under the nickel.

4.2.1A.1.6 Shell finish on plugs and sockets

It is necessary to standardize the plated finish on the shells to ensure compatibility of products from different sources. Both shells shall be electroplated with a minimum of 3.03 μm (120 μin) of tin or tin alloy over a suitable barrier underplate.

4.2.1A.1.7 Connector durability

The requirements of different end-use applications call for connectors that can be mated and unmated a different number of times, without degrading performance beyond acceptable limits. Accordingly, this standard specifies minimum performance criteria of 1000 mating cycles.

4.2.1A.1.8 Printed circuit board footprints

The dimensional specifications recommended for the footprint of a surface-mount printed circuit board socket are illustrated by figure 4-11G.

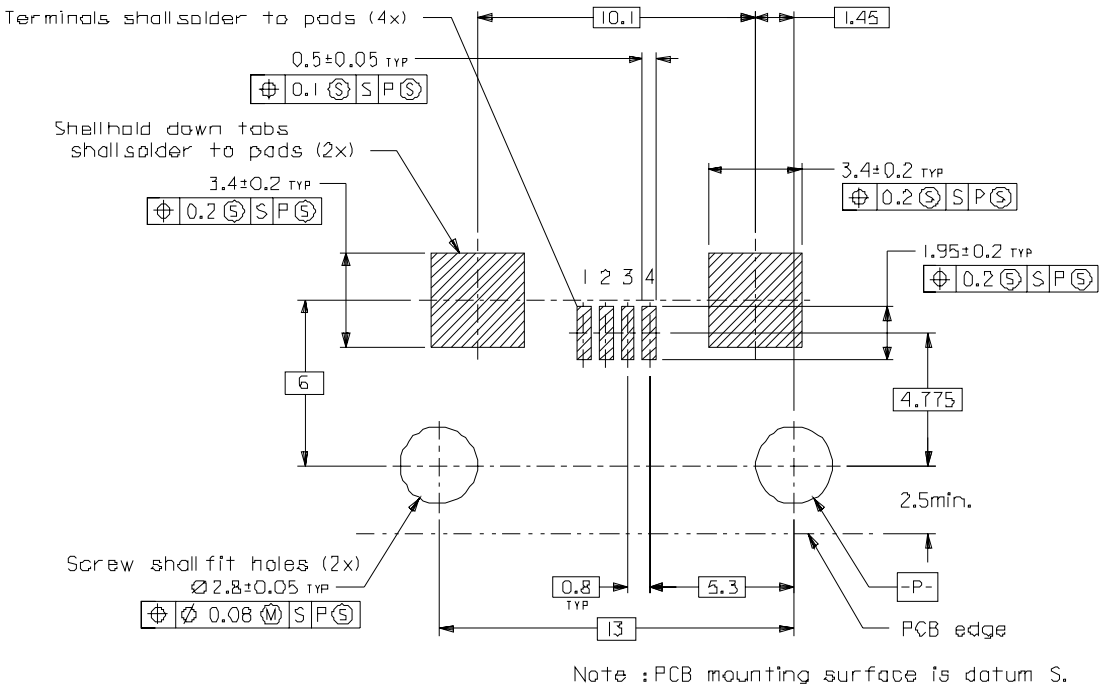


Figure 4-11G — Flat surface-mount printed circuit board socket footprint

The dimensional specifications recommended for the footprint of a through-hole printed circuit board socket are illustrated by figure 4-11H.

4.2.1A.2 Cables

All cables and cable assemblies shall meet assembly criteria and test performance in the normative portions of this standard and should be tested in accordance with the procedures described in annex K.

4.2.1A.2.1 Cable material (informative)

Linear cable material typically consists of two twisted-pair conductors. The two twisted pairs carry the balanced differential data signals. Figure 4-11I illustrates a *reference design* adequate for a 4.5 m cable. Subclause 4.2.1A.4 describes the performance requirements for the cable.

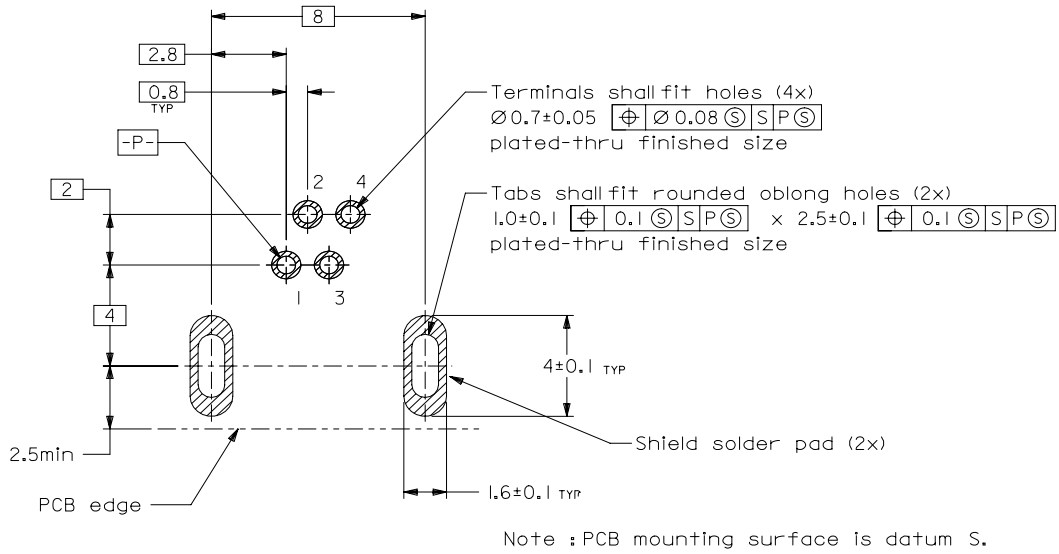
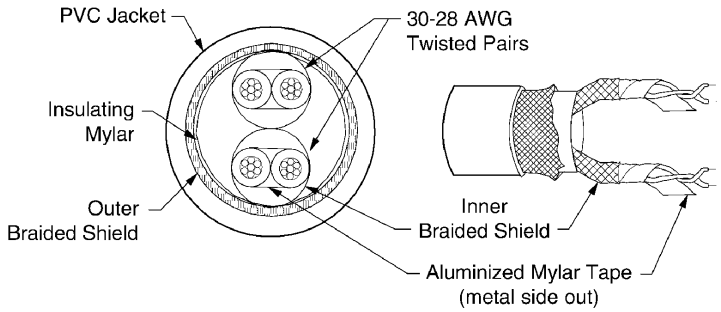


Figure 4-11H — Flat through-hole mount printed circuit board socket footprint



NOTE—This construction is illustrated for reference only; other constructions are acceptable as long as the performance criteria are met.

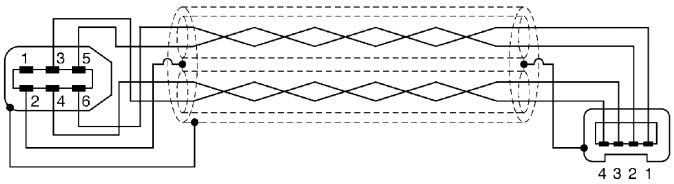
Figure 4-11I — Cable material construction example (for reference only)

4.2.1A.2.2 Cable assemblies

Cable assemblies consist of two plug connectors, either the 6-pin connector or the 4-pin connector, joined by a length of cable material. Subclause 4.2.1A.3 describes the performance requirements for the cable assemblies.

The suggested maximum length is 4.5 m. This is to assure that a maximally configured cable environment does not exceed the length over which the end-to-end signal propagation delay would exceed the allowed time. Longer cable lengths are possible if special consideration is given to the actual Serial Bus system topology to be used, as discussed in greater detail in annex M.

Both cable configurations, 6-pin connector to 4-pin connector and 4-pin connector to 4-pin connector, are illustrated in figure 4-11J and figure 4-11K. The connector pins are terminated as shown by figure 4-11J and figure 4-11K. The two signal pairs “cross” in the cable to effect a transmit-to-receive interconnection.



Note: Connectors are viewed as looking at the front plug face.

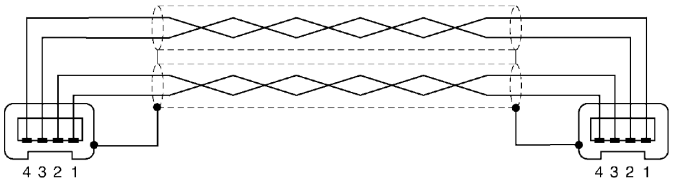
Signal Names for reference only

Pin No.	Signal
1	VP
2	VG
3	TPB*
4	TPB
5	TPA*
6	TPA

Signal Names for reference only

Pin No.	Signal
1	TPB*
2	TPB
3	TPA*
4	TPA
Shell	VG

Figure 4-11J — Cable assembly and schematic (6-pin to 4-pin connector)



Note: Connectors are viewed as looking at the front plug face.

Signal Names
Both ends identical; for reference only

Pin No.	Signal
1	TPB*
2	TPB
3	TPA*
4	TPA
Shell	VG

Figure 4-11K — Cable assembly and schematic (4-pin connectors)

4.2.1A.3 Connector and cable assembly performance criteria

To verify the performance requirements, performance testing is specified according to the recommendations, test sequences, and test procedures of ANSI/EIA 364-C-94; table 1 in that standard shows operating class definitions for different end-use applications. For Serial Bus, the test specifications follow the recommendations for environmental class 1.3, which is defined as “no air conditioning or humidity control with normal heating and ventilation.” The Equipment Operating Environmental Conditions shown for class 1.3 in table 2 are

— Temperature: +l5 °C to +85 °C

— Humidity: 95% maximum

Class 1.3 is further described as operating in a “harsh environmental” state, but with no marine atmosphere.

Accordingly, the performance groupings, sequences within each group, and the test procedures shall follow the recommendations of ANSI/EIA 364-C-94, except where the unique requirements of the Serial Bus connector and cable assembly may call for tests which are not covered in ANSI/EIA 364-C-94, or where the requirements deviate substantially from those in that document. In those cases, test procedures of other recognized authorities or specific procedures described in the annexes are cited.

Sockets, plugs, and cable assemblies shall perform to the requirements and pass all the following tests in the groups and sequences shown.

Testing may be done as follows:

a) *Plug and socket only.* In this case, for those performance groups that require it, the plugs may be assembled to the cable, to provide a cable assembly, by the connector manufacturer or by a cable assembly supplier.
b) *Cable assembly (with a plug on each end) and socket.* In this case, a single supplier may do performance testing for both elements or a connector supplier may team up with a cable assembly supplier to do performance testing as a team.
c) *Cable assembly only (with a plug on each end).* In this case, the cable assembly supplier should use a plug connector source that has successfully passed performance testing, according to this standard.
d) *Plug only or socket only.* In this case, the other half shall be procured from a source that has successfully passed performance testing, according to this standard. For those performance groups that require it, the plugs may be assembled to the cable, to provide a cable assembly, by the connector manufacturer or by a cable assembly supplier.

NOTES

1—All performance testing is to be done with cable material that conforms to this standard. In order to test to these performance groups, ANSI/EIA 364-C-94 requires that the test results specify the cable construction used.

2—All resistance values shown in the performance groups in 4.2.1A.3.1 through 4.2.1A.3.7 are for connectors only, including their terminations to the wire and/or PC board, but excluding the resistance of the wire. Resistance measurements shall be performed in an environment of temperature, pressure, and humidity specified by ANSI/EIA 364-C-94.

3—The number of units to be tested is a recommended minimum; the actual sample size is to be determined by requirements of users. This is not a qualification program.

4.2.1A.3.1 Performance group A: Basic mechanical dimensional conformance and electrical functionality when subjected to mechanical shock and vibration

Number of samples

[2] Sockets, not assembled to printed circuit board used for Phase 1, A1, and A2 (one each)
[2] Sockets, assembled to printed circuit board
[2] Plugs, not assembled to cable used for Phase 1, A1, and A2 (one each)
[2] Cable assemblies with a plug assembled to one end, 25.4 cm long

See table 4-11B for testing of performance group A.

Table 4-11A — Performance group A

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance Level
A1	Visual and dimensional inspection	ANSI/EIA 364-18A-84	Unmated connectors	Dimensional inspection	Per figure 4-11A through figure 4-11E	No defects that would impair normal operations. No deviation from dimensional tolerances.
A2	Plating thickness measurement	—	—	—	—	Record thickness; see 4.2.1A.1.4
A3	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated pair.
A4	Vibration	ANSI/EIA 364-28C-97	Condition II (See following paragraph)	Continuity	ANSI/EIA 364-46A-98	No discontinuity at 1 μs or longer. (Each contact)
A5	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
A6	Mechanical shock (specified pulse)	ANSI/EIA 364-27B-96	Condition G (See following paragraph)	Continuity	ANSI/EIA 364-46A-98	No discontinuity at 1 μs or longer. (Each contact)
A7	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

Connectors are to be mounted on a fixture that simulates typical usage. The socket shall be mounted to a panel that is permanently affixed to the fixture. The mounting means shall include typical accessories such as

a) An insulating member to prevent grounding of the shell to the panel.
b) A printed circuit board in accordance with the pattern shown in figure 4-11G or figure 4-11H for the socket being tested. The printed circuit board shall also be permanently affixed to the fixture.

The plug shall be mated with the socket and the other end of the cable shall be permanently clamped to the fixture. Refer to figure 4-10 for details.

4.2.1A.3.2 Performance group B: Low-level contact resistance when subjected to thermal shock and humidity stress

Number of samples

[0] Sockets, not assembled to printed circuit board
[2] Sockets, assembled to printed circuit board
[0] Plugs, not assembled to cable
[2] Cable assemblies with a plug assembled to one end, 25.4 cm long

See table 4-11C for testing of performance group B.

Table 4-11B — Performance group B

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
B1	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
B2	Thermal shock	ANSI/EIA 364-32B-92	Condition I 10 cycles (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
B3	Humidity	ANSI/EIA 364-31A-83	Condition A (96 hours) Method III (cycling) nonenergized Omit steps 7a and 7b (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

4.2.1A.3.3 Performance group C: Insulator integrity when subjected to thermal shock and humidity stress

Number of samples

[2] Sockets, not assembled to printed circuit board
[0] Sockets, assembled to printed circuit board
[2] Plugs, not assembled to cable used for Phase 1, A1, and A2 (one)
[0] Cable assemblies with a plug assembled to one end, 2 m long

See table 4-11D for testing of performance group C.

Table 4-11C — Performance group C

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
C1	Withstanding voltage	ANSI/EIA 364-20A-83	Test voltage 100 V dc ± 10 V dc Method C (unmated and unmounted)	Withstanding voltage	ANSI/EIA 364-20A-83	No flashover. No sparkover. No excess leakage. No breakdown.
C2	Thermal shock	ANSI/EIA 364-32B-92	Condition I 10 cycles (unmated)	Withstanding voltage (same conditions as C1)	ANSI/EIA 364-20A-83	No flashover. No sparkover. No excess leakage. No breakdown.

Table 4-11C — Performance group C (continued)

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
C3	Insulation resistance	ANSI/EIA 364-21B-95	Test voltage 100 V dc ± 10 V dc (unmated and unmounted)	Insulation resistance	ANSI/EIA 364-21B-95	100 MΩ, minimum, between adjacent contacts and contacts and shell.
C4	Humidity (cyclic)	ANSI/EIA 364-31A-83	Condition A (96 hours) Method III nonenergized Omit steps 7a and 7b	Insulation resistance (same conditions as C3)	ANSI/EIA 364-21B-95	100 MΩ, minimum.

4.2.1A.3.4 Performance group D: Contact life and durability when subjected to mechanical cycling and corrosive gas exposure

Number of samples

[0] Sockets, not assembled to printed circuit board
[4] Sockets, assembled to printed circuit board
[0] Plugs, not assembled to cable used for Phase 1, A1, and A2 (one)
[4] Cable assemblies with a plug assembled to one end, 25.4 cm long

See table 4-11E for testing of performance group D.

Table 4-11D — Performance group D

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D1	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
D2	Continuity-housing (shell)	—	See figure 4-11L for measurement points	Contact resistance, braid to socket shell	ANSI/EIA 364-06A-83	50 mΩ, maximum, initial from braid to socket shell at 100 mA, 5 V dc open circuit maximum.
D3	Durability	ANSI/EIA 364-09B-91	Class II Exposures: (a) 2 mated pairs, 5 cycles (b) 2 mated pairs, automatic cycling to 500 cycles, rate 500 cycles/h ± 50 cycles.	—	—	—
D4	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

Table 4-11D — Performance group D (continued)

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D5	Continuity-housing (shell)		See figure 4-11L for measurement points	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell at 100 mA, 5 V dc open circuit maximum.
D6	Mixed flowing gas	ANSI/EIA 364-65A-97	Class II Exposures: (a) 2 mated pairs; unmated for 1 day (b) 2 mated pairs; mated 10 days	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D7	Durability	ANSI/EIA 364-09B-91	Class II Exposures: (a) 2 mated pairs, 5 cycles (b) 2 mated pairs, automatic cycling to 500 cycles, rate 500 cycles/h ± 50 cycles	—	—	—
D8	Mixed flowing gas	ANSI/EIA 364-65A-97	Class II Exposures: Expose mated for 10 days	Low-level contact resistance at end of exposure	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D9	Continuity-housing (shell)		See figure 4-11L for measurement points	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell at 100 mA, 5 V dc open circuit maximum.

4.2.1A.3.5 Performance group E: Contact resistance and unmating force when subjected to temperature life stress

Number of samples

[0] Sockets, not assembled to printed circuit board
[2] Sockets, assembled to printed circuit board
[0] Plugs, not assembled to cable used for Phase 1, A1, and A2 (one)
[2] Cable assemblies with a plug assembled to one end, 2 m long

See table 4-11F for testing of performance group E.

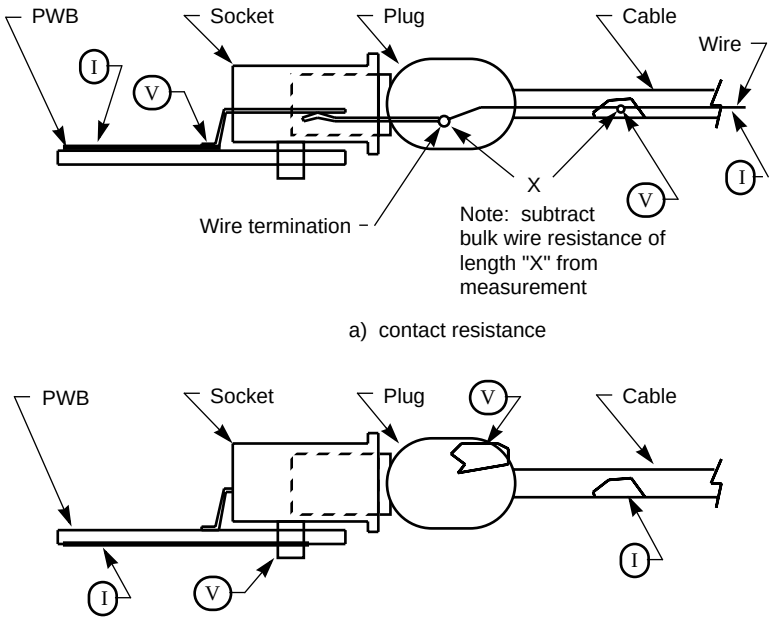


Figure 4-11L — Shield and contact resistance measuring points

Table 4-11A — Performance group E

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
E1	Mating and unmating forces	ANSI/EIA 364-13A-83	Mount socket rigidly; insert receptacle by hand	Mating only	—	—
			Auto Rate: 25 mm/min	Unmating only	ANSI/EIA 364-13A-83	Unmating force: 4.9 N minimum 39.0 N maximum
E2	None	—	—	Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
E3	Continuity-housing (shell)		See figure 4-11L	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum initial from braid to socket shell at 100 mA, 5 V dc open circuit maximum.
E4	Temperature life	ANSI/EIA 364-17A-87	Condition 2 (79° C) 96 hours Method A (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

Table 4-11A — Performance group E (continued)

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
E5	Continuity-housing (shell)	—	—	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell at 100 mA, 5 V dc open circuit maximum.
E6	Mating and unmating forces	ANSI/EIA 364-13A-83	Mount socket rigidly; insert plug by hand.	Mating only	—	—
			Auto Rate: 25 mm/min	Unmating only	ANSI/EIA 364-13A-83	Unmating force: 4.9 N minimum 39.0 N maximum

4.2.1A.3.6 Performance group F: Mechanical retention and durability

Number of samples

[0] Sockets, not assembled to printed circuit board
[2] Sockets, assembled to printed circuit board
[0] Plugs, not assembled to cable
[2] Plugs, assembled to cable, one end only, 25 cm long

See table 4-11G for testing of performance group F.

Table 4-11B — Performance group F

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
F1	Mating and unmating forces	ANSI/EIA 364-13A-83	Mount socket rigidly; insert plug by hand.	Mating only	—	—
F2	Mating and unmating forces	ANSI/EIA 364-13A-83	Auto rate: 25 mm/min	Unmating only	ANSI/EIA 364-13A-83	Unmating force: 4.9 N minimum 39.0 N maximum
F3	Durability	ANSI/EIA 364-09B-91	Automatic cycling to 1000 cycles. 500 cycles/h ± 50 cycles	Unmating only	ANSI/EIA 364-13A-83	Unmating force at end of durability cycles: 4.9 N minimum 39.0 N maximum

4.2.1A.3.7 Performance group G: General tests

Suggested procedures to test miscellaneous but important aspects of the interconnect.

Since the tests listed in table 4-11H may be destructive, separate samples must be used for each test. The number of samples to be used is listed under the test title.

Table 4-11C — Performance group G

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
G1	Electrostatic Discharge [1 plug] [1 socket]	IEC 61000-4-2 (1999-05)	1–8 kV in 1 kV steps. Use 8 mm ball probe. Test unmated.	Evidence of discharge	—	No evidence of discharge to any of the 4 contacts; discharge to shield is acceptable.
G2	Cable axial pull test [2 cable assemblies]	—	Fix plug housing and apply a 49.0 N load for 1 min on cable axis	Continuity, visual	ANSI/EIA 364-46A-98	No discontinuity on contacts or shield greater than 1 µs under load. No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.
G3	Cable flexing [2 cable assemblies]	ANSI/EIA 364-41B-89	Condition I, dimension X = 5.5 × cable diameter; 100 cycles in each of two planes (see figure 4-11M).	(a) Withstanding voltage	Per C1	Per C1
				(b) Insulation resistance	Per C3	Per C3
				(c) Continuity	ANSI/EIA 364-46A-98	No discontinuity on contacts or shield greater than 1 µs during flexing.
				(d) Visual	—	No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.

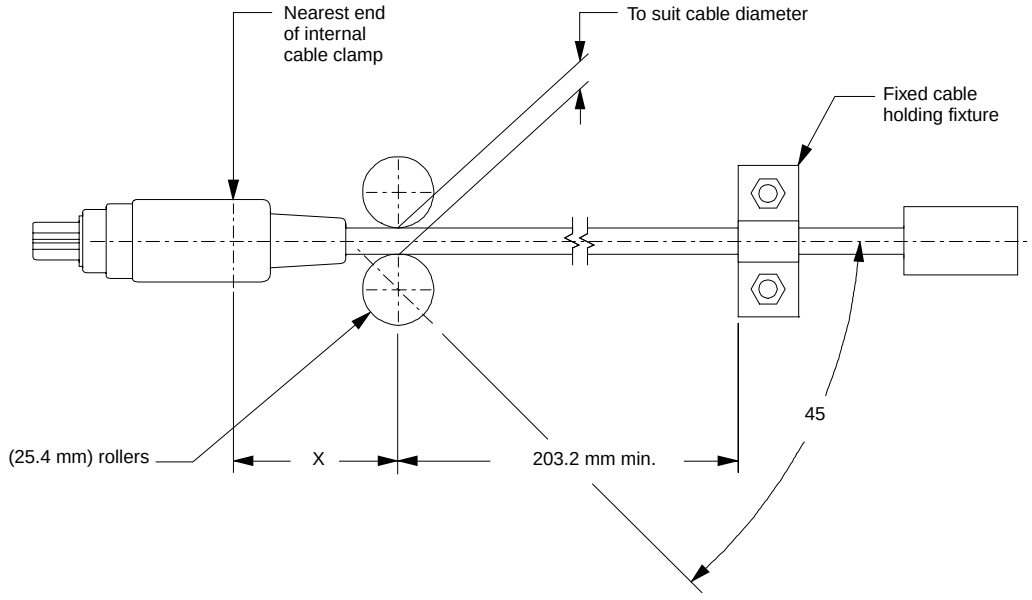


Figure 4-11M — Fixture for cable flex test

4.2.1A.4 Signal propagation performance criteria

The test procedures for all parameters listed in this clause are described in annex K.

4.2.1A.4.1 Signal impedance

The differential mode characteristic impedance of the signal pairs within the cable section shall be measured by time domain reflectometry at less than 100 ps rise time; the recommended procedure is described in K.3.

Z_{TPA} = (110 ± 6) Ω (differential)
Z_{TPB} = (110 ± 6) Ω (differential)

The common-mode characteristic impedance of the signal pairs within the cable section shall be measured by time domain reflectometry at less than 100 ps rise time; the recommended procedure is described in K.3.

Z_{TPACM} = (33 ± 6) Ω (common mode)
Z_{TPBCM} = (33 ± 6) Ω (common mode)

The differential mode discrete impedance of the signal pairs within the mated connector section shall be measured by time domain reflectometry at less than 500 ps rise time; the recommended procedure is described in K.3.

$Z_{TPACONN}$ = (110 ± 25) Ω (differential)
$Z_{TPBCONN}$ = (110 ± 25) Ω (differential)

4.2.1A.4.2 Signal pairs attenuation

A signal pairs attenuation requirement applies only to the two signal pairs, for any given cable assembly. Attenuation is measured using the procedure described in K.4.

Frequency	Attenuation
100 MHz	Less than 2.3 dB
200 MHz	Less than 3.2 dB
400 MHz	Less than 5.8 dB

4.2.1A.4.3 Signal pairs propagation delay

The differential propagation delay of the signal pairs through the cable shall be measured in the frequency domain; the recommended procedure is described in K.5.

$V_{TPA} \leq 5.05$ ns/m
$V_{TPB} \leq 5.05$ ns/m

The common-mode propagation delay of the signal pairs shall be less than or equal to the differential propagation delay. For typical cable constructions this is correct by design; consequently no test procedure is described for this in K.5.

$V_{TPACM} \leq 5.05$ ns/m
$V_{TPBCM} \leq 5.05$ ns/m

4.2.1A.4.4 Signal pairs relative propagation skew

The difference between the differential mode propagation delay of the two signal twisted pairs shall be measured in the frequency domain; the recommended procedure is described in K.6.1.

$S \leq 400$ ps

4.2.1A.4.5 Crosstalk

The TPA-TPB and signal-power crosstalk shall be measured in the frequency domain using a network analyzer in the frequency range of 1–75 MHz; the recommended procedures are described in K.8.

$X \leq -26$ dB

4.2.2 Media signal interface

General Background

Cable physical layer performance enhancement specifications. The following amendments to clause 4 specify a set of related enhancements to the physical layer of Serial Bus. When implemented as a group they can significantly increase both the efficiency and robustness of Serial Bus. The enhancements address the following:

— *Connection hysteresis (debounce).* When a connector is inserted or removed from a socket, electrical contact is made and broken countless times in a short interval. The existing standard does not take this into account and, as a consequence, a storm of bus resets occurs when a connection is made or broken. These resets are highly disruptive to isochronous traffic on the bus. IEEE Std 1394a-2000 specifies a connection time-out to avoid the problem.
— *Arbitrated (short) bus reset.* The current definition of bus reset assumes that the state of the bus is not known when a reset is initiated. The minimum reset assertion time must be long enough to complete any packet transmission that may have been in progress. However, if reset is asserted after first arbitrating for the bus, the minimum reset time can be significantly reduced.
— *Multiple-speed packet concatenation.* There is a defect in IEEE Std 1394-1995 in that PHYs are required to transmit a speed-signal only for the first packet of a multiple packet sequence, yet they are expected to receive a separate speed-signal for each packet. Faced with this contradiction, different vendors have attempted sensible interpretations; the interpretations have not been uniform and this has already resulted in observed interoperability problems with PHYs from different vendors. New requirements for PHYs in IEEE Std 1394a-2000 correct the defect and promote interoperability between existing PHYs and those that conform to IEEE Std 1394a-2000.
— *Arbitration improvements.* There are two circumstances identified in which a node may arbitrate for the bus without first observing a subaction gap. One occurs when an isochronous or acknowledge packet is received from a child port—fly-by concatenation. The other occurs subsequent to the observation of an acknowledge packet—ack-accelerated arbitration.
— *Transmission delay calculation (PHY pinging).* The ability to transmit a “ping” packet to a PHY and time its return permits the inclusion of cables longer than 4.5 m or PHYs with delays longer than 144 ns into Serial Bus topologies.
— *Remote PHY register read.* The ability to interrogate another PHY’s registers; the response to a remote register read may be used to time round-trip delay in the same fashion as a “ping” packet.
— *Extended speed encoding.* Although speeds in excess of S400 are not specified by IEEE Std 1394a-2000, the coding infrastructure in the PHY/link interface, PHY registers, and self-ID packets is established for future use.

— *Per port disable, suspend, and resume.* The model of PHY behavior is extended to recognize that the arbitration state machines defined by IEEE Std 1394a-2000 are applicable to active, connected ports. Inactive ports may be in other states (disabled, disconnected, or suspended) in which the PHY may reduce its power consumption.

These enhancements affect virtually all characteristics of the PHY, from reset detection to the normal arbitration state machines. As a consequence they are difficult to specify in isolation.

Cable topology. Informative clauses of IEEE Std 1394-1995 assume that Serial Bus cable topologies are limited by individual cable lengths less than or equal to 4.5 m and a maximum hop count (between the two most distant leaf nodes) of 16. There are no normative statements in IEEE Std 1394-1995 that mandate either of these requirements.

New facilities specified by IEEE Std 1394a-2000, the ping packet and the self-ID packet(s) sent in response, permit the configuration of usable Serial Bus topologies with longer cables or greater hop counts. Any topology whose arbitration behavior can be characterized by a gap count less than or equal to $3F_{16}$ is permitted.

NOTE—In the absence of a bus manager equipped to time the worst-case Serial Bus round trip delay, cable lengths less than or equal to 4.5 m and a maximum hop count of 16 are recommended.

4.2.2 Media signal interface

Background

The text at the beginning of 4.2.2 in IEEE Std 1394-1995 specifies requirements for the circuits of each port of a PHY. IEEE Std 1394a-2000 adds additional requirements for a current source and connect detect circuit; these facilities permit the detection of a connection with a peer PHY while this PHY is in a low-power state.

Replace the title of 4.2.2, as well as the next three paragraphs and figure 4-12 with the following:

4.2.2 Port interface

A port (previously called the *cable media signal interface* in IEEE Std 1394-1995) consists of two twisted pair interfaces, TPA/TPA* and TPB/TPB*, and an optional power distribution pair, VP/VG. A node may have from one to sixteen such ports. Each port has associated circuitry that provides separate signals for arbitration or for packet data reception and transmission, as shown in figure 4-12.

The TPA/TPA* pair transmit the Strb_Tx signal and receive the Data_Rx, Arb_A_Rx, and Speed_Rx signals. The TPB/TPB* pair transmits the Data_Tx and Speed_Tx signals and receives the Strb_Rx, Arb_B_Rx, and Bias_Detect signals. The Strb_Tx, Data_Tx, Strb_Enable, and Data_Enable signals are used together to generate the arbitration signals. The Arb_A_Rx and Arb_B_Rx signals are each generated by two comparators since they have three states—zero, one, or high impedance.

A PHY (or a circuit board which includes a PHY) shall not assert voltages greater than 0.4 V on TPA/TPA* while unpowered. Care should be taken to insure this requirement is met in the case that the peer PHY is powered and generating TpBias.

When enabled, the TPA/TPA* pair transmit TpBias while the TPB/TPB* pair receive the TpBias signal (this is used by the Bias_Detect comparator to determine presence or absence of TpBias). When TpBias is not generated, the connect detect circuit can detect the presence or absence of a peer PHY at the other end of a cable connection. Designers shall select appropriate threshold voltage and hysteresis values for the connect detect circuit so as to avoid erroneous detection of connection status changes while TpBias discharges.

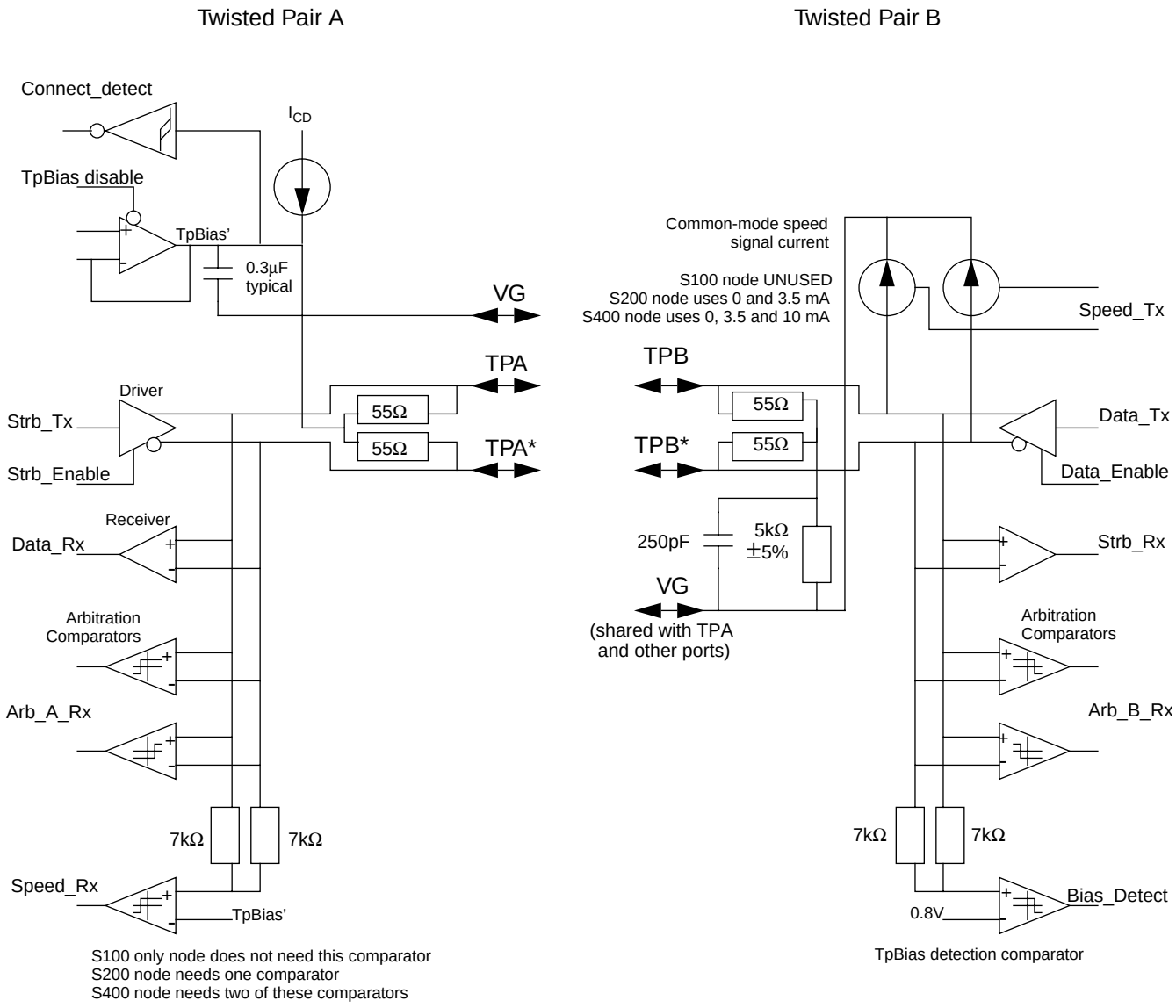


Figure 4-12 — Port interface

The bias generation circuit (including the external capacitor) shall be designed so that when TpBias is driven low for at least 0.5 ms, the capacitor shall be discharged and output bias shall be less than 0.1 V relative to VG. Contrariwise, when TpBias is generated the capacitor shall be recharged and the TPA/TPA* signals shall be within the specifications of table 4-14 (within 1.0 ms).

NOTE—Designers should examine the interrelationships between these times and BIAS_HANDSHAKE in order to avoid a fault during resume operations.

The current source used by the connect detect circuit, I_{CD}, shall not exceed 76 µA. This guarantees that the input to the peer PHY's bias detection circuit does not exceed 0.4 V.

Replace 4.2.2.7 with the following:

4.2.2.7 Cable power and ground

A node may be a power source, a power sink, or neither and may assume different roles at different times. The principal method by which a node identifies its power class is the self-ID packet transmitted subsequent to a bus reset (see 4.3.4.1). There may be other facilities, e.g., in a node's configuration ROM, that identify the power characteristics of a node in more detail than is possible in the self-ID packet; these are beyond the scope of this standard.

Serial Bus may be unpowered or powered; in the latter case, there may be more than one power source. The possibility of multiple sources requires that cable power sources be manufactured such that current from a node providing higher voltage does not flow into sources of lower output voltage. Cable power sources, whether or not identified as such by their self-ID packets, shall not permit the inflow of power from a higher voltage source. Cable power sources that supply a minimum of 20 V shall identify themselves with POWER_CLASS one, two, or three in their self-ID packet(s) and shall implement, for each of their ports, a diode and current-limiting scheme whose behavior is equivalent to that illustrated by figure 4-16. Different implementations are possible; the number or location of fuses or other current-limiting devices may vary to meet design needs or agency requirements.

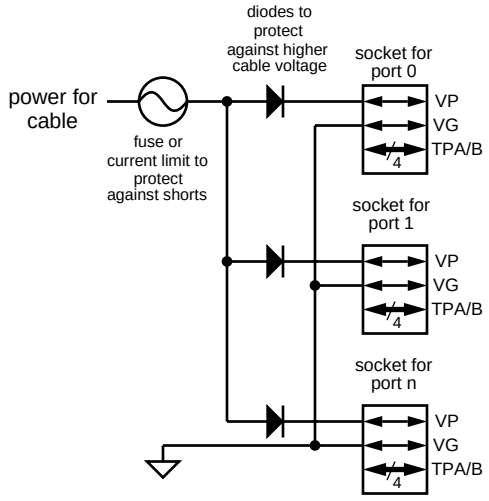


Figure 4-16—Node power interface for POWER_CLASS one, two, or three

Devices that implement three or more ports and identify themselves with POWER_CLASS four in their self-ID packet(s) shall implement, for each of their ports, a current-limiting scheme whose behavior is equivalent to that illustrated by figure 4-16A. The components shown in the shaded block are necessary only in the case that the device is also a cable power source. Different implementations are possible; the number or location of fuses or other current-limiting devices may vary to meet design needs or agency requirements.

All cable power sources shall meet the requirements of table 4-20 (which shall be measured at the printed circuit board side of the connector socket).

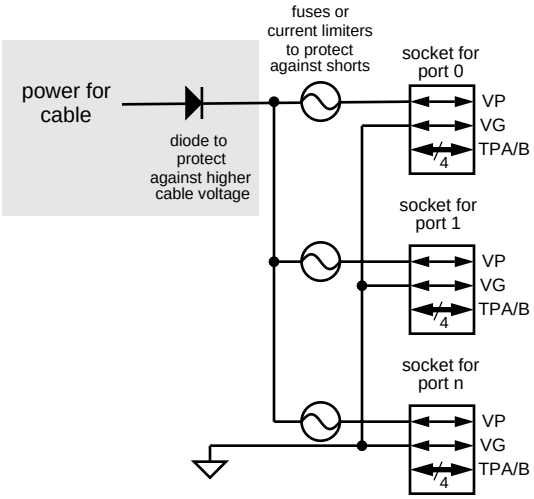


Figure 4-16A—Node power interface for POWER_CLASS four (three or more ports)

Table 4-20—Cable power source requirements

Condition	Limit	Units
Maximum output current per port	Specified by IEC 60950 (1999-04)	
Minimum output voltage (POWER_CLASS one, two, or three)	20	V dc
Minimum output voltage (POWER_CLASS four, if power is provided)	8	V dc
Maximum output voltage	30	V dc
Maximum output ripple (1 kHz to 400 MHz)	100	mV (peak-to-peak)

In addition, cable power sources shall provide over-voltage and short-circuit protection in compliance with the limited power source requirements in clause 2.5 of IEC 60950 (1999-04). Implementations might utilize impedance protection, such as polymer PTC devices, compliant with table 2B of IEC 60950 (1999-04) or overcurrent protection, such as a fuses, in compliance with table 2C of that standard.

When cable power is available, it is in the nominal range 8–30 V. A PHY that detects cable power at the connector of at least 7.5 V shall set the PS bit (cable power active) in its registers to one. The PS bit shall be cleared to zero when detectable voltage is below this value. When the PS bit transitions from one to zero, the PHY shall generate a PH_EVENT.*indication* of CABLE_POWER_FAIL.

A cable powered PHY may not be operational if the voltage falls below 7.5 V. If a cable PHY remains operational at the reduced voltage, it shall report the loss of cable power as specified previously.

If a node uses cable power, it shall meet the following requirements:

a) It shall sink no more than 1.5 A of current.

b) It shall consume no more than 3 W of power after a power reset or after being initially connected to the bus (transition from all ports unconnected to any port connected). The receipt of a PHY link-on packet enables the node to consume additional power up to the limit specified by the node's self-ID packet(s).

c) Inrush energy shall not exceed 18 mJ in 3 ms.

d) The node's current consumption, expressed as a function of the node's maximum current requirements, I_{load} in A(s), shall meet the following requirements:

   1) The peak-to-peak ripple shall be less than or equal to (I_{load} / 1.5 A) × 100 mA.
   2) The slew rate (change in load current) shall be less than I_{load} in any 100 µs period.

The sum of the dc currents on VG and VP, for any node that consumes cable power, should be less than 50 µA. This does not imply a requirement for galvanic isolation but encourages good design (the return of power supply current via VG rather than an alternate path).

When power is available from electric mains or from batteries, all nodes with two or more ports shall repeat bus signals on all ports that are both connected and enabled. When power is not available the node shall either

— Power its PHY from cable power, if available, and repeat bus signals on all ports that are both connected and enabled, or
— In the case where the PHY is off, prevent cable power from flowing from any port to any other port.

Nodes may be part of a module that implements a "soft" power switch. When the module connected to the electric mains is powered off, the preferred method to power the PHY is a trickle source from the electric mains. For battery powered modules that are powered off or for other modules when trickle power is not feasible, the preferred alternative is to power the PHY from the cable. The last alternative, an inactive PHY and a break in the bus power distribution, is not recommended.

A node that repeats cable power shall have a maximum resistance between any two connector sockets of 0.5 Ω. The measurement shall be made at the printed circuit board side of the connector socket.

4.2.3 Media signal timing

4.2.3.2 Data signal rise and fall times

Background

Output rise and fall times for data signals, given in table 4-22 of IEEE Std 1394-1995, are measured from 10% to 90% at the connector and are dependent on the data rate. IEEE Std 1394a-2000 adds minimum rise and fall times to the specification.

Replace table 4-22 with the following table and note:

Table 4-22—Output rise and fall times

Speed	Rise or fall time	
	Minimum (ns)	**Maximum (ns)**
S100	0.5	3.2
S200	0.5	2.2
S400	0.5	1.2

NOTE—The differential received signal amplitude specification in table 4-13 is not a receiver sensitivity specification for PHY inputs. Designers should consider factors such as the worst-case received waveform (e.g., slow rise or fall times near the signal thresholds) and board design characteristics when choosing receiver sensitivity.

4.3 Cable PHY facilities

4.3.3 Cable PHY line states

Background

New rules by which a PHY encodes arbitration line states on the two transmit arbitration signals, Arb_A_Tx and Arb_B_Tx, are defined.

Insert the following rows into table 4-27:.

Table 4-27—Cable PHY transmitted arbitration line states

Arbitration transmit		Line state name	Comment
Arb_A_Tx	**Arb_B_Tx**		
Z	1	TX_DISABLE_NOTIFY	Request the peer PHY port to enter the suspended state. The transmitting port will be disabled.
0	0	TX_SUSPEND	Request the peer PHY to handshake TpBias and enter the suspended state. The request is also propagated by the peer PHY to its other active ports.

Background

New rules by which a PHY decodes the interpreted arbitration signals, Arb_A and Arb_B, into a line state are defined.

Insert the following rows into table 4-28:

Table 4-28—Cable PHY received arbitration line states

Interpreted arbitration signals		Line state name	Comment
Arb_A	**Arb_B**		
0	0	RX_SUSPEND	Exchange TpBias handshake with the peer PHY and place the port into the suspended state. Also initiate suspend (i.e., propagate TX_SUSPEND) on all other active ports.
1	Z	RX_DISABLE_NOTIFY	Enter the suspended state and initiate short bus reset on all other active ports. The peer PHY port will be disabled.

4.3.4 Cable PHY packets

Background

IEEE Std 1394a-2000 extends the definition of PHY packets. For convenience of reference and to correct typographical errors, all of the existing PHY packet definitions are reproduced, followed by the new definitions.

Replace the text in 4.3.4 with the following:

Nodes send and receive a small number of short packets, which are used for bus management. These PHY packets all consist of 64 bits, with the second 32 bits being the logical inverse of the first 32 bits; they are all sent at the S100 speed. If the first 32 bits of a received PHY packet do not match the complement of the second 32 bits, the PHY packet shall be ignored. All received PHY packets are transferred to the link; all transmitted PHY packets, except those originated by the link, are also transferred to the link.

The cable physical layer packet types are

a) The self-ID packet
b) The link on packet
c) The PHY configuration packet
d) The extended PHY packets

NOTE—The PHY packets can be distinguished from an isochronous packet with no data payload (the only primary packet with exactly two quadlets) since the latter uses a 32-bit CRC as the second quadlet, while the PHY packets use the bit inverse of the first quadlet as the second.

PHY packets originated by the link shall be transmitted only when the bus has been granted as the result of either fair or priority arbitration.

Self-ID packets autonomously generated by the PHY shall also be transferred to the attached link. This differs from the behavior of PHYs compliant with IEEE Std 1394-1995 (but unmodified by IEEE Std 1394a-2000).

PHY packets originated by the attached link shall be processed by the PHY as if they were received from Serial Bus.

4.3.4.1 Self-ID packets

The cable PHY sends one to three self-ID packets at the base rate during the self-ID phase of arbitration or in response to a “ping” packet. The number of self-ID packets sent depends on the maximum number of ports implemented. The cable PHY self-ID packets have the format shown in figure 4-18.

Self-ID packets with sequence numbers, n, between 2 and 7, inclusive, are reserved for future standardization.

NOTE—This standard defines self-ID packet formats that permit a maximum of four self-ID packets to be generated by a PHY. Although IEEE Std 1394a-2000 defines only three self-ID packets, link designers should accommodate the reception of up to 252 self-ID packets during the self-identify process. Such a link design both supports this standard and permits future Serial Bus standards to define an additional self-ID packet without adverse impact on contemporary links.

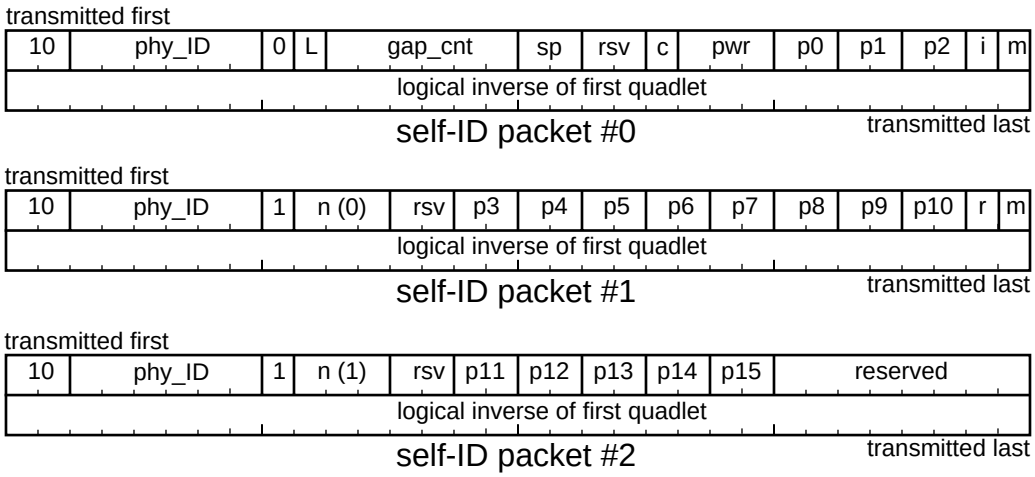


Figure 4-18—Self-ID packet formats

Table 4-1—Self-ID packet fields

Field	Derived from	Comment
phy_ID	`physical_ID`	Physical node identifier of the sender of this packet.
L	`LPS` `LCtrl`	If set, this node has active link and transaction layers. In discrete PHY implementations, this shall be the logical AND of LCtrl and LPS active.
gap_cnt	`gap_count`	Current value for this node's PHY_CONFIGURATION.gap_count field.
sp	`PHY_SPEED`	Speed capabilities: 00_2 S100 01_2 S100 and S200 10_2 S100, S200, and S400 11_2 Reserved for future standardization See 4.2.3.1 for exact definitions in Mbit/s.
c	`CONTENDER`	If set and the L bit is set, this node is a contender for the bus or isochronous resource manager as described in 8.4.2 (as amended by IEEE Std 1394a-2000).
pwr	`POWER_CLASS`	Power consumption and source characteristics: 000_2 Node does not need power and does not repeat power. 001_2 Node is self-powered and provides a minimum of 15 W to the bus. 010_2 Node is self-powered and provides a minimum of 30 W to the bus. 011_2 Node is self-powered and provides a minimum of 45 W to the bus. 100_2 Node may be powered from the bus and is using up to 3 W. No additional power is needed to enable the link.[a] 101_2 Reserved for future standardization. 110_2 Node is powered from the bus and is using up to 3 W. An additional 3 W is needed to enable the link.[a] 111_2 Node is powered from the bus and is using up to 3 W. An additional 7 W is needed to enable the link.[a]
p0 … p15	`NPORT`, `child[n]`, `active[n]`	Port connection status: 00_2 Not present on this PHY. 01_2 Not active (may be disabled, disconnected or suspended). 10_2 Active and connected to parent node. 11_2 Active and connected to child node.

Table 4-1—Self-ID packet fields (continued)

Field	Derived from	Comment
i	initiated_reset	If set, this node initiated the current bus reset (i.e., it started sending a BUS_RESET signal before it received one).[b] Once set, it remains one through the bus reset and clears to zero upon a subsequent bus reset only if the node is not an initiator of that bus reset.
m	more_packets	If set, another self-ID packet for this node immediately follows (i.e., if this bit is set and the next self-ID packet received has a different phy_ID, then a self-ID packet was lost).
n	—	Extended self-ID packet sequence number.
rsv	—	Reserved for future standardization; set to zeros.

[a]The link is enabled by the PHY link-on packet described in 4.3.4.2; this packet may also enable application layers.
[b]There is no guarantee that exactly one node has this bit set. More than one node may request a bus reset at the same time.

Some of the information in the self-ID packets changes in accordance with the node's operating mode. For example, a node that is initially a power consumer but subsequently supplies power would report a different value for the pwr field. Whenever any part of the node's configuration described by the self-ID packets changes and there is no expectation that interested parties would otherwise discover the change(s), the node should initiate a bus reset in order to transmit updated self-ID packets.

4.3.4.2 Link-on packet

Reception of the following cable PHY packet shall cause a PH_EVENT.*indication* of LINK_ON if the *phy_ID* field matches the PHY's physical ID (See 8.4.4 for more information).

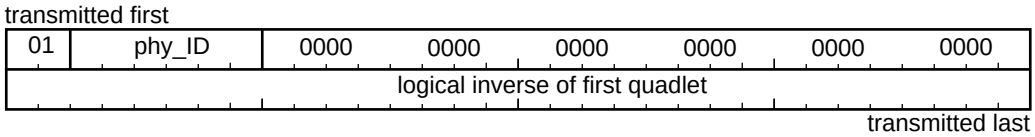


Figure 4-19—Link-on packet format

Table 4-1—Link-on packet fields

Field	Derived from	Comment
phy_ID	physical_ID	Physical node identifier of the destination of this packet.

NOTE—A link-on packet is advisory. A PHY that receives a link-on packet shall provide a PH_EVENT.*indication* of LINK_ON to its associated link but the link is not required to take any action. If a link does become functional in response to a link-on packet, there is no maximum time requirement.

4.3.4.3 PHY configuration packet

It is possible to configure Serial Bus performance in the following ways:

a) Optimize the gap_count used by all nodes to a smaller value (appropriate to the actual worst case round-trip delay between any two nodes), and
b) Force a particular node to be the root after the next bus initialization (e.g., to insure that the root is cycle master capable).

Both of these actions shall be effected for all nodes (including the originator) by means of the PHY configuration packet shown in figure 4-20 and table 4-31. The PH_CONTROL.*request* service affects only the local node and is not recommended for changes to either `gap_count` or `force_root`. The procedures for using this PHY packet are described in clause 8.

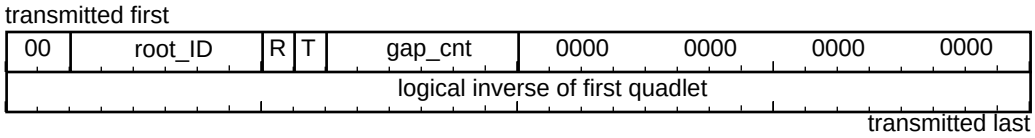


Figure 4-20—PHY configuration packet format

Table 4-1—PHY configuration packet fields

Field	Affects	Comment
root_ID		Physical ID of node to have its `force_root` bit set (only meaningful if R bit set)
R	`force_root`	If one, then the node with its `physical_ID` equal to this packet's *root_ID* shall have its `force_root` bit set, all other nodes shall clear their `force_root` bit. If cleared, the *root_ID* field shall be ignored.
T	`gap_count_reset_disable`	If one, all nodes shall set their `gap_count` variable to the value in this packet's *gap_cnt* field and set the `gap_count_reset_disable` variable to TRUE.
gap_cnt	`gap_count`	New value for all nodes' `gap_count` variables. This value goes into effect immediately on receipt and remains valid through the next bus reset. A second bus reset without an intervening PHY configuration packet resets `gap_count` to $3F_{16}$, as described in `reset_start_actions()` in 4.4.3.1.2)

It is not valid to transmit a PHY configuration packet with both R and T bits set to zero. This would cause the packet to be interpreted as an extended PHY packet.

4.3.4.4 Extended PHY packets

A PHY configuration packet with R = 0 and T = 0 is utilized to define extended PHY packets according to the value in the *gap_cnt* field (this is renamed the type field in figure 4-20A through figure 4-20F). The extended PHY packets have no effect upon either the `force_root` or `gap_count` variables of any node.

4.3.4.4.1 Ping packet

The reception of the cable PHY packet shown in figure 4-20A shall cause the node identified by phy_ID to transmit self-ID packet(s) that reflect the current configuration and status of the PHY. Because of other actions, such as the receipt of a PHY configuration packet, the self-ID packet(s) transmitted may differ from those of the most recent self-identify process.

A PHY shall transmit a self-ID packet within RESPONSE_TIME after the receipt of a ping packet.

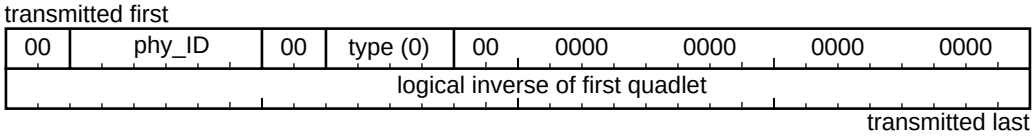


Figure 4-20A—Ping packet format

Table 4-31A—Ping packet fields

Field	Comment
phy_ID	Physical node identifier of the destination of this packet.
type	Extended PHY packet type (zero indicates ping packet).

4.3.4.4.2 Remote access packet

The reception of the cable PHY packet shown in figure 4-20B shall cause the node identified by phy_ID to read the selected PHY register and subsequently return a remote reply packet that contains the current value of the PHY register (see 4.3.4.4.3).

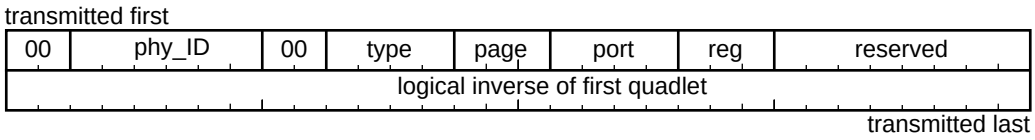


Figure 4-20B—Remote access packet format

Table 4-31B—Remote access packet fields

Field	Comment
phy_ID	Physical node identifier of the destination of this packet.
type	Extended PHY packet type: 1 Register read (base registers) 5 Register read (paged registers)
page	This field corresponds to the *Page_select* field in the PHY registers. The register read behaves as if *Page_select* was set to this value.
port	This field corresponds to the *Port_select* field in the PHY registers. The register read behaves as if *Port_select* was set to this value.
reg	This field, in combination with page and port, specifies the PHY register. If type indicates a read of the base PHY registers reg directly addresses one of the first eight PHY registers. Otherwise the PHY register address is 1000_2 + reg.

4.3.4.4.3 Remote reply packet

Subsequent to the reception of a remote access packet, the PHY shall transmit the packet shown in figure 4-20C.

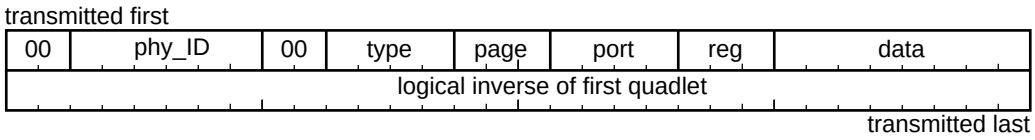


Figure 4-20C—Remote reply packet format

Table 4-31C—Remote reply packet fields

Field	Comment
phy_ID	Physical node identifier of the source of this packet.
type	Extended PHY packet type: 3 Register contents (base registers) 7 Register contents (paged registers)
page	This field corresponds to the *Page_select* field in the PHY registers; in conjunction with port and reg, it identifies the register whose contents are returned in data.
port	This field corresponds to the *Port_select* field in the PHY registers; in conjunction with page and reg, it identifies the register whose contents are returned in data.
reg	This field, in combination with page and port, identifies the register whose contents are returned in data. If type indicates a base PHY register, reg directly addresses one of the first eight PHY registers. Otherwise the PHY register address is 1000_2 + reg.
data	The current value of the PHY register addressed by the immediately preceding remote access packet. If the register is reserved or unimplemented, data shall be zero.

A PHY shall transmit a remote reply packet within RESPONSE_TIME after the receipt of a remote access packet.

4.3.4.4.4 Remote command packet

The reception of the cable PHY packet shown in figure 4-20D shall request the node identified by phy_ID to perform the operation specified and subsequently return a remote confirmation packet (see 4.3.4.4.5).

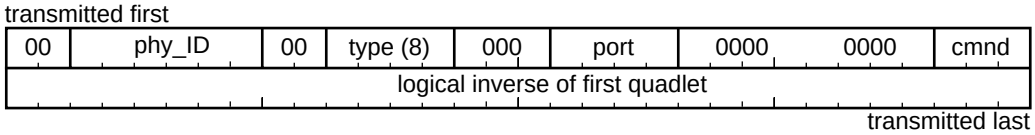


Figure 4-20D — Remote command packet format

Table 4-31D — Remote command packet fields

Field	Comment
phy_ID	Physical node identifier of the destination of this packet.
type	Extended PHY packet type (8 indicates command packet).
port	This field selects one of the PHY's ports.
cmnd	Command: 0 NOP 1 Transmit TX_DISABLE_NOTIFY then disable port 2 Initiate suspend (i.e., become a suspend initiator) 4 Clear the port's *Fault* bit to zero 5 Enable port 6 Resume port

Because a remote command packet may alter the power state of the addressed PHY, such a packet shall not be transmitted to any device unless the device has indicated, by means beyond the scope of this standard, that its power state may be managed by others. The absence of any such indication shall be interpreted as a refusal to grant power management privileges to others.

NOTE—Although this standard does not define any method for a device to advertise whether or not it participates in power management protocols, configuration ROM may provide the necessary information. If that is the case, simple devices without link and transaction layers (such as power bricks) would be exempt from power management.

4.3.4.4.5 Remote confirmation packet

Subsequent to the reception of a remote command packet, the PHY shall transmit the packet shown in figure 4-20E; it reports current status for the port and confirms whether or not the PHY initiated the requested action.

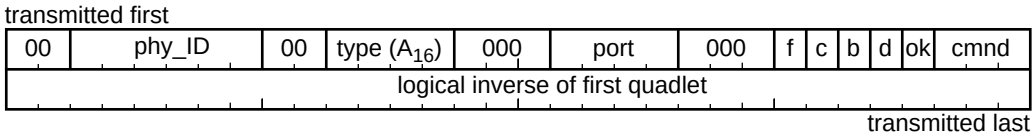


Figure 4-20E — Remote confirmation packet format

Table 4-31E — Remote confirmation packet fields

Field	Comment
phy_ID	Physical node identifier of the source of this packet.
type	Extended PHY packet type (A_{16} indicates remote confirmation packet).
port	This field shall specify the PHY port to which this packet pertains.
fault	Abbreviated as f in figure 4-20E, this bit is the current value of the *Fault* bit from PHY register 1001_2 for the addressed port.
connected	Abbreviated as c in figure 4-20E, this bit is the current value of the *Connected* bit from PHY register 1000_2 for the addressed port.
bias	Abbreviated as b in figure 4-20E, this bit is the current value of the *Bias* bit from PHY register 1000_2 for the addressed port.
disabled	Abbreviated as d in figure 4-20E, this bit is the current value of the *Disabled* bit from PHY register 1000_2 for the addressed port.
ok	One if the command was accepted by the PHY and zero otherwise.
cmnd	The cmnd value (from the preceding remote command packet) with which this confirmation packet is associated.

A PHY shall transmit a remote confirmation packet within RESPONSE_TIME after the receipt of a remote command packet. If the port is not implemented, the fault, connected, bias, disabled, and ok bits shall be zero.

4.3.4.4.6 Resume packet

The reception of the cable PHY packet shown in figure 4-20F shall cause any node to commence resume operations for all PHY ports that are both connected and suspended. This is equivalent to setting the `resume` variable to TRUE for each of these ports. The resume packet is broadcast; there is no reply.

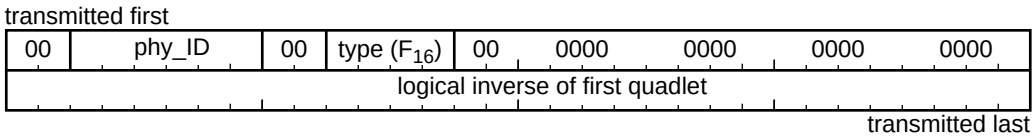


Figure 4-20F — Resume packet format

Table 4-31F — Resume packet fields

Field	Comment
phy_ID	Physical node identifier of the source of this packet.
type	Extended PHY packet type (F_{16} indicates resume packet).

4.3.5 Cable PHY timing constants

Background

IEEE Std 1394a-2000 defines new values and changes some existing constants from which the configuration and timing of the physical layer in the cable environment may be derived

Replace 4.3.5 with the following:

4.3.5 Cable interface timing constants

There are significant timing differences between originating and repeating ports. Repeating ports shall be designed to account for clock frequency and phase differences and still guarantee relevant times from table 4-32.

Note that the constant RESET_WAIT is redefined by IEEE Std 1394a-2000 as a delta to be applied to the reset time in order to derive a reset time. The reset wait time specified by IEEE Std 1394-1995 is expressed by IEEE Std 1394a-2000 as RESET_TIME + RESET_WAIT; the corresponding arbitrated (short) reset wait time is SHORT_RESET_TIME + RESET_WAIT.

Figure 4-20G through figure 4-20J are normative; if a conflict exists between them and C code elsewhere in this standard that describes the operation of the PHY state machines, the figures shall take precedence.

Table 4-32 — Cable interface timing constants

Timing constant	Minimum	Maximum	Comment
ACK_RESPONSE_TIME	—	—	This timing constant is no longer defined; see RESPONSE_TIME.
ARB_RESPONSE_DELAY	See comment		Delay through a PHY from the start of reception of an arbitration line state to the start of transmission of the associated arbitration line state at all transmitting ports. Arbitration line states shall be repeated by the PHY at least as fast as clocked data but not more than 80 ns faster.
ARB_SPEED_SIGNAL_START	–0.02 µs	—	Time delay between a transmitting port signaling TX_DATA_PREFIX and the same port transmitting a speed-signal for either an unconcatenated packet or the first packet in a concatenated sequence.
BASE_RATE	98.294 Mbit/s	98.314 Mbit/s	Base bit rate (98.304 Mbit/s ± 100 ppm).
BIAS_FILTER_TIME	41.6 µs	52.1 µs	Time to filter Bias_Detect upon the detection of TpBias before updating the PHY register *Bias* bit (~4096/BASE_RATE).

Table 4-32 — Cable interface timing constants (continued)

Timing constant	Minimum	Maximum	Comment
BIAS_HANDSHAKE	5.3 ms	16.0 ms	When used to detect a fault during a suspend or resume handshake, this is either the time a suspend initiator waits for the suspend target to remove bias (measured from the transmission of TX_SUSPEND) before a suspend fault exists or the time a resume initiator waits for the resume target to assert bias (measured from the generation of TpBias by the resume initiator) before a resume fault exists. Also the time a suspend target waits, measured from the time it drives TpBias low, before it suspends.
CONCATENATION_PREFIX_TIME	0.16 µs	—	For concatenated packets, the time a transmitting port shall signal TX_DATA_PREFIX between the end of clocked data for one packet and the start of speed-signaling time for the next.
CONFIG_TIMEOUT	166.6 µs	166.9 µs	Loop detect time (~16384/BASE_RATE).
CONNECT_TIMEOUT	330 ms	350 ms	Connection debounce time.
DATA_END_TIME	0.24 µs	0.26 µs	End of packet signal time (~24/BASE_RATE).
DATA_PREFIX_HOLD	0.04 µs	—	At a transmitting port, the time between the end of speed-signaling (when present) and the start of clocked data.
DATA_PREFIX_TIME	—	—	This timing constant is no longer defined; see CONCATENATION_PREFIX_TIME, DATA_PREFIX_HOLD and MIN_DATA_PREFIX.
FORCE_ROOT_TIMEOUT	83.3 µs	CONFIG_TIMEOUT	Time to wait in state T0: Tree-ID Start (see 4.4.3.2) before acknowledging RX_PARENT_NOTIFY (between ~8192/BASE_RATE and ~16384/BASE_RATE). NOTE—Designers are encouraged to use 162.0 µs as the maximum; this value prevents false detection of a loop in cases where more than one node has its force_root variable set to TRUE.
MAX_ARB_STATE_TIME	200 µs	400 µs	Maximum time in any state (before a bus reset shall be initiated) except A0: Idle, T0: Tree-ID Start, or a state that exits after an explicit time-out.
MAX_BUS_HOLD	—	1.63 µs	Maximum time an originating node may transmit TX_DATA_PREFIX between concatenated packets. The link shall ensure that this time is not exceeded.
MAX_BUS_OCCUPANCY	—	—	This timing constant is no longer defined; see MAX_DATA_TIME.
MAX_DATA_PREFIX_DELAY	—	—	This timing constant is no longer defined; see ARB_RESPONSE_DELAY.
MAX_DATA_TIME	—	84.34 µs	The maximum time that clocked data may be transmitted continuously. If this limit is exceeded, unpredictable behavior may result.
MIN_DATA_PREFIX	0.14 µs	—	The time a transmitting port shall signal TX_DATA_PREFIX prior to clocked data for either an unconcatenated packet or the first packet in a concatenated sequence.

Table 4-32 — Cable interface timing constants (continued)

Timing constant	Minimum	Maximum	Comment
MIN_IDLE_TIME	0.04 µs	—	Minimum idle time between packets at either an originating or repeating port (~4/BASE_RATE).
MIN_PACKET_SEPARATION	0.34 µs	—	Minimum time that an originating port shall signal TX_DATA_PREFIX between concatenated packets (~34/BASE_RATE).
NOMINAL_CYCLE_TIME	124.988 µs	125.013 µs	Average time between the start of one isochronous period and the next (125 µs ± 100 ppm).
PHY_DELAY	0.06 µs	See PHY registers	Measured from the receipt of the first data bit to its retransmission by the repeating port(s). Best-case repeater data delay has a fixed minimum.
PORT_ENABLE_TIME	—	1.0 ms	Time necessary for TpBias output to become valid when driven high. The minimum value is implementation-dependent and may incorporate other design timing requirements for port resumption.
RESET_DETECT	80.0 ms	85.3 ms	Time for an active port to confirm a reset signal.
RESET_TIME	166.6 µs	166.7 µs	Reset hold time. (~16384/BASE_RATE).
RESET_WAIT	0.16 µs		Reset wait delta time. (~16/BASE_RATE).
RESPONSE_TIME	0.04 µs	PHY_DELAY + 0.1 µs	Idle time at all ports of either a responding node or a node engaged in isochronous arbitration, measured at the cable connector, from the end of RX_DATA_END or TX_DATA_END that follows a PHY packet or primary packet to the start of the next arbitration line state, TX_DATA_PREFIX, TX_DISABLE_NOTIFY, TX_REQUEST, or TX_SUSPEND.
ROOT_CONTEND_FAST	0.76 µs	0.85 µs	Time to wait in state T3: Root Contention if the random bit is zero, as described in 4.4.3.2 (~80/BASE_RATE).
ROOT_CONTEND_SLOW	1.59 µs	1.67 µs	Time to wait in state T3: Root Contention if the random bit is one, as described in 4.4.3.2. (~160/BASE_RATE).
SHORT_RESET_TIME	1.26 µs	1.40 µs	Short reset hold time. (~128/BASE_RATE).
SID_SPEED_SIGNAL_START	–0.02 µs	0.02 µs	Time between a child port signaling TX_IDENT_DONE and the same port sending its speed capability signal.
SPEED_SIGNAL_LENGTH	0.10 µs	0.12 µs	Time that speed-signal output is driven (~10/BASE_RATE).

Some of the cable interface timing constants are more easily understood with the aid of diagrams. Figure 4-20G illustrates the relationship between data prefix and the concurrent speed-signal at the start of packet transmission. The packet may be either an unconcatenated packet or the first packet of a concatenated sequence.

Whether or not speed is explicitly signaled, TX_DATA_PREFIX shall be signaled by any transmitting port for at least the minimum time shown before clocked data is transmitted. For improved interoperability with legacy devices, TX_DATA_PREFIX should be signaled by the originating port(s) for a minimum of 180 ns prior to clocked data. Speed-signaling occurs concurrently with data prefix but may commence up to 20 ns before the start of data prefix. This is difficult to show graphically and is represented by a negative value for ARB_SPEED_SIGNAL_START. Once speed-signaling is initiated by a transmitting port, it shall be continued for SPEED_SIGNAL_LENGTH time. Speed-signaling shall complete no less than DATA_PREFIX_HOLD time before the start of clocked data.

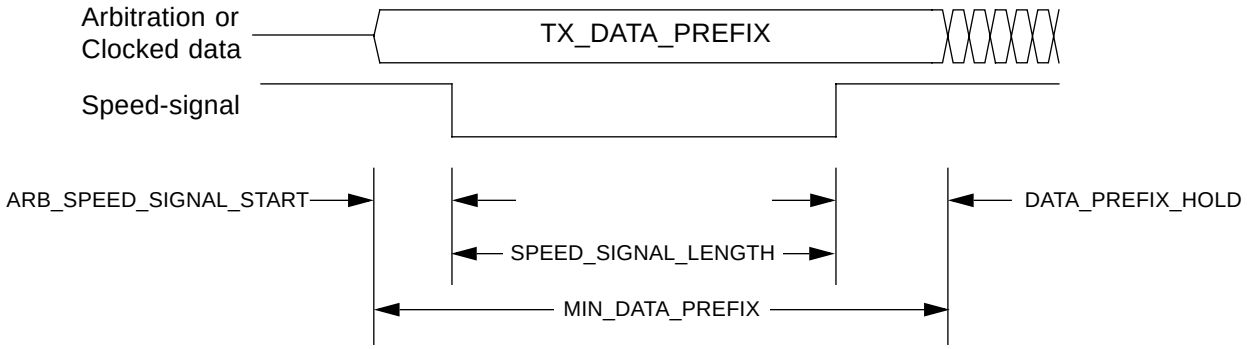


Figure 4-20G — Start of packet transmission

Data prefix and speed-signaling between two concatenated packets is similar to that for the first packet, but the speed-signal is shifted later into the data prefix time as illustrated by figure 4-20H.

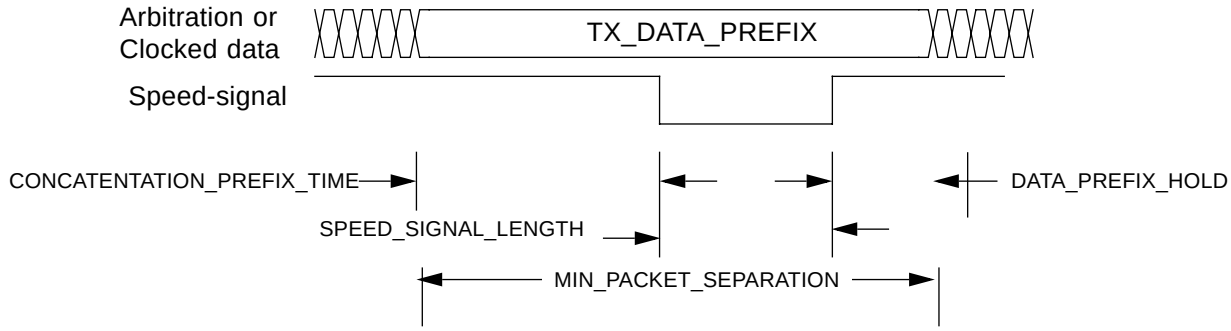


Figure 4-20H — Concatenated packet transmission

Originating ports shall guarantee MIN_PACKET_SEPARATION time between the clocked data of any two concatenated packets. Clock frequency and phase differences may cause a smaller separation at repeating ports, but in no case shall the minimum packet separation be less than the sum of CONCATENATION_PREFIX_TIME, SPEED_SIGNAL_LENGTH, and DATA_PREFIX_HOLD. After the end of clocked data for the previous packet, a transmitting port shall guarantee a delay of CONCATENATION_PREFIX_TIME before signaling the speed of the next packet. The requirements for SPEED_SIGNAL_LENGTH and DATA_PREFIX_HOLD are the same as for start of packet transmission.

When a packet ends and Serial Bus returns to the *Idle* state prior to the next subaction, the relevant timings are illustrated by figure 4-20I.

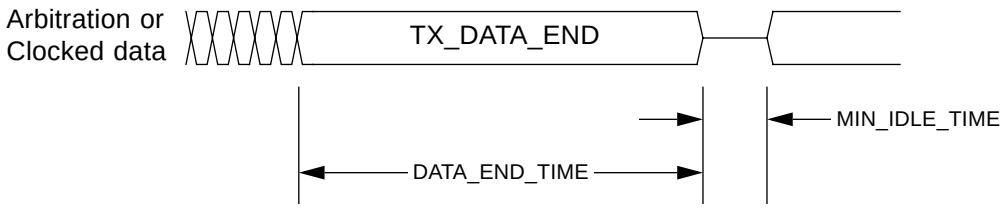


Figure 4-20I — End of packet transmission

A transmitting port shall transmit the data end indication at least DATA_END_TIME after either an unconcatenated packed or the last packet of a concatenated sequence. TX_DATA_END shall be followed by an idle gap of at least MIN_IDLE_TIME.

When a subaction requires a response, the responding node shall guarantee the timings shown by figure 4-20J. The upper bound on the idle gap prevents other nodes from erroneously observing a subaction gap.

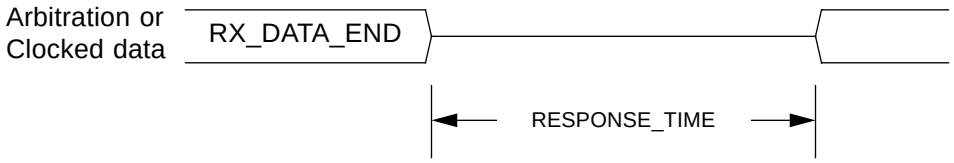


Figure 4-20J — Subaction response

The packets that require a response are any primary packet (except broadcast and stream packets), a “ping” packet, or a PHY remote access or remote command packet. The timing requirements are identical for all of these packets; only the data end arbitration line state that follows the packet is shown at the left of figure 4-20J. The arbitration line state that follows idle (shown at the right of the figure) shall be TX_DATA_PREFIX, TX_DISABLE_NOTIFY, or TX_SUSPEND. The responding node shall guarantee that the idle gap does not exceed RESPONSE_TIME at any transmitting port, not solely the port that received the packet that required the response.

NOTE—For some subactions the link is responsible for guaranteeing RESPONSE_TIME while the PHY is responsible for others.

Replace 4.3.6 with the following:

4.3.6 Gap timing

An interpacket gap is the period of time during which the received arbitration line state of the PHY is IDLE, as specified by 4.3.3. There are four types of gaps

— *Acknowledge gap.* Occurs between an asynchronous primary packet and its corresponding acknowledge packet.
— *Isochronous gap.* Precedes an unconcatenated isochronous subaction.
— *Subaction gap.* Precedes an unconcatenated asynchronous subaction within the current fairness interval.
— *Arbitration reset gap.* Precedes an unconcatenated asynchronous subaction and indicates the start of a new fairness interval.

Acknowledge and isochronous gaps are nominally 4/BASE_RATE at the PHY that originated the acknowledgment or arbitrated for the isochronous subaction, respectively. They occur as a consequence of the design of the arbitration state machines specified in 4.4.3.

A node may detect an arbitration reset or subaction gap as soon as the externally observable idle time at any port meets the minimum duration specified by table 4-33. A node shall detect an arbitration reset or subaction gap no later than when the externally observable idle time at any port meets the maximum duration specified.

Table 4-33 — Gap detection times

Gap type	Minimum	Maximum
Subaction	$\frac{(27 + \text{gap\_count} \times 16)}{\text{BASE\_RATE}_{max}} - \text{PHY\_DELAY}_{max}$	$\frac{(29 + \text{gap\_count} \times 16)}{\text{BASE\_RATE}_{min}} + \text{PHY\_DELAY}_{max}$
Arbitration reset	$\frac{(51 + \text{gap\_count} \times 32)}{\text{BASE\_RATE}_{max}} - \text{PHY\_DELAY}_{max}$	$\frac{(53 + \text{gap\_count} \times 32)}{\text{BASE\_RATE}_{min}} + \text{PHY\_DELAY}_{max}$

A node that detects an arbitration reset or subaction gap and elects to arbitrate shall wait an additional time, `arb_delay` (defined as gap_count × 4/BASE_RATE), before commencing arbitration. This delay guarantees that if some node detects an arbitration reset or subaction gap then all nodes, no matter what their relative location in the bus topology, correctly detect the same gap. As a consequence, externally observable gap times at a node that originates arbitration conform to the times specified by table 4-34.

Table 4-34 — Gap times at originating node

Gap type	Minimum	Maximum
Subaction	$\frac{(27 + \text{gap\_count} \times 20)}{\text{BASE\_RATE}_{max}}$	$\frac{(29 + \text{gap\_count} \times 20)}{\text{BASE\_RATE}_{min}} + \text{RESPONSE\_TIME}_{max} - \text{MIN\_IDLE\_TIME}$
Arbitration reset	$\frac{(51 + \text{gap\_count} \times 36)}{\text{BASE\_RATE}_{max}}$	—

Replace 4.3.8 with the following:

4.3.8 Node variables

Each node's PHY has a set of variables (see Table 4-35) that are referenced in the C code and state machines in 4.4. The values of these variables may be affected by writes to PHY registers, the transmission or reception of PHY configuration packets, or by arbitration state actions—including bus reset. A reset of the PHY/link interface affects none of these variables.

Table 4-35 — Node variables

Variable name	Power reset value	Comment
`accelerating`	TRUE	Set TRUE or FALSE by accelerate or decelerate requests issued by the link via LReq (see 5A.3) and used by the arbitration state machines. See also `enab_accel` below.
`arb_enable`	—	TRUE if the PHY may arbitrate on behalf of a fair request within the current fairness interval.
`boundary_node`	—	TRUE if the PHY has both an active and a suspended port.
`cable_power_active`	—	TRUE if cable power is within normal operating range (see 4.2.2.7).
`enab_accel`	FALSE	Globally enables or disables all PHY accelerations specified by 4.4. This variable is visible as the PHY register bit *Enab_accel*.
`force_root`	FALSE	When TRUE, this modifies the PHY's tree identification behavior and increases the likelihood that the node becomes root (see 4.4.2.2). If only one node on a bus has `force_root` set to TRUE, that node is guaranteed to become the root.

Table 4-35 — Node variables (continued)

Variable name	Power reset value	Comment
gap_count	$3F_{16}$	This value determines the length of arbitration reset and subaction gaps and may be used to optimize bus performance. All nodes on the bus should have the same gap_count value, else unpredictable arbitration behavior may occur.
gap_count_reset_disable	FALSE	Permits gap_count to retain its value through one bus reset.
initiated_reset	TRUE	TRUE if this node initiated the bus reset in progress. The variable remains TRUE through and after the bus reset and is cleared to FALSE only by a subsequent bus reset not initiated by this node.
lctrl	TRUE	TRUE if the node's link intends to advertise, by means of the L bit in the self-ID packets, that it is present and enabled.
more_packets	—	Flag that indicates whether or not additional self-ID packets are to be sent.
parent_port	—	The port number that is connected to the parent node; this variable is meaningless if the node is root.
physical_ID	—	The node's 6-bit physical ID established by the self-identify process.
receive_port	—	The port number that is receiving encoded data (determined by the arbitration states).
root	—	TRUE if the node is the root, as determined by tree-ID.

Replace 4.3.9 with the following:

4.3.9 Port variables

In addition to the variables described in 4.3.8, each node's PHY has a set of variables replicated for each port (see table 4-36). A reset of the PHY/link interface affects none of these variables.

Table 4-36 — Port variables

Variable name	Power reset value	Comment
active	—	TRUE if this port is neither disabled, disconnected, nor suspended.
bias	—	TRUE if TpBias is present.
child	—	TRUE if this port is connected to a child node.
child_ID_complete	—	TRUE when the child node connected to this port has finished its self-ID.
connected	FALSE	TRUE if there is a peer PHY connected to this port.
max_peer_speed	—	Maximum speed capability of the peer PHY connected to this port.
speed_OK	—	The connected port can accept a packet at the requested speed.

Replace 4.4 with the following:

4.4 Cable physical layer operation

The operation of the cable physical layer can best be understood with reference to the architectural diagram shown in figure 4-21.

The main controller of the cable physical layer is the block labeled "arbitration control," which responds to arbitration requests from the link layer (PH_ARB.*request*) and changes in the state of its ports. It provides the management and timing signals for transmitting, receiving, and repeating packets. It also provides the bus reset and configuration functions. The operation of this block is described in 4.4.3.

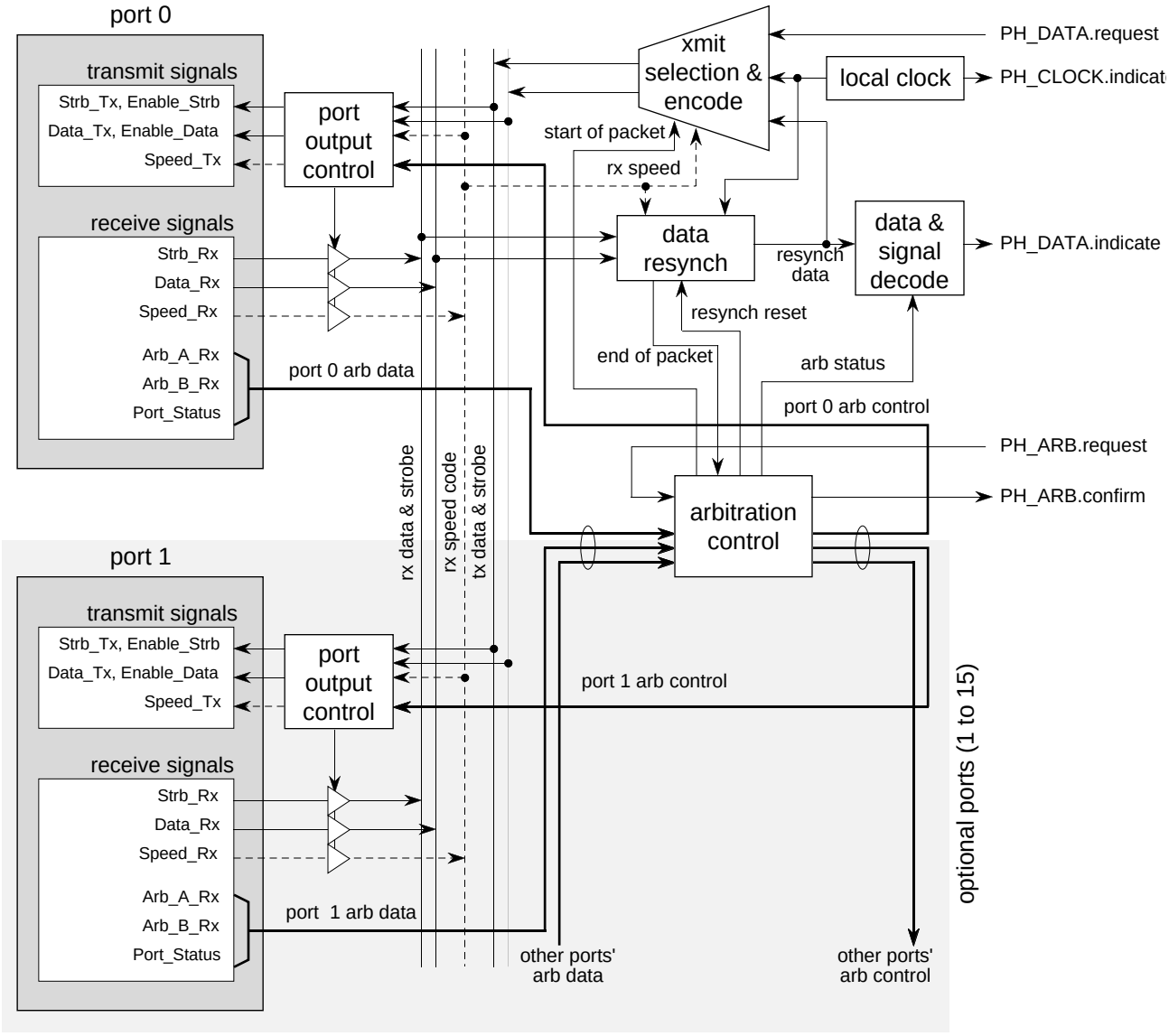


Figure 4-21 — Cable physical layer architecture

The “data resynch” block decodes the data-strobe signal and retimes the received data to a local fixed frequency clock provided by the “local clock” block. Since the clocks of receiving and transmitting nodes can be up to 100 ppm different from the nominal, the data resynch function must be able to compensate for a difference of 200 ppm over the maximum packet length of 84.31 µs (1024 byte isochronous packet at S100). The operation of this block is described in 4.4.2.2.

The “data & signal decode” block provides a common interface to the link layer for both packet data and arbitration signals (gaps and bus reset indicators).

The “xmit selection & encode” block is the selector between repeated data and data sent by the link layer. It also generates the strobe signal for the transmitted data. Its operation is described in 4.4.2.1.

Each port has an associated “port output control” that selects either the arbitration control signals or the data-strobe pair for transmission.

All of the procedures in this subclause use the syntax specified in 1.6.7 and the definitions in table 4-32 and table 4-35 through table 4-38, inclusively.

NOTE—Although not part of the C language, the type `dataBit` is used to represent a bit of information, 0 or 1.

The C language code and state machines are not normative descriptions of implementations; they are normative descriptions of externally apparent PHY behaviors. Different implementations are possible. In particular, the C language code and state machines do not contain provisions to enforce the LReq rules specified by 5A.3.1; if the link issues bus requests that do not follow the rules the PHY behavior is unspecified.

Subclauses 4.4.1 through 4.4.4 normatively describe the operation of the cable physical layer by means of state machine diagrams, C code, and expository text in the body of the standard (which includes the notes that accompany the state machine diagrams but excludes any text within figures or tables). In case of conflict, precedence shall be given first to the state machine diagrams, second to the C code, and last to the expository text.

Table 4-37 — Cable PHY code definitions (Sheet 1 of 2)

```
const int FIFO_DEPTH = ?;           // IMPLEMENTATION-DEPENDENT! At least 6 for S100,
                                    // 12 for S200 and 24 for S400 PHYs
enum breq {NO_REQ, IMMED_REQ,      // Types of bus requests made by link across PHY/link interface
           ISOCH_REQ,
           PRIORITY_REQ, FAIR_REQ};
enum PHY_state {R0, R1,             // Tracks the PHY state (names per state diagrams)
                T0, T1, T2, T3,
                S0, S1, S2, S3, S4,
                A0, A1, A2, RX, TX};
enum speedCode {S100, S200, S400};  // Speed codes
enum tpSig {L, H, Z};               // Differential signal on twisted pair
struct portData {tpSig TpA; tpSig TpB;};    // Port data structure
enum phyData(portData signals);     // Encoded types DATA_ZERO, DATA_ONE, DATA_PREFIX or DATA_END

boolean ack;                        // Set if last packet observed was exactly 8 bits
boolean all_child_ports_identified; // Set if all child ports have been identified
int arb_delay;                      // Arbitration delay time (4 * gap_count / BASE_RATE)
int arb_reset_gap;                  // Duration of an arbitration reset gap
                                    // ((52 + 32 * gap_count) / BASE_RATE)
timer arb_timer;                    // Timer for arbitration state machines
boolean bias_detect[NPORT];         // Output of bias detection comparator
boolean bus_initialize_active;      // Set while the PHY is initializing the bus
int children;                       // Number of child ports
boolean concatenated_packet;        // TRUE if a concatenated packet is being received
boolean connect_detect[NPORT];      // Output of connect detection comparator
boolean connect_detect_valid[NPORT];// Presence of bias renders connection detect circuitry
useless
int contend_time;                   // Amount of time to wait during root contention
timer connect_timer;                // Timer for connection status monitor
boolean connection_in_progress[NPORT];    // TRUE when debouncing a new connection
boolean disable_notify[NPORT];      // Set when PHY port is requested to disable
dataBit fifo[FIFO_DEPTH];           // Data resynch buffer
unsigned fifo_rd_ptr, fifo_wr_ptr;  // Data resynch buffer pointers
boolean fly_by_OK;                  // TRUE when fly-by concatenation permitted
boolean ibr;                        // Set when a long bus reset is needed
boolean isbr;                      // Set when an arbitrated (short) bus reset should be attempted
boolean isbr_OK;                // Set when asynchronous or immediate arbitration conditions permit
                                    // an arbitrated (short) bus reset to be commenced
```

Table 4-37 — Cable PHY code definitions (Sheet 2 of 2)

```
boolean isolated_node;              // Set if no ports active
boolean link_concatenation          // TRUE if the link is transmitting a concatenated packet
int lowest_unidentified_child;      // Lowest numbered active child that has not sent its self-ID
boolean OK_to_detect_reset;        // TRUE if a resuming port is permitted to monitor for BUS RESET
boolean own_request;                // Latch the value of arb_OK() at the time it is evaluated
boolean phy_response;            // TRUE to indicate that a PHY response packet is to be transmitted
boolean ping_response;              // Set if self-ID packet(s) needed in response to a ping
portData portR(int port_number);    // Return current rxData signal from indicated port
speedCode portRspeed[NPORT];        // Filtered and latched receive speed for indicated port
void portT(int port_number, portData txData);   // Transmit txData on indicated port
void portTspeed(int port_number, speedCode speed); // Set transmit speed on indicated port
boolean random_bool();              // Returns a random TRUE or FALSE value
int read_phy_reg(int page, int port, int reg);  // Return current value of specified PHY register
int requesting_child;               // Lowest numbered requesting child
int reset_time;                     // Duration to assert bus reset signal
boolean resume[NPORT];              // Set when PHY port is requested to resume
boolean resume_fault[NPORT];        // Set when peer port fails to complete resume
boolean resumption_done;          // Resumption by at least one resuming port forces all to be done
int rx_dribble_bits;                // Keep track of dribble bits in FIFO
boolean rx_S200, rx_S400;           // Outputs from speed-signal comparators
speedCode rx_speed, tx_speed;       // Current packet speeds
speedCode speed;                    // Speed-signaled by the link across the PHY/link interface
void signal(int event);             // Sets event status to the value specified
boolean signaled;                   // Indicate transmission of TX_DISABLE_NOTIFY or TX_SUSPEND
int subaction_gap;                // Duration of a subaction gap ((28 + 16 * gap_count) / BASE_RATE)
boolean suspend[NPORT];             // Set when PHY port is requested to suspend
boolean suspend_fault[NPORT];       // Set when peer port fails to complete suspend
int test_event(int event_mask);     // Test the indicated event(s), return those signaled
int wait_event(int event_mask);     // Await the indicated event(s), return events signaled
```

Table 4-38 — Cable PHY packet definitions (Sheet 1 of 2)

```
typedef union {
   struct {
      union {
         quadlet dataQuadlet;
         dataBit dataBits[32];
         struct {           // First self-ID packet
            quadlet type:2;
            quadlet phy_ID:6;       // Physical_ID
            quadlet :1;             // Always 0 for first self-ID packet
            quadlet L:1;            // Link active
            quadlet gap_cnt:6;      // Gap count
            quadlet sp:2;           // Speed code
            quadlet :2;
            quadlet c:1;            // Isochronous resource manager contender
            quadlet pwr:3;          // Power class
            quadlet p0:2;           // Port 0 connection status
            quadlet p1:2;           // Port 1 connection status
            quadlet p2:2;           // Port 2 connection status
            quadlet i:1;            // Initiated reset
            quadlet m:1;            // More self-ID packets...
         };
         struct {           // Subsequent self-ID packets
            quadlet :8;
            quadlet ext:1;          // Nonzero for second and subsequent self-ID packets
            quadlet n:3;            // Sequence number
            quadlet :2;
            quadlet pa:2;           // Port connection status...
            quadlet pb:2;           //                    pa  pb  pc  pd  pe  pf  pg  ph
            quadlet pc:2;           // Self-ID packet 2  P3  P4  P5  P6  P7  P8  P9 P10
            quadlet pd:2;           // Self-ID packet 3 P11 P12 P13 P14 P15 --- --- ---
            quadlet pe:2;
```

Table 4-38 — Cable PHY packet definitions (Sheet 2 of 2)

```
            quadlet pf:2;
            quadlet pg:2;
            quadlet ph:2;
            quadlet :2;
         };
         struct {           // PHY configuration packet
            quadlet :2;
            quadlet root_ID:6;      // Intended root
            quadlet R:1;            // If set, root_ID field is valid
            quadlet T:1;            // If set, gap_cnt field is valid
            quadlet gap_cnt:6;      // Gap count
            quadlet :16;
         };
         struct {           // Extended PHY packets (ping and other remote packets)
            quadlet :2;
            quadlet :6;             // Physical_ID
            quadlet :2;             // Always zero (identifies extended PHY packet)
            quadlet ext_type:4;     // Extended type
            quadlet page:3;         // Page_select
            quadlet port:4;         // Port_select
            quadlet reg:3;          // Register address (add 0b1000 if paged register)
            union {
               quadlet data:8;      // Register data (remote reply)
               struct {             // Remote confirmation
                  quadlet fault:1;  // Copies of equivalent PHY register bits...
                  quadlet connected:1;
                  quadlet bias:1;
                  quadlet disabled:1;
                  quadlet ok:1;     // Confirm command accepted or not
                  quadlet cmnd:3;   // Remote command
               };
            };
         };
      };
      union {
         quadlet checkQuadlet;
         dataBit checkBits[32];
      };
   };
} PHY_PKT;
```

4.4.1 Speed-signal sampling and filtering

Speed-signaling in the cable environment occurs during the self-ID phase of bus initialization and during packet transmission in the normal arbitration phase. In the self-ID phase, connected peer PHYs exchange speed-signals to configure their maximum speed capabilities. In the normal arbitration phases, the speed of an acknowledge, PHY, or primary packet is signaled concurrently with the data prefix that precedes the packet. In either case, speed is signaled as common-mode current on TPB and is observed as a common-mode voltage drop (relative to TpBias) on TPA.

A speed-signal is a single-ended, common-mode signal of small amplitude that is detected on TPA by differential comparator(s). An S100 PHY requires no comparators, an S200 PHY requires one comparator, while an S400 PHY requires different comparators for the S200 and S400 signals. Each comparator has one side tied to an internally generated reference voltage and the other to the common-mode voltage of the twisted pair (referenced at the midpoint of a resistor/divider network between the twisted pair inputs).

Reliable reception and detection of a speed-signal may be hampered by several factors

— *Common-mode noise.* Because the speed-signal is a common-mode signal, it is more susceptible to noise than the differential arbitration and data signals. Spurious speed-signals may be observed because of cross-talk, mismatches in differential signal transition times, or common-mode ground noise.
— *Receive comparator delay mismatch.* The two sets of speed-signal comparators may have different delays, which may vary with the input signal amplitude. For example, when receiving the leading edge of an S400 speed-signal, the S200 comparator typically switches before the S400 comparator.
— *RC effects.* The speed-signal rise and fall times may be degraded because of the RC filtering effects of the cable and bias network.

For all of these reasons it is desirable to filter the outputs of the speed-signal comparators to enhance the reliable detection of a speed-signal. A speed-signal should be continuously present for at least 20 ns before it is considered valid.

One method is to monitor the comparator outputs and consider the speed-signal valid only if the outputs remain stable for some number of consecutive samples. This is illustrated by the informative C code in table 4-39. The sampling frequency is governed by the 50 MHz PHY clock and the code latches the fastest speed-signal observed.

Table 4-39 — Digital speed filtering (informative)

```
void speed_filter(void) {           // Continuously sample speed-signals
   int i;
   boolean OK_to_sample;
   speedCode raw_speed[NPORT][2];   // Unfiltered speed (moving window of two samples)

   wait_event(PH_CLOCK.indication);       // Wait for 50 MHz clock
   for (i = 0; i < NPORT; i++) {
      raw_speed[i][1] = raw_speed[i][0];  // Save prior sample
      if (rx_S400[i]) {                   // S400 observed?
         if (rx_S200[i])
            raw_speed[i][0] = S400;
      } else if (rx_S200[i])              // S200 observed?
         raw_speed[i][0] = S200;
      else
         raw_speed[i][0] = S100;          // No to both: default S100
      OK_to_sample =    (PHY_state == S4)
                     || (   (PHY_state == S2 || PHY_state == S3)
                         && portR(i) == RX_IDENT_DONE)
                  || (   (PHY_state == A0 || PHY_state == A1 || PHY_state == A2 || PHY_state == RX)
                         && portR(i) == RX_DATA_PREFIX);
      if (!OK_to_sample)
         portRspeed[i] = S100;      // Reset to S100 whenever it's not OK to sample
      else if (raw_speed[i][0] == raw_speed[i][1]) // Consecutive identical samples?
         if (raw_speed[i][0] == S200 && portRspeed[i] < S400)
            portRspeed[i] = S200;   // Latch S200 only if S400 not yet confirmed
         else if (raw_speed[i] == S400)
            portRspeed[i][0] = S400;
   }
}
```

The C code in table 4-39 is not intended to preclude other implementations. For example, some implementation may require that the outputs remain stable for three consecutive samples. Others might implement different sampling algorithms for S200 and S400. The behavior of any implementation shall conform to the signal and timing requirements of this standard.

4.4.2 Data transmission and reception

Data transmission and reception are synchronized to a local clock that shall be accurate within 100 ppm. The nominal data rates are powers of two multiples of 98.304 Mbit/s for the cable environment.

4.4.2.1 Data transmission

Data transmission (see table 4-40) entails sending the data bits to the connected PHY along with the appropriately encoded strobe signal using the timing provided by the PHY transmit clock. If the connected port cannot accept data at the requested speed (indicated by the `speed_OK[i]` flag being FALSE), then a null packet is transmitted, i.e., the drivers remain in the "01" data prefix condition.

Table 4-40 — Data transmit actions

```
static dataBit tx_data, tx_strobe;     // Memory of last signal sent (reinitialized by DATA_PREFIX
                                        // at start of each packet to 1 and 0, respectively)

void tx_bit(dataBit bit) {              // Transmit a bit
   int i;

   wait_event(PH_CLOCK.indication);     // Wait for clock
   if (bit == tx_data)                  // If no change in data
      tx_strobe = ~tx_strobe;           // Invert strobe
   tx_data = bit;
   for (i = 0; i < NPORT; i++)
      if (active[i] && i != receivePort)
         if (speed_OK[i]) {
            portData pd = {tpSig(tx_strobe), tpSig(tx_data)};
            portT(i, pd);
         } else
            portT(i, TX_DATA_PREFIX);
}
```

The edge rates and jitter specifications for the transmitted signal are given in 4.2.3.

Starting data transmission requires sending a special data prefix signal and a speed code (see table 4-41). The `speed_OK[i]` flag for each port is TRUE if the connected PHY has the capabilities to receive the data.

Table 4-41 — Start data transmit actions

```
void start_tx_packet(speedCode speed) {   // Send data prefix and speed code
   int signal_port = NPORT, i;

   for (i = 0; i < NPORT; i++) {
      if (!active[i])
         speed_OK[i] = FALSE;
      else if (phy_response) {
         portT(i, TX_DATA_PREFIX);        // Data prefix before PHY packet
         speed_OK[i] = (speed <= max_peer_speed[i]);
         if (speed_OK[i])
            portTspeed(i, speed);         // Almost always S100
      } else if (disable_notify[i])
         portT(signal_port = i, TX_DISABLE_NOTIFY);
      else if (suspend[i])
         portT(signal_port = i, TX_SUSPEND);
      else {
         portT(i, TX_DATA_PREFIX);        // Send data prefix
         speed_OK[i] = (speed <= max_peer_speed[i]);
         if (speed_OK[i])
```

Table 4-41 — Start data transmit actions

```
            portTspeed(i, speed);         // Receiver can accept, send speed intentions
      }
   }
   wait_time(SPEED_SIGNAL_LENGTH);
   for (i = 0; i < NPORT; i++)
      if (active[i])
         portTspeed(i, S100);             // Go back to normal signal levels
   wait_time(DATA_PREFIX_HOLD);           // Finish data prefix
   signaled = (signal_port != NPORT);
}
```

Except in the case of a null packet, ending a data transmission (see table 4-42) requires sending extra bits (known as “dribble bits”) that flush the last data bit through the receiving circuit. The number of *dribble bits* required varies with the transmission speed—one, three, or seven extra bits for S100, S200, and S400, respectively. An extra bit is required to put the two signals TPA and TPB into the correct state; the value of the bit depends upon whether the bus is being held (PH_DATA.*request* of DATA_PREFIX) or not (PH_DATA.*request* of DATA_END). The `stop_tx_packet` procedure is not invoked for null packets.

Table 4-42 — Stop data transmit actions

```
void stop_tx_packet(phyData ending_status, speedCode speed, int hold_time) {
   int i;
   portData tx_data = (ending_status == DATA_PREFIX) ? TX_DATA_PREFIX : TX_DATA_END;

   switch (speed) {     // Bit width of PHY/link interface may require pad bits
      case S400:        // Pad with six extra (dribble) bits, 8 total
         tx_bit(1);
         tx_bit(1);
         tx_bit(1);
         tx_bit(1);
      case S200:        // Pad with two extra (dribble) bits, 4 total
         tx_bit(1);
         tx_bit(1);
      default:
         break;         // No need for extra (dribble) bits
   }
   tx_bit((ending_status == DATA_PREFIX) ? 1 : 0); // Penultimate bit...
   wait_event(PH_CLOCK.indication());              // Wait for clock
   for (i = 0; i < NPORT; i++)
      if (active[i] && i != receive_port)
         portT(i, tx_data);                        // ...and the last dribble bit
   wait_time(hold_time);                           // Speed-signal after this time
}
```

NOTE—This algorithm works to force the ending port state to TX_DATA_PREFIX or TX_DATA_END and relies on two characteristics of packet transmission; there are an even number of bits between the beginning and the end of a packet and a packet starts with `tx_strobe` at 0 and `tx_data` at 1. Thus, when `stop_tx_packet` is called, the port state is either 01 or 10. If the desired port state is 01 (TX_DATA_PREFIX), this algorithm sets port state to 11 for one bit time and then to 01. If the desired ending state is 10 (TX_DATA_END), the port state sequence is 00 followed by 10.

4.4.2.2 Data reception and repeat

Data reception for the cable environment physical layer has three major functions—decoding the data-strobe signal to recover a clock, synchronizing the data to a local clock for use by the link layer, and repeating the synchronized data out all other connected ports (see table 4-43). This process can be described as two routines communicating via a small FIFO.

Table 4-43 — Data reception and repeat actions

```
static tpSig old_data, old_strobe;          // Memory of last signal seen (reinitialized by DATA_PREFIX
                                             // at start of each packet to 1 and 0, respectively)

// Decode data-strobe stream and load FIFO -- this routine is always running
// (speed code recording is also done here)

void decode_bit(void) {
   portData new_signal;
   tpSig new_data, new_strobe;

   while (TRUE) {
      if (portRspeed(receive_port) > S100) {
         rx_speed = portRspeed(receive_port);// Default to previous speed if no speed-signal present
         signal(SPEED_SIGNAL_RECEIVED);      // Notify start_rx_packet
      }
      new_signal = portR(receive_port);      // Get signal
      if (new_signal == IDLE)
         signal(IDLE_DETECTED);
      else if (new_signal == RX_DATA_END)
         signal(DATA_END_DETECTED);
      else if (new_signal == BUS_RESET)
         signal(RESET_DETECTED);
      else {
         new_data = new_signal.TPA;          // Received data is on TPA
         new_strobe = new_signal.TPB;        // Received strobe is on TPB
         if ((new_strobe != old_strobe) || (new_data != old_data)) {
                                             // Either data or strobe changed
            fifo[fifo_wr_ptr] = new_data;    // Put data in FIFO
            fifo_wr_ptr = ++fifo_wr_ptr % FIFO_DEPTH; // Advance or wrap FIFO pointer
            signal(DATA_STARTED);            // Signal rx_bit to start
         }
         old_strobe = new_strobe;
         old_data = new_data;
      }
   }
}

// Unload FIFO and repeat data (but suppress dribble bits!)

void rx_bit(dataBit *rx_data, boolean *end_of_data) {
   wait_event(PH_CLOCK.indication);          // Wait for clock
   if (((fifo_wr_ptr + FIFO_DEPTH - fifo_rd_ptr) % FIFO_DEPTH) <= rx_dribble_bits)  // FIFO empty?
      *end_of_data = TRUE;                   // If so, set flag
   else {
      *end_of_data = FALSE;                  // If not, clear flag...
      *rx_data = fifo[fifo_rd_ptr];          // ... and get data bit
      fifo_rd_ptr = ++fifo_rd_ptr % FIFO_DEPTH; // Advance or wrap FIFO pointer
      tx_bit(*rx_data);                      // Repeat the data bit
   }
}
```

Starting data reception requires initializing the data resynchronizer and sampling the speed-signal from the sender of the data. In the absence of a speed-signal, the PHY interprets the speed as either S100 or else the speed of the immediately preceding concatenated packet. At the same time, the node starts the transmitting ports by sending a special data prefix signal and repeating the received speed code. As in the `start_tx_packet( )` function, the node must do the speed-signaling exchange for each transmitting port.

When a null packet is transmitted as a consequence of a cancelled arbitration request, possible interpretations of the normal arbitration state machine and C code permit the null packet to include a speed-signal. Subsequently, a repeating node may concatenate another packet (see table 4-47B) which contains a different speed-signal. The C code in

table 4-44 incorrectly requires PHYs to recognize the fastest speed-signal observed instead of the most recent. In order to minimize interoperability problems, PHY designers should implement two remedies

a) When a packet is observed to contain multiple speed-signals, use the most recent one, and

b) Do not concatenate onto a null packet if multiple conflicting speed-signals would result.

One possible implementation of the second recommendation is to permit concatenation onto a null packet only if its speed is S100.

Table 4-44 — Start data reception and repeat actions

```
void start_rx_packet() {            // Send data prefix and do speed-signaling
   int event, i;

   arb_timer = 0;                           // Timer in case there's no speed-signal
   fifo_rd_ptr = fifo_wr_ptr = 0;           // Reset data resynch buffer
   portT(receive_port, IDLE);               // Turn off grant, get ready to receive
   for (i = 0; i < NPORT; i++)
      if (active[i] && i != receive_port) {
         portT(i, TX_DATA_PREFIX);          // Send data prefix out repeat ports
         speed_OK[i] = (rx_speed <= max_peer_speed[i]);  // Default if no speed-signal
      }
   while (   arb_timer < SPEED_SIGNAL_LENGTH + DATA_PREFIX_HOLD // Guarantee times for other PHY's...
          && ((event = test_event(SPEED_SIGNAL_RECEIVED)) == 0)   // ...or wait for speed-signal
      ;
   if (event == 0) {                        // Speed-signal was not observed
      event = wait_event(  SPEED_SIGNAL_RECEIVED | DATA_STARTED   // Await normal start of packet...
                         | IDLE_DETECTED | DATA_END_DETECTED | RESET_DETECTED);  // ...or lack of data
      if (((event & (IDLE_DETECTED | DATA_END_DETECTED | RESET_DETECTED)) != 0)
         return;                            // There is no incoming packet
   }
   tx_speed = rx_speed;                     // Get speed of packet to repeat
   if (rx_speed == S100)
      rx_dribble_bits = 2;                  // Need for FIFO empty test
   else
      rx_dribble_bits = (rx_speed == S200) ? 4 : 8;
   if ((event & SPEED_SIGNAL_RECEIVED) != 0) {  // Repeat the speed-signal (if it was observed)
      for (i = 0; i < NPORT; i++)
         if (active[i] && i != receive_port) {
            speed_OK[i] = (tx_speed <= max_peer_speed[i]);
            if (speed_OK[i])
               portTspeed(i, tx_speed);     // Receiver can accept, send speed intentions
         }
      wait_time(SPEED_SIGNAL_LENGTH);
      for (i = 0; i < NPORT; i++)
         if (active[i] && i != receive_port)
            portTspeed(i, S100);            // Go back to normal signal levels
      wait_time(DATA_PREFIX_HOLD);          // Finish data prefix
      event = wait_event(  DATA_STARTED     // Wait for decoder to start...
                         | DATA_END_DETECTED | IDLE_DETECTED | RESET_DETECTED);  // ...or error
   }
   if (event == DATA_STARTED)               // Actually receiving a packet?
      for (i = 0; i < 2 * rx_dribble_bits - 1; i++)   // Buffer enough bits to allow for variation
         wait_event(PH_CLOCK.indication);   // in clock frequencies (same value as for dribble
                                            // bits) and for the dribble bits themselves
}
```

The value of all dribble bits, except for the last dribble bit, is unspecified. A PHY shall not depend upon the value of any dribble bit except the last. The last dribble bit shall be zero when the preceding packet is terminated by DATA_END and one when terminated by DATA_PREFIX.

NOTE—The description of the FIFO buffer in table 4-41 and table 4-42 assumes that the PHY strips dribble bits from incoming packets and regenerates them for repeated packets. Alternate implementations, e.g., one which repeats dribble bits, may be valid so long as no dribble bits are transferred to the link.

4.4.3 Arbitration

The cable environment supports the immediate, priority, isochronous, and fair arbitration classes. Immediate arbitration is used to transmit an acknowledge immediately after packet reception; the bus is expected to be available. Priority arbitration is used by the root for cycle start requests or may be used by any node to override fair arbitration. Isochronous arbitration is permitted between the time a cycle start is observed and the subaction gap that concludes an isochronous period; isochronous arbitration commences immediately after packet reception. Fair arbitration is a mechanism whereby a PHY succeeds in winning arbitration only once in the interval between arbitration reset gaps.

Some of these arbitration classes may be enhanced as defined by this standard. Ack-accelerated arbitration permits a PHY to arbitrate immediately following an observed acknowledge packet; this enhancement can reduce the arbitration delay by a subaction gap time. Fly-by arbitration permits a transmitted packet to be concatenated to the end of a packet for which no acknowledge is permitted—acknowledge packets themselves or isochronous packets. A PHY shall not use fly-by arbitration to concatenate an S100 packet after any packet of a higher speed.

Cable arbitration has two parts—a three phase initialization process (bus reset, tree identify, and self-identify) and a normal operation phase. Each of these four phases is described using a state machine, state machine notes, and a list of actions and conditions. The state machine and the list of actions and conditions are the normative part of the specification. The state machine notes are informative.

4.4.3.1 Bus reset

The bus reset process starts when a bus reset signal is recognized on an active port or generated locally (see figure 4-22). Its purpose is to guarantee that all nodes propagate the reset signal. Two types of bus reset are defined—long bus reset and arbitrated (short) bus reset. The PHY variable `reset_time` controls the length of the bus reset generated or propagated.

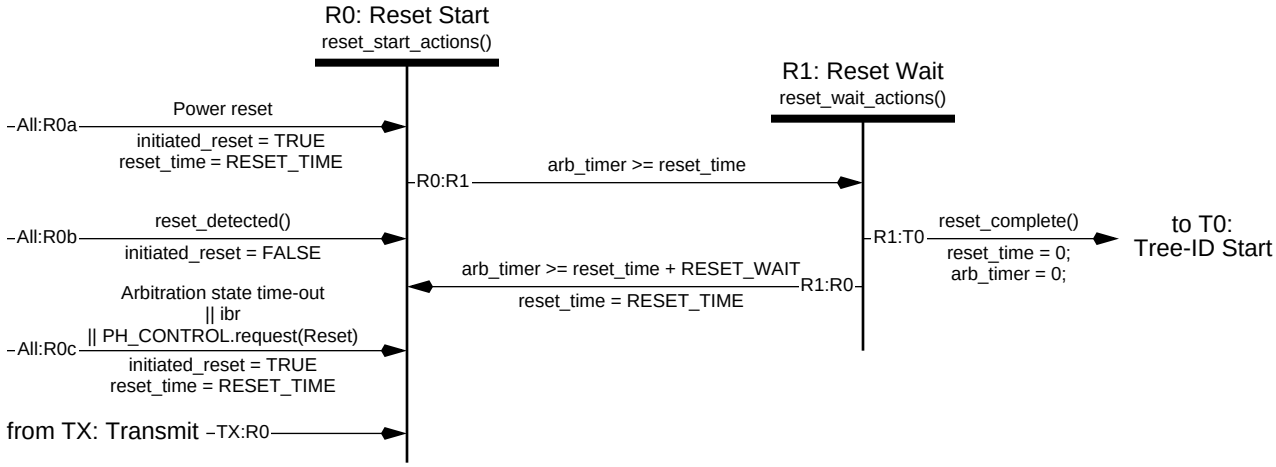


Figure 4-22 — Bus reset state machine

4.4.3.1.1 Bus reset state machine notes

Transition All:R0a. This is the entry point to the bus reset process if the PHY experiences a power reset. On power reset, PHY register values and internal variables are set as specified in this subclause; in particular all ports are marked disconnected. A solitary node transitions through the reset, tree-identify, and self-identify states and enters A0: Idle as the root node.

Transition All:R0b. This is the entry point to the bus reset process if the PHY senses BUS_RESET on any active or resuming port's arbitration signal lines (see table 4-28). This transition shall be made in preference to any other transition that might be simultaneously eligible. The initiation of a bus reset cannot occur until a state's actions have been completed.

Transition All:R0c. This is the entry point to the bus reset process if this node is initiating the process. This happens under the following conditions:

1) Serial Bus management makes a PH_CONTROL.*request* that specifies a long reset.
2) The PHY detects a disconnect on its parent port.
3) The PHY stays in any state (except A0: Idle, T0: Tree-ID Start, or a state that has an explicit time-out) for longer than MAX_ARB_STATE_TIME. This condition is not explicitly represented in either the state diagrams or the C code.

With the exception of the last condition, the initiation of a bus reset cannot occur until a state's actions have been completed.

Transition TX:R0. This is the entry point to the bus reset process if this node is initiating an arbitrated (short) reset. If arbitration succeeded and the `isbr_OK` variable is set, there is no packet to transmit and the short bus reset commences immediately.

State R0:Reset Start. The node sends a BUS_RESET signal whose length is governed by `reset_time`. In the case of a standard bus reset, this is long enough for all other bus activity to settle down (RESET_TIME is longer than the worst case packet transmission plus the worst case bus turn-around time). SHORT_RESET_TIME for an arbitrated (short) bus reset is significantly shorter since the bus is already in a known state following arbitration.

Transition R0:R1. The node has been sending a BUS_RESET signal long enough for all its connected neighbors to detect it.

State R1:Reset Wait. The node sends out IDLE, waiting for all its active ports to receive IDLE or RX_PARENT_NOTIFY (either condition indicates that the connected PHYs have left their R0 state).

Transition R1:R0. The node has been waiting for its ports to go *Idle* for too long (this can be a transient condition caused by multiple nodes being reset at the same time); return to the R0 state again. This time-out period is a bit longer than the R0:R1 time-out to avoid a theoretically possible oscillation between two nodes in states R0 and R1.

Transition R1:T0. All the connected ports are receiving IDLE or RX_PARENT_NOTIFY (indicating that the connected PHYs are in reset wait or starting the tree ID process).

4.4.3.1.2 Bus reset actions and conditions

Table 4-45 shows C code for bus reset actions and conditions.

Table 4-45 — Bus reset actions and conditions (Sheet 1 of 2)

```
boolean reset_detected() {          // Qualify BUS_RESET with port status / history
   int i;

   if (   PHY_state == R0 || PHY_state == R1 // Ignore during (or just before) reset...
       || isbr_OK || phy_response)           // ...or if busy with a PHY command
      return(FALSE);
   for (i = 0; i < NPORT; i++)
      if (disabled[i])              // Ignore completely if disabled
         continue;
      else if (bias[i] && portR(i) == BUS_RESET)
         if (active[i]) {
            reset_time = (PHY_state == RX) ? SHORT_RESET_TIME : RESET_TIME;
            return(TRUE);
         } else if (resume [i] && OK_to_detect_reset && !bus_initialize_active) {
            resumption_done = TRUE;
            reset_time = (boundary_node) ? RESET_TIME : SHORT_RESET_TIME;
            return(TRUE);
         }
   return(FALSE);
}

void reset_start_actions() {     // Transmit BUS_RESET for reset_time on all ports
   int i;

   root = FALSE;
   if (isolated_node)
      force_root = FALSE;        // No point in waiting to become root if isolated
   PH_EVENT.indication(BUS_RESET_START);  // Optional upon reentry to R0 from R1
   ibr = isbr = isbr_OK = FALSE;          // Don't replicate resets!
   phy_response = ping_response = FALSE;  // Invalidate stale information
   breq = NO_REQ;                // Discard any and all link requests for the bus
   children = physical_ID = 0;
   if (!bus_initialize_active) {
      bus_initialize_active = TRUE;
      if (gap_count_reset_disable)  // First reset since setting gap_count?
         gap_count_reset_disable = FALSE;  // If so, leave it as is and arm it for next
      else
         gap_count = 0x3F;          // Otherwise, set it to the maximum
   }
   for (i = 0; i < NPORT; i++) {
      if (active[i])             // For active ports, propagate appropriate signal
         portT(i, BUS_RESET);
      else if (resume[i] && resumption_done)
         portT(i, BUS_RESET);    // Also propagate on resuming ports (if they're ready...)
      else                       // Ignore all other ports
         portT(i, IDLE);
      child[i] = FALSE;
      child_ID_complete[i] = FALSE;
      max_peer_speed[i] = S100;  // Reset default speed for all ports
   }
   arb_timer = 0;                // Start timer
}

void reset_wait_actions() {      // Transmit IDLE
   int i;

   for (i = 0; i < NPORT; i++)
      portT(i, IDLE);
   arb_timer = 0;                // Restart timer
}
```

Table 4-45 — Bus reset actions and conditions (Sheet 2 of 2)

```
boolean reset_complete() {       // TRUE when all ports idle or in tree-ID
   int i;

   for (i = 0; i < NPORT; i ++)
      if (active[i] && (portR(i) != IDLE) && (portR(i) != RX_PARENT_NOTIFY))
         return(FALSE);
   rx_speed = S100;              // For leaf node's self-ID packet(s)
   return(TRUE);                 // Transition to tree identify
}
```

4.4.3.2 Tree identify

NOTE—The tree identify process configures the bus into a tree with a singular root node and labels each node's active ports as connected either to the node's parent or to one of its children (see figure 4-23).

This state machine does not include bus reset if it occurs during tree ID, since the state diagram would be difficult to comprehend. See the All:R0 transitions in figure 4-22.

4.4.3.2.1 Tree ID state machine notes

State T0: Tree-ID Start. In this state, a node waits up to a CONFIG_TIMEOUT period to receive the RX_PARENT_NOTIFY signal from at least all but one of its active ports. When RX_PARENT_NOTIFY is observed, that port is marked as a child port.

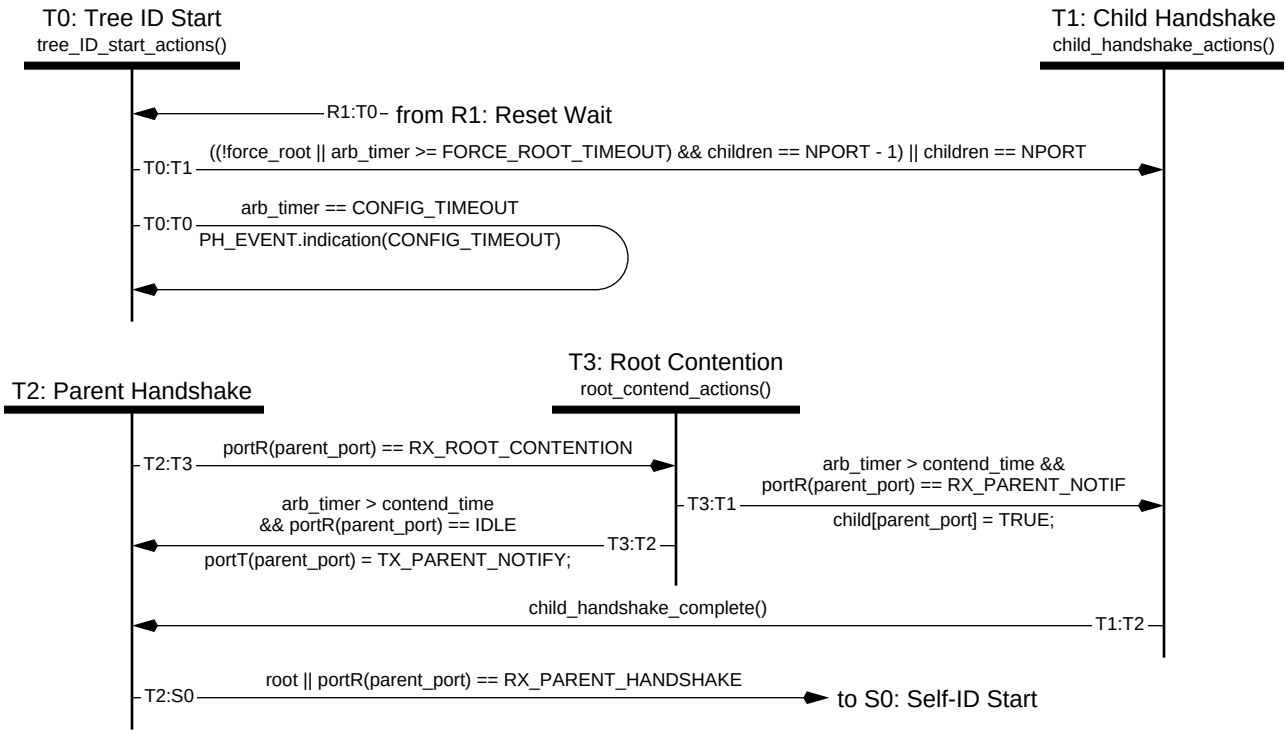


Figure 4-23 — Tree ID state machine

Transition T0:T0. A node stuck in this state waiting for an RX_PARENT_NOTIFY on more than one port means that the user has configured the bus in a loop. This error condition prevents completion of the tree identify process.

Transition T0:T1. If a node detects the RX_PARENT_NOTIFY signal on all of its ports, or all but one of its ports, it knows it is either the root or a branch; it can start the handshake process with its children. Leaf nodes (those with only one connected port) or root nodes (RX_PARENT_NOTIFY on all ports) take this transition immediately. If the `force_root` flag is set, the test for the "all but one port" condition is delayed long enough that all other nodes on the bus should have transitioned at least to state T1 so all the ports should then be receiving the RX_PARENT_NOTIFY signal (this extra delay is the FORCE_ROOT_TIMEOUT value).

State T1: Child Handshake. All ports that have been labeled as child ports transmit the TX_CHILD_NOTIFY signal. This allows the nodes attached to this node's child port(s) to transition from T2 to S0. Leaf nodes have no children, so they exit this state immediately via transition T1:T2. If all ports are labeled as child ports, then the node knows it is the root.

Transition T1:T2. When all of a node's children stop sending TX_PARENT_NOTIFY, it then observes the RX_CHILD_HANDSHAKE signal on all of its child ports. It then knows they have all transitioned to the self-ID start state, so the node can now handshake with its own parent.

State T2: Parent Handshake. At this point, a node is waiting to receive RX_PARENT_HANDSHAKE signal (the result of the node sending TX_PARENT_NOTIFY and its parent sending TX_CHILD_NOTIFY). This step is bypassed if the node is root (receiving RX_PARENT_NOTIFY on all connected ports). Another way this state can be exited is if it receives the RX_ROOT_CONTENTION signal from its parent.

Transition T2:S0. When the node receives the RX_PARENT_HANDSHAKE signal, it starts the self-ID process by sending the IDLE signal (see State S0: Self-ID Start in 4.4.3.3.1). It also takes this transition if it is root, since it doesn't have a parent.

Transition T2:T3. If a node receives an RX_PARENT_NOTIFY signal on the same port that it is sending a TX_PARENT_NOTIFY signal, the merged signal is called RX_ROOT_CONTENTION. This can only happen for a single pair of nodes if each bids to make the other node its parent.

State T3: Root Contention. At this point, both nodes back off by sending an IDLE signal, starting a timer, and picking a random bit. If the random bit is one, the node waits longer than if it is a zero. When the timer has expired, the node samples the contention port once again.

Transition T3:T2. If a node receives an IDLE signal on its proposed parent port at the end of the delay, it once again sends the TX_PARENT_NOTIFY signal. If the other node is taking longer it takes the T3:T1 transition and allows this node to exit state T2 via the self-ID start path. Otherwise the two nodes again see the RX_ROOT_CONTENTION signal and repeat the root contention process with a new set of random bits.

Transition T3:T1. If a node receives an RX_PARENT_NOTIFY signal on the proposed parent port at the end of the delay it means the other node has already transitioned to state T2, so this node returns to state T1 and becomes the root.

4.4.3.2.2 Tree ID actions and conditions

Table 4-46 shows the tree ID actions and conditions.

Table 4-46 — Tree ID actions and conditions

```
void child_handshake_actions() {
   int i;

   parent_port = NPORT;          // No parent port (in case we become root)
   root = TRUE;                  // Remains TRUE if all the ports are child ports
   for (i = 0; i < NPORT; i++)
      if (active[i])             // Only the connected, active ports participate
         if (child[i])
            portT(i, TX_CHILD_NOTIFY);    // Tell peer PHY, “You are my child”
         else {
            portT(i, TX_PARENT_NOTIFY);   // Ask peer PHY, “Please be my parent”
            parent_port = i;              // Only one parent port possible
            root = FALSE;                 // Tentative---see root contention
         }
}

boolean child_handshake_complete() {   // TRUE once all active children are in S0: Self-ID Start
   int i;

   for (i = 0; i < NPORT; i++)
      if (active[i] && child[i] && portR(i) != RX_CHILD_HANDSHAKE)
         return(FALSE);                // One as yet uncompleted handshake...
   return(TRUE);                       // All child ports have finished their handshakes
}

void root_contend_actions() {
   int i;

   contend_time = (random_bool() ? ROOT_CONTEND_SLOW : ROOT_CONTEND_FAST);
   for (i = 0; i < NPORT; i++)
      if (active[i] && child[i])    // Only the connected, active ports matter
         portT(i, TX_CHILD_NOTIFY); // Continue to tell peer PHY, “You are my child”
      else
         portT(i, IDLE);            // Withdraw “Please be my parent” request
   arb_timer = 0;                   // Restart arbitration timer
}

void tree_ID_start_actions() {
   int i;

   do {
      children = 0;              // Count the kids afresh on each loop
      for (i = 0; i < NPORT; i++) {
         if (!active[i]) {       // Child if disabled, disconnected or suspended
            child[i] = TRUE;
            children++;
         } else if (portR(i) == RX_PARENT_NOTIFY) {
            child[i] = TRUE;     // Child if other PHY asks us to be the parent
            children++;
         }
         if (children == NPORT - 1 && (!force_root || arb_timer >= FORCE_ROOT_TIMEOUT))
            return;              // Only one port left as the parent
         else if (children == NPORT)
            return;              // We are the root
      }
   } while (!(reset_detected() || ibr || PH_CONTROL.request(RESET) || arb_timer ==
CONFIG_TIMEOUT));
}
```

4.4.3.3 Self-identify

The self-identify process has each node uniquely identify itself and broadcast its characteristics to any management services.

4.4.3.3.1 Self-ID state machine notes

State S0: Self-ID Start. At the start of the self-ID process, the PHY is waiting for a grant from its parent or the start of a self-ID packet from another node. This state is also entered whenever a node is finished receiving a self-ID packet and all its children have not yet finished their self-identification.

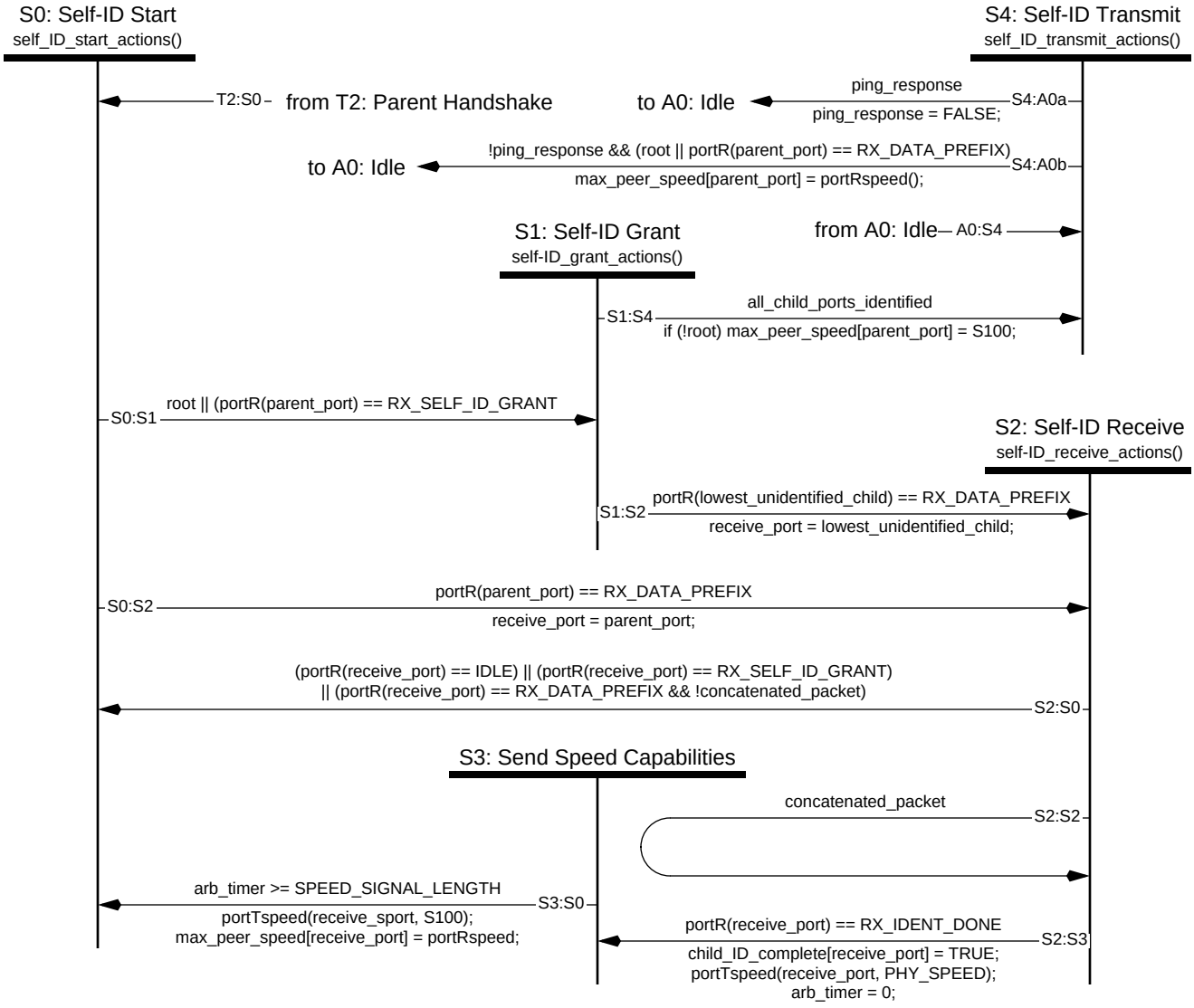


Figure 4-24 — Self-ID state machine

Transition S0:S1. If a node is the root, or if it receives an RX_SELF_ID_GRANT signal from its parent, it enters the Self-ID Grant state.

Transition S0:S2. If a node receives an RX_DATA_PREFIX signal from its parent, it knows that a self-ID packet is coming from a node in another branch in the tree.

State S1: Self-ID Grant. This state is entered when a node is given permission to send a self-ID packet. If it has any unidentified children, it sends a TX_GRANT signal to the lowest numbered of those. All other connected ports are sent a TX_DATA_PREFIX signal to warn them of the start of a self-ID packet.

Transition S1:S2. When the PHY receives an RX_DATA_PREFIX signal from its lowest numbered unidentified child, it enters the Self-ID Receive state.

Transition S1:S4. If there are no more unidentified children, it immediately transitions to the Self-ID Transmit state.

State S2: Self-ID Receive. As data bits are received from the bus they are passed on to the link layer as PHY data indications. This process is described in 4.4.1.2. Note that multiple self-ID packets may be received in this state. The parent PHY must also monitor the received speed-signal whenever RX_IDENT_DONE is received from the child. Because of resynchronization delays in repeating the packet, the parent PHY may not complete retransmission of the packet data and data end signal for up to 144 ns or more (as specified by PHY_DELAY) after the start of the RX_IDENT_DONE signal. Since the child sends its speed-signal for no more than 120 ns from the start of the RX_IDENT_DONE signal, the parent could miss the speed-signal from the child if it entered S3 before completing a speed-signal sample.

Transition S2:S0. When the receive port goes IDLE, gets an RX_SELF_ID_GRANT or observes RX_DATA_PREFIX for a unconcatenated packet it enters the Self-ID Start state to continue the self-ID process for the next child. The last case guards against a possible failure to observe IDLE.

Transition S2:S2. Multiple self-ID packets are received by the PHY and self_ID_receive_actions is reinvoked for each one.

Transition S2:S3. If the PHY gets an RX_IDENT_DONE signal from the receiving port, it flags that port as identified and starts sending the speed capabilities signal. It also starts the speed-signaling timer and sets the port speed to the S100 rate.

State S3: Send Speed Capabilities. If a node is capable of sending data at a higher rate that S100, it transmits on the receiving child port its speed capability signals as defined in 4.2.2.3 for a fixed duration SPEED_SIGNAL_LENGTH. The parent PHY must also monitor the received speed-signal whenever RX_IDENT_DONE is received from the child.

Transition S3:S0. When the speed-signaling timer expires, any signals sent by the child have been latched, so it is safe to continue with the next child port.

State S4: Self-ID Transmit. This state may be entered either as part of the self-identify process or as the result of the receipt of a PHY ping packet. In the latter case, any pending link requests are cancelled and the set of self-ID packets are transmitted. When state S4 is entered as part of the self-identify process, the actions are more complex, as described in the following paragraph.

At this point, all child ports have been flagged as identified, so the PHY can now send its own self-ID packet (see clause 4.3.4) using the process described in 4.4.1.1. When a non-root node is finished, it sends a TX_IDENT_DONE signal while simultaneously transmitting a speed capability signal (as defined in 4.2.2.3) to its parent and IDLE to its children. The speed capability signal is transmitted for a fixed time duration of SPEED_SIGNAL_LENGTH. Simultaneously it monitors the bus for a speed capability transmission from the parent. The highest indicated speed is recorded as the speed capability of the parent. The root node just sends IDLE to its children. Note that the children then enter the *Idle* state described in 4.4.3.3.2, but they do not start arbitration since an adequate arbitration gap does not elapse until the self-ID process is completed for all nodes.

While transmitting the TX_IDENT_DONE signal in the S4 state, the child monitors the received speed-signal from the parent. The child PHY then transitions to the A0:Idle state when it receives an RX_DATA_PREFIX signal from its parent. The parent PHY will be in the S2:Self-ID Receive state to receive the self-ID packet(s) from the child. When the parent PHY receives an RX_IDENT_DONE signal from the child PHY, the parent transitions to the S3:Send Speed Capabilities state. In the S3 state, the parent transmits a speed-signal for 100–120 ns to indicate its own speed capability, and monitors the received speed-signal from the child. The highest indicated speed is recorded as the speed capability of the child. After transmitting its own speed-signal, the parent PHY transitions to the S0:Self-ID Start state.

Transition S4:A0b. The PHY then enters the *Idle* state described in the next clause when the self-ID packet has been transmitted and if either of the following conditions are met:

a) *The node is the root.* When the root enters the *Idle* state, all nodes are now sending IDLE signals and the gap timers eventually measure enough elapsed time to allow normal arbitration to start.

b) *The node starts to receive a new self-ID packet (RX_DATA_PREFIX – 10).* This is the self-ID packet for the parent node or another child of the parent. This event shall cause the PHY to transition immediately out of A0:Idle into A5:Receive.

4.4.3.3.2 Self-ID actions and conditions

Table 4-47 shows self-ID actions and conditions.

Table 4-47 — Self-ID actions and conditions (Sheet 1 of 3)

```
void self_ID_start_actions() {
   int i;

   all_child_ports_identified = TRUE;     // Will be reset if any active children are unidentified
   concatenated_packet = FALSE;           // Prepare in case of multiple self-ID packets
   for (i = 0; i < NPORT; i++)
     if (child_ID_complete[i])
        portT(i, TX_DATA_PREFIX);         // Tell identified children to prepare to receive data
     else {
        portT(i, IDLE);                   // Allow parent to finish
        if (child[i] && active[i]) {      // If active child
           if (all_child_ports_identified)
              lowest_unidentified_child = i;
           all_child_ports_identified = FALSE;
        }
     }
}

void self_ID_grant_actions() {
   int i;

   for (i = 0; i < NPORT; i++)
      if (!all_child_ports_identified && (i == lowest_unidentified_child))
         portT(i, TX_GRANT);        // Send grant to lowest unidentified child (if any)
      else if (active[i])
         portT(i, TX_DATA_PREFIX);  // Otherwise, tell others to prepare for packet
}

void self_ID_receive_actions() {
   int i;

   portT(receive_port, IDLE);             // Turn off grant, get ready to receive
   receive_actions();                     // Receive (and repeat) packet
   if (!concatenated_packet) {            // Only do this on the first self-ID packet
      if (physical_ID < 63)               // Stop at 63 if malconfigured bus
         physical_ID = physical_ID + 1;   // Otherwise, take next PHY address
```

Table 4-47 — Self-ID actions and conditions (Sheet 2 of 3)

```
      for (i = 0; i < NPORT; i++)
         portT(i, IDLE);                  // Turn off all transmitters
   }
}

void self_ID_transmit_actions() {
   int last_SID_pkt = (NPORT + 4) / 8;
   int SID_pkt_number;              // Packet number counter
   int port_number = 0;             // Port number counter
   quadlet self_ID, ps;

   receive_port = NPORT;           // Indicate that we are transmitting (no port has this number)
   for (SID_pkt_number = 0; SID_pkt_number <= last_SID_pkt; SID_pkt_number++) {
      start_tx_packet(S100);        // Send data prefix and 98.304 Mbit/sec speed code
      PH_DATA.indication(DATA_START, S100);
      self_ID_pkt.dataQuadlet = 0;  // Clear all zero fields in self-ID packet
      self_ID.type = 0b10;
      self_ID.phy_ID = physical_ID;
      if (SID_packet_number == 0) { // First self-ID packet?
         self_ID.L = LPS && lctrl;  // Link active or not?
         self_ID.gap_cnt = gap_count;
         self_ID.sp = PHY_SPEED;
         self_ID.c = CONTENDER;
         self_ID.pwr = POWER_CLASS;
         self_ID.i = initiated_reset;
      } else {
         self_ID.seq = 1;        // Indicates second and subsequent packets
         self_ID.n = SID_pkt_number - 1;  // Sequence number
      }
      ps = 0;                     // Initialize for fresh group of ports
      while (port_number < ((SID_pkt_number + 1) * 8 - 5)) {    // Concatenate port status
         if (port_number >= NPORT)
            ;                     // Unimplemented
         else if (!active[port_number])
            ps |= 0b01;           // Disabled, disconnected or suspended
         else if (child[port_number])
            ps |= 0b11;           // Active child
         else
            ps |= 0b10;           // Active parent
         port_number++;
         ps <<= 2;                // Make room for next port's status
      }
      self_ID |= ps;
      if (SID_pkt_number == last_SID_pkt) { // Last packet?
         tx_quadlet(self_ID);
         tx_quadlet(~self_ID);
         stop_tx_packet(DATA_END, S100, DATA_END_TIME); // Yes, signal data end
         PH_DATA.indication(DATA_END);
      } else {
         self_ID.m = 1;           // Other packets follow, set "more" bit
         tx_quadlet(self_ID_pkt);
         tx_quadlet(~self_ID_pkt);
         stop_tx_packet(DATA_PREFIX, S100, CONCATENATION_PREFIX_TIME);  // Keep bus for
concatenation
         PH_DATA.indication(DATA_PREFIX);
      }
   }
   if (!ping_response) {        // Skip if self-ID packet was in response to a ping
      for (port_number = 0; port_number < NPORT; port_number++)
         if (root || port_number != parent_port)
            portT(port_number, IDLE);           // Turn off transmitters to children
         else
            portT(port_number, TX_IDENT_DONE);  // Notify parent that self-ID is complete
```

Table 4-47 — Self-ID actions and conditions (Sheet 3 of 3)

```
      if (!root) {                          // If we have a parent...
         portTspeed(parent_port, PHY_SPEED); // Send speed-signal (if any)
         wait_time(SPEED_SIGNAL_LENGTH);
         portTspeed(parent_port, S100);      // Stop sending speed-signal
      }
      PH_EVENT.indication(SELF_ID_COMPLETE, physical_ID, root);  // Register 0
   }
}

void tx_quadlet(quadlet quad_data) {
   int i;

   if (breq != NO_REQ) {                        // Is a request pending? (only fair and priority
possible)
      breq = NO_REQ;                               // If so, cancel it...
      PH_ARB.confirmation(LOST);                   // ...and advise the link
   }
   for (i = 0; i < 32; i++) {                      // Send the quadlet a bit at a time
      tx_bit(quad_data & 0x80000000);              // From the most significant downwards
      PH_DATA.indication(quad_data & 0x80000000);  // Copy our own self-ID packet to the link
      quad_data <<= 1;                             // Shift to next bit
   }
}
```

4.4.3.4 Normal arbitration

Normal arbitration is entered as soon as a node has finished the self-identification process (see figure 4-25). At this point, a simple request-grant handshake process starts between a node and its parent (and all parents up to the root).

4.4.3.4.1 Normal arbitration state machine notes

State A0: Idle. All inactive nodes stay in the *Idle* state until an internal or external event. All ports transmit the IDLE arbitration signal. Transitions into this state from states where idle was not being sent reset an idle period timer.

Transition A0:A0. If a subaction gap or arbitration reset gap occurs, the PHY notifies the link layer. The first subaction gap after a bus reset also signals the completion of the self-identify process, in which case the PHY notifies the node controller. The detection of an arbitration reset gap marks the end of a fairness interval; the PHY sets the arbitration enable flag.

Transition A0:A1. If the PHY has a queued request (other than an immediate request) from its own link or receives an RX_REQUEST signal from one of its children (and is not the root), it passes the request on to its parent. The `arb_OK()` function qualifies asynchronous requests according to the time elapsed since A0: Idle was last entered. In particular, notice that the test for a subaction gap is performed for a single value (equality), not a greater-than comparison. If arbitration were to be initiated at other times between the detection of a subaction gap and an arbitration reset gap, some nodes could mistakenly observe an arbitration reset gap.

Transition A0:A2. If, on the other hand, the PHY receives an RX_REQUEST signal from one of its children, has no queued requests from its own link and is the root, it starts the bus grant process.

Transition A0:PH. If an extended PHY packet (other than the ping packet) has been received, a response is required.

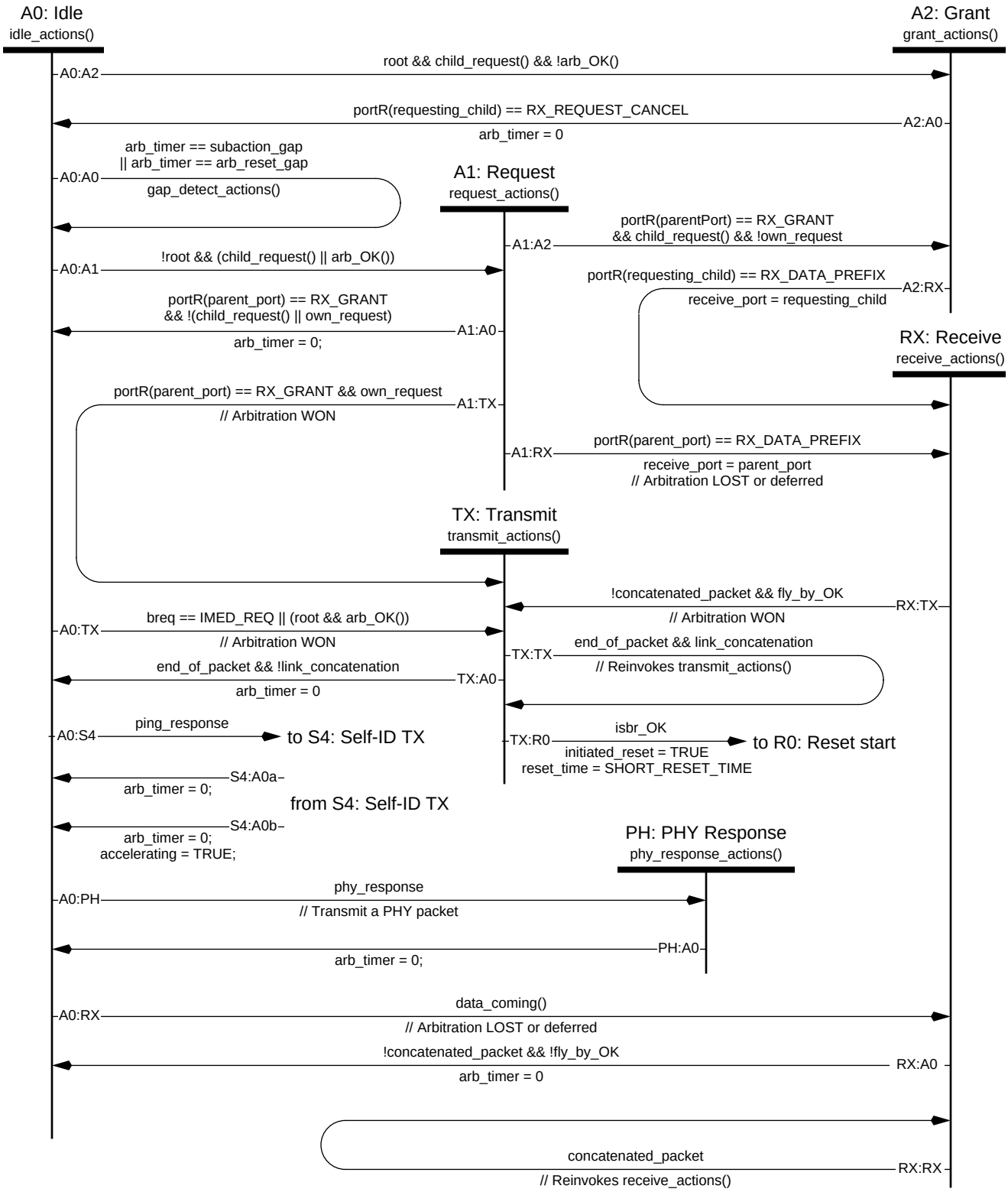


Figure 4-25 — Cable arbitration state machine

Transition A0:TX. If the PHY has a queued request and is the root or if the PHY has a queued immediate request (generated during packet reception if the link layer needs to send an acknowledge), the PHY notifies the link layer that it is ready to transmit and enters the Transmit state.

Transition A0:S4. In response to the receipt of a PHY "ping" packet, the variable ping_response is set to TRUE and a transition is made to the Self-ID Transmit State to send the self-ID packet(s).

State A1: Request. At this point, the PHY sends a TX_REQUEST signal to its parent and a data prefix to all its connected children. This signals all children to get ready to receive a packet.

Transition A1:A0. If the PHY receives an RX_GRANT signal from its parent and the requesting child has withdrawn its request, the PHY returns to *Idle* state.

Transition A1:A2. If the PHY receives an RX_GRANT signal from its parent and the requesting child is still making a request, the PHY grants the bus to that child.

Transition A1:RX. If the PHY receives an RX_DATA_PREFIX signal from its parent, then the PHY knows that it has lost the arbitration process and prepares to receive a packet. If the link layer was making the request, it is notified.

Transition A1:TX. If the PHY receives an RX_GRANT signal from its parent and the link layer has an outstanding request (asynchronous or isochronous), the PHY notifies the link layer that it can now transmit and enters the Transmit state.

State A2: Grant. During the grant process, the requesting child is sent a TX_GRANT signal and the other children are sent a TX_DATA_PREFIX so that they prepare to receive a packet.

Transition A2:A0. If the requesting child withdraws its request, the granting PHY sees its own TX_GRANT signal coming back as an RX_REQUEST_CANCEL signal and returns to the *Idle* state.

Transition A2:RX. If the data prefix signal is received from the requesting child, the grant handshake is complete and the node goes into the Receive state.

State PH: PHY Response. When the node has received an extended PHY packet (other than the ping packet) that requires a response, this state takes the appropriate actions, builds a remote reply or remote confirmation packet, and transmits the packet from all active ports.

Transition PH:A0. After transmitting the PHY response packet, return unconditionally to the *Idle* state.

State RX: Receive. When the node starts the receive process, it notifies the link layer that the bus is busy and starts the packet receive process that follows. Outstanding fair and priority requests are cancelled—immediately if arbitration enhancements are globally disabled, otherwise by the receipt of any packet other than an acknowledgment—and the link will have to reissue them later. Note that the packet received could be a PHY packet (self-ID, link-on, or PHY configuration), acknowledge, or normal data packet. PHY configuration and link-on packets are interpreted by the PHY, as well as being passed on to the link layer.

Transition RX:A0. If transmitting node stops sending any signals (received signal is ZZ) or if a packet ends normally when the received signal is RX_DATA_END, the bus is released and the PHY returns to the *Idle* state.

Transition RX:RX. If a packet ends and the received signal is RX_DATA_PREFIX (10), then there may be another packet coming, so the receive process is restarted.

Transition RX:TX. If fly-by arbitration is enabled and an acknowledge packet ends, a queued fair or priority request may be granted.

State TX: Transmit. Unless an arbitrated (short) bus reset has been requested, the transmission of a packet starts by the node sending a TX_DATA_PREFIX and speed-signal as described in 4.2.2.3 for 100 ns, then sending PHY clock indications to the link layer. For each clock indication, the Link sends a PHY data request. The clock indication/data request sequence repeats until the Link sends a DATA_END. Concatenated packets are handled within this state whenever the Link sends at least one data bit followed by a DATA_PREFIX. The arbitration enable flag is cleared if this was a fair request.

Transition TX:A0. If the link layer sends a DATA_END, the PHY shuts down transmission using the procedure described in 4.4.1.1 and returns to the *Idle* state.

Transition TX:R0. If arbitration succeeded and the `isbr_OK` variable is set, there is no packet to transmit. The PHY transitions to the Reset start state to commence a short bus reset.

Transition TX:TX. The link is holding the bus in order to send a concatenated packet. Remain in the transmit state and restart the transmit process for the next packet.

4.4.3.4.2 Normal arbitration actions and conditions

Table 4-47A shows the normal arbitration actions and conditions.

Table 4-47A — Normal arbitration actions and conditions (Sheet 1 of 3)

```
boolean fly_by_permitted() {        // TRUE if fly-by acceleration OK

   if (!enab_accel)
      return(FALSE);
   else if (receive_port == parent_port)
      return(FALSE);
   else if (speed == S100 && rx_speed != S100)
      return(FALSE);
   else if (breq == ISOCH_REQ)
      return(TRUE);
   else if (ack && accelerating)
      return(breq == PRIORITY_REQ || (breq == FAIR_REQ && arb_enable));
   else
      return(FALSE);
}

boolean child_request() {              // TRUE if a child is requesting the bus
   int i;

   for (i = 0; i < NPORT; i++)
      if (active[i] && child[i] && (portR(i) == RX_REQUEST)) {
         requesting_child = i;         // Found a child that is requesting the bus
         return(TRUE);
      }
   return(FALSE);
}

boolean data_coming() {                // TRUE if data prefix is received on any port
   int i;

   for (i = 0; i < NPORT; i++)
      if (active[i] && (portR(i) == RX_DATA_PREFIX)) {
         receive_port = i;             // Remember port for later...
         return(TRUE);                 // Found a port that is sending a data prefix signal
      }
   return(FALSE);
}

void gap_detect_actions() {
```

Table 4-47A — Normal arbitration actions and conditions (Sheet 2 of 3)

```
  if (arb_timer >= arb_reset_gap) {       // End of fairness interval?
     arb_enable = TRUE;                   // Reenable fair arbitration
     PH_DATA.indication(ARBITRATION_RESET_GAP);  // Alert link
  } else if (arb_timer >= subaction_gap) {
     PH_DATA.indication(SUBACTION_GAP);   // Notify link
     if (bus_initialize_active) {         // End of self-identify process for whole bus?
        PH_EVENT.indication(BUS_RESET_COMPLETE);
        bus_initialize_active = FALSE;
     }
  }
}

void idle_actions() {
   int i;

   rx_speed = S100;                 // Default in anticipation of no explicit receive speed code
   for (i = 0; i < NPORT; i++)      // Turn off all transmitters
      portT(i, IDLE);
   wait_time(MIN_IDLE_TIME);        // Do not exit A0: Idle before this minimum has elapsed
}


void request_actions() {
   int i;

   for (i = 0; i < NPORT; i++)
      if (active[i] && child[i] && (own_request || i != requesting_child))
         portT(i, TX_DATA_PREFIX);  // Send data prefix to all non-requesting children
   portT(parent_port, TX_REQUEST);  // Send request to parent
}

boolean arb_OK() {               // TRUE if OK to request the bus
   boolean async_arb_OK = FALSE; // Timing window OK for asynchronous arbitration?

   if (arb_timer < subaction_gap + arb_delay)   // Only window for accelerations
      async_arb_OK = enab_accel && accelerating && ack;
   if (arb_timer >= subaction_gap && child_request())
      async_arb_OK = TRUE;       // Small window for stealing a child's request
   else if (arb_timer == subaction_gap + arb_delay)
      async_arb_OK = TRUE;       // Window for first fair request and priority requests
   else if (arb_timer >= arb_reset_gap + arb_delay)
      async_arb_OK = TRUE;       // Window for all requests (new fairness interval)
   if (breq == ISOCH_REQ)
      own_request = TRUE;
   else if (isbr)
      own_request = isbr_OK = async_arb_OK;
   else if (breq == PRIORITY_REQ)
      own_request = async_arb_OK;
   else if (breq == FAIR_REQ)
      own_request = async_arb_OK && arb_enable;
   else
      own_request = FALSE;
   return(own_request);
}
```

Table 4-47A — Normal arbitration actions and conditions (Sheet 3 of 3)

```
void grant_actions() {
   int i;

   for (i = 0; i < NPORT; i++)
      if (i == requesting_child)
         portT(i, TX_GRANT);        // Send grant to requesting child
      else if (active[i] && child[i])
         portT(i, TX_DATA_PREFIX);  // Send data prefix to all non-requesting children
}
```

4.4.3.4.3 Receive actions and conditions

The C code in table 4-47B is inadequate to capture some nuances of synchronized behavior between the PHY and link; these arise from the fact that the PHY may receive an arbitrary number of clocked data bits from the bus but is constrained by the nature of the PHY/link interface to transfer multiples of 2, 4, or 8 bits (dictated by the speed-signaled to the link). In particular, table 5A-16 specifies that if arbitration acceleration is enabled and the PHY transfers exactly 8 data bits to the link then any outstanding fair or priority arbitration request is retained, otherwise it is cancelled. The requirements of 5A.3.1 shall take precedence over the C code in table 4-47B without regard for the actual number of clocked data bits received from the bus. If the PHY transfers an 8-bit packet to the link, the PHY shall behave as if it had received a packet whose length is exactly 8 bits. That is, an outstanding fair or priority arbitration request shall be retained. In this circumstance, the `ack` flag is cleared to FALSE to prevent any erroneous attempt at fly-by arbitration; this has no significant effect on the retained, since the PHY arbitrates on behalf of the retained link request at the next appropriate opportunity.

Table 4-47B — Receive actions and conditions (Sheet 1 of 2)

```
void receive_actions() {
   unsigned bit_count = 0, i, rx_data;
   boolean end_of_data;
   PHY_PKT rx_phy_pkt;

   ack = concatenated_packet = fly_by_OK = isbr_OK = FALSE;
   if (!enab_accel && (breq == FAIR_REQ || breq == PRIORITY_REQ)) {
      breq = NO_REQ;                  // Cancel the request
      PH_ARB.confirmation(LOST);      // And let the link know
   }
   PH_DATA.indication(DATA_PREFIX);   // Send notification of bus activity
   start_rx_packet();                 // Start up receiver and repeater
   PH_DATA.indication(DATA_START, rx_speed); // Send speed indication
   do {
      rx_bit(&rx_data, &end_of_data);
      if (!end_of_data) {             // Normal data, send to link layer
         PH_DATA.indication(rx_data);
         if (bit_count < 64)          // Accumulate first 64 bits
            rx_phy_pkt.dataBits[bit_count] = rx_data;
         bit_count++;
         ack = (bit_count == 8);      // For acceleration, any 8-bit packet is an ack
         if (bit_count > 8 && (breq == FAIR_REQ || breq == PRIORITY_REQ)) {
            breq = NO_REQ;            // Fly-by impossible
            PH_ARB.confirmation(LOST); // Let the link know (immediately) on 9th bit
         }
      }
   } while (!end_of_data);
   if (!ack && (breq == FAIR_REQ || breq == PRIORITY_REQ)) {
      breq = NO_REQ;                  // Fly-by impossible
      PH_ARB.confirmation(LOST);      // Advise the link
   }
   if (portR(receive_port) == BUS_RESET || portR(receive_port) == IDLE) {  // No data?
      ack = FALSE;      // Disable ack-accelerated arbitration
```

Table 4-47B — Receive actions and conditions (Sheet 2 of 2)

```
      return;
   } else if (   (portR(receive_port) != RX_DATA_END)
               && (portR(receive_port) != RX_DATA_PREFIX)) { // Unexpected end of data...
      ack = FALSE;      // Disable ack-accelerated arbitration
      for (i = 0; i < NPORT; i++)
         if (active[i] && i != receive_port)
            portT(i, TX_DATA_END);
      wait_time(DATA_END_TIME);
      return;
   }
   switch(portR(receive_port)) {       // Send appropriate end of packet indicator
      case RX_DATA_PREFIX:
         concatenated_packet = TRUE;
         PH_DATA.indication(DATA_PREFIX);    // Concatenated packet coming
         stop_tx_packet(DATA_PREFIX, rx_speed, CONCATENATION_PREFIX_TIME);
         break;

      case RX_DATA_END:
         PH_DATA.indication(DATA_END);
         fly_by_OK = fly_by_permitted();
         if (bit_count < 8) {          // Null packet? Don't add dribble bits if so...
            for (i = 0; i < NPORT; i++)
               if (active[i] && i != receive_port)
                  portT(i, (fly_by_OK) ? TX_DATA_PREFIX : TX_DATA_END);
             wait_time(DATA_END_TIME); // Guarantee MIN_IDLE_TIME on receive port during
concatenation
         } else if (fly_by_OK)
            stop_tx_packet(DATA_PREFIX, tx_speed, DATA_END_TIME); // Fly-by concatenation
                                          // Intended to guarantee MIN_IDLE_TIME on receive port
         else
            stop_tx_packet(DATA_END, tx_speed, DATA_END_TIME);    // Normal end of packet
         break;
   }
   if (bit_count == 64) {              // We have received a PHY packet
      for (i = 0; i < 32; i++)         // Check PHY packet for good format
         if (rx_phy_pkt.dataBits[i] == rx_phy_pkt.checkBits[i])
            return;                   // Check bits invalid - ignore packet
      decode_phy_packet(rx_phy_pkt);   // Parse valid PHY packets
   }
}
```

4.4.3.4.4 Transmit actions and conditions

Table 4-48 shows transmit actions and conditions.

Table 4-48 — Transmit actions and conditions (Sheet 1 of 2)

```
void transmit_actions() {        // Send a packet as link transfers it to the PHY

   int bit_count = 0, i;
   boolean end_of_packet = FALSE;
   phyData data_to_transmit;
   PHY_PKT tx_phy_pkt;

   if (breq == FAIR_REQ)         // Just used our one fair request per fairness interval?
      arb_enable = FALSE;        // Yes, clear permission (set again on next reset gap)
   breq = NO_REQ;
   tx_speed = speed;             // Assume speed has been set correctly by link...
                                 // (from PH_ARB.request or concatenated packet speed code)
   receive_port = NPORT;         // Impossible port number ==> PHY transmitting
   start_tx_packet(tx_speed);    // Send data prefix & speed-signal
   if (isbr_OK)                  // Avoid phantom packets...
```

Table 4-48 — Transmit actions and conditions (Sheet 2 of 2)

```
      return;
   PH_ARB.confirmation(WON);     // Signal grant on Ctl[0:1]
   while (!end_of_packet) {
      PH_CLOCK.indication();     // Tell link to send data
      data_to_transmit = PH_DATA.request();     // Wait for data from the link
      switch(data_to_transmit) {
         case DATA_ONE:
         case DATA_ZERO:
            tx_bit(data_to_transmit);
            if (bit_count < 64)                 // Accumulate possible PHY packet
               tx_phy_pkt.dataBits[bit_count] = data_to_transmit;
            bit_count++;
            break;

         case DATA_PREFIX:
            end_of_packet = link_concatenation = TRUE;
            stop_tx_packet(DATA_PREFIX, tx_speed, CONCATENATION_PREFIX_TIME); //
MIN_PACKET_SEPARATION
          break;     // needs to be guaranteed by stop_tx_packet() and subsequent start_tx_packet()

         case DATA_END:
            end_of_packet = TRUE;         // End of packet indicator
            link_concatenation = FALSE;
            stop_tx_packet(DATA_END, tx_speed, DATA_END_TIME);
            break;
      }
   }
   ack = (bit_count == 8);    // Used elsewhere to (conditionally) accelerate
   if (bit_count == 64) {     // We have transmitted a PHY packet
      for (i = 0; i < 32; i++)            // Check PHY packet for good format
         if (tx_phy_pkt.dataBits[i] == tx_phy_pkt.checkBits[i])
            return;                       // Check bits invalid - ignore packet
      decode_phy_packet(tx_phy_pkt);      // Parse any valid transmitted packet
   }
}
```

4.4.3.4.5 PHY response actions and conditions

Table 4-49 shows physical layer response actions and conditions.

Table 4-49 — PHY response actions and conditions (Sheet 1 of 3)

```
PHY_PKT phy_resp_pkt;                    // Create remote confirmation or reply packet here

void decode_phy_packet(PHY_PKT phy_pkt) {
   int i;

   if (phy_pkt.type == 0b10)             // Self-ID packet?
      ;                                  // If so, ignore
   else if (phy_pkt.type == 0b01) {      // Link-on packet?
      if (phy_pkt.phy_ID == physical_ID)
         PH_EVENT.indication(LINK_ON);    // LinkOn asserted only if either LPS or LCtrl FALSE
   } else if (phy_pkt.R != 0 || phy_pkt.T != 0) {  // PHY configuration packet?
      if (phy_pkt.R)                     // Set force_root if address matches
         force_root = (phy_pkt.phy_ID == physical_ID)
      if (phy_pkt.T) {                   // Set gap_count regardless of physical ID
         gap_count = phy_pkt.gap_cnt;
         gap_count_reset_disable = TRUE;
      }
   } else if (phy_pkt.ext_type == 0)     // Ping packet?
      ping_response = (phy_pkt.phy_ID == physical_ID);
   else if ((phy_pkt.ext_type == 1 || phy_pkt.ext_type == 5) && phy_pkt.phy_ID == physical_ID)
```

Table 4-49 — PHY response actions and conditions (Sheet 2 of 3)

```
      remote_access(phy_pkt.page, phy_pkt.port, phy_pkt.reg);
   else if (phy_pkt.ext_type == 8 && phy_pkt.phy_ID == physical_ID)
      remote_command(phy_pkt.cmnd, phy_pkt.port);
   else if (phy_pkt.ext_type == 0xF)      // Resume packet?
      for (i = 0; i < NPORT; i++)
         if (!active[i] && !disabled[i] && connected[i])
            resume[i] = TRUE;             // Resume all suspended/suspending ports
}

void remote_access(int page, int port, int reg) {  // Current value of remotely read register
   phy_resp_pkt.dataQuadlet = 0;
   phy_resp_pkt.phy_ID = physical_ID;
   phy_resp_pkt.ext_type = (phy_pkt.ext_type == 1) ? 3 : 7;
   phy_resp_pkt.page = page;
   phy_resp_pkt.port = port;
   phy_resp_pkt.reg = reg;
   phy_resp_pkt.data = read_phy_reg(page, port, reg);
   phy_response = TRUE;
}

void remote_command(int cmnd, int port) {          // Conditionally execute requested command
   phy_resp_pkt.dataQuadlet = 0;
   phy_resp_pkt.phy_ID = physical_ID;
   phy_resp_pkt.ext_type = 0x0A;
   phy_resp_pkt.port = port;
   phy_resp_pkt.ok = TRUE;
   if (cmnd == 1)                // Disable port
      if (port == receive_port)  // What? Disable the port that received the packet?
         phy_resp_pkt.ok = FALSE;   // No, we're not going to lock ourselves out...
      else if (active[port])     // Active, TX_DISABLE_NOTIFY to peer PHY first
         disable_notify[port] = TRUE;
      else                       // Otherwise (inactive), just mark disabled
         disabled[port] = TRUE;
   else if (cmnd == 2)           // Suspend port
      if (!active[port] || resume_in_progress() || suspend_in_progress())
         phy_resp_pkt.ok = FALSE;
      else
         suspend[port] = TRUE;
   else if (cmnd == 4)           // Clear fault conditions
      resume_fault[port] = suspend_fault[port] = FALSE;
   else if (cmnd == 5)           // Enable port
      disabled[port] = FALSE;
   else if (cmnd == 6)           // Resume port
      if (     active[port] || !connected[port] || disabled[port]
            || resume_in_progress() || suspend_in_progress())
         phy_resp_pkt.ok = FALSE;
      else
         resume[port] = TRUE;
   phy_resp_pkt.fault = resume_fault[port] || suspend_fault[port];
   phy_resp_pkt.connected = connected[port];
   phy_resp_pkt.bias = bias[port];
   phy_resp_pkt.disabled = disabled[port];
   phy_resp_pkt.cmnd = cmnd;
   phy_response = TRUE;
}

void phy_response_actions() {
   receive_port = NPORT;                 // We are transmitting (no port has this number)
   start_tx_packet(S100);                // Send data prefix and 98.304 Mbit/sec speed code
   PH_DATA.indication(DATA_START, S100);  // CC: the link (always)
   tx_quadlet(phy_resp_pkt);
   tx_quadlet(~phy_resp_pkt);
   stop_tx_packet(DATA_END, S100, DATA_END_TIME);     // Signal data end
```

Table 4-49 — PHY response actions and conditions (Sheet 3 of 3)

```
   PH_DATA.indication(DATA_END);          // And also inform the link
   if (     phy_resp_pkt.ext_type == 0x0A
         && (disable_notify[phy_resp_pkt.port] || phy_resp_pkt.cmnd == 2)
         && phy_resp_pkt.ok) {
      breq = IMMED_REQ;                  // Bus reset active ports if disable or suspend
      isbr = isbr_OK = TRUE;
   } else
      breq = NO_REQ;
   phy_response = FALSE;
}
```

4.4.4 Port connection

The port connection state machines (see figure 4-26) operate independently for each port, i, where i is a positive integer less than NPORT. While a port is in the active state its arbitration, data transmission, reception, and repeat behaviors are specified by the state machines in 4.4.3. When a PHY port is in any state other than active, it is permissible for it to lower its power consumption; the only functional components of a PHY required to be active in these states are the bias detect and physical connection detect circuits.

Although the port state machines are described as if they operate independently of the arbitration state machine, in fact it is necessary for the arbitration state machine to be in the *Idle* state for the P2:P4 transition to be permitted. In addition, the arbitration line state should be qualified by other information to guarantee its validity. For example, either RX_SUSPEND or RX_DISABLE_NOTIFY are spurious unless immediately preceded by a receipt of a PHY response packet on the same port.

One of the critical procedures that is an input to these state machines is `connection_status()` in table 4-50. This procedure runs constantly and monitors observed TpBias and, when a port is inactive, the connect detect circuitry. The C code describes how new connections are debounced and how the bias detection circuit output is filtered before the corresponding *Bias* bit in the PHY registers is updated.

NOTE—PHY designers are advised that insertion of a bus-powered device may result in the transient loss of bias (sometimes on ports other than the newly connected port). A possible remedy is to filter the loss of bias for 200–300 ns by either analog or digital logic methods. If a PHY implements this strategy, it is essential that it be able to detect a loss of bias in less time than the minimum subaction gap (440 ns).

The C code is an imperfect abstraction that does not capture the fact that Bias_Detect shall not be sampled while speed-signaling is active (see 4.2.2.2) nor the implicit requirements for a noise filter (see 4.2.2.6).

4.4.4.1 Port connection state machine notes

Transition All:P0. A power reset of the PHY initializes each port as disconnected.

Transition All:P6. The local link may immediately disable a port by setting the *Disabled* bit to one. This transition may also be caused by a remote command packet, in which case the port, if active, shall transmit TX_DISABLE_NOTIFY before the port is disabled.

State P0: Disconnected. The generation of TpBias is disabled and the outputs are in a high-impedance state. The PHY may place most of the port's circuitry in a low-power consumption state. The connection detect circuit shall be active even if other components of the PHY port are in a low-power state.

Transition P0:P1. When a port's connection detect circuitry signals that its peer PHY port is physically connected, the PHY port transitions to the Resuming state.

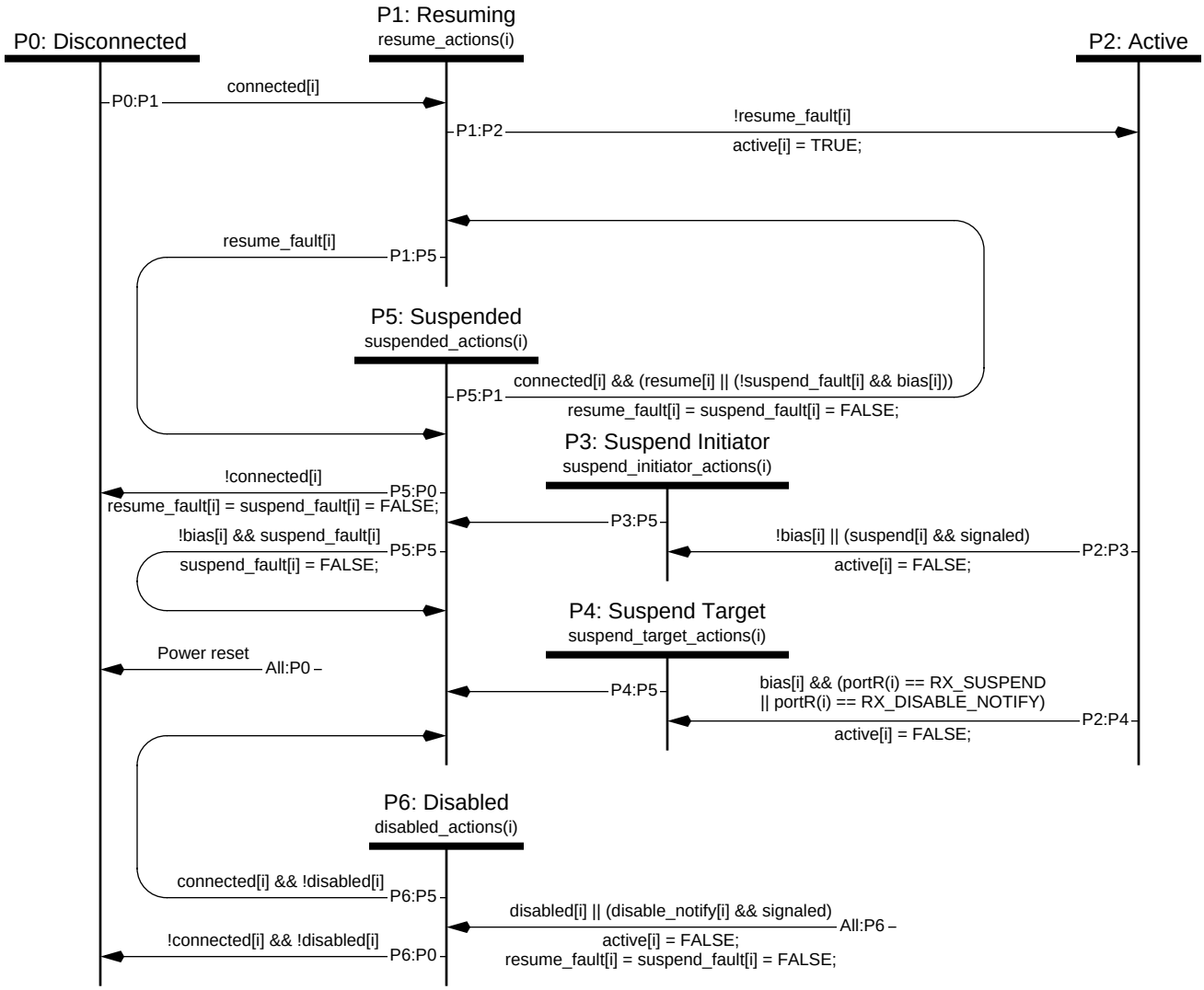


Figure 4-26 — Port connection state machine

State P1: Resuming. The PHY port tests both the connection status and the presence of TpBias to determine if normal operations may be resumed. If the port is connected, TpBias is present and there are no other active ports, the PHY waits seven RESET_DETECT intervals before any state transitions. Otherwise, in the case of a boundary node with one or more active ports, the PHY waits three RESET_DETECT intervals before any state transitions. A detected bus reset overrides either of these waits. Otherwise, when the wait elapse, the PHY initiates a bus reset.

Transition P1:P2. If the PHY port did not fault during the resume handshake, it waits for bus reset to start and then transitions to the active state.

Transition P1:P5. A resuming PHY port that faults during the resume handshake transitions to the suspended state.

State P2: Active. The PHY port is fully operational, capable of transmitting or receiving and repeating arbitration signals or clocked data. While the port remains active, the behavior of this port and the remainder of the PHY are subject to the cable arbitration states specified in 4.4.

Transition P2:P3. The PHY port leaves the active state to start functioning as a suspend initiator after either of two events—the loss of observed bias or the receipt of a PHY remote command packet that set the port's suspend variable to one. A loss of bias is usually the result of a physical disconnection or the loss of power to the connected peer PHY port. If the transition is the result of a remote command packet, exit from the active state is delayed until the PHY transmits a remote confirmation packet with the ok bit set to one and subsequently signals TX_SUSPEND to its connected peer PHY.

Transition P2:P4. If an active port observes an RX_DISABLE_NOTIFY or RX_SUSPEND signal, it becomes a suspend target and leaves the active state.

State P3: Suspend Initiator. A suspend initiator waits for bias to be zero. If BIAS_HANDSHAKE elapses and the connected peer PHY has not driven TpBias low, the suspend operation has faulted and the *Fault* bit is set to one. In either case, the suspend initiator first drives TpBias low for BIAS_HANDSHAKE time and then places all outputs in a high-impedance state.

Transition P3:P5. Upon completion of the actions associated with this state, the PHY port unconditionally transitions to the suspended state.

State P4: Suspend Target. A suspend target sets the suspend variable for all the other active ports, which in turn causes them to propagate the RX_SUSPEND signal as TX_SUSPEND. In the meantime the suspend target drives its TpBias outputs below 0.1 V in order to signal the suspend initiator that RX_SUSPEND was detected. When either the connected peer PHY drives TpBias low or a BIAS_HANDSHAKE time-out expires (whichever occurs first), the suspend target disables the generation of TpBias and places the outputs in a high-impedance state.

Transition P4:P5. Upon completion of the actions associated with this state, the PHY port unconditionally transitions to the suspended state.

State P5: Suspended. The PHY may place most of the port's circuitry in a low-power consumption state. The connection detect circuit shall be active even if other components of the PHY port are in a low-power state.

Transition P5:P0. A suspended PHY port that loses its physical connection to its peer PHY port transitions to the disconnected state.

Transition P5:P1. Either of two conditions cause a suspended PHY port to transition to the resuming state

a) A nonzero value for the port's *resume* variable, or

b) The detection of *bias* if the port's *suspend_fault* variable is zero.

The second condition also results if software zeros suspend_fault while bias is still present. A port's resume variable may be set indirectly as the result of the resumption of other PHY ports.

Transition P5:P5. If the port entered the suspended state in a faulted condition (i.e., TpBias was still present), the fault is cleared if and when TpBias is removed by the peer PHY.

State P6: Disabled. While disabled, the PHY may place most of the port's circuitry in a low-power consumption state. The connection detect circuit shall be active even if other components of the PHY port are in a low-power state.

Transition P6:P0. If the *Disabled* bit is zero and the PHY port is not physically connected to its peer PHY port, it transitions to the disconnected state.

Transition P6:P5. Otherwise, if the *Disabled* bit is zero and the PHY port is connected, it transitions to the suspended state.

4.4.4.2 Port connection actions and conditions

Table 4-50 shows port connection actions and conditions.

Table 4-50 — Port connection actions and conditions (Sheet 1 of 4)

```
void activate_connect_detect(int i, int delay) {
   tpBias(i, 0);                    // Drive TpBias low
   if (delay != 0) {
      connect_timer = 0;
      while (connect_timer < delay) // Enforce minimum hold time for TpBias low
         ;
   }
   while (!connect_detect[i])       // Wait for connect_detect circuit to stabilize
      ;                             // (this assumes sufficient hysteresis in analog circuit)
   connect_detect_valid[i] = TRUE;  // Signal OK to connection_status()
   tpBias(i, Z);                    // Release TpBias
}

void connection_status() {          // Continuously monitor port status in all states
   static timer bias_timer;             // Timer for bias filter (initially zero)
   static boolean bias_filter[NPORT];   // TRUE when applying hysteresis to bias_detect circuit
                                        // (initially FALSE)
   int active_ports = 0, i, suspended_ports = 0;

   isolated_node = TRUE;            // Remains TRUE if no active port(s) found
   for (i = 0; i < NPORT; i++) {
      if (active[i]) {
         active_ports++;            // Necessary to deduce boundary node status
         isolated_node = FALSE;     // ALL ports must be inactive at an isolated node
      } else if (connected[i] && !disabled[i])
         suspended_ports++;         // Other part of boundary node definition
      boundary_node = (active_ports > 0 && suspended_ports > 0);
   }
   for (i = 0; i < NPORT; i++) {
      if (bias_detect[i] == FALSE) {// 200 - 300 ns hysteresis recommended
         bias_filter[i] = FALSE;    // Cancel filtering (if in progress)
         bias[i] = FALSE;           // Report immediately
      } else if (bias_filter[i]) {  // Filtering positive bias transition?
         if (bias_timer >= BIAS_FILTER_TIME) {
            bias_filter[i] = FALSE; // Done filtering
            bias[i] = TRUE;         // Confirm new value in PHY register bit
         }
      } else if (   !disabled[i]    // Detected and reported bias differ on enabled port?
                 && bias_detect[i] != bias[i]) {
         bias_filter[i] = TRUE;     // Yes, start a filtering period
         bias_timer = 0;
      }
      if (connection_in_progress[i]) {
         if (!connect_detect[i])
            connection_in_progress[i] = FALSE;  // Lost attempted connection
         else if (connect_timer >= CONNECT_TIMEOUT) {
            connection_in_progress[i] = FALSE;
            connected[i] = TRUE;                // Confirmed connection
            if (disabled[i] && int_enable[i] && !port_event) { // Notify disabled ports
               port_event = TRUE;               // (Resumption itself notifies enabled ports)
               PH_EVENT.indication(INTERRUPT);
            }
         }
      } else if (   !connected[i]            // Recognize connection if port or interrupts enabled
                 && (!disabled[i] || int_enable[i])) {
         if (connect_detect[i]) {   // Possible new connection?
            connect_timer = 0;      // Start connect timer
            connection_in_progress[i] = TRUE;
```

Table 4-50 — Port connection actions and conditions (Sheet 2 of 4)

```
         }
      } else if (connect_detect_valid[i] && !connect_detect[i]) {
         connected[i] = FALSE;      // Detect disconnect instantaneously
         if (int_enable[i] && !port_event) {
            port_event = TRUE;
            PH_EVENT.indication(INTERRUPT);
         }
      }
   }
}

void disabled_actions(int i) {
   if (int_enable[i] && !port_event) {
      port_event = TRUE;
      PH_EVENT.indication(INTERRUPT);
   }
   disable_notify[i] = signaled = FALSE;
   disabled[i] = TRUE;
   activate_connect_detect(i, 0);      // Enable the connect detect circuit
}

boolean resume_in_progress() {         // TRUE if any port resuming
   int i;

   for (i = 0; i < NPORT; i++)
      if (resume[i])
         return(TRUE);
   return(FALSE);
}

void resume_actions(int i) {
   boolean resume_bias_OK;             // AND of bias on all resuming ports
   int j;

   while (suspend_in_progress())       // Let any other suspensions complete
      ;                                // (we may resume those ports later)
   connect_timer = 0;
   OK_to_detect_reset = resumption_done = FALSE;
   connect_detect_valid[i] = FALSE;    // Bias renders connect detect circuit useless
   tpBias(i, 1);                       // Generate TpBias
   if (!resume[i] && !boundary_node) {
      for (j = 0; j < NPORT; j++)
         if (!active[j] && !disabled[j] && connected[j])
            resume[j] = TRUE;          // Resume all suspended ports
   } else
      resume[i] = TRUE;                // Guarantee resume_in_progress() returns TRUE
   do {                                // Need to wait until bias is OK on all resuming ports
      resume_bias_OK = TRUE;           // Assume this is true until proven otherwise
      for (j = 0; j < NPORT; j++)
         resume_bias_OK &= (!resume[j] || bias[j]);
   } while ((connect_timer < BIAS_HANDSHAKE) && !resume_bias_OK);
   if (bias[i]) {                      // OK to continue to resume if TpBias is present
      if ((int_enable[i] || watchdog) && !port_event) {
         port_event = TRUE;
         PH_EVENT.indication(INTERRUPT);
      }
      while (bias[i] && !resumption_done) {  // Defer bus reset until “ready”
         if (bus_initialize_active)          // Do nothing if reset commences
            ;
         else if (connect_timer >= PORT_ENABLE_TIME && !OK_to_detect_reset)
           OK_to_detect_reset = TRUE;       // Now safe to detect bus reset on all resuming ports
         else if (boundary_node && (connect_timer >= 3 * RESET_DETECT))
            isbr = resumption_done = TRUE;   // If we can arbitrate, short reset NOW!
```

Table 4-50 — Port connection actions and conditions (Sheet 3 of 4)

```
         else if (connect_timer >= 7 * RESET_DETECT)
            ibr = resumption_done = TRUE;    // Sigh! We'll have to use long reset
      }
      while (bias[i] && !bus_initialize_active)    // Wait for bus reset to start
         ;
   }
   resume_fault[i] = ~bias[i];         // Resume attempt failed if TpBias is absent
   resume[i] = FALSE;                  // Resume attempt complete
   if (!resume_in_progress())          // Last resuming port does housekeeping upon completion
      OK_to_detect_reset = resumption_done = FALSE;
}

boolean suspend_in_progress() {        // TRUE if any port suspending
   int i;

   for (i = 0; i < NPORT; i++;)
      if (suspend[i])
         return(TRUE);
   return(FALSE);
}

void suspend_initiator_actions(int i) {
   connect_timer = 0;            // Used to debounce bias or for bias handshake
   if (!suspend[i]) {            // Unexpected loss of bias?
      suspend[i] = TRUE;         // Insure suspend_in_progress() returns TRUE
      if (child[i])              // Yes, parent still connected?
         isbr = TRUE;            // Arbitrate for short reset
      else
         ibr = TRUE;             // Transition to R0 for reset
      activate_connect_detect(i, 0);
      while (connected[i] && connect_timer < CONNECT_TIMEOUT / 2)
         ;                       // See if bias lost because of physical disconnection
   } else {                      // Instructed to suspend
      signaled = FALSE;
      while ((connect_timer < BIAS_HANDSHAKE) && bias[i])
         ;                       // Wait for suspend target to deassert bias
      suspend_fault[i] = bias[i]);  // Suspend handshake refused by target?
      activate_connect_detect(i, BIAS_HANDSHAKE);  // Also guarantees handshake timing
   }
}

void suspend_target_actions(int i) {
   int j;

   if (resume_in_progress())     // Other ports resuming?
      resume[i] = TRUE;          // OK, do suspend handshake but resume afterwards
   suspend[i] = TRUE;            // Insure suspend_in_progress() returns TRUE
   if (portR(i) == RX_DISABLE_NOTIFY) {   // Is our peer PHY going away?
      breq = IMMED_REQ;          // Topology change! Reset on other (active) ports
      isbr = isbr_OK = TRUE;
   } else if (portR(i) == RX_SUSPEND && !resume[i]) { // Don't propagate if resume in progress
      for (j = 0; j < NPORT; j++)
         if (active[j])          // Otherwise all active ports become suspend initiators
            suspend[j] = TRUE;
      breq = IMMED_REQ;          // Invoke transmitter to propagate TX_SUSPEND
      isbr = isbr_OK = TRUE;     // Alert link that we're now isolated
   }
   while (portR(i) == RX_DISABLE_NOTIFY || portR(i) == RX_SUSPEND)
      ;                          // Let signals complete before bias handshake
   activate_connect_detect(i, BIAS_HANDSHAKE);
}

void suspended_actions(int i) {
```

Table 4-50 — Port connection actions and conditions (Sheet 4 of 4)

```
   suspend[i] = FALSE;
   if (int_enable[i] && !port_event) {
      port_event = TRUE;
      PH_EVENT.indication(INTERRUPT);
   }
   if (resume_fault[i]) {
      while (!bias[i] && (connect_timer < 12 * RESET_DETECT))
         ;
      if (!bias[i])
         activate_connect_detect(i, 0);
   }
}
```

Insert the following after clause 5:

5A. PHY/link interface specification

This clause standardizes the PHY/link interface previously described in informative annex J. It specifies the protocol and signal timing. It does not describe specific operation of the PHY except for behavior with respect to this interface.

The interface specified in this clause is a scalable method to connect one Serial Bus link chip to one Serial Bus PHY chip. It supports data rates of S25 and S50 in the backplane environment and S100, S200, and S400 in the cable environment. The width of the data bus scales with Serial Bus speed—two signals support speeds up to 100 Mbit/s while at faster speeds a total of two signals per 100 Mbit/s are necessary. The clock rate of the signals at this interface remains constant, independent of Serial Bus speed. The interface permits isolation for implementations where it is desirable.

The interface may be used by the link to transmit data, receive data or status, or issue requests. The link makes requests of the PHY via the dedicated LReq signal in order to read or write a PHY register, to ask the PHY to initiate a transmit operation, or to control arbitration acceleration. In response, the PHY may transfer control of the bidirectional signals to the link. At all other times the PHY controls the bidirectional signals and may autonomously transfer data to the link, for either a receive operation (when a packet is received from Serial Bus) or a status transfer.

Discrete PHY implementations shall support all of the PHY signals shown in figure 5A-1. Discrete link implementations shall support D[0:*n*], Ctl[0:1], LReq, and SClk; link support for the other signals is optional. For both PHY and link, the number of data bits implemented, n, depends upon the maximum speed supported by the device. The PHY/link interface signals are described in table 5A-1.

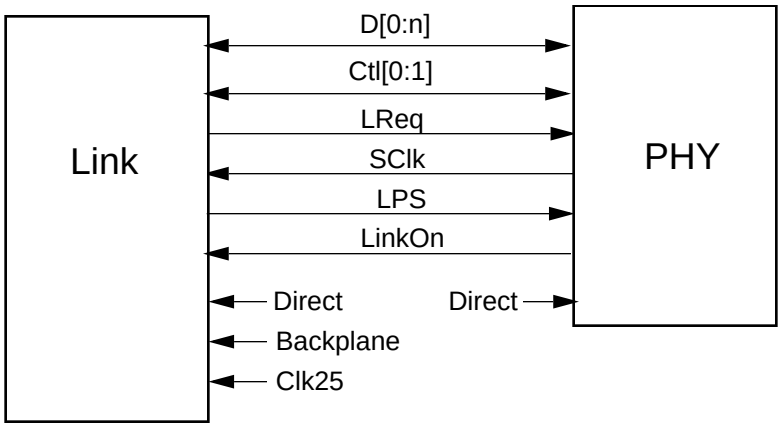


Figure 5A-1 — Discrete PHY/link interface

Table 5A-1 — PHY/link signal description

Name	Driven by	Description
D[0:*n*]	Link or PHY	Data
Ctl[0:1]	Link or PHY	Control
LReq	Link	Link request

Table 5A-1 — PHY/link signal description (continued)

Name	Driven by	Description
SClk	PHY	12.288 MHz, 24.576 MHz, or 49.152 MHz clock (synchronized to the PHY transmit clock)
LPS	Link	Link power status—indicates that the link is powered and functional
LinkOn	PHY	Occurrence of a link-on event
Direct	—[a]	Set high to disable differentiator outputs for the Ctl[0:1], D[0:*n*], and LReq signals
Backplane	—[a]	Set high if backplane PHY
Clk25	—[a]	Meaningful only if Backplane is high—set high to indicate a 24.576 MHz SClk; otherwise 12.288 MHz

[a]Usually determined by design or system configuration options (such as hardware strapping).

Data is transferred between the PHY and link on D[0:*n*]. The implemented width of D[0:*n*] depends on the maximum speed of the device—two bits for S100 or slower, four bits for S200, and eight bits for S400. At S100 or slower, packet data is transferred on D[0:1], at S200 on D[0:3], and at S400 on D[0:7]. Implemented but unused D[0:*n*] signals shall be driven low by the device that has control of the interface.

The PHY or the link may drive Ctl[0:1] to indicate the type of data transfer on D[0:*n*]. The encoding of the control bus signals is specified by table 5A-2 and table 5A-3.

Table 5A-2 — Ctl[0:1] when PHY is driving

Ctl[0:1]	Name	Meaning
00_2	*Idle*	No activity
01_2	*Status*	The PHY is sending status information to the link
10_2	*Receive*	A packet is being transferred from the PHY to the link
11_2	*Grant*	The link is granted the bus to send a packet

Table 5A-3 — Ctl[0:1] when the link is driving (upon a grant from the PHY)

Ctl[0:1]	Name	Meaning
00_2	*Idle*	Transmission complete, release bus
01_2	*Hold*	The link is holding the bus while preparing data or indicating that it wishes to reacquire the bus without arbitrating to send another packet
10_2	*Transmit*	The link is sending a packet to the PHY
11_2	—	Unused

The LReq signal is used by the link to request access to Serial Bus for packet transmission, to read or write PHY registers, or to control arbitration acceleration.

The presence of a stable SClk signal generated by the PHY is necessary for the PHY/link interface to be operational. When SClk is not shown in the timing diagrams in this clause, each Ctl[0:1], D[0:*n*] or LReq bit cell represents a single clock sample time. The specific timing relationships are described in 5A.8.2 and 5A.8.3.

NOTE—In cases where the PHY and link are powered independently of each other, the link implementation should be able to detect the loss of SClk from an otherwise initialized and operational PHY/link interface.

The LPS signal may be used by the link to disable SClk or reset the interface, as specified in 5A.1.

The LinkOn signal permits the PHY to indicate an interrupt to the link when LPS is logically false. The details are specified in 5A.2.

The Direct input controls digital differentiators on the D[0:*n*], Ctl[0:1], and LReq signals. When set high, the Direct input shall disable differentiator outputs on these signals (which shall be otherwise enabled). In the case that the link does not implement Direct, the link shall be configured so that output on these signals, differentiated or not, conforms to the value of Direct provided to the PHY.

NOTE—Differentiators may be required when the PHY and link are connected through an optional isolation barrier; see annex A for a discussion of electrical isolation in the cable environment. A digital differentiator drives its output signal for one clock period whenever the input signal changes, but places the output signal in a high-impedance state so long as the input signal remains constant. Figure 5A-2 illustrates this signal transformation.

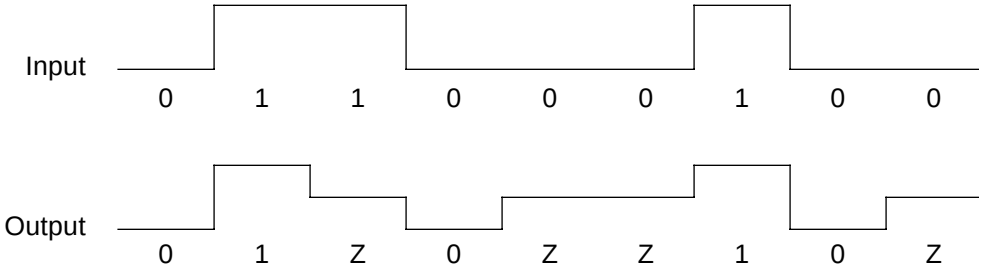


Figure 5A-2 — Digital differentiator signal transformation

When a backplane PHY is connected to a link, the Backplane input shall be strapped high. The Clk25 input is meaningful only to indicate the SClk frequency generated by a backplane PHY. In the backplane environment, data transfers use D[0:1]. SClk is used to clock the transfers at either 12.288 MHz (for TTL applications) or 24.576 MHz (for BTL and ECL applications). This yields PHY data rates at the backplane of S25 and S50, respectively.

5A.1 Initialization and reset

The LPS input requests the PHY to disable or enable the PHY/link interface. The output characteristics of LPS, if provided by the link, depend upon the interface mode, differentiated or undifferentiated. When the interface mode is differentiated, LPS shall be a pulsed output while logically asserted. When logically deasserted, LPS shall be driven low in either interface mode. The characteristics of LPS are specified by figure 5A-3 and table 5A-4.

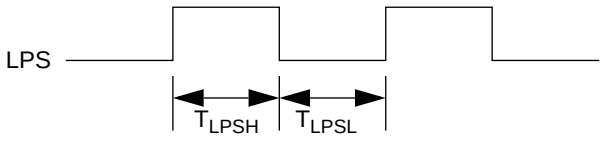


Figure 5A-3 — LPS waveform when differentiated

Table 5A-4 — LPS timing parameters

Parameter	Description	Unit	Minimum	Maximum
T_{LPSL}	LPS low time (when pulsed)	µs	0.09	1.00
T_{LPSH}	LPS high time (when pulsed)	µs	0.09	1.00
	Duty cycle (when pulsed)	%	20	60
T_{LPS_RESET}	Time for PHY to recognize LPS logically deasserted and reset the interface	µs	1.2	2.75
$T_{LPS\_DISABLE}$	Time for PHY to recognize LPS logically deasserted and disable the interface	µs	25	30
$T_{RESTORE}$	Time to permit the optional differentiator and isolation circuits to restore during an interface reset	µs	15	20[a]

[a]This maximum does not apply when the PHY/link interface is disabled (see figure 5A-5), in which case an indefinite time may elapse before LPS is reasserted. Otherwise, in order to reset but not disable the interface, it is necessary that the link insure that LPS is logically deasserted for less than $T_{LPS\_DISABLE}$.

The link requests the PHY to reset the interface by deasserting LPS. Within 1.0 µs after it deasserts LPS, the link shall place Ctl[0:1] and D[0:*n*] in a high-impedance state and condition LReq according to the interface mode; if undifferentiated, LReq shall be driven zero. Otherwise, it shall be placed in a high-impedance state.

If the PHY observes LPS logically deasserted for T_{LPS_RESET}, it shall reset the interface. The voltage levels shown in figure 5A-4 for Ctl[0:1], D[0:*n*], and LReq while LPS is logically deasserted are accurate only for an undifferentiated interface, but the timing relationships remain accurate for both modes. When the interface is undifferentiated, the PHY drives Ctl[0:1] and D[0:*n*] to zero. Otherwise, the PHY places the Ctl[0:1] and D[0:*n*] signals in a high-impedance state.

If the link continuously deasserts LPS for a longer period, it requests the PHY not only to reset, but also to disable the interface (see figure 5A-5). The link shall condition its outputs as already described for reset.

If the PHY observes LPS logically deasserted for $T_{LPS\_DISABLE}$, it shall disable the interface. The PHY has already reset the interface as described previously; it now disables the interface by stopping SClk. The voltage levels shown in figure 5A-5 for Ctl[0:1], D[0:*n*], LReq, and SClk, while LPS is logically deasserted, are accurate only for an undifferentiated interface, but the timing relationships remain accurate for both modes. When the interface is undifferentiated, the PHY disables the interface by driving SClk to zero while continuing to drive Ctl[0:1] and D[0:*n*] to zero. Otherwise, the PHY disables the interface by placing SClk in a high-impedance state while continuing to maintain the Ctl[0:1] and D[0:*n*] signals in a high-impedance state.

NOTE—When the PHY/link interface is disabled and none of the PHY's ports are active or in a transitional state, the PHY may place most of its circuitry in a low-power state.

When the PHY/link interface is reset, the PHY shall cancel any outstanding bus request or register read request. Although the cancellation of bus requests may affect PHY arbitration states in ways not described in clause 5B, the PHY's behaviors (as observable from Serial Bus) shall be consistent with that clause. For example, the PHY may have initiated arbitration in response to a bus request but reset of the PHY/link interface might cancel the request before it is granted. Appropriate PHY behavior would be the transmission of a null packet.

The C code and state machines in clause 5B describe the PHY's operation as if the interface to the link is always operational. If the PHY/link interface is reset while the link is transmitting a packet, the PHY shall behave as if the link had signaled *Idle* and terminated the packet. Similarly, any S[0:3] status bit information generated by the PHY while the interface is not operational (whether reset, disabled, or in the process of initialization) shall be zeroed and shall not cause a status transfer upon restoration of the interface.

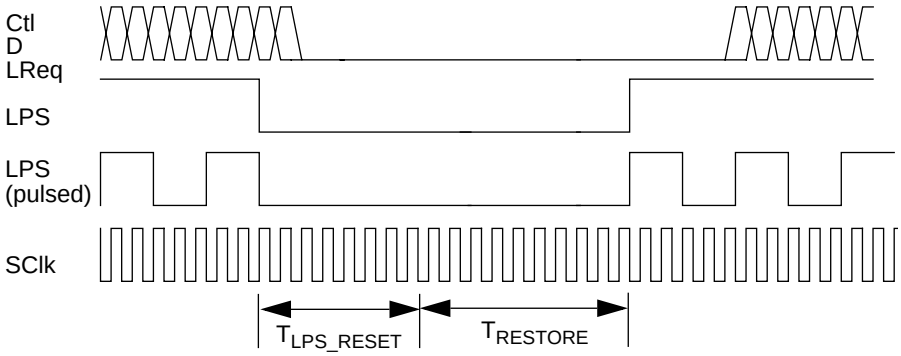


Figure 5A-4 — PHY/link interface reset via LPS

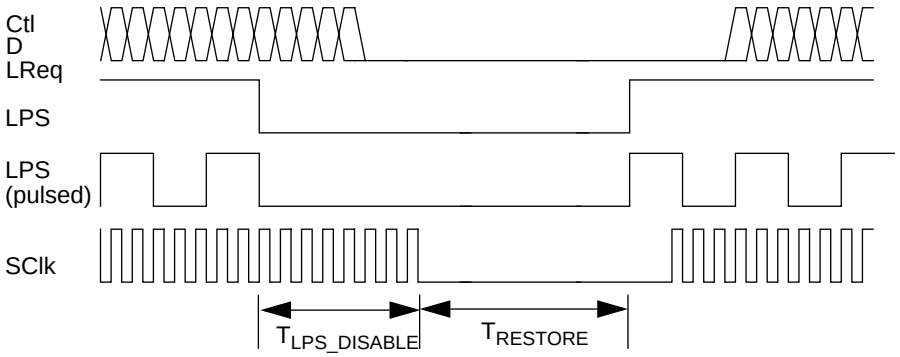


Figure 5A-5 — PHY/link interface disable via LPS

The handshake just described either resets or resets and then disables the interface when the link deasserts LPS for a minimum of 2.75 µs or 30 µs, respectively. In either case (or after power reset), normal operations may be restored if the link asserts LPS. After observing LPS, if SClk is not already provided by the PHY, it shall resume SClk as soon as possible, but no earlier than 10 µs since the PHY's most recent power reset. If the PHY/link interface is differentiated and SClk is resumed, the PHY shall commence by driving SClk low for a minimum of 5 ns. In either mode, the PHY shall ensure that duty cycle and period requirements of table 5A-21 are met from the first rising edge of SClk onwards. Once SClk is available, the PHY and link shall condition their Ctl[0:1] and D[0:*n*] outputs in accordance with table 5A-5. The reference point is the first rising edge of SClk after LPS is asserted.

The PHY may not be able to determine its appropriate interface mode, differentiated or undifferentiated, immediately after a power reset. While the mode is indeterminate, the PHY shall place its outputs in a high-impedance state.

Upon the eighth SClk cycle, the PHY shall assert *Receive* on Ctl[0:1] while simultaneously providing data prefix indication on D[0:*n*] for at least one SClk cycle. Upon the subsequent SClk cycles, the PHY shall drive Ctl[0:1] and D[0:*n*] as follows:

— The PHY shall continue to indicate data prefix while it is in a state in which (if initialization of the PHY/link interface were complete) it would normally assert other than *Idle* on Ctl[0:1], and

— Subsequently it shall assert *Idle* on Ctl[0:1] for at least one cycle in order to indicate the completion of PHY/link interface initialization and the resumption of normal operations.

Table 5A-5 — Initialization of the PHY/link interface

Device	Interface mode	
	Differentiated	**Undifferentiated**
PHY	For one and only one of the first six cycles of SClk after the reference point, drive Ctl[0:1] and D[0:*n*] to zero and otherwise, for these cycles and the seventh, place them in a high-impedance state.	Continue to drive Ctl[0:1] and D[0:*n*] to zero for the first seven cycles of SClk after the reference point.
Link	For one and only one of the first six cycles of SClk after the reference point, drive Ctl[0:1], D[0:*n*], and LReq to zero and otherwise place them in a high-impedance state.	For one and only one of the first six cycles of SClk after the reference point, drive Ctl[0:1] and D[0:*n*] to zero; prior to this place them in a high-impedance state. Once these signals have been driven low, return them to a high-impedance state until after the reset completes. LReq shall be driven low once the operating mode of the interface is determined and shall continue to be driven low until after the reset completes.

The link may examine Ctl[0:1] once it has driven Ctl[0:1], D[0:*n*], and LReq to zero for one cycle subsequent to the SClk reference point. When the link simultaneously observes *Receive* on Ctl[0:1] and data prefix on D[0:*n*], and subsequently observes *Idle* on Ctl[0:1], the reset of the PHY/link interface is complete. The PHY shall insure that no more than 10 ms elapse from the reassertion of LPS until the interface is reset. The link shall not assert LReq until the reset is complete.

Once initialization of the PHY/link interface completes successfully, the link shall not issue an isochronous request until a cycle start packet is either observed or generated by the link.

5A.2 Link-on and interrupt indications

The PHY LinkOn output provides a method to signal the link at times when the PHY/link interface is not operational. The PHY/link interface is not operational when the LPS signal is logically deasserted (see 5A.1). The characteristics of the LinkOn signal, specified by table 5A-6, permit the link to detect LinkOn in the absence of SClk and also permit the signal to cross an optional isolation barrier. When LinkOn is logically deasserted it shall be driven low.

Table 5A-6 — LinkOn timing parameters

Description	Unit	Minimum	Maximum
Frequency	MHz	4	8
Duty cycle	%	30	60
Persistence—time, measured from the point at which both LPS is active and *LCtrl* is one, after which the PHY shall not signal LinkOn	ns	—	500

When necessary to communicate a PH_EVENT.*indication* of LINK_ON or INTERRUPT, the PHY shall assert LinkOn if LPS is logically deasserted and may assert LinkOn if the PHY register *LCtrl* bit is zero.[6]

[6]Although the PHY should determine whether or not to signal LinkOn solely on the status of the PHY/link interface, some implementations assert LinkOn if the PHY register *LCtrl* bit is zero. This is redundant but harmless; software may simply set the *LCtrl* bit to one in acknowledgment.

NOTE—While the PHY/link interface is operational, all link-on packets are transferred to the link. When the *phy_ID* field matches the PHY's physical ID, this transfer is an implicit PH_EVENT.*indication* of LINK_ON; the PHY need not assert LinkOn in this case.

Once asserted, the LinkOn signal shall persist so long as the LPS signal is logically deasserted and may persist so long as the PHY register *LCtrl* bit is zero, with one exception. A bus reset shall clear the LinkOn signal unless

a) The PHY register *Port_event* bit is one, or

b) The PHY register *Watchdog* bit is one and a loop, power failure, or time-out condition exists.

5A.3 Link requests

To request the bus, access a PHY register, or control arbitration acceleration, the link sends a bit sequence (request) to the PHY on the LReq signal. The link always signals all bits of the request. The information sent includes the type of request and parameters that depend upon the type of request. Examples of parameters are packet transmission speed, priority, PHY register address, or data. With the exception of the bus request, each request is terminated by a stop bit of zero. The size of the request, inclusive of the stop bit, varies between 6 bits and 17 bits. When the link transmits zeros on LReq the request interface is *Idle*.

The timing for this signal and the definition of the bits in the transfer are shown in figure 5A-6.

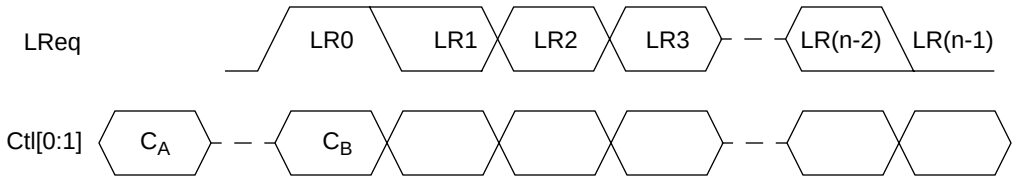


Figure 5A-6 — LReq and Ctl timings

If the LReq transfer is a bus request in the cable environment, it is 7 bits or 8 bits long and has the format given in table 5A-7. Links compliant with this standard shall send all 8 bits. PHYs shall accept bus requests in both the format specified by table J-4 and the format given in table 5A-7.

Table 5A-7 — Bus request format for cable environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always one.
1–3	Request Type	Indicates type of bus request—immediate, isochronous, priority, or fair. See table 5A-12 for the encoding of this field.
4–6	Request Speed	The speed at which the PHY is to transmit the packet on Serial Bus. This field has the same encoding as the speed code from the first symbol of the receive packet. See table 5A-13 for the encoding of this field.
7	Stop Bit	Indicates end of transfer. Always zero. If bit 6 is zero, this bit may be omitted.

If the LReq transfer is a bus request in the backplane environment, it is 11 bits long and has the format given in table 5A-8.

Table 5A-8 — Bus request format for backplane environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always one.
1–3	Request Type	Indicates type of bus request—immediate, isochronous, priority, or fair. See table 5A-12 for the encoding of this field.
4–5	—	Reserved.
6–9	Request Priority	Indicates priority of urgent requests. (Only used with FairReq request type.) All zeros indicates fair request. All ones is reserved (this priority is implied by a PriReq). Other values are used to indicate the priority of an urgent request.
10	Stop Bit	Indicates end of transfer. Always zero.

If the transfer is a register read request, it is 9 bits long and has the format given in table 5A-9.

Table 5A-9 — Register read request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always one.
1–3	Request Type	Indicates that this is a register read. See table 5A-12 for the encoding of this field.
4–7	Address	The internal PHY address to be read.
8	Stop Bit	Indicates end of transfer. Always zero.

If the transfer is a register write request, it is 17 bits long and has the format given in table 5A-10.

Table 5A-10 — Register write request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always one.
1–3	Request Type	Indicates that this is a register write. See table 5A-12 for the encoding of this field.
4–7	Address	The internal PHY address to be written.
8–15	Data	For a write transfer, the data to be written to the specified address.
16	Stop Bit	Indicates end of transfer. Always zero.

If the transfer is an acceleration control request, it is 6 bits long and has the format given in table 5A-11.

Table 5A-11 — Acceleration control request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always one.
1–3	Request Type	Indicates that this is an acceleration control request. See table 5A-12 for the encoding of this field.
4	Accelerate	When zero, instructs the PHY to disable arbitration accelerations. A value of one requests the PHY to enable arbitration accelerations.
5	Stop Bit	Indicates end of transfer. Always zero.

The request type field is encoded as shown in table 5A-12.

Table 5A-12 — Request type field

Request Type	Name	Meaning
000_2	ImmReq	Take control of the bus immediately upon detecting *Idle*; do not arbitrate. Used for acknowledge packets.
001_2	IsoReq	Arbitrate for the bus after an isochronous gap. Used for isochronous stream packets.
010_2	PriReq	Ignore the PHY's fairness protocol and, unless accelerating, arbitrate after a subaction gap. Used for cycle master or other packets for which the link need not wait for a fairness interval.
011_2	FairReq	Arbitrate within the current fairness interval if permitted by the PHY's fairness interval. Otherwise, arbitrate after an arbitration reset gap.
100_2	RdReg	Return specified register contents through status transfer.
101_2	WrReg	Write to specified register.
110_2	AccCtrl	Disable or enable PHY arbitration accelerations.
111_2	—	Reserved for future standardization.

The request speed field is encoded as shown in table 5A-13. The actual data rates for the S100, S200, and S400 speed codes are specified by 4.2.3.1. Although encoding for speeds up to S3200 is specified in table 5A-13, the PHY/link interface does not support speeds in excess of S400.

Table 5A-13 — Request speed field

LR[4:6]	Data rate
000_2	S100
001_2	S1600
010_2	S200
011_2	S3200
100_2	S400
110_2	S800
All other values	Reserved

The PHY continuously monitors LReq for link requests and sets internal variables in response to the parameters of the request. These actions occur independently of the state of the PHY arbitration control; the effects upon PHY arbitration, if any, are a consequence of the values of the internal variables, as specified in 4.4. Table 5A-14 summarizes the effects of the various link requests.

Table 5A-14 — Link request effects on PHY variables

Request	PHY variables affected	Note
ImmReq, IsoReq, PriReq, FairReq	`breq`, `speed` `accelerating` (see note)	The `breq` variable is set to IMMED_REQ, ISOCH_REQ, PRIORITY_REQ, or FAIR_REQ according to the type of request. The `speed` variable is set to S100, S200, etc., according to the encodings specified by table 5A-13. The `accelerating` variable is affected only by an IsoReq, which sets it to TRUE.
RdReg	—	The values returned in response to a register read are an instantaneous snapshot. Asynchronous events, such as bus reset, may cause the PHY to autonomously change the values of PHY register(s).

Table 5A-14 — Link request effects on PHY variables (continued)

Request	PHY variables affected	Note
WrReg	See table 5B-1	The PHY updates the addressed register with the data field value from the request and sets the value of any PHY variables that correspond to register bits or fields.
AccCtrl	`accelerating`	If the Accelerate bit in the request is zero, `accelerating` is cleared to FALSE; otherwise it is set to TRUE.

To request the bus for fair or priority access, the link sends a FairReq or PriReq after the interface has been *Idle* for at least one clock. The expected response to a bus request is *Grant* on Ctl[0:1] which the PHY asserts after it has won arbitration. Under other circumstances, the PHY may cancel the bus request or retain it, pending the completion of other PHY activity (see table 5A-16). The link may reissue a cancelled request when the interface is subsequently *Idle*.

The cycle master link uses a priority request (PriReq) to send the cycle start packet. To request the bus to send isochronous data, the link issues an IsoReq while sending or receiving a cycle start or, during the same isochronous period, while sending or receiving an isochronous packet. The PHY cancels an isochronous request when a subaction gap or bus reset is observed

In order to meet timing requirements, a link may issue an isochronous request after observing *tcode* 8 in a putative cycle start packet but before verifying the CRC. If the CRC fails, the link shall not transmit isochronous packet(s) but shall cancel any isochronous request as soon as possible. It is not necessary for the link to issue an AccCtrl request with a zero Accelerate bit after the invalid CRC is detected.

To send an acknowledge, the link issues an ImmReq during or immediately after packet reception. This ensures that the ACK_RESPONSE_TIME requirement is met and that other nodes do not detect a subaction gap. After the packet ends, the PHY immediately takes control of Serial Bus and asserts *Grant* on Ctl[0:1]. If the packet header CRC passed, the link transmits an acknowledge. Otherwise, the link asserts *Idle* on Ctl[0:1] for three SClk cycles after observing *Grant* on Ctl[0:1].

NOTE—Although unlikely, more than one node may perceive (one correctly, the others mistakenly) that a packet is intended for it and issue an immediate request before checking the CRC. The PHYs of all such nodes would grab control of the bus immediately after the packet is complete. This condition would cause a temporary, localized collision of DATA_PREFIX somewhere between the PHYs intending to acknowledge while all the other PHYs on the bus would see DATA_PREFIX. This collision would appear as “ZZ” line state and would not be interpreted as a bus reset. The mistaken node(s) should drop their request(s) as soon as they check the CRC; the spurious “ZZ” line state would vanish. The only side effect of such a collision might be the loss of the intended acknowledge packet, which would be handled by the higher layer protocol.

A bus reset causes the PHY to cancel any pending bus request.

In response to register write requests, the PHY takes the value from the data field of the transfer and updates the addressed register. For register read requests, the PHY returns the contents of the addressed register at the next opportunity through a status transfer. If the status transfer is interrupted by a packet received or generated by the PHY, the PHY restarts the status transfer at the next opportunity.

Once the link issues a request for access to the bus, it shall not issue another bus request until the request is either granted or cancelled. The PHY shall ignore bus requests issued while a previous request is pending.

5A.3.1 LReq rules

In general, the link issues requests asynchronously with respect to activities on Serial Bus. However, certain requests are allowed only at specific times. Even when a request is issued at a valid time, Serial Bus activity may cause the PHY to cancel the request or to defer the request until the other activity has been completed. This subclause specifies when a link may issue a request and the corresponding PHY behavior; these rules permit the link to unambiguously determine the state—satisfied, cancelled, or deferred—of a request.

For the purpose of these rules, two specific cycles are defined—labeled C_A and C_B in figure 5A-6. The rules are specified in terms of the values of the Ctl[0:1] lines during these cycles. The sample point at which the link decides whether or not to initiate a request is C_A, which is one or more SClk cycles before C_B, the cycle in which the link sends the request's start bit. The disposition of the request is determined by the value of Ctl[0:1] from C_B onward.

General rules that govern link and PHY use of the request interface are as follows:

— The link shall not initiate a bus request (fair, priority, immediate, or isochronous) or use the *Hold* protocol to concatenate a packet until any outstanding request (whether the result of a bus request or the *Hold* protocol) has been granted or the link has been able to determine that it has been cancelled.
— The link should not issue a register read or write request when a previous register read request is outstanding. PHY behavior in this case is undefined.
— All pending bus requests (but not register read requests) are cancelled on a bus reset.

Additional rules for issuing a request are given in table 5A-15.

Table 5A-15 — Link rules to initiate a request on LReq

Request	Permitted when PHY has control of the interface and Ctl[0:1] at CA is	Permitted when link controls the interface	Additional requirements
Fair, Priority	Idle, Status	No	No fair or priority request shall be issued until any outstanding bus request completes.
Immediate	Receive, Idle	No	Sent after *destination_ID* decode during packet reception when the link is ready to transmit an acknowledge packet. The start bit of an immediate request shall be transmitted no later than the fourth cycle subsequent to that in which Ctl[0:1] went *Idle* following packet reception.
Isochronous	Any	Yes	Sent during an isochronous period when the link is ready to transmit an isochronous packet. The start bit of an isochronous request shall be transmitted no later than a) The eighth cycle subsequent to that in which Ctl[0:1] went from *Transmit* to *Idle*, or b) The fourth cycle subsequent to that in which Ctl[0:1] went from *Receive* to *Idle*. The link shall not issue an isochronous request if it intends to concatenate a packet after the current transmission.

Table 5A-15 — Link rules to initiate a request on LReq (continued)

Request	Permitted when PHY has control of the interface and Ctl[0:1] at CA is	Permitted when link controls the interface	Additional requirements
Register read, Register write	Any	Yes	Shall not be issued while there are pending register read requests.
AccCtrl	Any	Yes	Accelerate bit is zero. Issued by cycle slaves if `enab_accel` is TRUE and shall be issued once every isochronous period, as soon as possible after the local clock indicates the start of a new isochronous period. Accelerate bit is one. May only be issued by cycle slaves once every isochronous period, after a cycle start packet has been recognized and after all of the link's isochronous requests (if any).

In general, the PHY behavior varies dependent on whether another Serial Bus packet is detected before it has successfully completed processing the LReq.

The link may determine the PHY disposition of the LReq by monitoring the value of Ctl[0:1] from C_B onward, as shown in the table 5A-16.

Table 5A-16 — PHY disposition of link request

Request	Ctl[0:1] (at CB or later)	PHY behavior	Link action
Fair, Priority	Receive	If arbitration acceleration is enabled, and the packet transferred by the PHY is exactly 8 bits, then request is retained. Otherwise, request is discarded as soon as the PHY determines that the packet has other than 8 bits. Request is always discarded if arbitration acceleration is not enabled.	Continue to monitor for the next change on Ctl[0:1] if request is retained. Once the link starts shifting out the fair or priority LReq, both the link and the PHY monitor the Ctl[0:1] lines to determine if the LReq is to be cancelled. On every rising edge of SClk from C_B onwards, if *Receive* is asserted on Ctl[0:1] and enab_accel is FALSE, the request is cancelled. Otherwise, if arbitration accelerations are globally enabled for the PHY, the request is cancelled only if the packet transferred by the PHY is not 8 bits in length.
	Grant	Arbitration won.	Transmit packet.
	Idle, Status	Retain the request unless a bus reset was reported in the status.	Unless there was a bus reset, monitor Ctl[0:1] in anticipation of *Grant*.
Immediate	Grant	—	Transmit the acknowledge packet.
	Receive	PHY is still transferring a packet; the request is retained.	Continue to receive packet, then monitor Ctl[0:1] in anticipation of *Grant*.
	Idle, Status	Retain the request unless a bus reset was reported in the status.	Unless there was a bus reset, monitor Ctl[0:1] in anticipation of *Grant*.

Table 5A-16 — PHY disposition of link request (continued)

Request	Ctl[0:1] (at CB or later)	PHY behavior	Link action
Isochronous	Transmit, Idle (driven by link)	Request retained by PHY.	Monitor Ctl[0:1] after releasing the interface.
	Grant	Arbitration won.	Transmit packet.
	Receive	Request retained by PHY.	Monitor Ctl[0:1].
	Status	Request discarded if status indicates subaction gap (this is an error condition and should not occur). Otherwise, request is retained unless status reports a bus reset.	Continue to monitor for the next change on Ctl[0:1] if request is retained.
	Idle	—	Continue to monitor for the next change on Ctl[0:1].
Register read	Any (driven by link)	—	Wait until link releases the interface then monitor for the next change on Ctl[0:1].
	Grant	Request retained. Bus request LReq was previously issued, and now takes priority.	Service previous bus request, then monitor for next change on Ctl[0:1].
	Receive	Request retained.	Receive packet, then monitor for next change on Ctl [0:1].
	Status	Request is retained by the PHY until corresponding register data is returned.	If unrelated status is received or the desired status is interrupted, monitor Ctl[0:1] for desired status.
	Idle	—	Monitor Ctl[0:1].
Register write, Acceleration control	Any	Request completed.	—

NOTE—When a PHY autonomously generates packets, e.g., during the self-identify phase or in the transmission of a PHY response packet, it cancels all outstanding bus requests. This behavior is not described in table 5A-16 because, at such times, a link does not have any outstanding isochronous or priority requests.

In addition to the preceding rules for the use of the PHY/link interface, the timing constants, in units of SClk cycles, of table 5A-17 shall apply. Measurements of Serial Bus arbitration line states are taken at the cable connector while those of PHY/link interface states are taken at the PHY. The reference point for the latter is the first SClk edge that occurs while Ctl[0:1] are in the indicated state.

Table 5A-17 — PHY/link interface timing constants

Timing constant	Minimum	Maximum	Comment
BUS_TO_LINK_DELAY	2	9	Time from the start of RX_DATA_PREFIX to the assertion of *Receive* on Ctl[0:1].
DATA_PREFIX_TO_GRANT	—	25	When a node originates a packet, the time from the start of TX_DATA_PREFIX at the parent port (or if the root, any port) to the PHY's assertion of *Grant* on Ctl[0:1].
LINK_TO_BUS_DELAY	2	5	At the end of packet transmission by the link, the time from the assertion of *Idle* on Ctl[0:1] to the start of TX_DATA_END on all transmitting ports.
MAX_HOLD	—	47	Maximum time that the link may continuously assert *Hold* on Ctl[0:1] after observing *Grant*.

5A.3.2 Acceleration control

The ack-accelerated and fly-by arbitration enhancements specified in 4.4 can have adverse effects on the isochronous period if continuously enabled. Serial Bus relies upon the natural priority of the cycle master (root) to win arbitration and transmit the cycle start packet as soon as possible after cycle synchronization. Ack-accelerated arbitration or fly-by concatenation by node(s) other than the root can prolong asynchronous traffic on the bus indefinitely and disrupt isochronous operations.

The link avoids this problem by selectively disabling and enabling these arbitration enhancements. The acceleration control request permits the link to disable and enable ack-accelerated and fly-by arbitration enhancements, while leaving the other arbitration enhancements unaffected. The cycle master does not issue the acceleration control request.

The time period in which ack-accelerated and fly-by arbitration enhancements shall not be used extends from the time of the local cycle synchronization event until a cycle start packet is observed. During this period, the link at any node that is not the cycle master shall use the acceleration control request as follows:

a) If accelerations have been globally enabled, the link shall not make a fair or priority request unless an acceleration control request with a zero Accelerate bit has been transmitted since the most recent local cycle synchronization event.

b) The link shall not use the *Hold* protocol to concatenate an asynchronous primary packet after an acknowledge packet, except after its own *ack_pending* in order to complete a split transaction with a concatenated subaction.

c) Upon conclusion of this time period, the link may reenable ack-accelerated and fly-by acceleration by transmitting an acceleration control request whose Accelerate bit is set to one.

A link that makes one or more bus requests to transmit an isochronous packet need not use the acceleration control request to re-enable fly-by accelerations, since the isochronous request sets the `accelerating` variable to TRUE.

For a bus that does not have an active cycle master, it is not necessary to use the acceleration control request. So long as arbitration enhancements are enabled by *Enab_accel* in the PHY registers, the fly-by accelerations are also enabled by the default value of `accelerating` after a power reset.

5A.4 Status

When the PHY has status information to transfer to the link, it initiates a status transfer. The PHY waits until the interface is *Idle* before performing the transfer. The PHY initiates the transfer by asserting *Status* on Ctl[0:1] while simultaneously presenting the first 2 bits of status information on D[0:1]. The PHY continues to assert *Status* on Ctl[0:1] for the duration of the status transfer but may prematurely end the transfer by asserting something other than *Status* on Ctl[0:1]. This may be done in the event that a packet arrives before the status transfer completes. There shall be at least one *Idle* cycle in between consecutive status transfers.

The PHY sends 16 bits of status in the following two cases:

a) In response to a register read request, or

b) After a bus reset, to indicate the node’s new physical ID.

The latter is the only condition for which the PHY sends a register to the link without a corresponding register read request. In the case of event indications initiated by the PHY, 4 bits of status are sent to the link. The timing for a status transfer is shown in figure 5A-7.

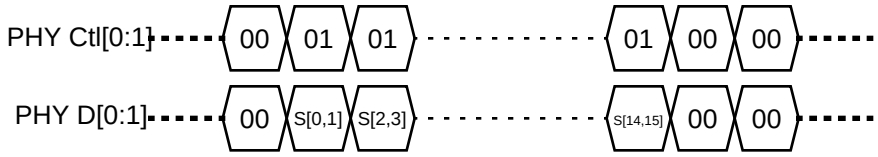


Figure 5A-7 — Status timing

The structure of the status data is specified by table 5A-18.

Table 5A-18 — Status bits

Bit(s)	Name	Description
0	ARB_RESET_GAP	The PHY has detected that Serial Bus has been *Idle* for an arbitration reset gap time.
1	SUBACTION_GAP	The PHY has detected that Serial Bus has been *Idle* for a subaction gap time.
2	BUS_RESET_START	The PHY has entered bus reset state.
3	INTERRUPT	This indicates one or more of the following interrupt conditions: — Loop detect interrupt — Cable power fail interrupt — Arbitration state machine time-out — Port event interrupt
4–7	Address	Register number
8–15	Data	Register contents

Upon successful completion of status transfer to the link, status bits S[0:3] shall be zeroed. The PHY shall also clear ARB_RESET_GAP and SUBACTION_GAP whenever it asserts *Grant* or *Receive* on Ctl[0:1], whether or not this status information has been successfully transferred to the link.

The PHY may truncate a status transfer by removing the status indication on Ctl[0:1]. In this event, the PHY shall set to zero whichever of the four status bits that have been successfully transferred to the link. That is, if only S[0:1] have been transferred, only S[0:1] shall be zeroed, while if S[0:3] have been transferred, all of S[0:3] shall be zeroed. The PHY shall reinitiate the status transfer at the earliest opportunity if either

a) At least one of the four status bits S[0:3] is nonzero, or

b) The truncated status transfer was intended to include PHY register data.

Status transfers shall commence with S[0:1] in all cases.

The PHY shall guarantee that neither subaction nor arbitration reset gap status information is lost because of a response to a register read request. During some period prior to the anticipated detection of a gap, it may be necessary for the PHY to defer completion of a register read request in order to avoid the loss of status information.

5A.5 Transmit

When the link requests access to Serial Bus through the LReq signal, the PHY arbitrates for access to Serial Bus. If the PHY wins the arbitration, it grants the bus to the link by asserting *Grant* on Ctl[0:1] for one SClk cycle, followed by *Idle* for one cycle. After observing *Grant* on Ctl[0:1], the link takes control of the interface by asserting *Idle*, *Hold*, or Transmit on Ctl[0:1] one cycle after sampling *Grant* from the PHY. The link should assert *Idle* for one cycle before changing the state of Ctl[0:1] to either *Hold* or *Transmit*, but shall not assert *Idle* for more than one cycle. PHY implementations shall tolerate *Idle* for one cycle prior to *Hold* or *Transmit*. The link asserts *Hold* to keep ownership of the bus while preparing data but shall not assert *Hold* on Ctl[0:1] for more than MAX_HOLD cycles subsequent to observing *Grant* on Ctl[0:1]. The PHY asserts DATA_PREFIX on Serial Bus during this time. When it is ready to begin transmitting a packet, the link asserts *Transmit* on Ctl[0:1] along with the first bits of the packet. After sending the last bits of the packet, the link asserts either *Idle* or *Hold* on Ctl[0:1] for one cycle, and then it asserts *Idle* for one additional cycle before placing those signals in a high-impedance state.

Whenever control of the bidirectional signals is transferred between the PHY and link, the device relinquishing control shall drive Ctl[0:1] and D[0:*n*] to logic zero levels for one clock before releasing the interface. This permits both devices to act upon registered versions of the interface signals while allowing the new owner a clock cycle in which to sample and respond. Note that when the link transfers control to the PHY without a *Hold* request, an additional clock with logic zero on the control and data signals is necessary so as not to place the signal lines in a high-impedance state before the PHY takes control.

An assertion of *Hold* after the last bits of a packet indicates to the PHY that the link needs to send another packet without releasing the bus. This function is used by the link to concatenate a packet after an acknowledge or to concatenate isochronous packets. With this assertion of *Hold*, the link simultaneously signals the speed of the next packet on the data lines, as encoded by table 5A-19. Once *Hold* is asserted, the PHY waits a MIN_PACKET_SEPARATION time and then asserts *Grant* as before. After observing *Grant* on Ctl[0:1], the link resumes control of the interface by asserting *Idle*, *Hold*, or *Transmit* on Ctl[0:1]. The link should assert *Idle* for one SClk cycle, but shall not assert *Idle* for more than one cycle before changing Ctl[0:1] to *Hold* or *Transmit*. In preparation for transmission of a concatenated packet, the link shall not assert *Hold* on Ctl[0:1] for more than MAX_HOLD cycles subsequent to observing *Grant* on Ctl[0:1].

The link may transmit concatenated packets at different speeds, with one exception—the link shall not concatenate an S100 packet after any packet of a higher speed. When the link wishes to send an S100 packet after any packet of a higher speed, it shall make a separate request.

If the multispeed capabilities of the PHY have not been enabled (see 5B.1), all concatenated packets shall be transmitted at the speed originally specified as part of the bus request. This requirement provides for backward compatibility when a PHY compliant with this specification is interfaced to a link that is not aware of the necessity to signal speed for each packet.

As noted above, when the link has finished sending the last packet, it releases the bus by asserting *Idle* on Ctl[0:1] for two SClk cycles. The PHY begins asserting *Idle* on Ctl[0:1] one cycle after sampling *Idle* from the link.

The timings for both a single and a concatenated packet transmit operation are illustrated in figure 5A-8. In the diagram, D0 through D_n are the data symbols of the packet, SP represents the speed code for the packet (encoded according to the values specified in table 5A-19), and ZZ represents high-impedance state. The link should assert the signals indicated by the shaded SClk cycles (this may be necessary in the presence of an isolation barrier).

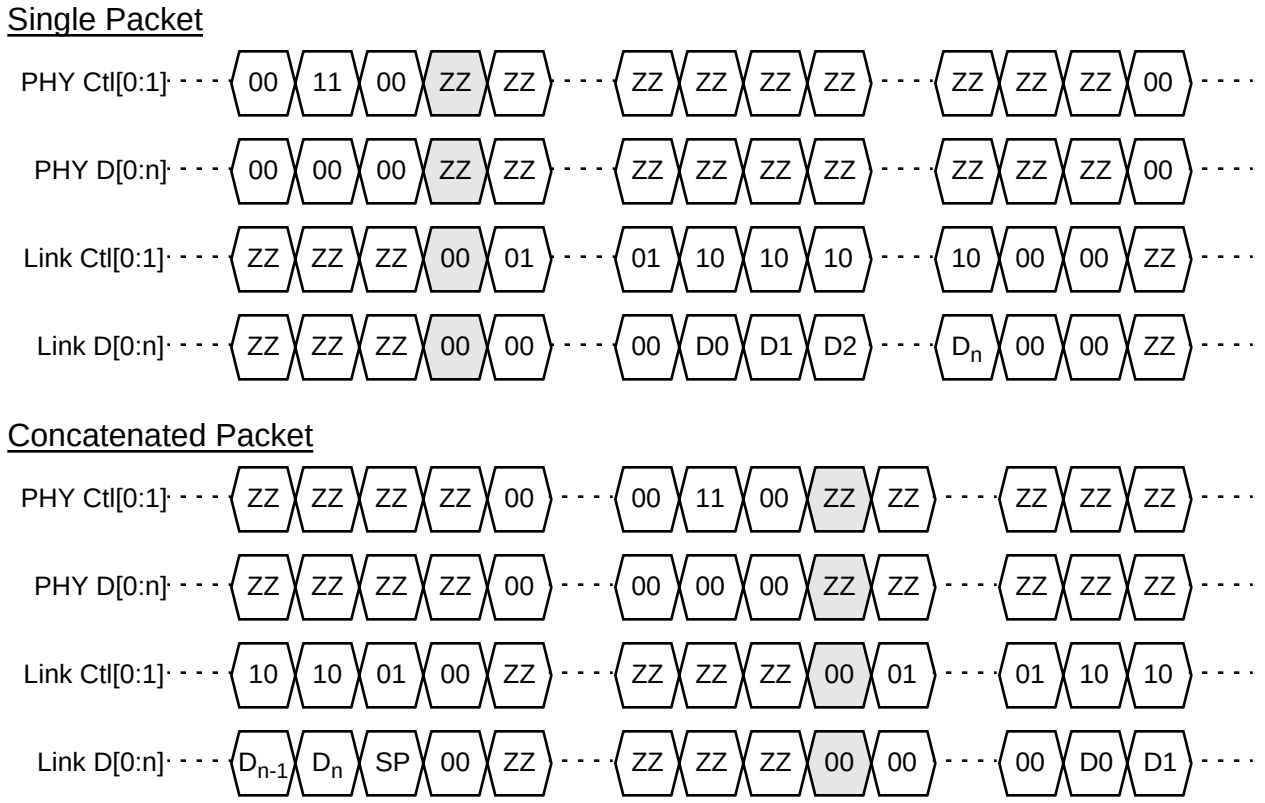


Figure 5A-8 — Transmit timing

NOTE—It is not required that the link assert *Hold* on Ctl[0:1] before sending a packet if the implementation permits the link to be ready to transmit as soon as bus ownership is granted.

5A.6 Cancel

The link may relinquish control of the PHY/link interface after a bus requested has been granted if data is not to be transmitted. This causes a null packet to appear on Serial Bus and effectively cancels the bus request. If the link cancels a request prior to the transmission of data, it shall use one of the two protocols described in the following paragraphs.

After observing *Grant* on Ctl[0:1], the link takes control of the interface by asserting *Idle*, *Hold*, or *Transmit* on Ctl[0:1] one cycle after sampling *Grant* from the PHY (as already described in 5A.5). If the link asserts *Idle* on Ctl[0:1], it shall continue to assert *Idle* for two additional cycles before placing those signals in a high-impedance state. This is illustrated by figure 5A-9.The PHY shall recognize the cancel on the second *Idle* cycle; the additional assertion of *Idle* by the link guarantees that Ctl[0:1] are not left in a high-impedance state before the PHY takes control of the interface.

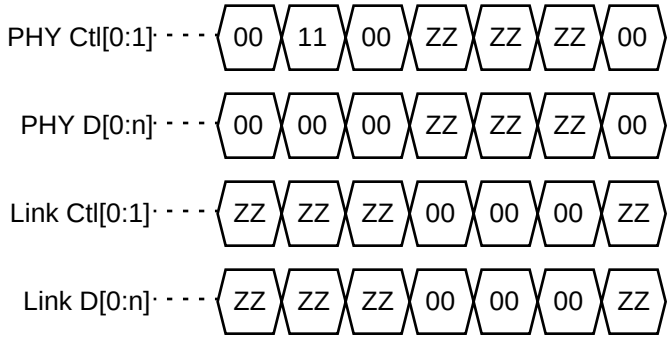


Figure 5A-9 — Link cancel timing (after Grant)

Otherwise, if the link had assumed or maintained control of the interface by asserting *Hold* on Ctl[0:1] and wishes to relinquish control of the interface, subsequent to the last *Hold* cycle it shall assert *Idle* on Ctl[0:1] for two cycles before placing those signals in a high-impedance state. This method may be used either after the initial *Grant* or to cancel the transmission of concatenated packet. The interface signaling is illustrated by figure 5A-10; the link should assert the signals indicated by the shaded SClk cycles (this may be necessary in the presence of an isolation barrier).

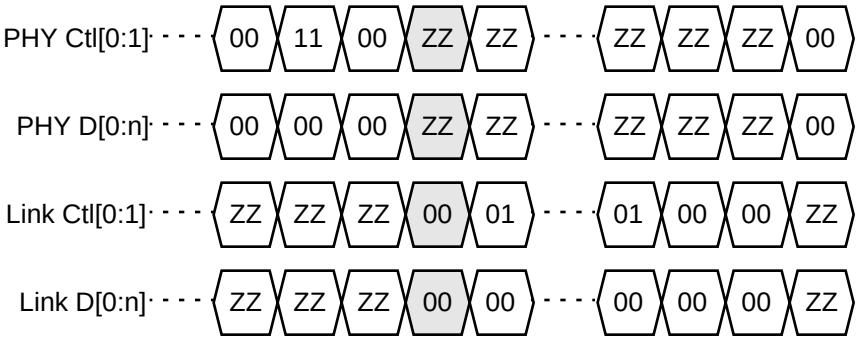


Figure 5A-10 — Link cancel timing (after Hold)

The PHY shall recognize the cancel on the first *Idle* cycle subsequent to *Hold*. The extra *Idle* cycle provided by the link avoids the problem of a high-impedance state on Ctl[0:1].

5A.7 Receive

Whenever the PHY sees data prefix on Serial Bus, it initiates a receive operation by asserting *Receive* on Ctl[0:1] and ones on D[0:*n*]. The PHY indicates the start of a packet by placing the speed code (encoding shown in table 5A-19) on D[0:*n*], followed by the contents of the packet. The PHY holds Ctl[0:1] in *Receive* until the last symbol of the packet has been transferred. The PHY indicates the end of the packet by asserting *Idle* on Ctl[0:1]. Note that signaling the speed code is a PHY/link protocol and not a data symbol to be included in the calculation of the CRC.

It is possible that a PHY can see data prefix appear and then disappear on Serial Bus without seeing a packet. This is the case when a packet of a higher speed than the PHY can receive is being transmitted. In this case, the PHY ends the packet by asserting *Idle* when data prefix goes away.

If the PHY is capable of a higher data rate than the link, the link detects the speed code as such and ignores the packet until it sees the *Idle* state again.

The timing for the receive operation is shown in figure 5A-11. In the diagram, SP refers to the speed code and D0 through D_n are the data symbols of the packet.

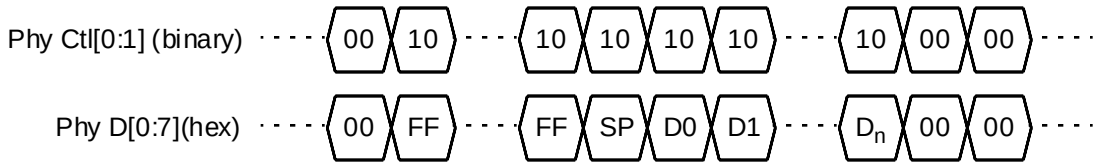


Figure 5A-11 — Receive timing

The speed code for the receive operation is defined as shown in table 5A-19. This is also the same speed encoding used by the link to signal speed to the PHY during concatenated packet transmission.

Table 5A-19 — Speed code signaling

D[0:*n*]		Data rate
Transmitted	Observed	
00000000_2	$00xxxxxx_2$[a]	S100
01000000_2	$0100xxxx_2$	S200
01010000_2	01010000_2	S400
01010001_2	01010001_2	S800
01010010_2	01010010_2	S1600
01010011_2	01010011_2	S3200
11111111_2	$11xxxxxx_2$	Data prefix indication

[a]An “x” indicates ignored on received.

NOTE—The speed code is only applicable for cable applications. For backplane applications, the speed code is set to $00xxxxxx_2$.

5A.8 Electrical characteristics (cable environment)

This subclause specifies the signal and timing characteristics of the interface between a discrete PHY and a link.

5A.8.1 DC signal levels and waveforms

DC parametric attributes of the PHY/link interface signals are specified by table 5A-20. Input levels may be greater than the power supply level (e.g., a 5 V output driving the undifferentiated output high voltage [V_{OH}] into a 3.3 V input); tolerance of mismatched input levels is optional. Devices not tolerant of mismatched input levels, but which otherwise meet the requirements in table 5A-20, are compliant with this standard. V_{DD} is obtained from the vendor’s specifications.

Table 5A-20 — DC specifications for PHY/link interface

Name	Description	Conditions	Unit	Minimum	Maximum
V_{OH}	Output high voltage (undifferentiated)	I_{OH} = –4 mA	V	2.8	—
V_{OHD}	Output high voltage (differentiated)	I_{OH} = –9 mA at V_{DD} = 3 V I_{OH} = –11 mA at V_{DD} = 4.5 V	V	V_{DD} – 0.4	—
V_{OL}	Output low voltage (undifferentiated)	I_{OL} = 4 mA	V	—	0.4
V_{OLD}	Output low voltage (differentiated)	I_{OL} = 9 mA at V_{DD} = 3 V I_{OL} = 11 mA at V_{DD} = 4.5 V	V	—	0.4
V_{IH}	Input high voltage (undifferentiated)	—	V	2.6	V_{DD}[a]+10%
V_{IL}	Input low voltage (undifferentiated)	—	V	—	0.7
V_{LIT+}	Input rising threshold (LinkOn and LPS)	—	V	—	V_{LREF} + 1[b]
$V_{LIT–}$	Input falling threshold (LinkOn and LPS)	—	V	V_{LREF} + 0.2[b]	—
V_{IT+}	Hysteresis input rising threshold (differentiated)[c]	—	V	V_{REF} + 0.3	V_{REF} + 0.9[d]
$V_{IT–}$	Hysteresis input falling threshold (differentiated)[c]	—	V	V_{REF} – 0.9[c]	V_{REF} – 0.3
V_{REF}	Reference voltage[e]	—	V	(V_{DD}/2) ± 1%	
V_{LREF}	Reference voltage[f] (LinkOn and LPS inputs)	—	V	0.5	1.6
C_{IN}	Input capacitance	—	pF	—	7.5

[a]Refers to driving device's power supply.

[b]The LinkOn and LPS receiver parameters are based on a swing of 2.4 V for the received signal. Links that only depend on receiving the initial edge of LinkOn may be capable of operating with less constrained values.

[c]When the PHY/link interface is in differentiated mode, the SClk input shall meet the V_{IT+} and $V_{IT–}$ requirements.

[d]When designing a device capable of both undifferentiated and differentiated operation, V_{IH} and V_{IL} effectively constrain these V_{IT+} and $V_{IT–}$ values to V_{REF} + 0.8 V and V_{REF} – 0.8 V, respectively.

[e]For some applications, a device can be compliant with these dc specifications even if a different V_{REF} is chosen.

[f]For a particular application, there is a single value for each device's nominal bias point, V_{LREF}, which shall be within the range specified. V_{LREF} should be chosen in conjunction with the receiver parameters so that a loss of power by the transmitting device is perceived as zero by the receiving device.

5A.8.2 AC timing

The rise and fall time measurement definitions, t_R and t_F, for SClk, Ctl[0:1], D[0:*n*], and LReq are shown in figure 5A-12.

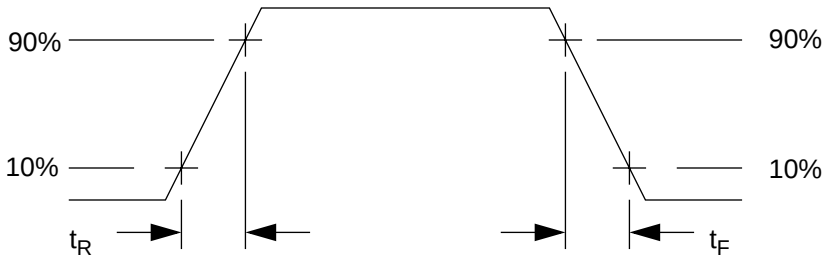


Figure 5A-12 — Signal levels for rise and fall times

Other signal characteristics of the PHY/link interface are specified by table 5A-21. If an isolation barrier is implemented it shall cause neither delay nor skew in excess of the values specified. AC measurements shall be taken from the 1.575 V level of SClk to the input or output Ctl[0:1], D[0:*n*], or LReq levels and shall assume an output load of 10 pF.

Table 5A-21 — AC timing parameters

Name	Description	Unit	Minimum	Maximum
	SClk frequency	MHz	49.152 ± 100 ppm	
	SClk duty cycle	%	45	55
t_R	Rise time	ns	0.7	2.4
t_F	Fall time	ns	0.7	2.4
idel	Delay through isolation barrier	ns	0	2
	Skew through isolation barrier	ns	0	0.5
	Isolation barrier recovery time	µs	0	10

Figure 5A-13 and figure 5A-14 illustrate the transfer waveforms as observed at the PHY. A PHY shall implement values for tpd1, tpd2, and tpd3 within the limits specified in table 5A-22 and shall not depend upon values for tpsu and tph greater than the minimums specified.

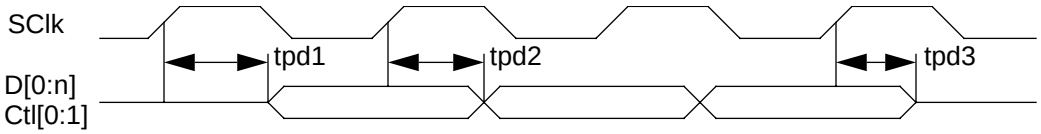


Figure 5A-13 — PHY to link transfer waveform at the PHY

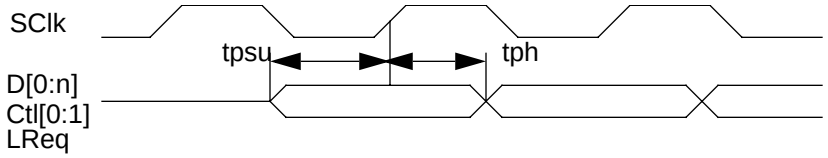


Figure 5A-14 — Link to PHY transfer waveform at the PHY

The values for the timing parameters illustrated in figure 5A-13 and figure 5A-14 are specified in table 5A-22.

Table 5A-22 — AC timing parameters at the PHY

Name	Description	Unit	Minimum	Maximum
tpd1	Delay time, SClk input high to initial instance of D[0:*n*] and Ctl[0:1] outputs valid	ns	0.5	13.5
tpd2	Delay time, SClk input high to subsequent instance(s) of D[0:*n*] and Ctl[0:1] outputs valid	ns	0.5	13.5
tpd3	Delay time, SClk input high to D[0:*n*]] and Ctl[0:1] invalid (high-impedance)	ns	0.5	13.5
tpsu	Setup time D[0:*n*], Ctl[0:1] and LReq inputs before SClk	ns	6	—
tph	Hold time D[0:*n*], Ctl[0:1] and LReq inputs after SClk	ns	0	—

Figure 5A-15 and figure 5A-16 illustrate the transfer waveforms as observed at the link. A link shall implement values for tld1, tld2, and tld3 within the limits specified in table 5A-23 and shall not depend upon values for tlsu and tlh greater than the minimums specified.

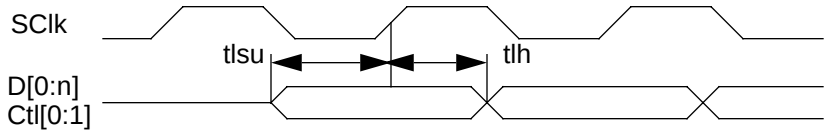


Figure 5A-15 — PHY to link transfer waveform at the link

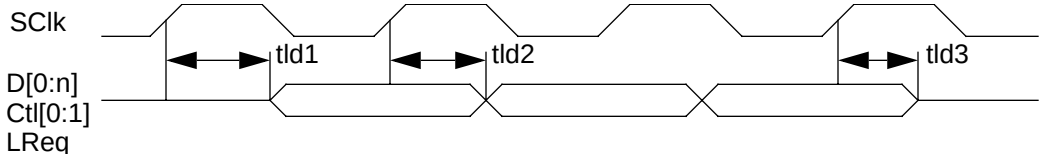


Figure 5A-16 — Link to PHY transfer waveform at the link

The values for the timing parameters illustrated in figure 5A-15 and figure 5A-16 are specified in table 5A-23.

Table 5A-23 — AC timing parameters at the link

Name	Description	Unit	Minimum	Maximum
tld1	Delay time, SClk input high to initial instance of D[0:*n*], Ctl[0:1], and LReq outputs valid	ns	1	10
tld2	Delay time, SClk input high to subsequent instance(s) of D[0:*n*], Ctl[0:1], and LReq outputs valid	ns	1	10
tld3	Delay time, SClk input high to D[0:*n*], Ctl[0:1], and LReq invalid (high-impedance)	ns	1	10
tlsu	Setup time, D[0:*n*] and Ctl[0:1] inputs before SClk	ns	6	—
tlh	Hold time, D[0:*n*] and Ctl[0:1] inputs after SClk	ns	0	—

5A.8.3 AC timing (informative)

The protocol of this interface is designed such that all inputs and outputs at this interface can be registered immediately before or after the I/O pad and buffer. No state transitions need be made that depend directly on the chip inputs; chip outputs can come directly from registers without combinational delay or additional loading. This configuration provides generous margins on setup and hold time.

In the direction from the PHY to the link, timing follows normal source-clocked signal conventions. A 0.5 ns allowance is made for skew through an (optional) isolation barrier.

In the direction from the link to the PHY, the data is timed at the PHY in reference to SClk, whose frequency allows a nominal budget of 20 ns for delay, inclusive of the PHY input setup time. Possible sources of delay are an isolation barrier or internal SClk delay at the link caused by a clock tree. Figure 5A-17 illustrates the relationship among these delays. Note that the maximum round-trip delay of 14 ns (calculated as $tdrt1_{max} = idel_{max} + tld1_{max} + idel_{max}$) provides generous delays for both the link and the PHY. The link, after the receipt of SClk, has 10 ns to assert valid data, while at the PHY, the minimum input setup for the subsequent SClk cycle is 6 ns (calculated as $tpsu_{min} = 20\ ns - tdrt1_{max}$). Also note that the minimum round-trip delay until the next change in data of 21 ns (calculated as $tdrt2_{min} = 20\ ns + idel_{min} + tld2_{min} + idel_{min}$) limits the hold time at the PHY to 1 ns (calculated as $tph_{min} = tdrt2_{min} - 20\ ns$); the hold time is further reduced to zero to provide a guard band of 1 ns.

The values for the delays illustrated in figure 5A-17 are given in table 5A-24.

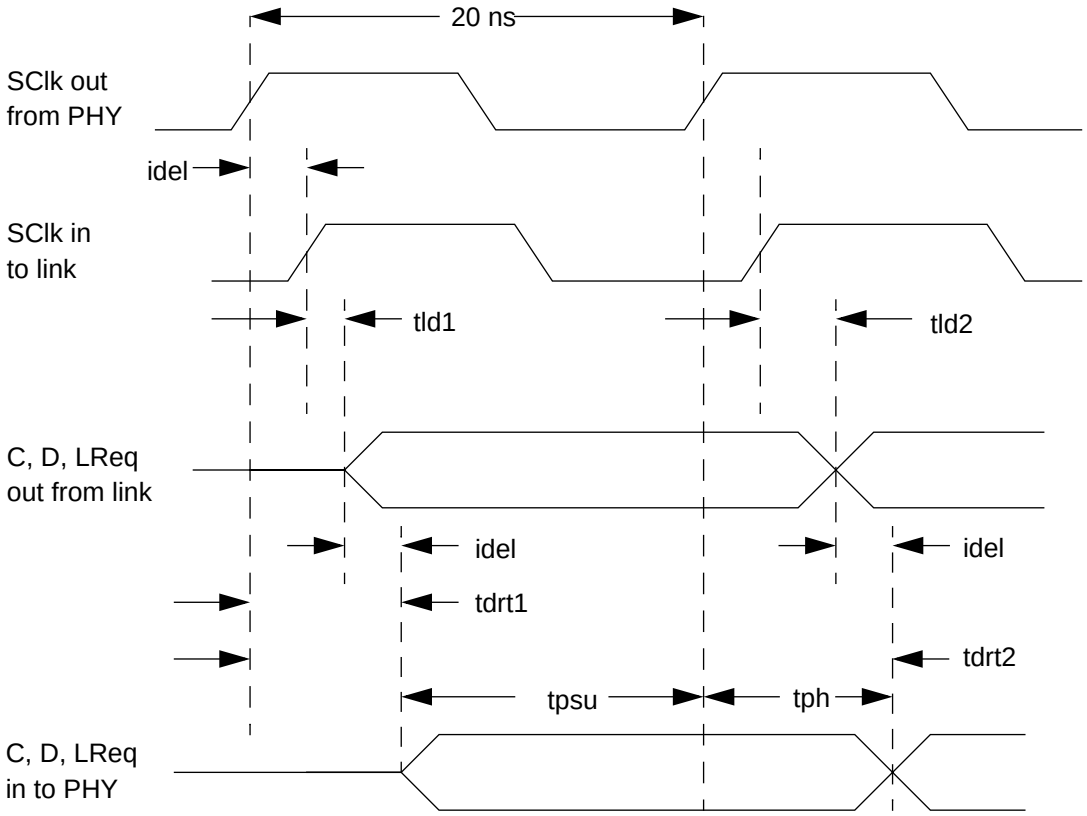


Figure 5A-17 — Link to PHY delay timing

Table 5A-24 — Link to PHY delay timing parameters

Name	Description	Unit	Minimum	Maximum
tdrt1	Round-trip delay from SClk output at the PHY to valid Ctl[0:1], D:[0:7] and LReq at the PHY	ns	1	14
tdrt2	Round-trip delay from SClk output at the PHY to changed or invalid Ctl[0:1], D:[0:7] and LReq at the PHY	ns	21	34

5A.8.4 Isolation barrier (informative)

The example circuits shown in this subclause demonstrate how to achieve galvanic isolation between a discrete PHY and link by means of a capacitive isolation barrier. For applications that require isolation, other methods may be used. When capacitive isolation is used between the PHY and the link, the grounds of both devices must be coupled as shown in figure 5A-18. The details of this ground coupling are omitted from figure 5A-19 through figure 5A-23.

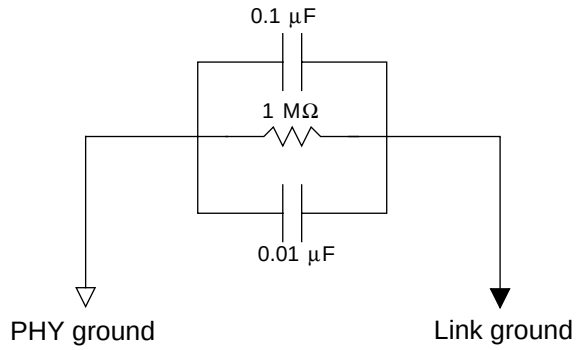


Figure 5A-18 — Ground coupling circuit example

The example circuits in figure 5A-19 through figure 5A-23 illustrate different requirements of the various signals of the PHY/link interface.

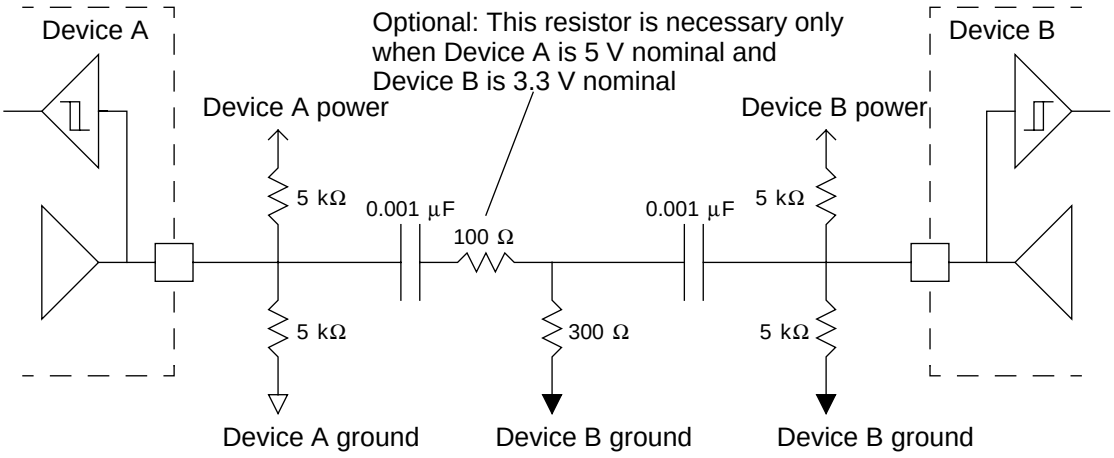
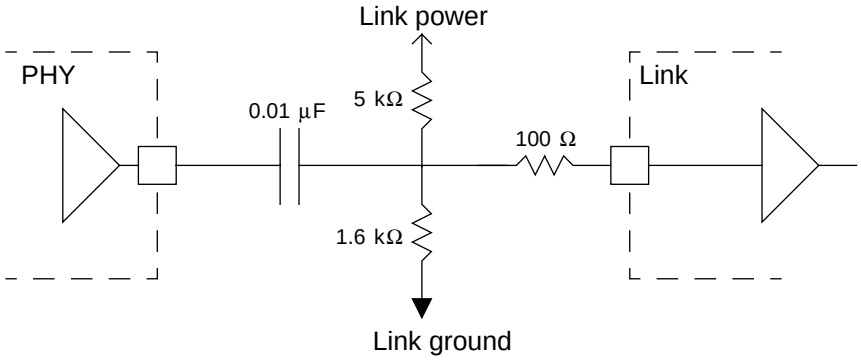
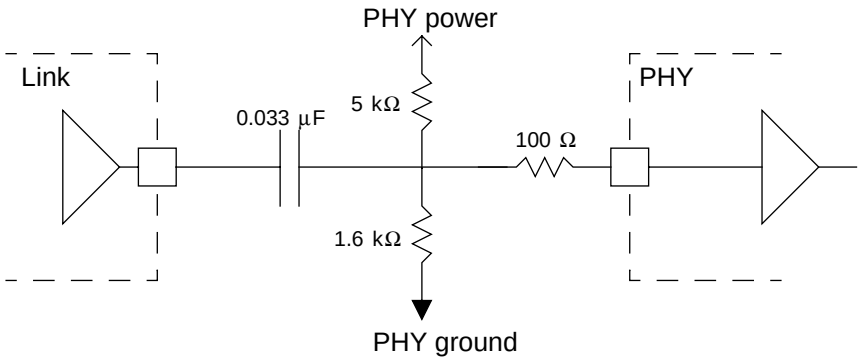


Figure 5A-19 — Capacitive isolation barrier circuit example for Ctl[0:1] and D[0:n]



NOTE—The values of the resistors between signal and ground or signal and power should be chosen to suit the implemented value of V_{LREF}. The values shown are appropriate when V_{LREF} is nominally 0.8 V.

Figure 5A-20 — Capacitive isolation barrier circuit example for LinkOn



NOTE—The values of the resistors between signal and ground or signal and power should be chosen to suit the implemented value of V_{LREF}. The values shown are appropriate when V_{LREF} is nominally 0.8 V.

Figure 5A-21 — Capacitive isolation barrier circuit example for LPS

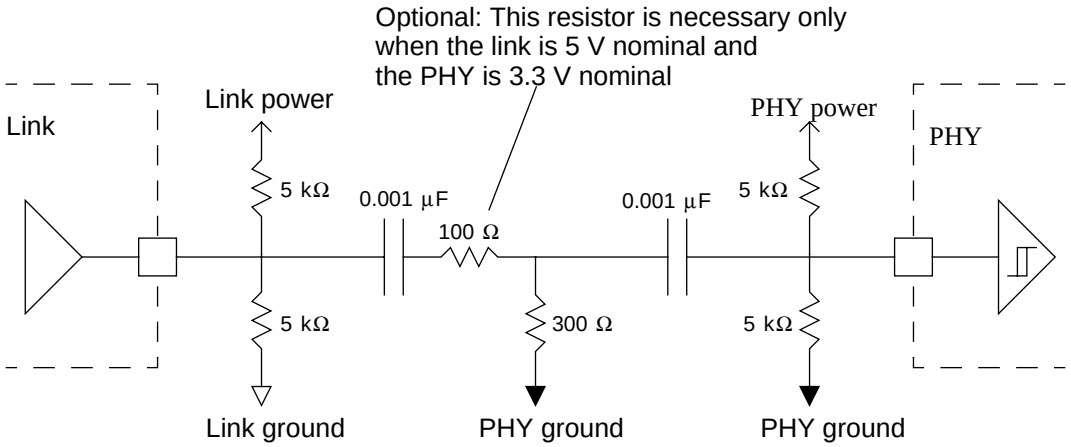


Figure 5A-22 — Capacitive isolation barrier circuit example for LReq

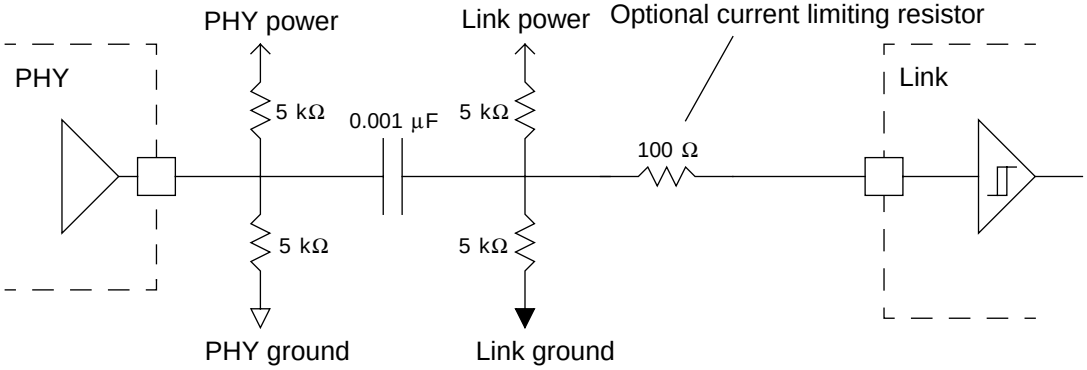


Figure 5A-23 — Capacitive isolation barrier circuit example for SClk

Insert the following after clause 5A of IEEE Std 1394a-2000.

5B. PHY register map

Although annex J in IEEE Std 1394-1995, from which this annex is derived, originally described an interface to a discrete PHY, the material in this clause is normative for both discrete and integrated PHY and link implementations. In addition, link implementations shall provide a means for software or firmware to access the PHY registers defined in this clause.

5B.1 PHY register map (cable environment)

In the cable environment, the extended PHY register map illustrated by figure 5B-1 shall be implemented by all designs compliant with this standard. Reserved fields are shown shaded in gray.

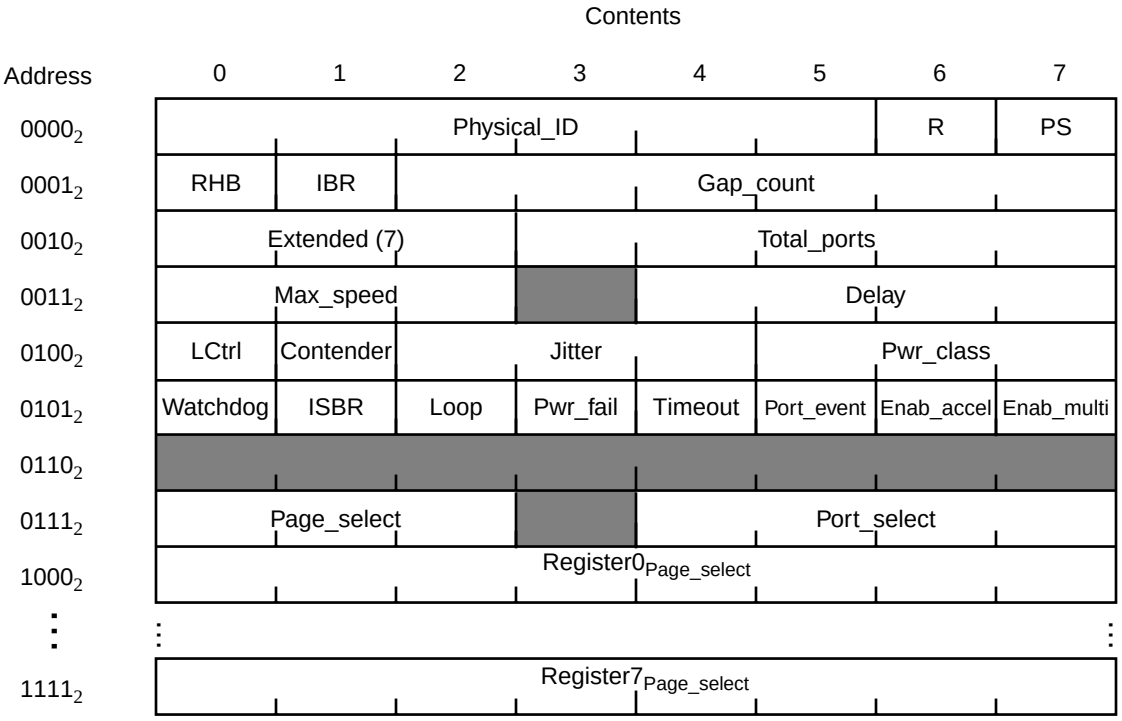


Figure 5B-1 — Extended PHY register map for the cable environment

The meaning, encoding, and usage of all the fields in the extended PHY register map are summarized by table 5B-1. Power reset values not specified, as well as all bus reset values, are resolved by the operation of the PHY state machines subsequent to the reset (see 4.4.3).

Table 5B-1 — PHY register fields for the cable environment

Field	Size	Type	Power reset value	Description
Physical_ID	6	r	—	The address of this node is determined during self-identification. A value of 63 indicates a malconfigured bus; the link shall not transmit any packets.
R	1	r	—	When set to one, indicates that this node is the root.
PS	1	r	—	Cable power active (see 4.2.2.7).
RHB	1	rw	0	Root hold-off bit. When one, the force_root variable is TRUE, which instructs the PHY to attempt to become the root during the next tree identify process. The PHY sets this bit to zero if it determines itself to be an isolated node.
IBR	1	rw	0	Initiate bus reset. When set to one, instructs the PHY to set `ibr` TRUE and `reset_time` to `RESET_TIME`. These values in turn cause the PHY to initiate a bus reset without arbitration; the reset signal is asserted for 166 µs. This bit is self-clearing.
Gap_count	6	rw	$3F_{16}$	Used to derive arbitration gap detection times to optimize performance according to the topology of the bus. See 4.3.6 for the encoding of this field.
Extended	3	r	7	This field shall have a constant value of seven, which indicates the extended PHY register map.
Total_ports	5	r	vendor-dependent	The number of ports implemented by this PHY.
Max_speed	3	r	vendor-dependent	Indicates the speed(s) this PHY supports 000_2 S100 001_2 S100 and S200 010_2 S100, S200, and S400 011_2 S100, S200, S400, and S800 100_2 S100, S200, S400, S800, and S1600 101_2 S100, S200, S400, S800, S1600, and S3200 All other values are reserved for future definition.
Delay	4	r	vendor-dependent	Worst-case repeater delay, measured from the receipt of the first data bit to its retransmission by the repeating port(s), expressed as $(0.144 + 2 \times Delay / \text{BASE\_RATE})$ µs.
LCtrl	1	rw	See description	Cleared or set by software to control the value of the L bit transmitted in the node's self-ID packet 0, which shall be the logical AND of this bit and LPS active. If hardware implementation-dependent means are not available to configure the power reset value of the *LCtrl* bit, the power reset value shall be one.
Contender	1	rw	See description	Cleared or set by software to control the value of the C bit transmitted in the node's self-ID packet. If hardware implementation-dependent means are not available to configure the power reset value of this bit, the power reset value shall be zero.
Jitter	3	r	vendor-dependent	The maximum variance of either arbitration or data repeat delay, expressed as $\lceil 2 \times (Jitter + 1) / \text{BASE\_RATE} \rceil$ µs.
Pwr_class	3	rw	vendor-dependent	Power class. Controls the value of the *pwr* field transmitted in the self-ID packet. See 4.3.4.1 for the encoding of this field.
Watchdog	1	rw	0	Watchdog enable. Controls whether or not loop, power fail, and time-out interrupts are indicated to the link when the PHY/link interface is not operational. Also determines whether or not interrupts are indicated to the link when resume operations commence for any port (regardless of the value of *Int_enable* for the port).

Table 5B-1 — PHY register fields for the cable environment (continued)

Field	Size	Type	Power reset value	Description
ISBR	1	rw	0	Initiate short (arbitrated) bus reset. A write of one to this bit instructs the PHY to set `isbr` TRUE and `reset_time` to `SHORT_RESET_TIME`. These values in turn cause the PHY to arbitrate and issue a short bus reset. This bit is self-clearing.
Loop	1	rw	0	Loop detect. A write of one to this bit clears it to zero.
Pwr_fail	1	rw	1	Cable power failure detect. Set to one when the PS bit changes from one to zero or upon a PHY power reset. A write of one to this bit clears it to zero.
Timeout	1	rw	0	Arbitration state machine time-out (see `MAX_ARB_STATE_TIME`). A write of one to this bit clears it to zero.
Port_event	1	rw	0	Port event detect. The PHY sets this bit to one if any of *Bias* (unless the port is disabled), *Connected, Disabled,* or *Fault* change for a port whose *Int_enable* bit is one. The PHY also sets this bit to one if resume operations commence for any port and *Watchdog* is one. A write of one to this bit clears it to zero.
Enab_accel	1	rw	See footnote[a]	Enable arbitration acceleration. When set to one, the PHY shall use the enhancements specified in 4.4. PHY behavior is unspecified if the value of *Enab_accel* is changed while a bus request is pending.
Enab_multi	1	rw	See footnote[a]	Enable multi-speed-packet concatenation. When set to one, the link shall signal the speed of all packets to the PHY.
Page_select	3	rw	vendor-dependent	Selects which of eight possible PHY register pages are accessible through the window at PHY register addresses 1000_2 through 1111_2, inclusive.
Port_select	4	rw	vendor-dependent	If the page selected by *Page_select* presents *per* port information, this field selects which port's registers are accessible through the window at PHY register addresses 1000_2 through 1111_2, inclusive. Ports are numbered monotonically starting at zero, p0.

[a]For a discrete PHY, the power reset value of these bits shall be zero unless hardware implementation-dependent means are available to configure the power reset value. Integrated PHY/link implementations may implement either bit as read-only, in which case the power reset value shall be one.

Because a write to the PHY register at address 0001_2 sets the PHY's `gap_count_reset_disable` variable TRUE whether or not the values of *RHB, IBR,* or *Gap_count* are altered, a write that sets *Gap_count* to any value other than 63 shall either set *IBR* to one in the same write or, as soon as possible thereafter, set *ISBR* to one in order to initiate a bus reset; see 8.4.6.2 for details.

The *RHB* bit should be zero unless it is necessary to establish a particular node as the root; see 8.4.2.6A for details.

The *Watchdog* bit serves two purposes. One is to determine whether or not an interrupt condition shall be indicated to the link. The other is to create an interrupt condition whenever resume operations commence for any port. An interrupt condition is created either when any of the *Loop, Pwr_fail, Timeout,* or *Port_event* bits transition from zero to one or when the PHY detects the absence of LPS and any of these same bits are one. Whether or not an interrupt condition shall be indicated to the link is determined by table 5B-2.

The second function of *Watchdog* affects the creation of an interrupt condition. If this bit is one, *Port_event* shall be set to one and an interrupt condition shall be created whenever resume operations commence for any port.

Once an interrupt condition has been created, the method used to indicate it to the link depends upon the operational state of the PHY/link interface. When the interface is operational, INTERRUPT is reported as S[3] in a PHY status transfer; otherwise the INTERRUPT event causes the assertion of the LinkOn signal (see 5A.2 for details).

Table 5B-2 — PH_EVENT.indication(INTERRUPT) sources

Interrupt source	LPS	Watchdog	Action
Port_event	—	—	INTERRUPT
Loop Pwr_fail Timeout	0	0	—[a]
		1	INTERRUPT
	1	—	INTERRUPT

[a]Although the existence of the interrupt condition is not indicated to the link, the PHY register bits that describe the interrupt condition remain set until cleared by a PHY register write.

The *Loop*, *Pwr_fail*, *Timeout*, and *Port_event* bits are unaffected by PHY register writes if the corresponding bit position is zero. When the bit written to the PHY register is one, the corresponding bit is set to zero.

The upper half of the PHY register space, addresses 1000_2 through 1111_2, inclusive, provides a windows through which additional pages of PHY registers may be accessed. This standard defines pages zero, one, and seven—the Port Status page, the Vendor Identification page, and a vendor-dependent page. Other pages are reserved.

The Port Status page is used to access configuration and status information for each of the PHY's ports. The port is selected by writing zero to *Page_select* and the desired port number to *Port_select* in the PHY register at address 0111_2. If the value of *Port_select* specifies an unimplemented port, all of the registers of the Port Status page shall be reserved. The format of the Port Status page is illustrated by figure 5B-2; reserved fields are shown shaded in gray.

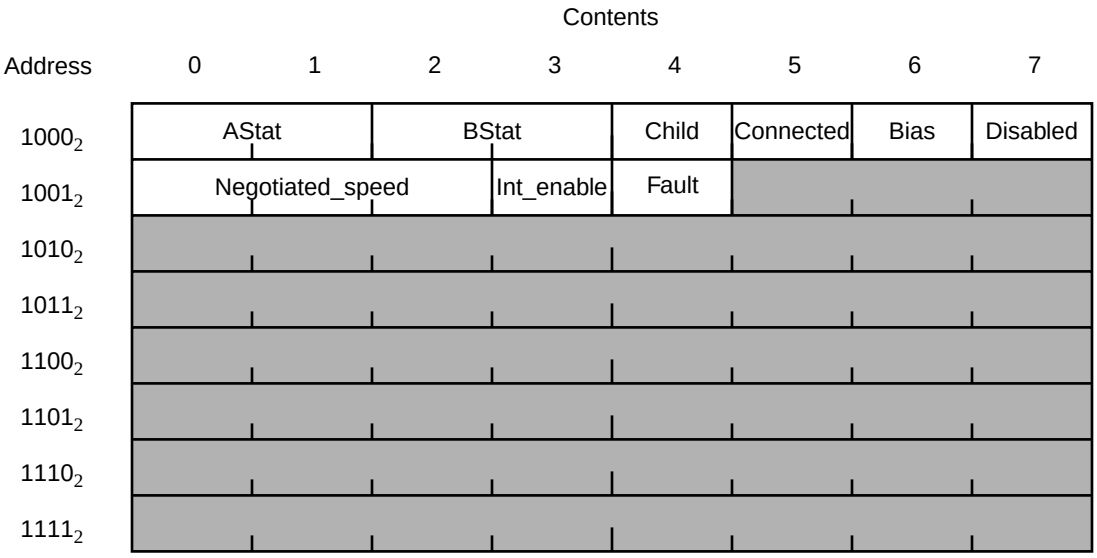


Figure 5B-2 — PHY register page 0: Port Status page

The meanings of the register fields with the Port Status page are defined by table 5B-3.

Table 5B-3 — PHY register Port Status page fields

Field	Size	Type	Power reset value	Description
AStat	2	r	—	TPA line state for the port 00_2 = invalid 01_2 = 1 10_2 = 0 11_2 = Z
BStat	2	r	—	TPB line state for the port (same encoding as *AStat*).
Child	1	r	—	If equal to one, the port is a child. Otherwise, it is a parent. The meaning of this bit is undefined from the time a bus reset is detected until the PHY transitions to state T1: Child Handshake during the tree identify process (see 4.4.2.2).
Connected	1	r	0	If equal to one, the port is connected.
Bias	1	r	—	If equal to one, incoming TpBias is detected.
Disabled	1	rw	See description	If equal to one, the port is disabled. The value of this bit subsequent to a power reset is implementation-dependent. If a hardware configuration option is provided, a single option may control the power reset value for all ports.[a]
Negotiated_speed	3	r	—	Indicates the maximum speed negotiated between this PHY port and its immediately connected port; the encoding is the same as for the PHY register *Max_speed* field.
Int_enable	1	rw	0	Enable port event interrupts. When set to one, the PHY shall set *Port_event* to one if any of this port's *Bias* (unless the port is disabled), *Connected*, *Disabled*, or *Fault* bits change state.
Fault	1	rw	0	Set to one if an error is detected during a suspend or resume operation. A write of one to this bit or receipt of the appropriate remote command packet (see 4.3.4.4.4) shall clear it to zero. When this bit is set to zero, both resume and suspend errors are cleared.

[a]The *Disabled* bit may be set to one by a direct access to the PHY registers via the PHY/link interface or indirectly via a PHY remote command packet. When a remote command packet is used to disable an active port, a bus reset is initiated on all of the PHY's active ports. When *Disabled* is set to one via the PHY/link interface, a bus reset is not necessarily initiated by the PHY, and may need to be generated by the local link. As a consequence, PHY remote command packets should be used to control the *Disabled* bit.

The *AStat* and *BStat* fields and *Child* bit are present in both the legacy and extended PHY registers and have identical meanings, defined by table 5B-3, in both cases. The *Connected* field replaces the legacy PHY *Con* field and has a slightly altered meaning. For a legacy PHY, *Con* indicates a connected and active PHY port while *Connected* indicates a port with a physical connection to a peer PHY, but the port is not necessarily active.

The Vendor Identification page is used to identify the PHY's vendor and compliance level. The page is selected by writing one to *Page_select* in the PHY register at address 0111_2. The format of the Vendor Identification page is illustrated by figure 5B-3 below; reserved fields are shown shaded in gray.

The meanings of the register fields within the Vendor Identification page are defined in table 5B-4.

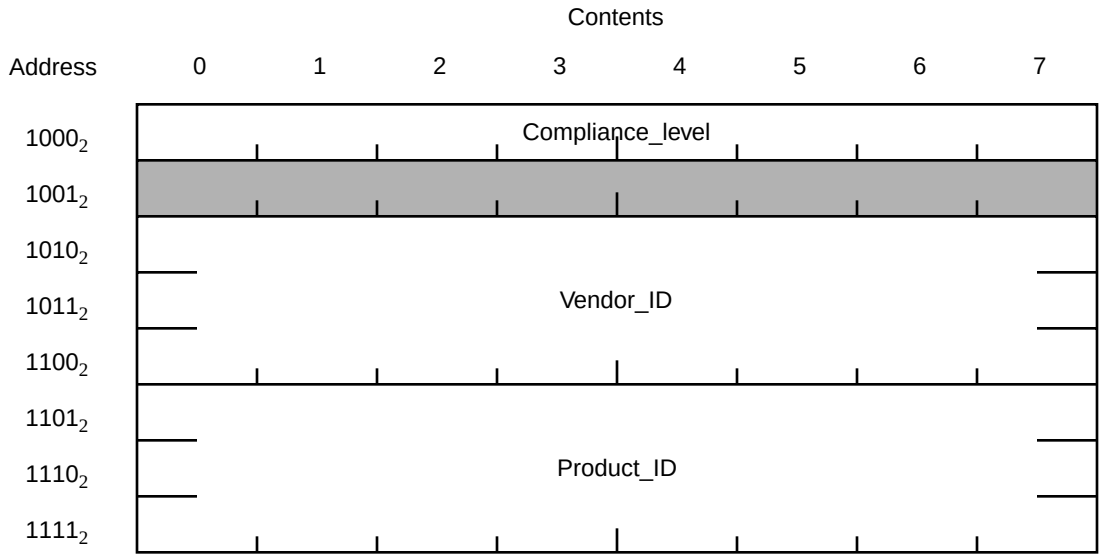


Figure 5B-3 — PHY register page 1: Vendor Identification page

Table 5B-4 — PHY register Vendor Identification page fields

Field	Size	Type	Description
Compliance_level	8	r	Standard to which the PHY implementation complies 0 = not specified 1 = IEEE P1394a All other values reserved for future standardization
Vendor_ID	24	r	The company ID or Organizationally Unique Identifier (OUI) of the manufacturer of the PHY. This number is obtained from the IEEE Registration Authority (RA). The most significant byte of *Vendor_ID* appears at PHY register location 1010_2 and the least significant at 1100_2.
Product_ID	24	r	The meaning of this number is determined by the company or organization that has been granted *Vendor_ID*. The most significant byte of *Product_ID* appears at PHY register location 1101_2 and the least significant at 1111_2.

The vendor-dependent page provides registers set aside for use by the PHY's vendor. The page is selected by writing seven to *Page_select* in the PHY register at address 0111_2. The PHY vendor shall determine the meaning of *Port_select* when *Page_select* equals seven. PHY vendors may implement and specify the format of up to eight vendor-dependent registers, at addresses 1000_2 through 1111_2, inclusive.

5B.2 PHY register map (backplane environment)

The PHY register map for the backplane environment is related to that of the cable environment; some fields are not present, while other fields changeable in the cable environment have a fixed value in the backplane environment, and vice versa. The PHY register map for the backplane environment is illustrated by figure 5B-4; reserved fields are shown shaded in gray.

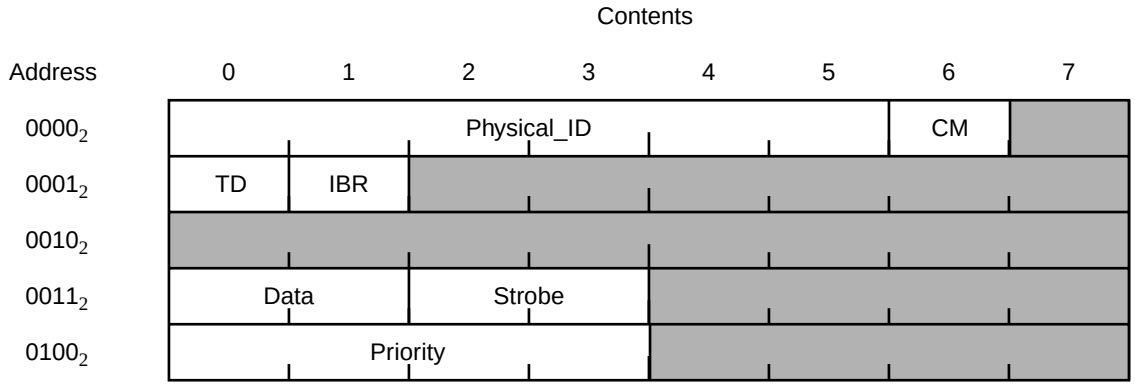


Figure 5B-4 — PHY register map for the backplane environment

The meaning, encoding, and usage of all the fields in the backplane PHY register map are summarized by table 5B-5.

Table 5B-5 — PHY register fields for the backplane environment

Field	Size	Type	Description
Physical_ID	6	rw	The address of this node. Unlike the equivalent field in the cable environment, the physical ID in the backplane environment is writable.
CM	1	rw[a]	Cycle master. The value of this bit does not affect the operation of the PHY, it is present in the backplane PHY register map for the sake of compatibility with some link designs that do not operate as cycle master unless this bit is set to one.
TD	1	rw	Transceiver disable. When set to one the PHY shall set all bus outputs to a high-impedance state and ignore any link layer service actions that would require a change to this bus output state.
IBR	1	rw	Initiate bus reset. When set to one, instructs the PHY to initiate a bus reset immediately (without arbitration). This bit causes assertion of the reset signal for approximately 8 µs and is self-clearing.
Data	2	r	Data line state (uses the same encoding as for cable).[b]
Strobe	2	r	Strobe line state (uses the same encoding as for cable).[b]
Priority	4	rw	This field shall contain the priority used in the urgent arbitration process and shall be transmitted as the *pri* field in the packet header.

[a]It is permitted to implement *CM* as a read-only bit, in which case its value shall be zero.
[b]The implementation of the *Data* and *Strobe* fields is optional; if unimplemented, these fields shall be zero.

5B.3 Integrated link and PHY

The register map described in 5B.1 and 5B.2 is specified to assure interoperability between discrete link and PHY implementations offered by different vendors. Because the PHY registers are the only means available to software to control or query the state of the PHY, these register definitions are also critical to software.

An integrated link and PHY implementation shall present the appropriate register map standardized in the preceding subclauses. The status that may be read from the registers and the behavior caused by a write to the registers shall be identical to that of a discrete link and PHY combination.

6. Link layer specification

6.1 Link layer services

Background

Subclauses 6.1.2.3 and 6.1.3.4 of IEEE Std 1394-1995 mandate that a link layer communicate packet status for each received packet to the transaction layer or isochronous application, respectively. DATA_CRC_ERROR, one of the values defined for packet status, specifies an error caused by an invalid data CRC. However, the definition of *ack_data_error* in both IEEE Std 1394-2995 and IEEE Std 1394a-2000 additionally characterize mismatches between a packet's *data_length* field and the data payload actually received as data errors.

This standard renames the DATA_CRC_ERROR packet status to DATA_ERROR and expands its definition to include malformed packets.

6.1.1 Link layer bus management services for the node controller

Replace 6.1.1.3 with the following:

6.1.1.3 Link event indication (LK_EVENT.indication)

Events detected by the link layer are communicated to the node controller by this service. This service provides no actions nor are there any responses defined for the indication. A single parameter, link event, may be communicated via this service; the parameter shall specify the event detected by the link layer, as defined below:

— BUS OCCUPANCY VIOLATION DETECTED. Clocked data was observed on the bus for longer than MAX_DATA_TIME.
— CYCLE TOO LONG. A cycle start packet was received and NOMINAL_CYCLE_TIME elapsed without the detection of a subaction gap.
— DUPLICATE CHANNEL DETECTED. A stream packet was received with a channel number equal to one of the node's active, transmit isochronous channels.
— HEADER CRC ERROR DETECTED. The CRC check for the header of a received primary packet failed (see 6.2.4.15).
— UNEXPECTED CHANNEL DETECTED. The isochronous resource manager observed a stream packet whose channel number is not allocated in the CHANNELS_AVAILABLE register; this event shall not be reported except by the active isochronous resource manager.

Link layer implementations may be incapable of detecting some or all of these events.

NOTE—The LK_EVENT.*indication* service is provided to report events detectable by the link layer but not reportable through the link layer data services. The node controller may log these events or it may take other implementation-dependent action.

6.1.2 Link layer asynchronous data services for the transaction layer

6.1.2.3 Link data indication (LK_DATA.indication)

Replace the list item that defines DATA_CRC_ERROR with the following:

3) DATA_ERROR. The CRC check (see 6.2.4.15) for the data portion of a block packet failed or the actual size of the packet's data payload, if present, does not match that specified by the *data_length* field. When the transaction layer receives this information (as the result of a LK_DATA.*indication*) it shall respond as specified by 7.3.1.3A.

6.1.3 Link layer isochronous data services for application layers

6.1.3.4 Link isochronous indication (LK_ISO.indication)

Replace list item g) with the following:

g) Packet status. This parameter shall contain the result of the receive packet operation performed by the link layer, as defined in 6.1.2.3. When an isochronous application receives a packet status of DATA_ERROR (as the result of LK_ISO.*indication*) it shall ignore the isochronous packet.

Add the following note after the existing note at the end of 6.1.3.4.

NOTE—In some isochronous applications, delivery jitter larger than a single isochronous cycle may be tolerable. For these applications, an isochronous talker that detects an error during transmission may elect to retransmit the packet during the next isochronous interval. The recipient of a malformed isochronous packet may not have a priori knowledge of the talker's error recovery behavior; as a consequence, the only reliable strategy is to discard the malformed packet.

6.2 Link layer facilities

6.2.2 Asynchronous packets

6.2.2.2 Asynchronous packet formats with data block payload

Replace 6.2.2.2.3 with the following:

6.2.2.2.3 Cycle start

The requirements placed upon a cycle master to transmit a cycle start packet are different from the criteria used by recipients to recognize a cycle start packet. The format of a cycle start packet is shown by figure 6-10.

The ***tcode*** field shall have a value of eight.

The ***source_ID*** field shall be the node ID of the cycle master that transmits the cycle start packet.

The ***cycle_time*** field shall contain the contents of the cycle master's CYCLE_TIME register (see 8.3.2.3.1).

The ***header_CRC*** field shall be calculated as specified by 6.2.4.15.

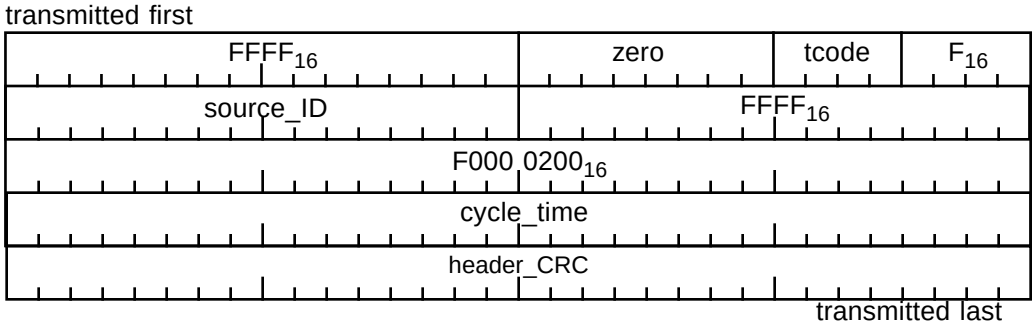


Figure 6-10 — Cycle start packet format

The cycle master shall signal the start of the isochronous period by transmitting a cycle start packet with the format defined above. No node except the cycle master shall transmit a cycle start packet.

A cycle start packet shall be recognized if *tcode* has a value of eight and the *header_CRC* is valid.

NOTE—The cycle start packet is a write request for data quadlet whose values can be interpreted as a broadcast write to the CYCLE_TIME register. Although this could be handled in an implementation by the transaction layer and node controller, this standard assumes that the link layer is responsible for the generation and detection of the start of an isochronous period.

6.2.2.3 Asynchronous packet formats with data block payload

Replace table 6-4 with the following:

Table 6-4 — Maximum data payload for asynchronous primary packets

Data rate	Maximum payload (bytes)	Comment
S25	128	TTL backplane
S50	256	BTL or ECL backplane
S100	512	Cable base rate
S200	1 024	—
S400	2 048	—
S800	4 096	—
S1600	8 192	—
S3200	16 384	—

6.2.2.3.3 Read response for data block

Insert the following at the end of the paragraph in 6.2.2.3.3:

When the response code (*rcode*) is *resp_complete*, the *data_length* field in the response packet shall be equal to the data length specified by the corresponding read request. If *data_length* is zero, no *data CRC* shall be transmitted.

6.2.3 Isochronous packets

6.2.3.1 Isochronous data block packet format

Background

Subclause 6.2.3.1 mandates that the total size of an isochronous stream packet (a primary packet with a *tcode* of A_{16} intended for transmission during the isochronous period) shall not exceed the bandwidth allocated for the channel. An additional requirement that the maximum data payload of an isochronous stream packet is speed-dependent is added.

Insert the following text and table at the end of 6.2.3.1:

The maximum data payload of an isochronous stream packet is speed dependent and shall conform to table 6-6A.

Table 6-6A — Maximum payload for isochronous stream packets

Data rate	Maximum payload (bytes)	Comment
S25	256	TTL backplane
S50	512	BTL or ECL backplane
S100	1 024	Cable base rate
S200	2 048	—
S400	4 096	—
S800	8 192	—
S1600	16 384	—
S3200	32 768	—

6.2.3A Asynchronous stream packets

Background

Asynchronous streams are an extension of the IEEE 1394 facilities to use an existing primary packet type with different arbitration requirements. As previously defined in IEEE Std 1394-1995, packets with a transaction code of A_{16} were called isochronous data block packets[7] and are subject to the following restrictions:

— An isochronous stream packet is transmitted only during the isochronous period. The isochronous period is controlled by the cycle master, which signals the start of the period with a cycle start packet. The period ends when a subaction gap is observed, which happens after all isochronous talkers have had a chance to transmit.
— Two resources, bandwidth and a channel number, are allocated from the isochronous resource manager registers BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE, respectively.
— For a given channel number, no more than one talker may transmit an isochronous stream packet with that channel number for each isochronous period.

This extension relaxes some of the above requirements in order to create something new—asynchronous stream(s). An asynchronous stream utilizes packets with a transaction code of A_{16} and is subject to the following requirements:

— An asynchronous stream packet shall be transmitted during the asynchronous period, subject to the same arbitration requirements, including fairness, as other asynchronous request subactions.
— The channel number shall be allocated from the isochronous resource manager register CHANNELS_AVAILABLE.
— Multiple nodes may transmit asynchronous stream packets with the same channel number or the same node may transmit multiple asynchronous stream packets with the same channel number as often as desired, subject to arbitration fairness.

An advantage of an asynchronous stream is that broadcast and multicast applications that do not have guaranteed latency requirements may be supported on Serial Bus without the allocation of a valuable resource, bandwidth. An additional advantage is that asynchronous streams may be easily filtered by contemporary hardware.

[7]Throughout this clause, primary packets with a transaction code of A_{16} are referred to as stream packets; the arbitration mode determines whether they are asynchronous or isochronous stream packets. The term *isochronous stream packet* is equivalent to *isochronous data block packet*, which was used previously.

Insert the following after 6.2.3:

6.2.3A Asynchronous stream packets

6.2.3A.1 Asynchronous stream packet format

The format of an asynchronous stream packet is identical to that of an isochronous stream packet, as specified by 6.2.3.1 and illustrated by figure 6-17A.

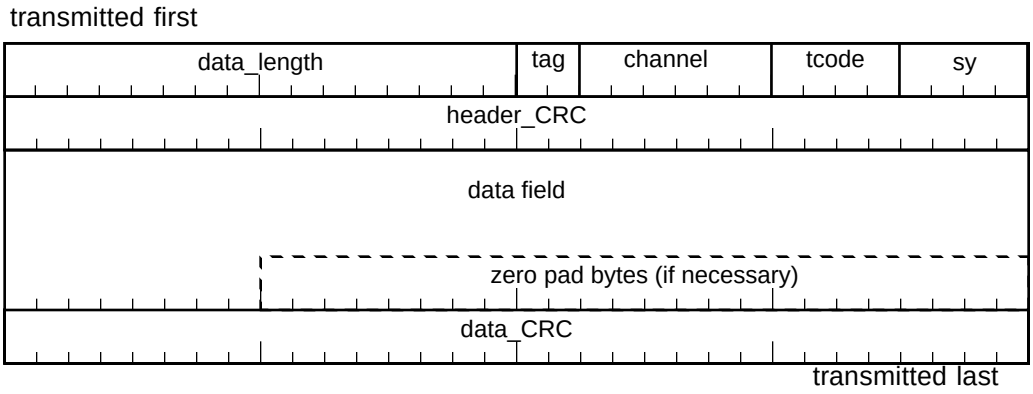


Figure 6-17A — Asynchronous stream packet format

The fields of an asynchronous stream packet shall conform to requirements that follow and those specified in 6.2.4.

The ***data_length*** field shall specify the length in bytes of the data field in the asynchronous stream packet The number of bytes in the data field is determined by the transmission speed of the packet and shall not exceed the maximums specified by table 6-4.

The tag field shall have a value of zero, unformatted data, or three, global asynchronous stream packet (GASP) format; other values of tag are reserved for future standardization. When tag is zero the content and format of the data field are unspecified. Otherwise, when tag is three, the format of the asynchronous stream packet is specified by 6.2.3A.2.

The channel field shall identify the stream; the channel identified shall be allocated from the isochronous resource manager CHANNELS_AVAILABLE register.

NOTE—Unlike isochronous stream packets, which may continue to be transmitted for up to 1 s subsequent to a bus reset without channel reallocation, asynchronous stream packets may not be transmitted until their channel number(s) are reallocated.

The *tcode* field shall have a value of A_{16}. The new name for this transaction code value is stream packet; the context in which the packet is sent determines whether it is an asynchronous or isochronous stream packet.

The usage of any fields not specified in this subclause remains as described for isochronous stream packets in 6.2.3.

6.2.3A.2 Global asynchronous stream packet (GASP) format

Motivated by ongoing efforts in the IEEE P1394.1 Working Group, this subclause defines an asynchronous stream packet format suitable for transport across a bridge from one Serial Bus to another.

The format of a global asynchronous stream packet is an extension of that specified by 6.2.3A.1, which utilizes the first two quadlets of the packet's data payload. The GASP format is illustrated by figure 6-17B.

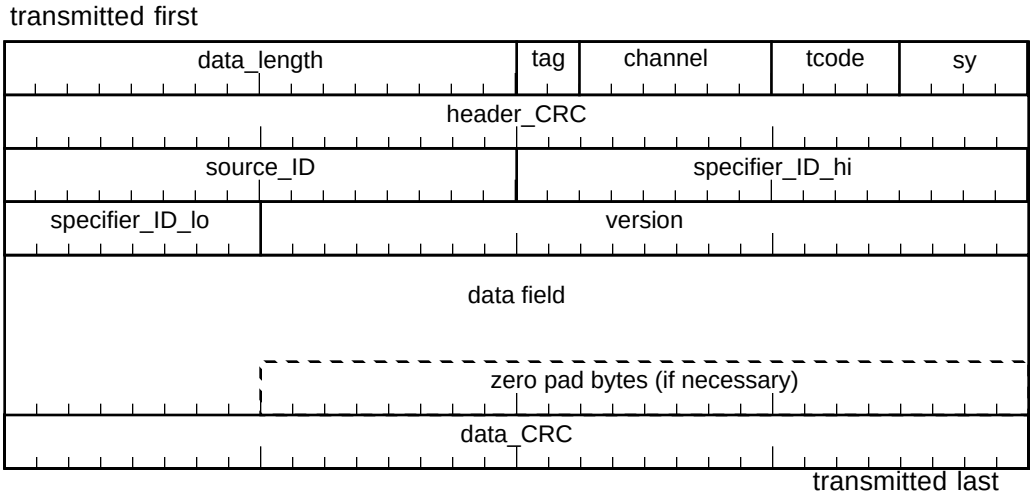


Figure 6-17B — Global asynchronous stream packet (GASP) format

Except as specified below, the definition and usage of the fields in figure 6-17B is contained within 6.2.3A.1.

The ***tag*** field shall have a value of three.

The ***sy*** field is reserved for future standardization and shall be zero.

The ***source_ID*** field shall specify the node ID of the sending node and shall be equal to the most significant 16 bits of the sender's NODE_IDS register.

The ***specifier_ID*** field shall contain a 24-bit organizationally unique identifier (OUI) assigned by the IEEE Registration Authority (RA). The owner of the OUI (company, accredited standards organization, or industry group) shall be responsible for defining the meaning and usage of the remainder of the data payload in the stream packet.

The meaning and usage of the ***version*** field shall be defined by the owner of *specifier_ID*.

6.2.3A.3 Loose vs. strict isochronous packet reception

Although this standard previously prohibited the reception of an isochronous stream packet outside of the isochronous period (strict isochronous), a significant number of contemporary Serial Bus link designs relax this requirement and permit the reception of an isochronous stream packet at any time (loose isochronous).

The strict isochronous requirement is removed; link designs compliant with this standard shall be capable of receiving stream packets (primary packets identified by *tcode* A_{16}) without regard to whether or not the packet falls within or without the isochronous period.

NOTE—It is possible for an isochronous packet properly transmitted by one node (i.e., during the isochronous period) to be observed by another node during the asynchronous period. This can happen if the cycle start packet is not received correctly. In this case, the reception of isochronous packets at any time is sensible, even without consideration of asynchronous streams. Many applications are able to make valid use of isochronous data that follows, so long as the link permits its reception.

6.2.4 Primary packet components

6.2.4.4 Retry code (rt)

Background

Subclause 6.2.4.4 in IEEE Std 1394-1995 specifies the encoding for the retry code (*rt*) in an asynchronous primary packet. It also specifies requirements for the usage of *rt* that are in conflict with the retry protocols contained within IEEE Std 1394a-2000. Consequently, 7.3.5.1, as amended in this document, takes precedence.

Replace the text in 6.2.4.4 with the following:

The encoding for the retry code (*rt*) in an asynchronous primary packet is given in table 6-8. Requirements for the usage of *rt* are also specified.

Table 6-8 — Retry code encoding

Value	Name	Comment
0	*retry_1*	Reservation requested; transmitter implements outbound dual-phase retry.
1	*retry_X*	No reservation requested.
2	*retry_A*	Reservation previously assigned by the receipt of *ack_busy_A;* transmitter implements outbound dual-phase retry.
3	*retry_B*	Reservation previously assigned by the receipt of *ack_busy_B;* transmitter implements outbound dual-phase retry.

The rules for the usage of the retry code are specified by 7.3.5.2.

6.2.4.5 Transaction codes (tcode)

Background

Subclause 6.2.4.5 defines the meaning of the *tcode* that identifies primary packets. IEEE Std 1394a-2000 extends the scope of *tcode* A_{16} in order to support asynchronous streams and also permanently reserves *tcode* E_{16}.

Replace table 6-9 with the following:

Table 6-9 — Transaction code encoding

Code	Header size (quadlets)	Name	Comment
0	5	Write request for data quadlet	Request subaction, quadlet payload
1	5	Write request for data block	Request subaction, block payload
2	4	Write response	Response subaction for both write requests types, no payload
3	—	Reserved	—
4	4	Read request for data quadlet	Request subaction, no payload
5	5	Read request for data block	Request subaction, quadlet (*data_length*) payload
6	5	Read response for data quadlet	Response subaction to read request for quadlet, quadlet payload
7	5	Read response for data block	Response subaction to read request for block, block payload
8	5	Cycle start	Request to start isochronous period, quadlet payload
9	5	Lock request	Request subaction, block payload

Table 6-9 — Transaction code encoding (continued)

Code	Header size (quadlets)	Name	Comment
A_{16}	2	Stream data	Asynchronous or isochronous subaction, block payload
B_{16}	5	Lock response	Response subaction for lock request, block payload
C_{16}	—	—	Reserved for future standardization
D_{16}	—	—	Reserved for future standardization
E_{16}	—	—	Utilized internally by some link designs; not to be standardized
F_{16}	—	—	Reserved for future standardization

6.2.4.12 Tag

Background

Subclause 6.2.4.12 defines the meaning of the *tag* field transmitted in an isochronous stream packet. This standard defines, by reference to IEC 61883-1 (1998-02), one of the previously reserved *tag* values.

Replace 6.2.4.12 with the following:

6.2.4.12 Tag (isochronous stream packets)

The *tag* field provides a label, useful to applications, that specifies the format of the payload carried by an isochronous stream packet.

The values defined for *tag* in table 6-10 only apply to isochronous stream packets; permissible values of *tag* for asynchronous stream packets are defined in 6.2.3A.1. Because it is necessary to allocate different channel numbers for asynchronous and isochronous stream packets, the context in which to interpret *tag* is unambiguous.

Table 6-10 — Tag field encoding

Value	Meaning
0	Data field format unspecified
1	Data format and *sy* field specified by IEC 61883-1 (1998-02)
2	Reserved for future standardization
3	Reserved for future standardization

6.2.5 Acknowledge packets

6.2.5.2 ACK packet components

6.2.5.2.2 Acknowledge codes (ack_code)

Background

Subclause 6.2.5.2.2 defines the meaning of the *ack_code* transmitted in immediate response to a nonbroadcast request or response packet. Three new acknowledge codes, *ack_address_error, ack_conflict_error,* and *ack_tardy* are defined.

Replace table 6-13 with the following:

Table 6-13 — Acknowledge codes

Code	Name	Comment
0	—	Not to be used in any future Serial Bus standard.
1	ack_complete	The node has successfully accepted the packet. If the packet was a request subaction, the destination node has successfully completed the transaction and no response subaction shall follow.
2	ack_pending	The node has successfully accepted the packet. If the packet was a request subaction, a response subaction is expected at a later time. This code shall not be returned for a response subaction.
3	—	Reserved for future standardization.
4	ack_busy_X	The packet could not be accepted. The destination transaction layer may accept the packet on a retry of the subaction.
5	ack_busy_A	The packet could not be accepted. The destination transaction layer may accept the packet in accordance with the retry protocol specified by 7.3.5.
6	ack_busy_B	The packet could not be accepted. The destination transaction layer may accept the packet in accordance with the retry protocol specified by 7.3.5.
$7–A_{16}$	—	Reserved for future standardization.
B_{16}	ack_tardy	The node could not accept the packet because the link and higher layers are in a suspended state; the destination node shall restore full functionality to the link and transaction layers and may accept the packet on a retransmission in a subsequent fairness interval.
C_{16}	ack_conflict_error	A resource conflict prevented the packet from being accepted.
D_{16}	ack_data_error	The node could not accept the block packet because the data field failed the CRC check or because the length of the data payload did not match the length contained in the *data_length* field. This code shall not be returned for any packet whose header does not contain a *data_length* field.
E_{16}	ack_type_error	A field in the request packet header was set to an unsupported or incorrect value, or an invalid transaction was attempted (e.g., a write to a read-only address).
F_{16}	ack_address_error	The node could not accept the packet because the *destination_offset* field in the request was set to an address not accessible in the destination node.

Insert the following text in 6.2.5.2.2, after table 6-13:

An ***ack_complete*** shall not be sent in response to a read or lock request.

In addition to the uses of ***ack_busy_X, ack_busy_A,*** and ***ack_busy_B*** specified in table 6-13, these acknowledge codes may be sent when the recipient of a packet desires its retransmission for other reasons. For example, packets with invalid data CRC or a mismatch between actual data payload and that specified by the *data_length* field are likely the result of transient conditions, which may not be present upon retransmission. Because link layer retry behavior after receipt of a busy acknowledgment is often implemented in hardware, one of ***ack_busy_X, ack_busy_A,*** or ***ack_busy_B*** should be used in place of other acknowledge or response codes that permit transaction layer retry of the entire transaction.

A node that returns *ack_busy_A* or *ack_busy_B* in cases when sufficient resources are available may introduce unintended consequences. As specified by 7.3.5, the node would be prohibited from accepting packets unless their retry code matched the preferred retry phase—even though it suffers from no lack of resources. Therefore, a node that elects to return a busy acknowledgment instead of a terminal acknowledgment shall select one of *ack_busy_X, ack_busy_A,* or *ack_busy_B* according to the following table.

Resources available	Retry code	Busy acknowledgment
—	retry_A	ack_busy_A
	retry_B	ack_busy_B
	retry_X	ack_busy_X
Yes	retry_1	
No	retry_1	ack_busy_A / ack_busy_B (per 7.3.5)

Although a resource conflict or an address error in a request packet is usually detected by the transaction or higher-application layer, there may be circumstances in which the link is capable of detecting these errors within the time permitted for a unified response to a request subaction. The acknowledge codes ***ack_conflict_error*** and ***ack_address_error*** are defined to provide the same utility as the existing *resp_conflict_error* and *resp_address_error.*

The ***ack_tardy*** code has been defined to enable low-power consumption states for Serial Bus devices. Such a device may be able to place its link layer in a partially functional state and suspend the transaction and all higher-application layers. The link layer shall be able to recognize nonbroadcast request packets whose *destination_ID* addresses the suspended node. Upon recognition of such a packet, the link shall send an *ack_tardy* and shall initiate the resumption of full link and transaction layer functionality. The recipient of an *ack_tardy* may retransmit the request packet in a subsequent fairness interval. The time required for the link and transaction layers to become fully operational is implementation-dependent.

NOTE—Transactions are completed by either an *ack_pending* and a subsequent response packet, or by an acknowledgment other than *ack_pending, ack_busy_X, ack_busy_A,* or *ack_busy_B.* At the transaction layer both methods are equivalent and the same criteria shall be used in either case to select the appropriate acknowledgment or response code. However, responses conveyed by acknowledge packets are preferred over separate response packets, since the former utilize less Serial Bus bandwidth.

6.3 Link layer operation

6.3.1 Overview of link layer operation

Insert the following subclause after 6.3.1.1:

6.3.1.1A Priority arbitration for PHY packets and response packets

When transmitting a PHY packet or a response packet, a link may use priority arbitration requests. If the node implements the PRIORITY_BUDGET register (see 8.3.2.3.5A), priority requests for PHY packets may count but response packets shall not count against the node's priority arbitration budget. A PHY packet is any 64-bit packet whose least significant 32 bits are the one's complement of the most significant 32 bits. A response packet is an asynchronous primary packet with a *tcode* of 2, 6, 7, or B_{16}.

If an *ack_busy_X, ack_busy_A,* or *ack_busy_B* is received in acknowledgment of a response packet, the node shall not retransmit the response packet until the next fairness interval.

7. Transaction layer specification

7.1 Transaction layer services

7.1.2 Transaction layer data services for applications and bus management

Background

The descriptions of the different types of primary Serial Bus packets in clause 6 require that the transaction codes (*tcode*) used in response to data requests correspond to the original *tcode* of the request. That is, a read response for data quadlet shall be sent only in response to a read request for data quadlet, a read response for data block shall be sent only in response to a read request for data block and a lock response shall be sent only in response to a lock request. A close examination of clause 7 reveals that insufficient information is communicated to the transaction layer in order for it to meet this requirement.

The portion of clause 7 that describes which *tcode* the transaction layer shall select for a READ or WRITE request mandates, at present, that a quadlet *tcode* shall be used if the data length of the transaction is four, independent of whether or not the destination offset is quadlet aligned. This does not conform to the expectations of most Serial Bus implementers, namely, that quadlet transactions should be used only if the address is quadlet aligned.

Uniform behavior of transaction layer implementations shall be achieved by conformance to the following specifications. Briefly, the specifications

a) Limit the use of quadlet READ and WRITE transactions to the case where the data length is four and the destination offset is quadlet aligned
b) Optionally permit the use of block READ and WRITE transactions in the case where the data length is four and the destination offset is quadlet aligned
c) Add new parameters to a transaction layer data service so that quadlet responses may be properly generated for quadlet requests and block responses for block requests
d) Emphasize that support for quadlet transactions is mandatory in all Serial Bus implementations but that support for block transactions is optional

7.1.2.1 Transaction data request (TRAN_DATA.request)

Insert the following above item b) in 7.1.2.1:

a1) *Local bus ID.* When TRUE, this indicates that the link shall use $3FF_{16}$ as the bus ID component of source_ID. Otherwise, the most significant 16 bits of the NODE_IDS register shall be used as source_ID.
a2) *Packet format.* In the case of READ or WRITE transactions with a data length of four and a quadlet-aligned destination address, this parameter shall govern the type of *tcode*, block or quadlet, generated by the transaction layer. This parameter shall have a value of BLOCK TCODE or QUADLET TCODE.

7.1.2.4 Transaction data response (TRAN_DATA.response)

Insert the following above item b) in 7.1.2.4:

a1) *Local bus ID.* When TRUE, this indicates that the link shall use $3FF_{16}$ as the bus ID component of source_ID. Otherwise, the most significant 16 bits of the NODE_IDS register shall be used as source_ID.
a2) *Packet format.* In the case of READ or WRITE transactions, this parameter shall indicate the type of *tcode*, block or quadlet, received by the transaction layer. This parameter shall have a value of BLOCK TCODE or QUADLET TCODE.

7.3 Transaction operation

7.3.1 Overview of transaction layer operations

Background

Despite the intended sufficiency of IEEE Std 1394-1995, discussions in a variety of forums have made it clear that the usage of the response code is subject to interpretation. Response code usage in ambiguous cases is clarified so as to assure equivalent behavior, and hence interoperability, of link layer, transaction layer, and application implementations from different vendors. The examples given are not exhaustive nor do they illustrate the common usage already specified by IEEE Std 1394-1995.

Insert the following after 7.3.1.3:

7.3.1.3A Response codes (rcode)

7.3.1.3A.1 No response

If a packet is received with a *tcode* value that is reserved by this standard, the node shall not respond.

7.3.1.3A.2 resp_complete

Nodes shall respond with *resp_complete* in the following circumstance (this is not exhaustive, and is only an example of a circumstance for which there might be confusion with other response codes):

— A write request is received for a writable address that contains read-only bits or fields. The transaction completes successfully and the write effects on the read-only bits are as specified in this standard or the document that describes the unit architecture. Generally an address is not considered writable if all bits are read-only; see the discussion of *resp_type_error* in 7.3.1.3A.5.

Nodes may respond with *resp_complete* in the following circumstances:

— A read request packet is addressed to a valid *destination_ID* but the *destination_offset* references an address that is not implemented by the node.
— A block read request packet is addressed to a valid *destination_ID* but the combination of the *destination_offset* and the *data_length* reference addresses, some of which are not implemented by the node.

In both of the previous circumstances, if the read request is addressed to configuration ROM, the data value returned in the response is unspecified. Configuration ROM includes not only the first kilobyte of ROM (quadlets in the address range FFFF F000 0400_{16} through FFFF F000 $07FC_{16}$, inclusive), but any directories or leaves that are indirectly addressed from the first kilobyte. Otherwise, for read requests addressed to any location not within configuration ROM, the returned data value shall be zero.

7.3.1.3A.3 resp_conflict_error

Nodes shall respond with resp_conflict_error in the following circumstance:

— An otherwise valid request packet is received but the resources required to act upon the request are not available. The requester may reasonably expect the same packet to succeed at some point in the future when the resources are available. Note that the distinction between *resp_conflict_error* and *ack_busy_X*, *ack_busy_A*, or *ack_busy_B* hinges upon the possibility of deadlock. The busy acknowledgments are appropriate for transient conditions of expected short duration that cannot cause a deadlock. On the other hand, *resp_conflict_error* shall be returned when an end-to-end retry is necessary to avoid the possibility of deadlock. Deadlocks may arise when a request cannot be queued and blocks a node's transaction resources.

7.3.1.3A.4 resp_data_error

Nodes shall respond with resp_data_error in the following circumstances:

— For read requests, an otherwise valid packet is received but a hardware error at the node prevents the return of the requested data. For example, an uncorrectable memory error shall be reported as *resp_data_error.*

— For write or lock requests, an otherwise valid packet is received but a hardware error at the node prevents the updates indicated by the data payload from initiation or completion.

Nodes may respond with resp_data_error in the following circumstances:

— An otherwise valid request packet is received but there is a data CRC error for the data payload.

— An otherwise valid packet is received but the actual size of the data payload differs from that specified by *data_length*.

7.3.1.3A.5 resp_type_error

Nodes shall respond with resp_type_error in the following circumstances:

— A request packet is received with a valid *tcode* (transaction code) value but the *extended_tcode* field value is reserved by this standard.

— A request packet is received with valid *tcode* and *extended_tcode* values, but the referenced address does not implement the indicated request. An example of this is a write request to an address that is entirely read-only (note that this is distinct from a write request that references read-only bits or fields at an otherwise writable location). Another example is a transaction whose *tcode* specifies a lock operation but the destination address supports only read and write operations.

— A request packet is addressed to a valid *destination_ID*, the *destination_offset* references an address implemented by the node, but the alignment of the destination offset does not match the node's alignment requirements. For example, a quadlet register is implemented but cannot respond to a 1-byte data block request.

In addition to the previously mandated responses, nodes should respond with resp_type_error in the following circumstance:

— A request packet is received with valid *tcode* and *extended_tcode* values, but the recipient accepts requests only from particular senders, as identified by *source_ID*. Some protocols protect certain addresses from both unintended and malicious interference by requiring a login procedure that identifies the *source_ID* of a valid requester.

7.3.1.3A.6 resp_address_error

An address error condition exists when the combination of the destination_offset and, when present in the request, the *data_length* fields reference addresses, some of which are not implemented by the node. In some circumstances, a node may respond with *resp_complete* (see 7.3.1.3A.2).

Unless a node responds with *resp_complete*, it shall respond with resp_address_error in the following circumstances:

— A request packet is addressed to a valid *destination_ID* but the *destination_offset* references an address that is not implemented by the node.

A block request packet is addressed to a valid *destination_ID* but the combination of the *destination_offset* and the *data_length* reference addresses, some of which are not implemented by the node.

7.3.3 Details of transaction layer operation

7.3.3.1 Outbound transaction state machine

7.3.3.1.2 Sending a transaction request

Replace the list after the second paragraph in 7.3.3.1.2 with the following:

- Write request for data quadlet, if the transaction type value in the transaction data request is WRITE, the data length is four, the destination address is quadlet aligned, and the packet format value is QUADLET TCODE.
- Write request for data block, if the transaction type value in the transaction data request is WRITE, the data length is four, the destination address is quadlet aligned, and the packet format value is BLOCK TCODE.
- Write request for data block, if the transaction type value in the transaction data request is WRITE and the data length is not four, or the destination address is not quadlet aligned.
- Read request for data quadlet, if the transaction type value in the transaction data request is READ, the data length is four, the destination address is quadlet aligned, and the packet format value is QUADLET TCODE.
- Read request for data block, if the transaction type value in the transaction data request is READ, the data length is four, the destination address is quadlet aligned, and the packet format value is BLOCK TCODE.
- Read request for data block, if the transaction type value in the transaction data request is READ and the data length is not four, or the destination address is not quadlet aligned; or
- Lock request, if the transaction type value in the transaction data request is LOCK.

7.3.3.1.3 Sending a transaction response

Replace the list after the second paragraph in 7.3.3.1.3 with the following:

- Write response for data quadlet, if the transaction type value in the transaction data request is WRITE, the data length is four, and the packet format value is QUADLET TCODE.
- Write response for data block, if the transaction type value in the transaction data request is WRITE and the data length is not four, or the packet format value is BLOCK TCODE.
- Read response for data quadlet, if the transaction type value in the transaction data request is READ, the data length is four, and the packet format value is QUADLET TCODE.
- Read response for data block, if the transaction type value in the transaction data request is READ and the data length is not four, or the packet format value is BLOCK TCODE.
- Lock response, if the transaction type value in the transaction data request is LOCK.

7.3.4 Transaction types

7.3.4.3 Lock transactions

Background

The change in the definition of mask_swap in table 7-5 conforms to ISO/IEC 13213:1994.

Replace table 7-5 with the following:

Table 7-5 — Summary of lock transaction functions

Lock function	Update action
mask_swap	`new_value = (data_value & arg_value) | (old_value & ~arg_value);`
compare_swap	`if (old_value == arg_value) new_value = data_value;`
fetch_add	`new_value = old_value + data_value;`
little_add	`(little) new_value = (little) old_value + (little) data_value;`
bounded_add	`if (old_value!= arg_value) new_value = old_value + data_value;`
wrap_add	`new_value = (old_value != arg_value) ? old_value + data_value : data_value;`

Insert the following at the end of 7.3.4.3:

In the preceding, *arg_value* and *data_value* are the fields of the same name from the lock request packet. The *old_value* field is the current value of the addressed CSR obtained as if from a read request; this is also the value returned in the lock response packet. The *new_value* field is the updated value of the CSR as if a write request were used to store the calculated value.

When a lock transaction addresses a CSR that has one or more reserved bits or fields, the results are not necessarily obvious. The behavior of a particular lock function shall be determined by applying rules for reserved fields (see 1.6.11) in order, as follows:

a) The CSR's *old_value* shall be obtained as if via a read request and shall be returned in the lock response; reserved fields are read as zeros.
b) An intermediate value shall be calculated according to the C code in table 7-5 (this is not explicitly shown but is the right-hand part of each of the assignment statements in the table).
c) The intermediate value shall be stored in the CSR as if via a write request; reserved fields shall be ignored and remain zero in the CSR. The contents of the CSR after this operation are the *new_value*.

7.3.5 Retry protocols

Background

Although 7.3.5 of IEEE Std 1394-1995 accurately specifies single-phase retry, IEEE Std 1394a-2000 clarifies the information. In addition, the following changes have been made:

— The requirements for the usage of *rt* that is specified in 6.2.4.4 of IEEE Std 1394-1995 are in conflict with the revised retry protocols contained within IEEE Std 1394a-2000. This is corrected in 6.2.4.4 of IEEE Std 1394a-2000.

— The specification of outbound retry behavior in IEEE Std 1394-1995 is somewhat confusing since it implies a transaction layer state machine when in fact no such state machine exists in the context of the entire transaction layer. This is corrected in 7.3.5.2 of IEEE Std 1394a-2000.

Replace the text in 7.3.5 with the following and renumber the remaining tables in clause 7:

Retry protocols define the complementary procedures used by the transaction layers of the sender of an asynchronous primary packet and its intended recipient when the recipient is unable to service the packet, i.e., is in some sense "busy." The packet originator utilizes an outbound retry protocol while the intended recipient participates in an inbound retry protocol. The retry protocols may be used with any packet, whether request or response, for which an acknowledgment is expected from the recipient.

In the case of dual-phase retry, revision of both the outbound and inbound state machines are necessary for the following reasons:

— *Reservation time limit ambiguities.* This standard specifies that retry reservations shall be held by the intended recipient for four arbitration fairness intervals before cancellation. Unfortunately, the point from which fairness intervals is measured is not clearly specified. Contemporary link implementations are known to have different (although reasonable) interpretations which are not interoperable.
— *No resynchronization.* The inbound dual-phase retry state machines in this standard do not resynchronize reservation histories (phase and count of outstanding reservations) when discrepancies exist between the outbound and inbound nodes.

In order to correct these problems and to add an enhancement (the ability to count the number of outstanding reservations for each phase), dual-phase retry behavior for both outbound and inbound nodes is redefined. The maintenance of reservation counters permits an inbound node to immediately accept subactions that lack a resource reservation if there are no reservations held for pending subactions.

7.3.5.1 Outbound subaction retry protocol

The transaction layer shall implement a decision table that selects a retry code, *rt*, each time an asynchronous primary packet is transmitted. The choice of retry code, specified by table 7-6, depends upon the packet's history, i.e., both its "age" and the last acknowledge code received for the subaction.

Table 7-6 — Outbound subaction retry code decision table

<table>
<tr><th rowspan="2">Subaction age</th><th rowspan="2">Prior acknowledge code</th><th colspan="2">Retry code</th></tr>
<tr><th>Single-phase</th><th>Dual-phase</th></tr>
<tr><td rowspan="4">Oldest</td><td>—</td><td rowspan="4">retry_X</td><td rowspan="2">retry_1</td></tr>
<tr><td>ack_busy_X</td></tr>
<tr><td>ack_busy_A</td><td>retry_A</td></tr>
<tr><td>ack_busy_B</td><td>retry_B</td></tr>
<tr><td rowspan="2">Not oldest</td><td>—</td><td colspan="2" rowspan="2">retry_X</td></tr>
<tr><td>ack_busy_X
ack_busy_A
ack_busy_B</td></tr>
</table>

Request and response retries and their associated reservations shall be processed independently of each other. At any point in time there may be both an oldest request and an oldest response.

The description of a subaction's age is not meant to imply the necessity for timers in a link design. When an asynchronous primary packet is available for transmission and there are no subactions awaiting retry, the packet is by definition the oldest packet and may be transmitted with a retry code of *retry_1*. When that subaction completes with a terminal acknowledge code (any acknowledgment, including *ack_pending*, other than *ack_busy_X*, *ack_busy_A*, or *ack_busy_B*), another subaction awaiting transmission (or retransmission) may be designated oldest. The details as to which other subaction is elected oldest are implementation-dependent and are not important to the proper behavior of the retry protocols so long as the following requirement is observed:

> An asynchronous primary packet shall not be transmitted with a retry code of *retry_1* so long as a different subaction of the same type (request or response) once transmitted with a retry code of *retry_1* has not yet been completed with a terminal acknowledge code or been abandoned.

Subactions that do not yet have a history, i.e., this is the first time transmission has been initiated, are indicated by the absence of a prior acknowledge code.

It is not necessary for the node transmitting a subaction to have a priori knowledge as to whether or not the intended recipient (inbound node) has implemented the single- or dual-phase retry protocol. Designs capable of dual-phase retry should select the initial retry code, *retry_X* or *retry_1*, from the right-hand column in table 7-6 while designs that restrict themselves to single-phase retry shall use a retry code of *retry_X* in all cases. A node that implements the dual-phase retry protocol may transmit *retry_X* if no reservation is requested.

In order for the dual-phase retry protocol to be able to guarantee forward progress, an outbound node should be capable of retransmission of a subaction within five fairness intervals; this is the period of time for which an inbound node guarantees a retry reservation. Although the inbound dual-phase retry protocol state machine resets itself properly if a reservation is not utilized within this time limit (see 7.3.5.3 for details), the outbound node may fail to make forward progress if it loses retry reservations because of delayed retransmission. Fairness intervals are counted from the receipt of the *ack_busy_A* or *ack_busy_B* that granted the reservation; if an outbound node is unable to retransmit the subaction before five arbitration reset gaps have been observed, it may assume that the reservation has been cancelled.

7.3.5.2 Inbound subaction single-phase retry protocol

Any time the transaction layer receives an LK_DATA.*indication* and resources are unavailable to service the subaction, the transaction layer shall communicate an LK_DATA.*response* with an acknowledge parameter of *ack_busy_X* without regard to the value of the subaction's retry code. The preceding shall not apply to subactions for which no acknowledgment is expected, e.g., broadcast requests.

7.3.5.3 Inbound subaction dual-phase retry protocol

The intended recipient of an asynchronous subaction, request or response, may be unable to accept the packet because of transient resource limitations—the node is busy. In a simple (single-phase) retry protocol, senders retransmit the subaction until resources are available or they abandon the transaction. Single-phase retry does not guarantee forward progress because it makes no attempt to reserve resources for the oldest subactions. The dual-phase protocol described in this subclause reserves resources when congestion is encountered and keeps them reserved for particular subactions identified by a retry code. As subactions complete, the inbound node resources are once again made available to all subactions, with or without reservations.

The transaction layer shall allocate resources independently for request and response queues; this is necessary to prevent interdependent live-locks or starvation conditions. The dual-phase retry protocol specified by this subclause shall be separately implemented for request and response subactions.

In the description that follows, the size of the reservation counter is implementation-dependent. These counters are unsigned numbers and shall not be decremented to a value smaller than zero nor incremented to a value larger than their limit (often 2^{n-1},where *n* is the size of the counter, in bits). Two procedures are defined in table 7-7 to specify saturated operations for reservation accounting.

Table 7-7 — Saturated arithmetic procedures

```
void reserve(int retry_phase) {        // Count reservations until saturated

   reservations[retry_phase] += (reservations[retry_phase) < MAX_RESERVATIONS);

}

void release(int retry_phase) {       // Release earlier reservation (unless saturated)

   reservations[retry_phase] -= (   reservations[retry_phase) > 0
                                 && reservations[retry_phase) < MAX_RESERVATIONS);

}
```

The name *ack_subaction_done* refers collectively to any acknowledge code defined in this standard except *ack_busy_X*, *ack_busy_A*, or *ack_busy_B*.

The state machine transitions are shown in figure 7-3 and briefly described in the paragraphs that follow.

Transition All:IDR. The receipt of a TR_CONTROL.*request* with an action of either Reset or Initialize shall cause the inbound dual-phase retry state machine to set its reservation preference to *retry_A* and zero all retry counters.

State IDR: Idle. The transaction layer is potentially ready to accept a request or response subaction from the link. Whether or not the subaction is serviced by the transaction layer is determined by the availability of resources (such as FIFO space), outstanding reservation counts and the retry history of the subaction itself.

Transition IDR:IDRa. The end of a fairness interval has been detected, indicated by an arbitration reset gap. If the accumulated count of reset gaps is less than four, the transaction layer shall increment the count.

Transition IDR:IDRb. The end of the last fairness interval available to the outbound node for the retry of a subaction for which the inbound node granted a reservation. The inbound node abandons all reservations for the currently preferred retry phase, switches its preference to the opposite retry phase and counts this fairness interval as the first of the four available to holders of reservations in the now preferred phase.

Transition IDR:IDRX. An LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_X*.

Transition IDR:IDR1. An LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_1*.

Transition IDR:IDRA. An LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_A*.

Transition IDR:IDRB. An LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_B*.

State IDRX: Retry_X received. The outbound node (originator of the request or response subaction) has not requested a reservation if resources are unavailable to service the subaction. Attempt to process the subaction so long as resources are free and not allocated to a different subaction that holds a reservation.

Transition IDRX:IDRa. Resources are available and there are no reservations outstanding for the currently preferred phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRX:IDRb. Resources are unavailable or there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction without creating a reservation.

State IDR1: Retry_1 received. The outbound node (originator of the request or response subaction) has requested a reservation if resources are unavailable to service the subaction. Attempt to process the subaction so long as resources are free and not allocated to a different subaction that holds a reservation. Otherwise, create a reservation in the opposite phase and indicate the phase of the reservation by means of the acknowledge code returned to the outbound node.

Transition IDR1:IDRa. Resources are available and there are no reservations outstanding for the currently preferred phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

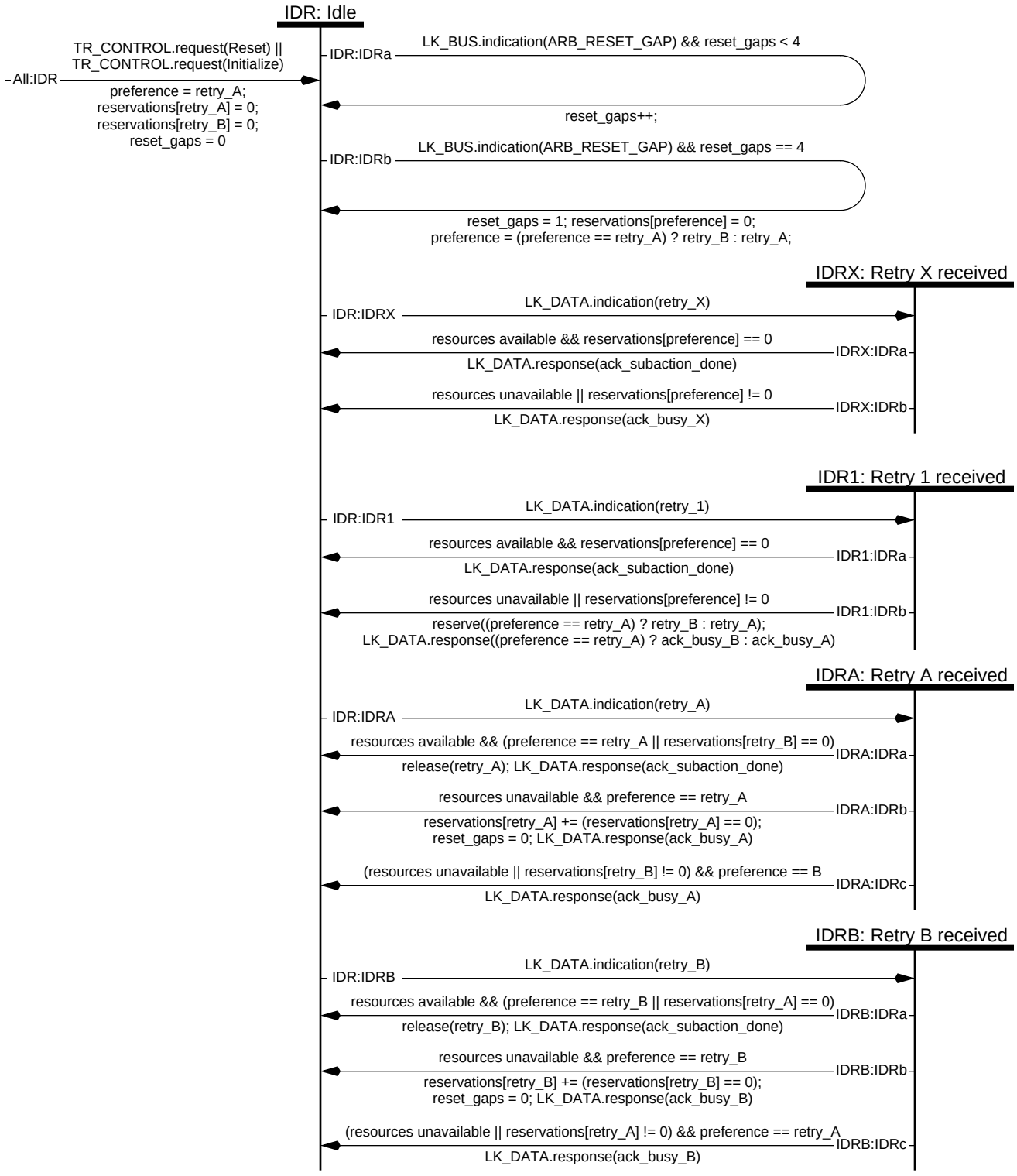


Figure 7-3—Inbound subaction dual-phase retry state machine

Transition IDR1:IDRb. Resources are unavailable or there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction but create a reservation for the opposite phase. The acknowledge code returned to the outbound node, either *ack_busy_A* or *ack_busy_B*, shall indicate the phase of the newly created reservation.

State IDRA: Retry_A received. The outbound node is attempting retransmission of an earlier request or response subaction for which the inbound node created a reservation. So long as resources are available, the inbound node grants preference to reservations earlier created for the current phase. Otherwise, if there are no outstanding reservations for the current phase, opposite phase retry attempts are serviced as resources permit.

Transition IDRA:IDRa. Resources are available and either *retry_A* is the currently preferred phase or else there are no reservations outstanding for the opposite phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRA:IDRb. Resources are unavailable and *retry_A* is the currently preferred phase. If the reservation count for *retry_A* is nonzero, there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction; the *ack_busy_A* acknowledge code returned indicates that the outbound node should continue retry attempts in the same phase. If the *retry_A* reservation count is zero, the outbound and inbound state machines are out of synchronization with respect to each other; the inbound dual-phase state machine corrects this by incrementing the reservation count.

Transition IDRA:IDRc. Although a retry code of *retry_A* was received from the outbound node, the currently preferred retry phase is for *retry_B*. If either resources are unavailable or there are *retry_B* reservations outstanding, the transaction layer shall refuse the subaction but continue to hold a *retry_A* reservation for the outbound node.

State IDRB: Retry_B received. The outbound node is attempting retransmission of an earlier request or response subaction for which the inbound node created a reservation. So long as resources are available, the inbound node grants preference to reservations earlier created for the current phase. Otherwise, if there are no outstanding reservations for the current phase, opposite phase retry attempts are serviced as resources permit.

Transition IDRB:IDRa. Resources are available and either *retry_B* is the currently preferred phase or else there are no reservations outstanding for the opposite phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRB:IDRb. Resources are unavailable and *retry_B* is the currently preferred phase. If the reservation count for *retry_B* is nonzero, there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction; the *ack_busy_B* acknowledge code returned indicates that the outbound node should continue retry attempts in the same phase. If the *retry_B* reservation count is zero, the outbound and inbound state machines are out of synchronization with respect to each other; the inbound dual-phase state machine corrects this by incrementing the reservation count.

Transition IDRB:IDRc. Although a retry code of *retry_B* was received from the outbound node, the currently preferred retry phase is for *retry_A*. If either resources are unavailable or there are *retry_A* reservations outstanding, the transaction layer shall refuse the subaction but continue to hold a *retry_B* reservation for the outbound node.

7.4 CSR Architecture transactions mapped to Serial Bus

Replace the second paragraph in 7.4 with the following:

All Serial Bus nodes shall implement support for transaction data requests with a transaction type of READ or WRITE, a data length of four, a destination address that is quadlet aligned, and a packet format of QUADLET TCODE. These correspond to the read4 and write4 requests of the CSR Architecture.

All other transaction support, i.e., transaction data requests with a data length other than four, a destination address that is not quadlet aligned, or lock requests, is optional.

NOTE—Transaction support for block reads or writes for some arbitrary data length n does not necessarily imply transaction support for any other length block read or write.

8. Serial Bus management specification

8.2 Serial Bus management services

8.2.1 Serial Bus control request (SB_CONTROL.request)

Insert the following after the second dashed list:

A Serial Bus reset has the potential to disrupt isochronous data flow. Isochronous devices may be designed to compensate adequately for occasional disruptions, but until some time elapses subsequent to the bus reset and equilibrium is reestablished they may be more vulnerable to disruption than before. Any additional bus resets that occur during this time increase the likelihood that users perceive an interruption in isochronous data flow.

For this reason, applications, the bus manager, and the node controller should not make an SB_CONTROL.*request* that specifies a Reset action until 2 s have elapsed subsequent to the completion of the self-identify process that follows a bus reset—except for the purpose of confirming that a uniform gap_count has been set by a transmitted PHY configuration packet.

When `gap_count` has a value other than 63, bus resets initiated by software should be immediately preceded by the transmission of a PHY configuration packet with a nonzero *T* bit and *gap_cnt* equal to the current value of `gap_count`. Without this precaution, the bus manager is almost certain to transmit a PHY configuration packet to restore the optimal value of `gap_count` and then generate an additional bus reset (see 8.4.6.2 for details).

8.2.3 Serial Bus event indication (SB_EVENT.indication)

Replace the definition of the DUPLICATE CHANNEL DETECTED and UNEXPECTED CHANNEL DETECTED bus event parameters with the following:

— DUPLICATE CHANNEL DETECTED (Optional). A stream packet was received with a channel number equal to one of the node's active, transmit isochronous channels.

— UNEXPECTED CHANNEL DETECTED (Optional, may be signaled only at the active isochronous resource manager). The isochronous resource manager observed a stream packet whose channel number is not allocated in the CHANNELS_AVAILABLE register.

8.3 Serial Bus management facilities

Background

As a consequence of experience with products designed in accordance with IEEE Std 1394-1995, as well as the creation of new responsibilities for the isochronous resource manager and the concomitant necessity for a hierarchy of root capabilities, the relationship of node capabilities is revised by IEEE Std 1394a-2000.

Replace 8.3.1 with the following:

8.3.1 Node capabilities taxonomy

Node capabilities are ranked according to functionality, from the least capable repeater nodes to the most fully capable bus manager nodes. For some of the capabilities, namely cycle master, isochronous resource manager, and bus manager, at most one instance shall be active on a bus at any given moment. The configuration procedures used to determine which nodes among the candidate nodes actually assume these roles are described in 8.4.1 and 8.4.2.

8.3.1.1 Repeater (cable environment)

All multiple port nodes present on Serial Bus in the cable environment are, by definition, repeater nodes. This is the minimum capability required and consists of an active physical layer. The PHY may be powered from the bus (via the power/ground pair in the Serial Bus cable) or from some other source. Repeater nodes shall

a) Have an active physical layer.
b) Function as an accurate signal repeater to propagate the signal state from the PHY port conditioned for reception to all other PHY ports conditioned for transmission.
c) Participate in the cable initialization and normal arbitration phases.
d) Be capable of functioning as the root of a Serial Bus.
e) Reconfigure their operational characteristics in response to PHY configuration packets.

8.3.1.2 Transaction capable

Any node on a Serial Bus with an enabled link layer shall be transaction capable, i.e., capable of origination of and response to asynchronous transactions with other transaction capable nodes on the bus. In the cable environment, a repeater node that contains an unpowered or inactive link layer may be transformed into a transaction capable node by means of a PHY link-on packet. Transaction capable nodes shall

a) In the backplane environment, have an active physical layer, or
   In the cable environment, implement all the capabilities of repeater nodes
b) Have an active link layer
c) Implement the STATE_CLEAR, STATE_SET, NODE_IDS, RESET_START, and SPLIT_TIMEOUT registers

NOTE—An active link layer may be in a state of reduced functionality in which it can detect packets addressed to the node and return an *ack_tardy* but not generate any other responses.

8.3.1.3 Isochronous capable

Any node on a Serial Bus that can either send or receive isochronous packets is isochronous capable. Isochronous capable nodes shall

a) Implement all the capabilities of transaction capable nodes
b) Implement configuration ROM in the general ROM format
c) Implement the CYCLE_TIME register and a free-running 24.576 MHz clock that updates the CYCLE_TIME register

8.3.1.4 Cycle master capable

For a Serial Bus to be capable of isochronous operations, there shall be a cycle master. Through a process described in 8.4.1.3 and 8.4.2.6, a single cycle master shall be selected from possible candidates after bus reset. A node that is capable of becoming the cycle master shall

a) Implement all the capabilities of isochronous capable nodes
b) Be able to generate cycle start events triggered by the 8 kHz clock synchronized to the CYCLE_TIME register and broadcast the corresponding cycle start packets
c) Implement the BUS_TIME register

8.3.1.5 Isochronous resource manager capable

For a Serial Bus to be capable of both isochronous operations and asynchronous stream packet transmission on a default broadcast channel, there shall be an isochronous resource manager. Through a process described for the backplane environment by 8.4.1.2 and for the cable environment by 8.4.2.3, a single isochronous resource manager shall be selected from possible candidates after a bus reset. A node that is capable of becoming the isochronous resource manager shall

a) Implement all the capabilities of transaction capable nodes
b) Implement the BUS_MANAGER_ID, BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE, and BROADCAST_CHANNEL registers
c) In the cable environment, be able to analyze received self-ID packets in order to correctly determine the physical ID of the isochronous resource manager node from all the contenders for this role
d) Implement configuration ROM in the general format
e) Execute the responsibilities of the isochronous resource manager specified by 8.4.2.3

NOTE—In a sense, the name *isochronous resource manager* is misleading since such a node does not directly "manage" isochronous resources such as bandwidth and channels. In actual fact, the isochronous resource manager provides a single location where other Serial Bus nodes may cooperatively record their usage of isochronous resources.

In the absence of a bus manager, the isochronous resource manager may perform the following bus management functions:

— Gap count optimization
— Limited power management
— Set a node's `force_root` flag to TRUE, subject to the requirements of 8.4.2.6A

8.3.1.6 Bus manager capable (cable environment)

The most fully capable nodes on a Serial Bus are those that may become bus managers. Bus manager capable nodes shall be isochronous resource manager capable, may provide advanced power management, and may provide facilities to optimize Serial Bus performance and describe the topology of the bus. Through a process described in 8.4.2.5, a single bus manager shall be selected from all bus manager capable nodes; this occurs any time there is a bus reset. A node that is capable of becoming the bus manager shall

a) Implement all the capabilities of isochronous resource manager capable nodes, and
b) Implement the TOPOLOGY_MAP registers.

NOTE—Just as the term *isochronous resource manager* is somewhat misleading, so is the description *bus manager* with respect to some of its functions. With respect to the topology information, the bus manager does not "manage" this information as much as it publishes it. However, for Serial Bus optimization and power management, the bus manager takes a more active role and may actually configure nodes on the Serial Bus.

8.3.2 Command and status registers

8.3.2.2 CSR Architecture core registers

8.3.2.2.1 STATE_CLEAR register

Background

As defined by IEEE Std 1394-1995, the operations of an incumbent cycle master may resume immediately after a bus reset. The intent is to disrupt isochronous operations as little as possible when a bus reset occurs. However, because of Serial Bus topology changes, there may be a new root node subsequent to a bus reset. As specified by IEEE Std 1394-1995, the new root shall not commence cycle master operations until enabled by either the bus manager or isochronous resource manager. If the new root is cycle master capable, it would be desirable for it to commence cycle master operations automatically.

Replace figure 8-2 and the sentence below it with the following:

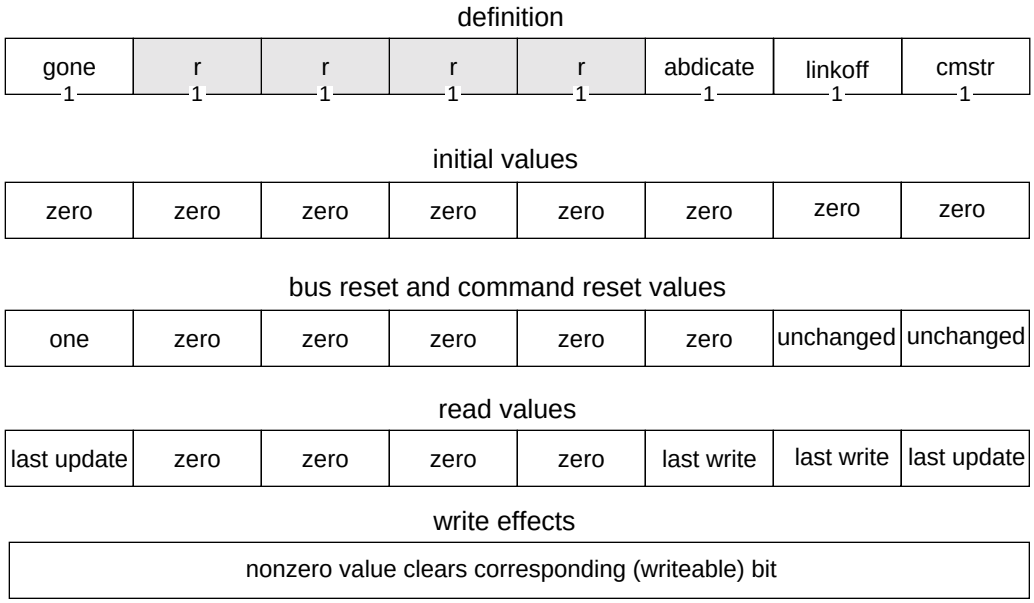


Figure 8-2 — STATE_CLEAR.bus_depend field

The four shaded ***r*** bits are reserved for future definition by Serial Bus.

Insert the following after the third paragraph below figure 8-2:

The ***abdicate*** bit shall be implemented by bus manager-capable nodes; it controls the behavior of the node during contention for the role of bus manager. When *abdicate* is zero, the incumbency of the node prior to a bus reset determines the amount of time the node waits before contending to become the bus manager. As specified by this standard, the incumbent manager contends immediately after the first subaction gap that follows a bus reset while nonincumbent, bus manager-capable nodes wait 125 ms before contending. When *abdicate* is one, the node shall wait 125 ms before contending, whether incumbent or not.

Replace the last two paragraphs in 8.3.2.2.1, including the ordered list, with the following:

Cycle master-capable nodes shall implement the *cmstr* bit. The *cmstr* bit enables the node as a cycle master. A *cmstr* value of one enables cycle master operations while a zero value disables cycle master operations. Only the bus manager or, in the absence of a bus manager, the isochronous resource manager may change the state of *cmstr* by means of a write transaction. Any request that attempts to set *cmstr* to one shall be ignored if the recipient is not the root.

An active cycle master that detects the CYCLE_TOO_LONG event shall clear the *cmstr* bit.

In the cable environment, the value of *cmstr* subsequent to a bus reset is determined as follows:

a) If this node is not the root, the *cmstr* bit shall be cleared to zero, else
b) If this node had been the root prior to the bus reset, *cmstr* shall retain its prior value
c) Otherwise, *cmstr* shall be set to the value of the cmc bit (from the bus information block)

Cycle master operations are controlled by the *cmstr* bit in the STATE_SET register defined in 8.3.2.2.2.

Replace 8.3.2.2.3 with the following:

8.3.2.2.3 NODE_IDS register

The NODE_IDS register reports and permits modification of a node's bus ID and physical ID. Together these form a 16-bit node ID used by the link to determine if a primary packet is addressed to the node. Serial Bus reserves the 16-bit bus-dependent field, as indicated by the shaded field within figure 8-3.

The 10-bit read/write ***bus_ID*** field provides software with a mechanism for reconfiguration of the initial node address space. The *bus_ID* field permits node addresses on one bus to be distinguished from those on another. All nodes on a bus shall have identical *bus_ID* values.

NOTE—A bus consists of all physically connected nodes that are within the same arbitration domain, i.e., nodes that receive their arbitration grant(s) from the same root node.

The 6-bit ***local_ID*** field shall have a value generated as a side-effect of the bus initialization process. Within this standard, the value of NODE_IDS.*local_ID* is also known as the physical ID of the node. This field is read-only in the cable environment and read/write in the backplane environment.

NOTE—The CSR Architecture requires that if there are any side-effects of a nonbroadcast write transaction to a register, the affected node delay the return of a transaction response until all effects of the write are complete. In the case of the NODE_IDS register, a return of *resp_complete* indicates that the node recognizes transactions to the newly assigned NODE_IDS value. The contents of the *source_ID* field of the response packet are required to be equal to the most significant 16 bits of the updated NODE_IDS register.

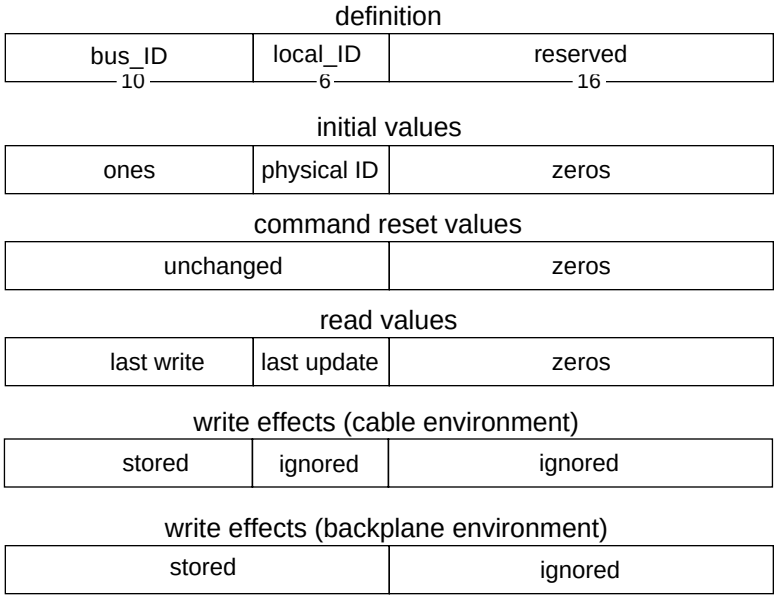


Figure 8-3 — NODE_IDS format

Replace 8.3.2.2.4 with the following:

8.3.2.2.4 Command reset effects

A write to the RESET_START register performs an immediate command reset, as defined in the CSR Architecture. A command reset shall have no effects upon any of the Serial Bus-dependent registers defined by this standard.

Replace 8.3.2.2.6 with the following:

8.3.2.2.6 SPLIT_TIMEOUT register

Split-transaction error detection requires that all nodes on Serial Bus share the same time-out value and that requester and responder behave in complementary fashion. The SPLIT_TIMEOUT register establishes the time-out value for the detection of split-transaction errors. The value of SPLIT_TIMEOUT is the maximum time permitted for the receipt of a response subaction after the transmission of a request subaction. After this time, a responder shall not transmit a response for the request subaction and a requester shall terminate the transaction with a request status of TIMEOUT. For a requester the time-out period commences when an *ack_pending* is received in response to a request subaction. A responder starts the time-out period when an *ack_pending* is transmitted. Figure 8-4 illustrates the portions of the SPLIT_TIMEOUT register implemented on Serial Bus.

NOTE—A requester should not reuse the transaction label from an expired request subaction in a subsequent request subaction to the same node unless at least twice the split time-out period has elapsed since the initiation of the expired subaction.

The ***sec*** field, in units of seconds, and the ***cycles*** field, in units of 125 µs, together specify the time-out value. The value of *cycles* shall be less than 8000. The bus manager, if present, shall insure that all nodes on the bus have identical values in their SPLIT_TIMEOUT registers.

The minimum time-out value is 800 cycles (0.1 s). If a value smaller than this is written to the SPLIT_TIMEOUT register the node shall not round the stored value but shall behave as if a value of 800 had been stored.

NOTE—The Serial Bus definition of the SPLIT_TIMEOUT register differs from that of the CSR Architecture. Serial Bus interprets the most significant 13 bits of the SPLIT_TIMEOUT_LO register as units of 1/8000 s, rather than a true binary fraction of a second with units of 1/8192 s. Since precise time-outs are not necessary, the bus manager may ignore this difference when calculating values for use within the SPLIT_TIMEOUT_LO register.

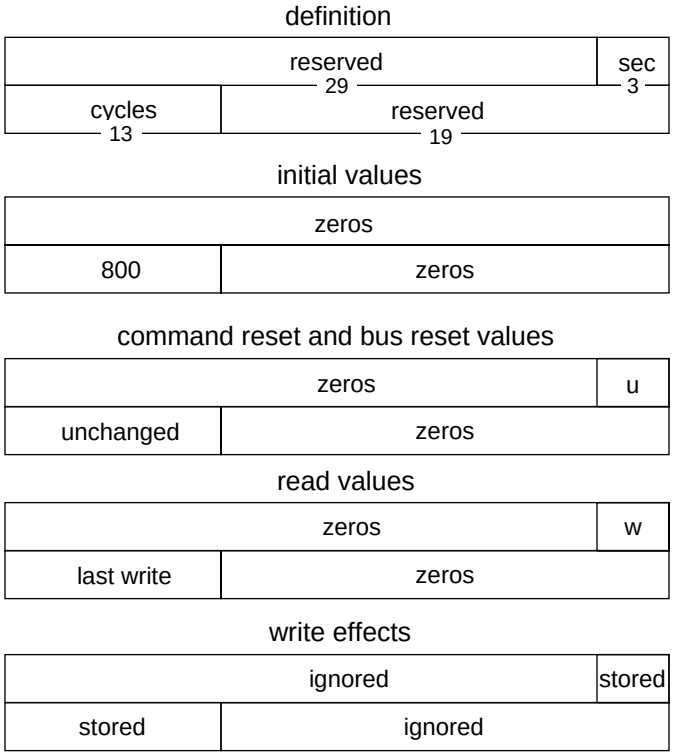


Figure 8-4 — SPLIT_TIMEOUT format

8.3.2.3 Serial-Bus-dependent registers

Replace table 8-3 with the following:

Table 8-3 — Serial-Bus-dependent registers

Offset	Name	Notes
200_{16}	CYCLE_TIME	For nodes providing isochronous services.
204_{16}	BUS_TIME	For nodes requiring synchronized bus time.
208_{16}	POWER_FAIL_IMMINENT	For nodes needing power-fail warning.
$20C_{16}$	POWER_SOURCE	
210_{16}	BUSY_TIMEOUT	For transaction-capable nodes.
214_{16}–218_{16}	PRIORITY_BUDGET	For priority arbitration control.

Table 8-3 — Serial-Bus-dependent registers (continued)

Offset	Name	Notes
$21C_{16}$	BUS_MANAGER_ID[a]	For selecting or locating the bus manager.
220_{16}	BANDWIDTH_AVAILABLE[a]	For bandwidth allocation of isochronous-resource-manager-capable nodes.
224_{16}–228_{16}	CHANNELS_AVAILABLE[a]	For channel allocation of isochronous-resource-manager-capable nodes.
$22C_{16}$	MAINT_CONTROL	For introducing diagnostic error conditions.
230_{16}	MAINT_UTILITY	
234_{16}–$3FC_{16}$	BROADCAST_CHANNEL	For communicating the channel number assigned for asynchronous stream broadcast.

[a]Only quadlet read and quadlet lock (compare and swap) transactions supported.

Insert the following after 8.3.2.3.5:

8.3.2.3.5A PRIORITY_BUDGET register

Reserved, backplane environment.

Optional, cable environment. This register shall be implemented on nodes that use asynchronous priority arbitration for the primary packets enumerated by table 8-3A and, if implemented, shall be located at offset 218_{16} within initial register space.

The PRIORITY_BUDGET register permits the bus manager to configure a node's asynchronous arbitration behavior. This register provides a mechanism for a node to be granted permission to use priority arbitration during the asynchronous period. The definition is given by figure 8-9A.

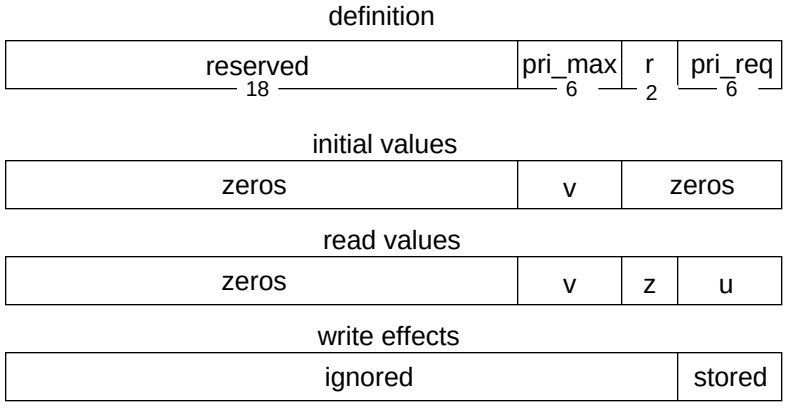


Figure 8-9A—PRIORITY_BUDGET format

The ***pri_max*** field shall specify the maximum value the node expects to be stored in *pri_req*. If a write request attempts to update *pri_req* to a larger value, the result is unspecified.

The ***pri_req*** field shall specify the maximum number of certain priority arbitration requests the link is permitted to make of the PHY during a fairness interval. The primary packet transaction codes for which priority arbitration may be used

are specified by table 8-3A; PHY packets and response packets may also use priority arbitration (see 6.3.1.1A). A Serial Bus fairness interval exists between the occurrence of an arbitration reset gap and the first subsequent arbitration reset gap. The *pri_req* default value of zero is equivalent to the fair arbitration behavior specified by this standard; any priority requests enabled by a nonzero value of *pri_req* are in addition to the fair arbitration request permitted each node.

Table 8-3A — Request subactions eligible for priority asynchronous arbitration

tcode	Name	Comment
0	Write request for data quadlet	Request subaction, quadlet payload
1	Write request for data block	Request subaction, block payload
4	Read request for data quadlet	Request subaction, no payload
5	Read request for data block	Request subaction, quadlet (*data_length*) payload
9	Lock request	Request subaction, block payload
A_{16}	Stream data	Asynchronous subaction, block payload

Each time a link receives PHY status of ARB_RESET_GAP, it shall reset an internal variable, *priority_request_count*, to the value of *pri_req*. The link may use priority asynchronous arbitration for any of the transaction codes specified by table 8-3A so long as *priority_request_count* is nonzero. The link may also issue a single priority arbitration request in place of a fair arbitration request if no fair arbitration request has been granted within the current fairness interval. Even if either of those two conditions is met, if a node receives an *ack_busy_X*, *ack_busy_A*, or *ack_busy_B* in acknowledgment of a request subaction, the node shall not retransmit the request packet until the next fairness interval. Each time a priority arbitration request is granted (either as the result of a link request or the use of the *Hold* protocol to concatenate a packet) for one of the transaction codes specified and *priority_request_count* is nonzero, the link shall decrement *priority_request_count*.

NOTE—IEEE Std 1394-1995 specifies only one use for the priority arbitration request from the link to the PHY to arbitrate for the bus in order to transmit a cycle start packet. IEEE Std 1394a-2000 does not change that use of a priority request and it does not count against the *priority_request_count* maintained by the link.

The bus manager shall ensure that the sum of the values of *pri_req* in the PRIORITY_BUDGET registers of all nodes on the local bus is less than or equal to 63 minus the number of nodes.

8.3.2.3.7 BANDWIDTH_AVAILABLE register

Insert the following text, table, and note at the end of 8.3.2.3.7.

Designers are strongly encouraged to implement behavior for the BANDWIDTH_AVAILABLE register described by table 8-3B. Although this algorithm is, strictly speaking, not compliant with the definition of compare_swap in 7.3.4.3, its behavior is functionally equivalent so long as lock requests addressed to the BANDWIDTH_AVAILABLE register conform to the assumption that the register represents an unsigned integer.

The variables *arg_value* and *data_value* correspond to the same fields in the lock request; *old_value* and new_value represent the value of BANDWIDTH_AVAILABLE during the operations, and *old_value* is returned in the lock response.

The performance of this algorithm is superior to that specified by 7.3.4.3, since bandwidth allocation attempts always succeed if there is enough bandwidth remaining, whether or not the requester knew the actual quantity of bandwidth remaining. A similar observation holds for bandwidth deallocation, although the performance improvements are insignificant since deallocation is not usually performed at times when Serial Bus is critically busy.

Table 8-3B — Lock (compare_swap) algorithm for BANDWIDTH_AVAILABLE

```
unsigned old_value;                          // Current value of BANDWIDTH_AVAILABLE

unsigned compare_swap_bandwidth(unsigned arg_value, unsigned data_value) {
   unsigned bandwidth;

   if (arg_value > 0x1FFF)                   // Impossible BANDWIDTH_AVAILABLE value?
      return(old_value);
   data_value &= 0x1FFF;                     // Mask for bw_remaining field
   if (arg_value >= data_value) {            // Allocation or deallocation?
      bandwidth = arg_value - data_value;    // Bandwidth to allocate
      if (old_value >= bandwidth) {          // Enough bandwidth remaining?
         new_value = old_value - bandwidth;
         return(arg_value);
      } else
         return(old_value);
   } else {
      bandwidth = data_value - arg_value;    // Bandwidth to deallocate
      if (old_value + bandwidth < 0x2000) {  // Will it overflow bw_remaining field?
         new_value = old_value + bandwidth;
         return(arg_value);
      } else
         return(old_value);
   }
}
```

NOTE—An independent observer capable of monitoring all Serial Bus subactions and their relative order in time could detect that this algorithm does not comply with the requirements of ISO/IEC 13213:1994. However, an individual requester lacking in such omnipotent knowledge has no way to determine that the behavior is noncompliant. By the time the originator of a lock request could transmit a read request to the isochronous resource manager's BANDWIDTH_AVAILABLE register another node might have changed its value.

8.3.2.3.8 CHANNELS_AVAILABLE register

Replace figure 8-12 with the following:

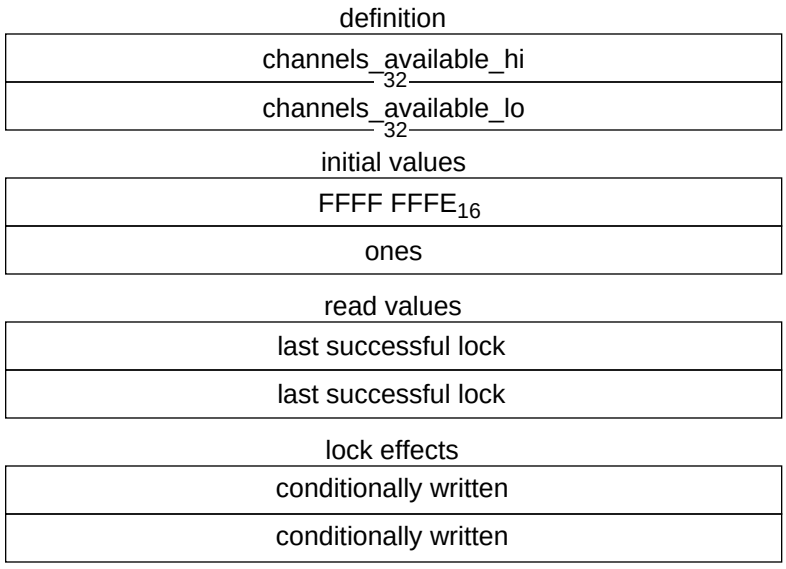


Figure 8-12 — CHANNELS_AVAILABLE format

Insert the following text and table at the end of 8.3.2.3.8:

The contents of the CHANNELS_AVAILABLE register are valid only at the isochronous resource manager. Channel 31 is automatically allocated by the isochronous resource manager for use as the default broadcast channel for asynchronous stream packets.

For reasons analogous to those already presented for the BANDWIDTH_AVAILABLE register, designers are strongly encouraged to implement behavior for each of the quadlets of the CHANNELS_AVAILABLE register described by table 8-3C. Although this algorithm is, strictly speaking, not compliant with the definition of compare_swap in 7.3.4.3, its behavior is functionally equivalent so long as lock requests addressed to the CHANNELS_AVAILABLE register conform to the assumption that the register represents a bit map.

The variables *arg_value* and *data_value* correspond to the same fields in the lock request; *old_value* and *new_value* represent the value of CHANNELS_AVAILABLE during the operations, and *old_value* is returned in the lock response.

Table 8-3C — Lock (compare_swap) algorithm for CHANNELS_AVAILABLE

```
unsigned old_value;                          // Current value of CHANNELS_AVAILABLE_xx

unsigned compare_swap_channels(unsigned arg_value, unsigned data_value) {
   unsigned affected_channels = arg_value ^ data_value;

   if (   (arg_value & affected_channels)       // Does expected value of affected channels
       == (old_value & affected_channels)) {    // match their actual value?
      new_value = old_value ^ affected_channels;
      return(arg_value);
   } else
      return(old_value);
}
```

Insert the following after 8.3.2.3.10:

8.3.2.3.11 BROADCAST_CHANNEL register

Optional. This register, if implemented, shall be a quadlet register located at offset 234_{16} within initial register space. All isochronous-resource-manager-capable nodes shall implement the BROADCAST_CHANNELS register.

The BROADCAST_CHANNEL register permits the isochronous resource manager to communicate the channel number assigned for asynchronous stream broadcast to other nodes on Serial Bus. The BROADCAST_CHANNEL register shall support quadlet read and write requests only. The definition is given by figure 8-14A.

The most significant bit (a constant one) differentiates the presence of the BROADCAST_CHANNEL register from the value (all zeros) that may be returned when this register's address is read at node(s) that do not implement the register.

NOTE—Nodes compliant with this standard return an address error response when unimplemented addresses are accessed, but some implementations are known to complete such requests with *resp_OK* and response data of zeros.

The ***valid*** bit (abbreviated as v above), when set to one, indicates that the default broadcast channel specified by the *channel* field may be used. Nodes shall not transmit stream packets that specify this channel while the *valid* bit in their own BROADCAST_CHANNEL register is zero.

The ***channel*** field shall identify the channel number shared by all nodes for asynchronous stream broadcast.

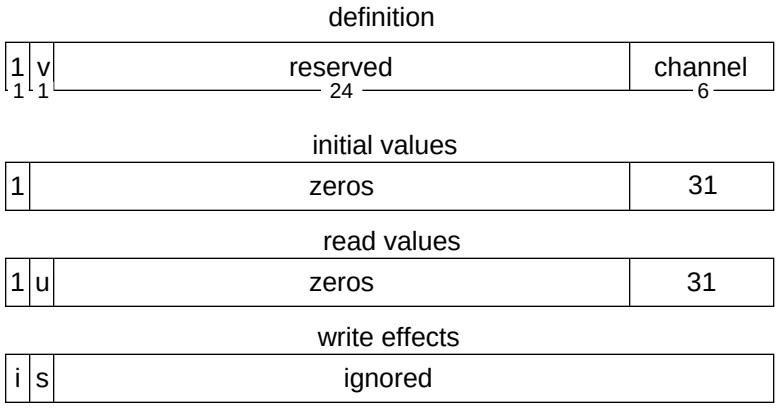


Figure 8-14A — BROADCAST_CHANNEL format

8.3.2.4 Unit registers

Background

Subclause 8.3.2.4 reserves a range of addresses in initial units space for Serial Bus-dependent or other uses, notably the TOPOLOGY_MAP and SPEED_MAP registers that are defined. Additional portions of that address space have been utilized both by this standard and by other draft standards.

Replace table 8-4 with the following table and text:

Table 8-4 — Serial Bus-dependent registers in initial units space

Offset	Name	Notes
$800_{16} – 8FC_{16}$	—	Reserved for Serial Bus
900_{16}	OUTPUT_MASTER_PLUG	Specified by IEC 61883-1 (1998-02)
$904_{16} – 97C_{16}$	OUTPUT_PLUG	
980_{16}	INPUT_MASTER_PLUG	
$984_{16} – 9FC_{16}$	INPUT_PLUG	
$A00_{16} – AFC_{16}$	—	Reserved for Serial Bus
$B00_{16} – CFC_{16}$	FCP command frame	Specified by IEC 61883-1 (1998-02)
$D00_{16} – EFC_{16}$	FCP response frame	
$F00_{16} – FFC_{16}$	—	Reserved for Serial Bus
$1000_{16} – 13FC_{16}$	TOPOLOGY_MAP	Present at the bus manager, only.
$1400_{16} – 1FFC_{16}$	—	Reserved for Serial Bus
$2000_{16} – 2FFC_{16}$	SPEED_MAP	Obsoleted
$3000_{16} – FFFC_{16}$	—	Reserved for Serial Bus

Except as specified by this standard or future IEEE Serial Bus standards, unit architectures shall not implement any CSRs that fall within the above address space.

8.3.2.4.2 SPEED_MAP registers (cable environment)

Background

As a consequence of the addition of the *link_spd* field to the bus information block, the SPEED_MAP registers specified by 8.3.2.4.2 may contain unreliable information and is obsoleted by this standard. This clause should be considered unimplemented (i.e., any access results in an address error). Bus managers that implement the SPEED_MAP registers as specified by IEEE Std 1394-1995 are compliant with this standard but users are cautioned that the addresses utilized by these registers may be redefined in future IEEE standards.

Delete 8.3.2.4.2 in its entirety.

8.3.2.5 Configuration ROM

Replace the title for 8.3.2.5.4 with the following:

8.3.2.5.4 Configuration ROM Bus_Info_Block

Background

Several new fields are specified for the Bus_Info_Block in order to support enhanced capabilities defined by this standard:

— Immediately subsequent to a bus reset nodes might generate a flurry of quadlet read requests to ascertain the identity, by means of the EUI-64 (Extended Unique Identifier, 64 bits), of nodes whose physical ID may have been reassigned in the self-identify process. The volume of read requests may be minimized if it is not necessary to reread all (or a significant part) of a node's configuration ROM. A new field, *generation*, has been added whose purpose is to indicate whether or not configuration ROM has changed from one bus reset to the next. In addition, the *max_ROM* field permits discovery of the largest block reads that may be used to read configuration ROM.

— Outside of IEEE P1394a standardization activities, work is underway to define power management for Serial Bus. A new entity, the power manager, may manage power distribution and consumption in the cable environment. A new bit, *pmc*, is defined to identify nodes capable of power management.

— IEEE Std 1394-1995 does not specify how both the link and the PHY maximum speed capabilities shall be reported when they differ—nor does it require all link and PHY combinations to support the same speed capabilities. IEEE Std 1394a-2000 adds a new field, *link_spd*, to the Bus_Info_Block to permit the speeds to be reported independently.

The revised format of the Bus_Info_Block follows. With the exception of *max_rec*, the definitions of all fields previously specified by IEEE Std 1394-1995 are unchanged.

Replace figure 8-20 in 8.3.2.5.4 with the following:

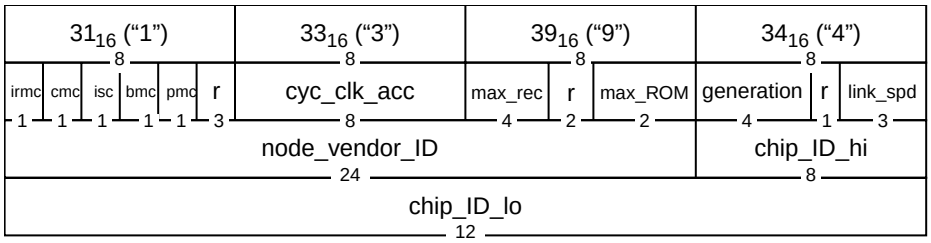


Figure 8-20 — Bus_Info_Block format

Insert the following paragraph after the definition of the bmc bit in 8.3.2.5.4:

The ***pmc*** bit when set to one indicates the node is power manager-capable. A node that sets its *pmc* bit to one shall also set its *bmc* bit to one to indicate bus manager capabilities. The capabilities and responsibilities of a power manager-capable node are beyond the scope of this standard.

Replace the definition of the max_rec field in 8.3.2.5.4 with the following:

The ***max_rec*** field defines the maximum data payload size that the node supports. The data payload size applies to block write requests or asynchronous stream packets addressed to the node and to block read responses transmitted by the node. When *max_rec* is zero the maximum data payload is unspecified. Otherwise, it is equal to 2^{max_rec+1} bytes, as defined by table 8-5. If *max_ROM* is nonzero, *max_rec* shall be greater than or equal to $2^{max\_ROM+1} + 1$.

Table 8-5 — Encoding of max_rec field

Code	Maximum data payload (bytes)
0	Not specified
1	4
2	8
3	16
4	32
5	64
6	128
7	256
8	512
9	1024
A_{16}	2048
B_{16}	4096
C_{16}	8192
D_{16}	16384
E_{16} to F_{16}	Reserved

The maximum isochronous data payload supported by the node, either as a talker or listener, is not governed by *max_rec*.

Insert the following in 8.3.2.5.4 before the definition of the node_vendor_id field:

The ***max_ROM*** field shall specify the size and alignment of read requests supported by configuration ROM, whether within the address range FFFF F000 0400_{16} through FFFF F000 $07FF_{16}$ inclusive or another portion of the node's address space, as specified by table 8-5A.

NOTE—Devices that report a *max_ROM* value of zero should support block read requests capable of returning the first five quadlets of configuration ROM (which includes the entire contents of the bus information block) in one transaction even if block read requests are not supported for any other portion of configuration ROM.

Table 8-5A — Encoding of max_ROM field

Code	Meaning
0	Quadlet read requests are supported. This encoding is suggested for legacy devices and should not be reported by devices compliant with this standard.
1	Quadlet read requests and block read requests aligned on 64-byte addresses with a data length of 64 bytes are supported.
2	Quadlet read requests and block read requests aligned on quadlet addresses with a data length less than or equal to 1024 bytes are supported.
3	Reserved for future standardization.

The ***generation*** field is used to indicate changes in configuration ROM. Devices that comply with IEEE Std 1394-1995 (but not with IEEE Std 1394a-2000) shall report a value of zero for the *generation* field. Devices compliant with IEEE Std 1394a-2000, whose configuration ROM never changes (so long as the device's link is continuously active) shall set the *generation* field to one. All other devices compliant with this standard shall set the *generation* field to a value between two and F_{16}, inclusive. For these devices, upon the detection or initiation of a bus reset, the *generation* field shall be modified if any portion of configuration ROM has changed since the prior bus reset. The updated value of the *generation* field shall not be equal to any values assumed by the field within the preceding 60 s. Configuration ROM includes not only the first kilobyte of ROM (quadlets in the address range FFFF F000 0400_{16} through FFFF F000 $07FC_{16}$, inclusive) but any directories or leaves that are indirectly addressed from the first kilobyte. The CRC in the first quadlet of configuration ROM shall be recalculated each time the *generation* field is updated.

NOTE—The *generation* field is usually incremented upon a change to configuration ROM; the value wraps from F_{16} back to two. If an update would result in a value used within the last 60 s, the device should defer any changes to configuration ROM until the requisite 60 s have elapsed.

The *link_spd* field shall report the maximum speed capability of the node's link layer; the encoding used is the same as for the PHY register *Max_speed* field defined in table 5B-1.

8.3.2.5.5.3 Node_Unique_Id entry

Background

The requirements of 8.3.2.5.5.3 for a Node_Unique_Id leaf and a root directory entry to address it are removed. Since the Node_Unique_Id leaf is no longer required, the corresponding root directory entry that addresses it is made obsolete.

Delete 8.3.2.5.5.3 in its entirety.

8.3.2.5.7.1 Node_Unique_Id leaf

Background

The requirements of 8.3.2.5.7.1 for a Node_Unique_Id leaf and a root directory entry to address it are removed. Since the same information is required in the Bus_Info_Block, the node unique Id leaf is made obsolete.

Delete 8.3.2.5.7.1 in its entirety.

8.4 Serial Bus management operations

8.4.2 Bus configuration procedures (cable environment)

Background

As a consequence of the definition of the BROADCAST_CHANNEL register, an isochronous resource manager compliant with IEEE Std 1394a-2000 implements greater functionality than that specified by IEEE Std 1394-1995.

Replace 8.4.2.3 with the following:

8.4.2.3 Determination of the isochronous resource manager (cable environment)

Subsequent to bus reset, each node participates in the self-identify process described in 4.4.3.3. The link and higher layers of all nodes that are contenders for the role of the isochronous resource manager shall observe all self-ID packets to determine the identity of the single node selected as isochronous resource manager.

From all the nodes that are capable of becoming the isochronous resource manager, one is selected as follows. An isochronous resource manager-capable node that wishes to contend for the role of isochronous resource manager shall, during the self-identify process, transmit its own self-ID packet with both the *c* bit (contender) and the *L* bit (link active) set to one. Each of these contenders shall also monitor all received self-ID packets in order to observe the largest physical ID from a packet with both the *c* and *L* bits set. The candidate node with the largest physical ID wins the role of the isochronous resource manager.

Contenders shall apply consistency checks to the observed self-ID packets to ensure that

— The second quadlet of each self-ID packet is the logical inverse of the first quadlet.

— The *phy_ID* fields of self-ID packet zero are in monotonically increasing sequence and the *phy_ID* fields of all self-ID packets other than packet zero are equal to the *phy_ID* field of the preceding self-ID packet.

— The set of self-ID packets for the node with the largest physical ID indicates that all its active PHY ports are connected to children.

If any of these requirements is not met, the contender shall initiate a bus reset as soon as possible.

A contender that becomes the isochronous resource manager shall update its own BROADCAST_CHANNEL register by setting the *valid* bit to one; it may notify other nodes that channel 31 has been allocated as the default broadcast channel by updating their BROADCAST_CHANNEL register(s) with the contents of its own BROADCAST_CHANNEL register, by means of either a broadcast write request or write requests addressed to individual nodes.

Because the self-ID packets do not indicate whether a contender is compliant only with IEEE Std 1394-1995 (and does not implement the BROADCAST_CHANNEL register) or is compliant with IEEE Std 1394a-2000, contenders shall determine whether or not the current isochronous resource manager is compliant with IEEE Std 1394a-2000 by using at least one of the following tests:

— By the receipt of a write request addressed to the node's own BROADCAST_CHANNEL register that sets the valid bit to one.

— By the successful completion of a read request addressed to the isochronous resource manager's BROADCAST_CHANNEL register and the return of response data that shows the most significant bit to be one.

— By the successful read of the isochronous resource manager's bus information block that shows the *generation* field to be nonzero.

— If an analysis of self-ID packets indicates that, subsequent to a bus reset, the prior isochronous resource manager has continued as the isochronous resource manager (but only if the prior isochronous resource manager was known to be compliant with this standard).

If the current isochronous resource manager is not compliant with this standard, a contender that additionally complies with the requirements of 8.4.2.6A shall set its own `force_root` variable to TRUE and initiate a bus reset in order to become the isochronous resource manager.

NOTE—A contender that is also bus manager capable may become the bus manager at a later step in the configuration process, whether or not it wins the role of the isochronous resource manager.

8.4.2.5 Determination of the bus manager (cable environment)

Background

Subclause 8.4.2.5 describes the use of the BUS_MANAGER_ID register to determine the identity of the bus manager subsequent to a bus reset. The text is misleading where it describes the return of an *old_value* of $3F_{16}$ as the only way in which a candidate bus manager is confirmed as the new bus manager. If a candidate bus manager successfully completes a lock (compare_swap) request to the BUS_MANAGER_ID register but the response packet is corrupted the candidate shall retry the lock request as described. In this case the *old_value* returned in response to the retry is not the anticipated $3F_{16}$, but is instead the physical ID of the bus manager.

Replace the last two sentences in the first paragraph of 8.4.2.5 with the following:

If the *old_value* received in the lock response packet is $3F_{16}$ or the physical ID of the incumbent bus manager, the incumbent bus manager has reestablished itself as the bus manager. If any other value is returned, it is the physical ID of the node that has won the role of bus manager; the incumbent manager has lost and shall not perform any of the functions of the bus manager.

8.4.2.6 Determination of the cycle master (cable environment)

Background

Subclause 8.4.2.6 requires the bus manager or, in the absence of a bus manager, the isochronous resource manager, to set the root node's `force_root` variable to TRUE if the root is cycle master-capable.[8] The `force_root` variable is cleared to FALSE at all other nodes; this promotes stability across bus resets since it increases the likelihood that the same node becomes the root after subsequent bus resets. Although no other rationale is given for setting a particular node's `force_root` variable to TRUE, IEEE Std 1394-1995 does not explicitly forbid the use of `force_root` for other purposes. IEEE Std 1394a-2000 establishes additional policy for the use of the `force_root` variable to influence the selection of the root node during the tree identify process that follows a bus reset.

Insert the following subclause after 8.4.2.6:

8.4.2.6A Determination of the root (cable environment)

A node's `force_root` variable shall not be set to TRUE unless the node is more capable than the current root.

[8]IEEE Std 1394-1995 also describes how the bus manager or isochronous resource manager selects a different, cycle master-capable node and sets its `force_root` variable to TRUE if the current root cannot function as the cycle master—this process eventually produces a root whose `force_root` variable is TRUE.

This standard creates an ordering of root node capabilities, which may be extended by future IEEE standards. The most basic of root node capabilities is inherent in all nodes whether or not link and transaction layers are implemented (see 8.3.1.1). Additional capabilities may be implemented by transaction-capable nodes and are enumerated below in increasing order of capability as follows:

a) *Cycle master-capable.* When an isochronous resource manager or bus manager is present, the root shall be cycle master-capable. A cycle master-capable node may be positively identified by the cmc bit in the bus information block or its presence may be inferred by the observation of cycle start packets.
b) *Enhanced isochronous resource manager capable.* This standard defines new functionality for the isochronous resource manager—the ability to allocate channel 31 as the default channel for asynchronous stream broadcast. Such a node may be positively identified by its implementation of the BROADCAST_CHANNEL register, by a nonzero *generation* field in its bus information block, or its presence may be inferred by the receipt of a write addressed to the BROADCAST_CHANNEL register.

In order for this scheme to be extensible, it is crucial that nodes compliant with this and future standards adhere to these requirements and not set their own `force_root` variable to TRUE unless they are more capable than the current root. This is particularly important in the case where the current root's `force_root` variable is cleared to FALSE in order that another node become the root. The former root shall not attempt to reestablish itself as the root if the new root is at least as capable as the former root. It is likely that the new root is more capable and that the former root is unable to detect its enhanced capabilities.

Subclause 8.4.2.3 conforms to these requirements; the only reason a candidate isochronous resource manager is permitted to set its own `force_root` variable TRUE is because it implements all root node capabilities defined by this standard. If a future IEEE standard specifies new functionality for which it is desirable that a particular node be the root, that node shall be both cycle master and enhanced isochronous resource manager capable in addition to its new capabilities.

8.4.3 Isochronous management (cable environment)

Replace the head for 8.4.3 with the following:

8.4.3 Isochronous resource allocation (cable environment)

8.4.3.1 Bandwidth allocation

Background

With respect to bandwidth allocation, the algorithm described by IEEE Std 1394-1995 requires that the BANDWIDTH_AVAILABLE register first be read to determine the bandwidth available before attempting an allocation or deallocation. This is unnecessary and discouraged. Instead the algorithm should start with a lock request with an *extended_tcode* of compare_swap. The *arg_value* should contain the most recently known value of the BANDWIDTH_AVAILABLE register (immediately subsequent to a bus reset its value is known to be 4915). The *data_value* shall be equal to *arg_value* less the bandwidth desired. If the lock request is unsuccessful because *arg_value* did not match the current value, the *old_value* returned in the lock response may be used in a retry attempt. The returned *old_value* becomes the new *arg_value* in the retry attempt and *data_value* is recalculated relative to the new *arg_value*.

Although there is less expectation that bandwidth deallocation attempts will succeed on their first try if the modified algorithm is used, there is no performance penalty for attempting a lock request whose *arg_value* is equal to the inferred value of the BANDWIDTH_AVAILABLE register after the most recent lock transaction.

Replace the procedural list and the two subsequent paragraphs with the following:

a) The bandwidth desired shall be calculated to include the bandwidth needed by the application plus all overhead associated with isochronous data transfer, e.g., isochronous gap, arbitration, data prefix and data end. The bandwidth desired shall not exceed the current bandwidth available. If a node desires more bandwidth than is available, the node shall either reduce its request for bandwidth or shall delay some period of time and retry the allocation later.

b) The bandwidth allocation shall be attempted by a lock request with an extended transaction code of compare and swap to the BANDWIDTH_AVAILABLE register at the isochronous resource manager. The lock packet shall have an *arg_value* equal to the value of the current bandwidth available and a *data_value* equal to the bandwidth available less the bandwidth desired.

c) If the lock transaction fails to complete, i.e., the Request Status returned is not COMPLETE or the Response Code is not resp_complete, the node may retry the entire bandwidth allocation procedure.

d) If the lock transaction is successful and the *old_value* received is equal to the *arg_value* transmitted in the lock request, the allocation of isochronous bandwidth is successful. In all other cases, the bandwidth allocation has failed and may be retried as appropriate. A subsequent read of the BANDWIDTH_AVAILABLE register is not necessary, since the *old_value* returned by the failing lock request reflects the current bandwidth available.

When the procedure described above succeeds, the requesting node becomes the owner of the isochronous bandwidth. Isochronous bandwidth shall not be deallocated by any node other than the owner of the bandwidth unless the owner of the bandwidth has requested, by means beyond the scope of this standard, another node to deallocate the bandwidth on behalf of the owner.

Bandwidth deallocation is performed by an essentially similar protocol. The owner of the bandwidth shall use a lock transaction to attempt to increase the currently available bandwidth by the amount of bandwidth returned. Lock requests that fail should be retried until the bandwidth is successfully deallocated.

8.4.3.2 Channel allocation

Background

Arguments analogous to those made for the BANDWIDTH_AVAILABLE register hold sway with respect to the CHANNELS_AVAILABLE register. The algorithm should dispense with a read of the CHANNELS_AVAILABLE register and instead commence with a compare_swap operation whose *arg_value* equals the most recently known value of pertinent quadlet of the CHANNELS_AVAILABLE register. The *data_value* should be derived from the *arg_value* by clearing or setting the bits that correspond to the channels intended for release or allocation, respectively. If the lock request is unsuccessful because *arg_value* did not match the current value of CHANNELS_AVAILABLE, the *old_value* returned in the lock response may be used in a retry attempt. The returned *old_value* becomes the new *arg_value* in the retry attempt and *data_value* is recalculated relative to the new *arg_value*.

Replace the procedural list and the two subsequent paragraphs with the following:

a) If unused channels are available, the request for a channel is made with a lock request with an extended transaction code of compare and swap to the CHANNELS_AVAILABLE register at the isochronous resource manager. The lock packet shall have an *arg_value* equal to the bit mask that represents the current state of channel availability and a *data_value* equal to the same value except that the bit(s) that represents the desired channel(s) shall be cleared to zero. A node shall not attempt to allocate a channel that is in use; in such a circumstance, the node shall either select a different, unused channel or shall delay some period of time and retry the allocation later.

b) If the lock transaction fails to complete, i.e., the Request Status returned is not COMPLETE or the Response Code is not resp_complete, the node may retry the entire channel allocation procedure.

c) If the lock transaction is successful and the *old_value* received is equal to the *arg_value* transmitted in the lock request, the allocation of isochronous channel(s) is successful. In all other cases, the channel allocation has failed and may be retried as appropriate. A subsequent read of the CHANNELS_AVAILABLE register is not necessary, since the *old_value* returned by the failing lock request reflects the current availability of isochronous channels.

When the procedure described above succeeds, the requesting node becomes the owner of the isochronous channel(s). Isochronous channel(s) shall not be deallocated by any node other than the owner of the channel(s) unless the owner of the channel(s) has requested, by means beyond the scope of this standard, another node to deallocate the channel(s) on behalf of the owner.

Channel deallocation is performed by an essentially similar protocol. The owner of the channel shall first read the value of the CHANNELS_AVAILABLE register at the isochronous resource manager in order to obtain an accurate picture of which channels are in use. A subsequent lock transaction shall be used to attempt to set the bit that corresponds to the channel being released. Lock requests that fail should be retried until the channel is successfully deallocated.

8.4.5 Speed management (cable environment)

Background

As a consequence of the addition of the *link_spd* field to the bus information block, the SPEED_MAP registers specified by 8.4.5 in IEEE Std 1394-1995 may contain unreliable information and is obsoleted by this standard. This clause should be considered unimplemented (i.e., any access results in an address error). Bus managers that implement the SPEED_MAP registers as specified by IEEE Std 1394-1995 are compliant with IEEE Std 1394a-2000 but users are cautioned that the addresses utilized by these registers may be redefined in future IEEE standards.

Delete 8.4.5 in its entirety.

8.4.6 Topology management (cable environment)

8.4.6.2 Gap count optimization

Background

The information contained in table 8-7 of IEEE Std 1394-1995 is known to contain errors and shall not be used. This standard provides more accurate methods to determine the optimal gap count for a particular bus topology; these are described in E.1 of IEEE Std 1394a-2000.

Replace the text in 8.4.6.2 with the following:

The value of `gap_count` shall not be reduced to a point where either

a) An asynchronous subaction is interrupted by node(s) other than the source and destination, or

b) Where all nodes fail to consistently and identically detect both subaction and arbitration reset gaps.

Once a value for `gap_count` is selected, the remainder of this subclause specifies the procedures that shall be used to establish that value uniformly for all nodes on the bus.

The bus manager[9] may optimize Serial Bus performance by transmitting a PHY configuration packet (see 4.2.2.7) with the *gap_cnt* field set to a value less than 63 and the *T* bit set to 1. Nodes other than the bus manager may transmit a PHY configuration packet with the *T* bit set to 1 so long as the value of the *gap_cnt* field is equal to the node's `gap_count` variable (obtained by a read of PHY register one).

A node that transmits a PHY configuration packet with the T bit set to one shall initiate a bus reset as soon as possible after the PHY configuration has been sent. This is essential so that the `gap_count_reset_disable` variable at all node(s) is cleared to FALSE. Without this precaution, the subsequent addition of a new node to the bus could result in different values of `gap_count` at different nodes and resultant unpredictable arbitration behavior.

Subsequent to a bus reset, the bus manager shall verify the self-ID packets generated as a result. If the *gap_cnt* field in all the self-ID packets is not identical, the bus manager shall initiate another bus reset. This additional bus reset should cause all nodes to have the same value for `gap_count`, 63. If a more optimal gap count value is desired, the bus manager may retransmit a PHY configuration packet prior to the bus reset.

NOTE—Differences in the reset state machines between PHYs compliant with IEEE Std 1394-1995 (but unmodified by IEEE Std 1394a-2000) and those compliant with IEEE Std 1394a-2000 may result in different gap counts (subsequent to a bus reset preceded by a PHY configuration packet) when a mixture of devices is present. The probability of this inconsistency may be reduced by asserting BUS_RESET for 166.6 µs instead of the arbitrated (short) bus reset specified by this standard, but this remedy should not be applied unless inconsistent gap counts have been observed subsequent to arbitrated (short) bus resets. This heuristic does not guarantee uniform values for `gap_count` at all nodes; nevertheless, because of the probabilistic nature of the problem, success is likely after a modest number of attempts.

8.5 Bus configuration state machines (cable environment)

Background

The following subclause provides an orderly method for a bus manager to yield its role to another bus manager candidate. Although the new intended bus manager is presumably more capable, in some fashion, than the current bus manager, the details are beyond the scope of this standard.

Insert the following subclause after 8.5.3:

8.5.4 Abdication by the bus manager

A bus manager-capable node that wishes to assume the role of bus manager shall proceed as follows:

a) The candidate bus manager shall set the *abdicate* bit in the incumbent bus manager's STATE_SET register.
b) The candidate bus manager shall initiate a Serial Bus reset.
c) Subsequent to the bus reset, the candidate bus manager shall attempt to become the bus manager in accordance with the procedures in this standard, with one exception. The candidate bus manager shall not wait 125 ms before making a lock transaction to the BUS_MANAGER_ID register at the isochronous resource manager node, but shall attempt to become the bus manager immediately upon the completion of the self-identify process.
d) If the candidate bus manager fails to become the bus manager, it may transmit a PHY configuration packet with the R bit set to one, and the *root_ID* field set to the value of the candidate's own physical ID. The candidate bus manager shall not transmit such a PHY configuration packet unless it meets the

[9]In the absence of a bus manager, the isochronous resource manager is permitted to assume some of the responsibilities of the bus manager, including gap count optimization. Throughout this subclause the phrase *bus manager* is understood to mean either the bus manager or the isochronous resource manager in its role as limited bus manager.

requirements of 8.4.2.6A. The effect of such a PHY configuration packet is to clear the `force_root` variable of other nodes to FALSE. The candidate bus manager shall insure that its own `force_root` variable is TRUE[10], initiate a Serial Bus reset, and attempt to become the bus manager as described in item c).

NOTE—The last step is necessary to wrest control of the bus manager role from an incumbent bus manager that is the root and does not implement the *abdicate* bit. When the candidate bus manager becomes the root after the bus reset it has the highest arbitration priority of all the nodes on the bus and should be able to be the first to complete a lock transaction to the BUS_MANAGER_ID register.

The means by which a candidate bus manager determines that it is more capable than the incumbent bus manager are not specified by this standard. The candidate may interrogate the incumbent bus manager's CSRs for the presence or absence of advanced features or the two nodes may engage in some negotiation to determine which is more capable.

[10]PHYs compliant with this standard set their own `force_root` variable appropriately when a PHY configuration packet is transmitted or received but those compliant with IEEE Std 1394-1995 (but unmodified by IEEE Std 1394a-2000) may require a separate PHY register write to set the value of `force_root`.

Annex A

(normative)

Cable environment system properties

Background

Annex A in IEEE Std 1394-1995 defines specifications in the following areas:

a) External shielded cable interconnection

b) Internal unshielded interconnection

c) Cable power sourcing and connection

d) Powering PHY integrated circuits (ICs) and devices

e) Electrical isolation requirements

Systems designers have concluded that, with the exception of the last item, the areas above are either adequately specified in other clauses of IEEE Std 1394-1995 or else are outside of the scope of the standard.

Although electrical isolation is an important system design issue for many Serial Bus devices, it is not possible to specify uniform isolation requirements for all devices. Electrical isolation is not a normative requirement of this standard. As a consequence, 4.2.1.4.8 in IEEE Std 1394-1995 was deleted by IEEE Std 1394a-2000.

Designers are cautioned that particulars of their application, e.g., industrial or medical usage, may require electrical isolation to comply with applicable standards. In other cases, the lack of electrical isolation may cause grounding problems that in turn make it difficult to comply with agency requirements.

Replace the annex A title with the following:

Cable environment electrical isolation

Replace the text of annex A with the following:

A.1 Grounding characteristics of ac-powered devices

AC-powered devices whose power cords provide for a connection to ground are typically wired as shown in figure A-1.

The ground wire is electrically connected to the metal chassis and does not carry power current to or from the powered device. The neutral wire is connected to earth ground but must also carry the full current used by the powered device; neutral wires exhibit significant voltages due to IR drops across them. In the event of an internal short-circuit of the hot wire to the chassis, significant current flows through the ground wire to ground and causes the activation of a current-limiting device in the hot wire circuit. This arrangement is intended to reduce the user’s risk of electrocution.

NOTE—Many consumer electronic products have power cords with only two conductors, hot and neutral, but they typically have insulated cases that protect against shock hazards.

A consequence of the grounding scheme illustrated above is that the device chassis potential floats to the local ground voltage level. For a number of reasons, e.g., the return of large currents to earth by nearby, unrelated equipment, lightning strikes, or as the result of different power transformer supply domains, earth ground potential may vary by many volts. System designers are cautioned not to assume that the ground wire from different pieces of equipment connects to the same earth ground at the same voltage.

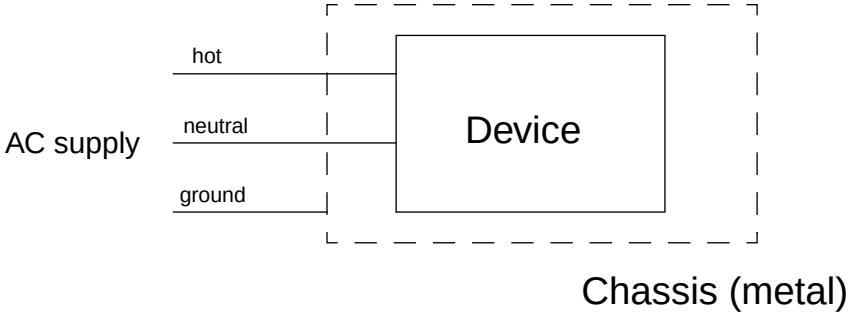


Figure A-1 — AC power supply with ground

A.2 Electrical isolation

The cables and connectors specified by this standard provide three ways in which chassis-to-chassis ground currents may flow

— The ground wire returns ground current to a chassis with a nonisolated power supply that provides cable power.

— The ground wire returns ground current to the chassis via the logic circuits of the receiving PHY in the case where the PHY is not electrically isolated from the rest of the logic circuitry in a node (e.g., the link or other ICs).

— The outer shield of the cable makes electrical connection, through the connector shield, to all connected chassis. Although this blocks RF emissions, it introduces another problem—a dc connection that forms a direct ground loop. The secondary problem may be solved by the use of RC circuits between the shield and the chassis that limit power line frequency currents while passing RF frequency currents.

A.3 Agency requirements

The following information provides guidance, valid at the time of publication of IEEE Std 1394-1995, for safety aspects relating to the interconnection and power distribution for Serial Bus devices. Because this standard permits cable power distribution at voltages greater than 24 V, international safety standards apply.

The cabling and interconnection requirements are applicable to installations of information-processing or business equipment intended for, or capable of, permanent or cord connection (during operation) to 600 V or lower potential branch circuits when such equipment is intended for installations covered under the National Electric Code® (NEC®) (NFPA 70-1999) [B9]. The equipment may also be installed according to NFPA 75-1999.

Examples of the types of equipment covered by these recommendations include but are not limited to accounting and calculating machines; cash registers; copiers; data-processing equipment; dictating and transcribing machines; duplicators; erasers; modems and other data-communication equipment; motor-driven filing cabinets, including cassette, CD, and tape accessing equipment; printers; staplers; tabulating machines; postal machines; typewriters; and other electrically operated equipment that separately, or assembled in systems, accumulate, process, and store data.

Specifically not covered by these guidelines are equipment covered by other safety standards, including but not limited to the following:

- HVAC systems
- Sensors
- Alarms
- Other equipment for the detection and signaling of conditions capable of causing damage or injury to persons
- Fire extinguishing systems
- Electrical power-supply equipment, such as motor-generator sets
- Branch-circuit supply wiring

Separate safety standards apply to this kind of equipment, and the cabling and distribution must be modified in accordance to the specifications covering that kind of equipment, in force in the location of the installed equipment.

Reference documents applicable in the United States include the following:

- UL 478-1984
- The NEC
- NFPA 75-1999

Reference documents applicable in Japan include the following:

- Electronic Equipment Technology Criteria by the Ministry of Trading and Industry (Similar to NFPA 70)
- Wired Electric Communication Detailed Law 17 by the Ministry of Posts and Telecom Law for Electric Equipment
- Dentori law made by the Ministry of Trading and Industry
- Fire law made by the Ministry of Construction

An equivalent citation of the Japanese references is given by figure A-2.

Reference documents applicable in Europe include materials to secure the European Union CE marking as follows:

- Telecommunications Terminal Equipment (91/263/EEC)
- EMC Directive (89/339/EEC)
- CE Marking Directive (93/68/EEC)
- LOW Voltage Directive (73/23/EEC) as amended by the CE Marking Directive (The CE Marking Directive is recommended as the basis for compliance)

The documents cited previously provide reference information for selection and installation of cabling in walls, temporary partitions, under floors, in overhead or suspended ceilings, or in adverse atmospheres.

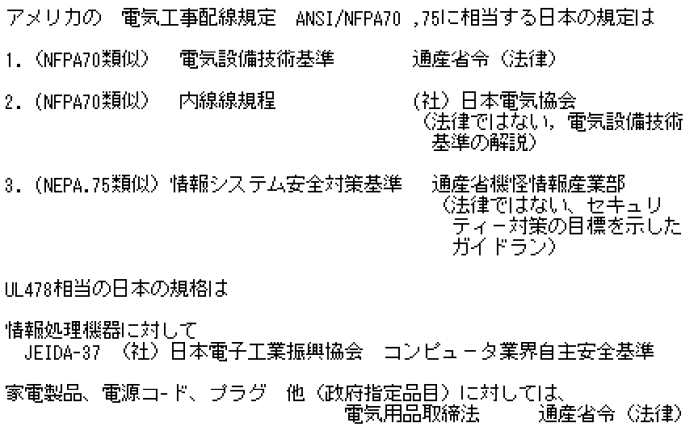


Figure A-2—Citation of Japanese references

Annex C

(normative)

Internal device physical interface

Background

Annex C specifies a physical interface suitable for Serial Bus devices mounted internal to a module's enclosure. When this optional interface is utilized, all clauses of annex C are normative with the exception of C.1. The overview is informative and describes the rationale for the internal device physical interface specified by the remainder of the annex. The other clauses in annex C are not affected.

Replace C.1 with the following:

C.1 Overview (informative)

The cable media attachment specification in 4.2.1 is suitable to external, box-to-box applications. (An example would be a computer, printer, and video camera connected via Serial Bus; the computer and printer are powered from different ac outlets while the camera takes power from the Serial Bus cable.) The external cable also provides power to all PHYs on the bus so that they can maintain their bus repeater capability even when their local power is off. When necessary to accommodate different power domains (i.e., from different ac power sources), each node provides isolation between the its local ac power and external cable power. The external environment requires mechanically strong shielded cables and connectors.

Internal devices may not have the same design criteria as external, box-to-box applications; they may be optimized for low-cost, low-power, minimum components and minimum package size (e.g., mass storage devices). Internal devices usually share a common power domain with other devices packaged within the enclosure and may not require mechanically strong or shielded connectors and cables. Internal devices may require other packaging options, such as hot-plug, auto-dock, and blind-mate; they may need various connector methodologies, such as cable or board attachment, with such connector systems as surface mount or card edge.

A goal of the internal device interface is to allow implementation options for both the device vendor and the system integrator. These options enable Serial Bus internal devices to accommodate a wide range of applications in a cost-effective manner. Device options include a second port that can be configured as either as a repeater (bus) or as a second independent port (dual path). Packaging options include cable attachment, board attachment, or a combination of the two. Pins are allocated in the internal device connector to support these options.

Insert the following after annex C:

Annex C1

(normative)

Transaction integrity safeguards

IEEE Std 1394-1995 makes little provision for facilities or implementation constraints that enhance resistance to tampering by malicious agents. Because Serial Bus may be connected to external gateways (such as cable network interface units) which may be reprogrammable from a remote location, there is a desire to provide building blocks upon which more tamper-resistant systems may be constructed. In particular it is important for Serial Bus modules to possess unforgeable identities and to not be able to snoop asynchronous request or response packets addressed to other nodes.

A module compliant with this standard shall meet the following requirements at the time of manufacture:

a) If a node's unique ID, EUI-64, is read from the configuration ROM bus information block by quadlet read requests, the value returned shall be the EUI-64 assigned by the manufacturer. In particular, the EUI-64 so returned shall not be alterable by software.

b) A node shall not originate an asynchronous request or response packet with a *source_ID* field that is not equal to either

   1) The most significant 16 bits of the node's NODE_IDS register, or
   2) The concatenation of $3FF_{16}$ and the physical ID assigned to the node's PHY during the self-identify process.

c) A node's link shall not receive nor make available to the transaction layer or any other application layer an asynchronous request or response packet unless the *destination_ID* field is equal to either

   1) The concatenation of the most significant 10 bits of the node's NODE_IDS register and either the physical ID assigned to the node's PHY during the self-identify process or $3F_{16}$, or
   2) The concatenation of $3FF_{16}$ and either the physical ID assigned to the node's PHY during the self-identify process or $3F_{16}$.

All exceptions to these requirements, if any, shall be explicitly specified in future standards developed and approved through the IEEE standards development process. At the time of writing, the only anticipated exceptions are for Serial Bus to Serial Bus bridges, whose development is in progress in the IEEE P1394.1 Working Group.

Annex E

(informative)

Cable operation and implementation examples

Replace E.1 with the following and renumber the remaining tables and figures in Annex E:

E.1 Performance optimization

Three of the simplest ways to improve the performance of a Serial Bus configuration are to

— Rearrange the devices to minimize the longest round-trip delay between any two leaf nodes. This may involve either minimizing the number of hops (cable connections) between the farthest devices, reducing cable lengths, or both.
— Group devices with identical speed capabilities next to one another. This avoids the creation of a “speed trap” when a slower device lies along the path between two faster devices.
— Set the PHY `gap_count` parameter to the lowest workable value for a particular topology.

The first two methods likely yield the most dramatic results, but they depend upon the designer of the topology (in those cases, such as inside an equipment enclosure, where it is fixed) or else upon the user’s willingness to reconfigure the bus. An application that analyzes the self-ID packets and offers suggestions for better arrangements likely would be of great value to naive users.

The third method may be employed with or without changes to bus topology. The bus manager or, in the absence of a bus manager, the isochronous resource manager, should optimize performance by setting the gap count according to the recommendations of this annex. Note that failure to optimize the gap count nullifies the benefit of a topology chosen to minimize round-trip delays.

Because of cable and PHY propagation delays, it is highly unlikely that any two nodes observe *Idle* gaps between packets of precisely the same duration. A node that completes data transmission and releases the bus observes *Idle* sooner than a node farther away. However, for different nodes’ arbitration state machines to interact correctly it is necessary for all nodes to observe the same type of gap, either an arbitration reset gap or a subaction gap. Each type of gap has a minimum and maximum time which is derived from `gap_count`, as specified by 4.3.6. The *Idle* time occupied by arbitration reset and subaction gaps is not available for either arbitration or data transfer, so it is reasonable to minimize this wasted time by choosing the smallest workable `gap_count`. The only constraint is that `gap_count` never be reduced to a value where either

a) An asynchronous subaction is interrupted by node(s) other than the source and destination, or
b) Where all nodes fail to consistently and identically detect subaction gaps (at the end of the self-identify process or the isochronous period and, at other times, if not interrupted by ack-accelerated arbitration) and arbitration reset gaps.

The worst disparity between observed *Idle* times occurs in one of the following two cases:

— Between whichever two nodes have the greatest round-trip delay for data transmission between them.
— Between whichever two nodes have the greatest accumulated disparity of one-way arbitration vs. data repeat delays of the PHYs on the path between them.

Dependent upon bus topology and cable lengths, either round-trip delay or one-way repeater disparity will dominate. The disparity between arbitration and data repeat delays for a PHY is 80 ns, as specified by ARB_RESPONSE_DELAY. Consequently, the accumulated disparity between any two nodes is simply 80 ns multiplied by the number of intervening PHYs between the nodes. Although it relies on informative assumptions, this standard suggests that round-trip delay may be obtained from the following formula:

$$\text{Round-trip delay} = 2 \times (\text{Hops} - 1) \times (\text{Cable delay} + \text{PHY delay}) + 2 \times \text{Cable delay}$$

With knowledge of the topology and individual PHY delays derived from the self-ID packets (and an assumed value for cable delay), the bus manager may use the preceding formula to calculate round-trip delay between all possible combinations of leaf nodes. The maximum round-trip delay may in turn be used to derive an optimal gap count. This method is approximate and may be improved by actually measuring the round-trip delays. Subclause 4.3.4.4.1 adds a new facility, the “ping” packet, which permits direct measurement of round-trip delay.

The bus manager may measure all leaf-to-leaf delays even if it is not itself a leaf. The possible topologies resolve into one of the following three categories:

a) The bus manager is a leaf and the round-trip delay is to be measured to another leaf.

b) The bus manager is not a leaf but is on the path that connects two leaves whose round-trip delay is to be measured.

c) The bus manager is neither a leaf nor on the path that connects two leaves whose round-trip delay is to be measured.

In all cases, the bus manager first measures propagation time(s) between itself and target node(s), then calculates the desired round-trip delay from the separately measured propagation times. The bus manager measures propagation time by transmitting a ping packet and timing the return of the first self-ID packet transmitted in response. This presupposes that the bus manager link hardware has a timer of sufficient accuracy and granularity to autonomously time the interval. The minimum and maximum propagation times may be calculated as follows:

$$\text{Propagation time}_{min} = \text{Ping time} - (\text{DATA\_END\_TIME}_{max} + \text{LINK\_TO\_BUS\_DELAY}_{max} + \text{RESPONSE\_TIME}_{max} + \text{BUS\_TO\_LINK\_DELAY}_{max}) - 2 \times \sum(\text{PHY jitter})$$

$$\text{Propagation time}_{max} = \text{Ping time} - (\text{DATA\_END\_TIME}_{min} + \text{LINK\_TO\_BUS\_DELAY}_{min} + \text{RESPONSE\_TIME}_{min} + \text{BUS\_TO\_LINK\_DELAY}_{min}) + 2 \times \sum(\text{PHY jitter})$$

Propagation time is the aggregate cable and PHY delay adjusted for jitter (which is obtained by PHY register reads of each node on the path); delay caused by arbitration or the PHY/link interface is subtracted out. The ping time, measured by link hardware, starts when the last (least significant) bit of the ping packet is transferred from the link to the PHY and ends when a data prefix indication is signaled by the PHY. The constants are obtained from table 5A-17 and table 4-32. The term for PHY jitter is the sum of individual PHY jitter for each of the repeating PHYs on the path between the bus manager and the pinged node (exclusive of the terminal nodes themselves); this can be obtained by a remote read of the PHY registers (see 5B.1 and 4.3.4.4.2). The resultant round-trip delay is expressed in units of microseconds; convert all values appropriately.

NOTE—The propagation time may be measured for any PHY packet that provokes a response from the addressed PHY, not only the ping packet. For example, a remote access packet may be used both to obtain PHY jitter from another PHY and measure the propagation time in the same step.

The three different cases for the derivation of leaf-to-leaf round-trip delays are illustrated by figure E-1; the bus manager is labeled M.

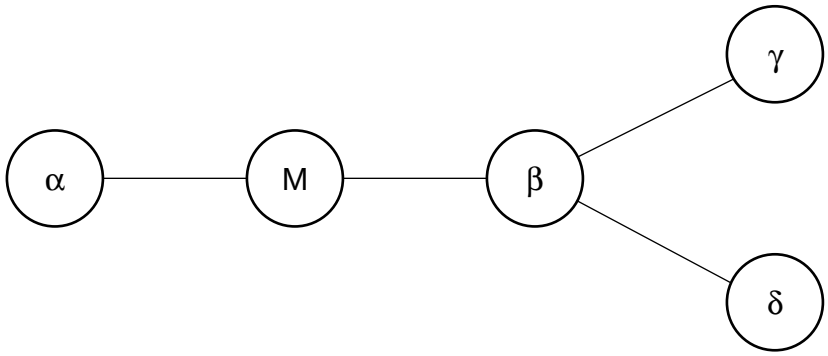


Figure E-1 — PHY pinging and round-trip times

In the first case, imagine that nodes β, γ, and δ are not present, i.e., that the bus manager and node α are both leaf nodes. The round-trip delay is the propagation time measured from the bus manager to node α, as already described.

For the second case, assume that the round-trip delay is to be calculated between nodes α and γ. The bus manager first measures the propagation time from itself to node α and then from itself to node γ. The round-trip delay between the two leaf nodes also needs to account for the bus manager's PHY delay and is expressed by

$$\text{Round-trip delay}_{(\alpha,\gamma)} = \text{Propagation time}_{\alpha} + \text{Propagation time}_{\gamma} + 2 \times \text{PHY delay}_{M}$$

In the formula above, all of the times are maxima; the bus manager's PHY delay is obtained from its own PHY registers.

The third and final case, for which the round-trip delay between nodes γ and δ is derived, is the most involved because the bus manager is not on the path between the two leaf nodes. First the bus manager measures propagation times to both nodes γ and δ. Then the bus manager measures the propagation time to the node closest to the bus manager that is also on the path between the leaf nodes—node β in this example. These measurements can be combined to eliminate the propagation time from the bus manager to node β and the excess PHY delay for node β (measured twice in the propagation times for nodes γ and δ) as follows:

$$\text{Round-trip delay}_{(\gamma,\delta)} = \text{Propagation time}_{\gamma} + \text{Propagation time}_{\delta} - 2 \times (\text{Propagation time}_{\beta} - \text{PHY delay}_{\beta}) - 240\ \text{ns}$$

In this final case, the propagation times measured for nodes γ and δ are maxima while the propagation time measured to node β is a minimum. The PHY delay for node β is a maximum and is obtained by remote access to that node's PHY registers.

In order to calculate an optimal gap count for a particular topology, two values are required for each possible leaf-to-leaf pair, the round-trip delay between them, and the accumulated disparity of arbitration vs. data repeat delays of their intervening PHYs. Use the methods described above to measure the round-trip delays and calculate the accumulated repeat delay disparity from a knowledge of the bus topology. For a particular pair of nodes, select the largest value yielded by the following formulas:

$$\frac{\text{BASE\_RATE}_{max} \times \left(\begin{array}{c} \text{Round-trip delay}_{max} + \text{RESPONSE\_TIME}_{j,\,max} \\ - \text{MIN\_IDLE\_TIME} + \text{PHY\_DELAY}_{i,\,max} \end{array} \right) + 29 \times \frac{\text{BASE\_RATE}_{max}}{\text{BASE\_RATE}_{min}} - 51}{32 - 20 \times \frac{\text{BASE\_RATE}_{max}}{\text{BASE\_RATE}_{min}}}$$

$$\frac{\text{BASE\_RATE}_{max} \times (\text{Accumulated repeat delay disparity}_{max} + \text{PHY\_DELAY}_{j,\,max}) + 53 \times \frac{\text{BASE\_RATE}_{max}}{\text{BASE\_RATE}_{min}} - 51}{36 - 32 \times \frac{\text{BASE\_RATE}_{max}}{\text{BASE\_RATE}_{min}}}$$

Note that although the round-trip delay measurement is not dependent upon the ordering of the two nodes, it is necessary to apply the preceding formulas twice for each pair of leaf nodes, since the results may be dominated by different timing constants for the two PHYs, designated *i* and *j*.

Repeat these measurements and calculations for all possible leaf-to-leaf node combinations and retain the largest value obtained from the formulas. After all combinations have been examined, the optimal `gap_count` for the topology is obtained by rounding the retained value up to the next larger integer. The bus manager or, in the absence of a bus manager, the isochronous resource manager may transmit `gap_count` in a PHY configuration packet to optimize Serial Bus performance.

If the bus manager does not have the link timer necessary to measure propagation delays, it may be appropriate to optimize the gap count in a more approximate fashion. If, by means beyond the scope of this standard, the bus manager knows that the maximum cable length used in the topology is 4.5 m and that the maximum PHY delay is 0.144 µs, the gap count may be obtained from table E-1.

Table E-1 — Gap count as a function of hops

Hops	Gap count
1	5
2	7
3	8
4	10
5	13
6	16
7	18
8	21
9	24
10	26
11	29
12	32
13	35
14	37
15	40
16	43
17	46
18	48
19	51
20	54
21	57
22	59
23	62

Annex K

(informative)

Serial Bus cable test procedures

General background

This annex amends annex K. Except as stated in the subclauses that follow, existing annex K remains unchanged. The updated test procedures attempt to characterize completely the electrical performance of the Serial Bus cable assembly and are intended to provide results of maximum relevance to system implementer(s).

The procedures presented in this annex provide an “end-to-end” characterization of the Serial Bus cable and connector system. This includes the cable itself, two assembled cable plugs, two printed circuit boards assembled with sockets, and a length of controlled impedance printed circuit board traces relevant for practical applications. Although only the cable assembly itself is under test, the sockets, the printed circuit board, and its traces are included in the test fixtures. The limit specifications and the test procedures described in this annex apply to complete cable assemblies of any length.

The measuring equipment listed in the following text or shown in the figures is provided for completeness and to guarantee maximum reproducibility of measurements. Equipment of equal capabilities may be used, although the procedures described in this annex may have to be modified accordingly.

Replace the Annex K title with the following:

Serial Bus cable assembly test procedures

Insert the following after K.2:

K.2A Differential test fixture

The test procedures utilize two different test fixtures, one described in K.2 and one described in this subclause. Each provides a transition between a Serial Bus board-mounted socket (which may receive the cable assembly under test) and 50 Ω SMA connectors (which may be connected to standard 50 Ω coaxial test equipment).

The fixture specified by figure K-1 provides an SMA connector to interface the Serial Bus socket pin (VG) to test equipment but does not isolate the socket shield from the fixture ground plane. A total of six SMA connectors port all socket pins to the test equipment.

The differential test fixture, illustrated by figure K-3A, provides a controlled RC shunt between the socket shield and the fixture ground plane while isolating them. The equivalent RC circuit values were chosen in accordance with those depicted by figure 3-30.

Construct the fixture using a multilayer board enclosed in a metal case. Isolate the case and the five SMA connector shields from direct contact to the shield of the Serial Bus socket but interface through a distributed RC shunt.

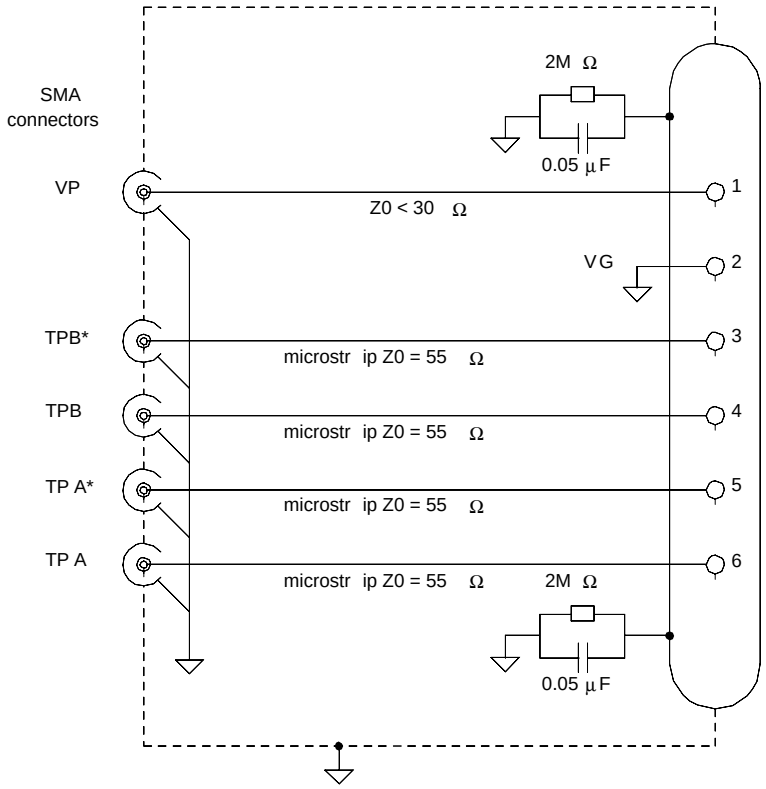


Figure K-3A — Differential test fixture schematic

The Serial Bus socket is surface mounted to the board. Connect the four signal pins of the socket (TPA, TPA*, TPB, and TPB*) to the four SMA connectors using microstrip lines with a characteristic impedance of (55 ± 3) Ω. It is important that the length of the connections be less than 50 mm and that the length mismatch between any two of the four connections be less than 2 mm. Minimize crosstalk within the fixture by using the ground plane to isolate the connections corresponding to different signal pairs.

Connect only one power pin on the Serial Bus socket (VP) to an SMA connector via a trace with a characteristic impedance of less than 30 Ω. Other power pin trace considerations remain the same as in the test fixture specified by annex K.

The differential test fixture is optimized for non-power pair crosstalk measurements and true differential impedance measurements (see the updated test procedures in K-3 and K-8).

Two test fixtures of the same type are used for every cable assembly test; their electrical performance becomes an integral part of the test results. The effect of the test fixtures upon the test results is not calibrated out during the test setup. Consequently, the test fixtures should be maintained in conditions representative of reasonable practical system usage. The socket should be replaced at least every 1000 connections.

The graphic symbol of the differential test fixture used in the test configuration diagrams contained in this annex is shown in figure K-3B.

The construction of the test fixture raises the issue of impedance matching between a pair of single-ended 50 Ω coaxial connectors and the differential mode 110 Ω Serial Bus signal lines. In order to assume there is reasonable differential mode matching between the connectors and the Serial Bus socket and cable assembly, it is necessary for the signal line connection traces within the fixture to be single-ended 55 Ω and have matched electrical lengths. This shifts the impedance matching problem to the level of the SMA connectors, where matching circuits may be added.

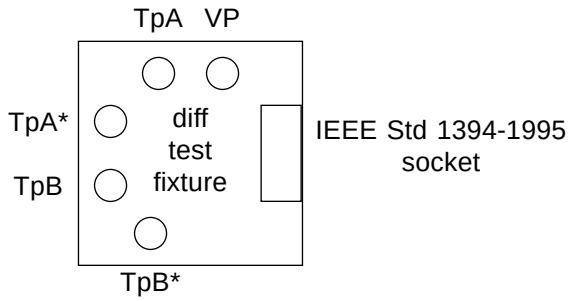


Figure K-3B — Differential test fixture graphic symbol

In order to improve accuracy, some of the tests use a minimum loss-resistive matching pad. The schematic diagram of such a pad is shown by figure K-3.

The test configurations may contain precision 20 dB attenuators, which isolate the test equipment from the cable/connector mismatch. Due to the relative low level of mismatch and the isolation of the attenuator, the matching pads may be omitted at the expense of a slight increase in the frequency domain ripple. This effect may be removed by data filtering algorithms available with the suggested test equipment.

If impedance matching pads are utilized, always include them in the test calibration setup to eliminate their effect upon the final test result.

K.3 Signal pairs characteristic impedance

Background

This subclause amends K.3 of IEEE Std 1394-1995 by adding connector tests to the cable assembly tests specified by the existing standard. A description of the use of differential time domain reflectometry (TDR) equipment, in addition to single-ended TDR equipment, is also incorporated.

Replace K.3 with the following:

K.3 Signal pairs characteristic and discrete impedance

Although a complete cable assembly, including both cable and connector parts is specified, individual impedance evaluations select either cable or connector parts as a consequence of time-of-flight into the assembly. Measure the differential mode characteristic impedance for the cable section in the time domain using a single-ended TDR with an edge rate of less than 0.2 ns. The differential mode impedance value is calculated from the single-ended measurements described in K.3.1 through K.3.3. Calculate the result of each single-ended measurement as the average of the impedance measured at two points along the cable. Select these points at 1 ns and 2.5 ns along the

cable from the plug closest to the launching connector. Because the TDR displays the round-trip propagation delay, make the measurements at 2 ns and 5 ns from the plug closest to the launching connector as measured by the TDR instrument. Repeat this test for both signal pairs.

The differential mode discrete impedance, through the connector section, is measured in the time domain using a differential TDR with an equivalent edge rate of 0.5 ns. A differential TDR may establish an equivalent edge rate by means of a filter algorithm within its data processing software, which may permit a wide range of equivalent rise times to be evaluated.[11]

Measure true differential mode impedance at three discrete points beyond the plane of signal insertion into the Serial Bus socket. Select these points at 50 ps, 100 ps, and 150 ps into the connector. Again, since the TDR displays the round-trip propagation delay, take the measurements at 100 ps, 200 ps, and 300 ps beyond the plane of signal insertion.

Evaluate each of the three connector section impedance values individually against the range of differential impedance allowed over the defined 100 ps exception window (50–150 ps). Repeat this for both signal pairs.

K.3.1 Signal pairs impedance setup calibration—short and load

This calibration should be performed as shown in figure K-4 using the calibration algorithms built into the TDR equipment suggested (HP 54120B and HP 54121A or equivalent). External short and load calibration is not necessary for some differential TDR equipment. The appropriate equipment setup procedure should be conducted as defined by the equipment manufacturer.

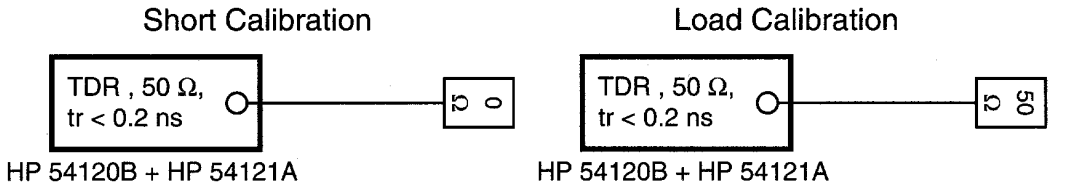


Figure K-4—Signal pairs impedance setup calibration

K.3.2 Signal pairs impedance test procedure (connector)

Using the test configuration described in figure K-5 and the connection matrix shown in table K-1, measure the discrete differential impedances of the signal wires through the connector section at three points and compare the results to the allowed limits through the exception window.

The true differential mode discrete impedance of connector signal pair TPA_{conn} is displayed as $ZTPA_{conn.}$ The true differential mode discrete impedance of connector signal pair TPB_{conn} is displayed directly as $ZTPB_{conn}$.

[11]Representative TDRs of this type include the HP 54750A and Tektronix 11801B, both configured with TDR sampling heads. Selection of a 0.5 ns filter algorithm causes the differential mode impedance to be displayed directly.

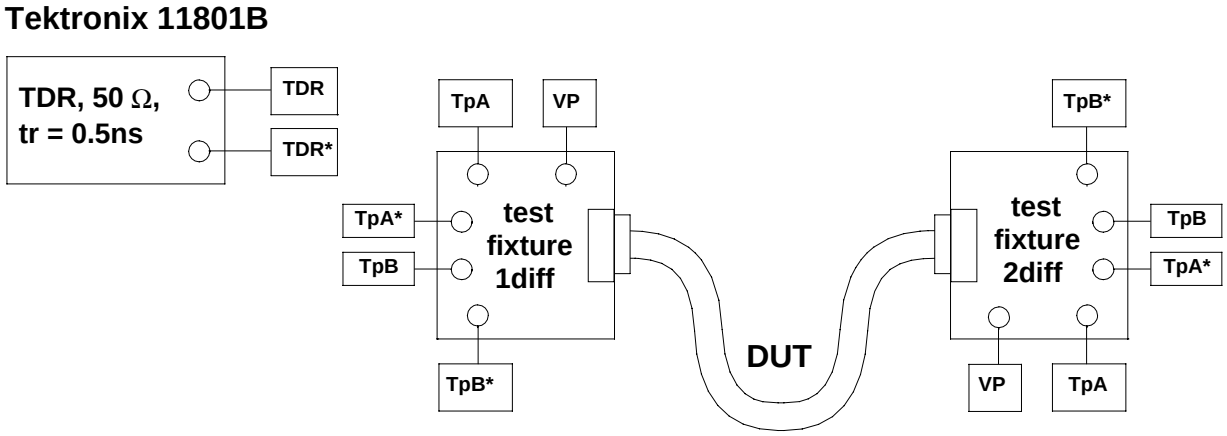


Figure K-5 — Signal pairs impedance measurement configuration (connector)

Table K-1 — Connection matrix for signal pairs impedance tests (connector)

Measured value	Fixture 1 (differential)					Fixture 2 (differential)				
	TPA	**TPA***	**TPB**	**TPB***	**VP**	**TPA**	**TPA***	**TPB**	**TPB***	**VP**
Differential mode TPA ($ZTPA_{conn}$)	TDR	TDR*	50 Ω	50 Ω	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω
Differential mode TPB ($ZTPB_{conn}$)	50 Ω	50 Ω	TDR	TDR*	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω

K.3.3 Signal pairs impedance limits (connector)

The test limits, for the 100 ps exception window, are

$ZTPA_{conn} = (110 \pm 25)\ \Omega$
$ZTPB_{conn} = (110 \pm 25)\ \Omega$

K.7.5 DC resistance test procedure

Change the first paragraph of K.7.5 as follows:

Using the test configuration described in figure K-11 and the connection matrix shown in ~~table K-2~~ table K-3, the dc resistance of the power pair shall be measured.

K.8 Crosstalk

Background

The provision in K.8 of IEEE Std 1394-1995 to measure crosstalk in the frequency range of 1–500 MHz is changed by IEEE Std 1394a-2000.

Replace the first paragraph in K.8 with the following:

Measure pair-to-pair crosstalk in the frequency domain using a network analyzer in the frequency range of 1–75 MHz.

NOTE—Although described as a measurement of pair-to-pair crosstalk, the test configurations are single-ended. The phrase *pair-to-pair* refers only to the location of the designated driven line and quiet line.

K.8.2 Crosstalk test procedure

Background

The test procedures specified by K.8.2 are modified so that they apply only to crosstalk between the power pair and a signal pair.

Replace the title of K.8.2 with the following:

K.8.2 Crosstalk test procedure (between power and signal pairs)

Insert the following text above table K-4 in K.8.2:

The test fixtures referenced in table K-4 are specified by figure K-1.

Replace table K-4 with the following:

Table K-4 — Connection matrix for crosstalk tests between power and signal pairs

Measured value	Fixture 1						Fixture 2					
	VP	**TPA**	**TPA***	**TPB**	**TPB***	**VG**	**VP**	**TPA**	**TPA***	**TPB**	**TPB***	**VG**
Crosstalk between VP and TPA (X_{PA})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	In	50 Ω	50 Ω	50 Ω	0 Ω
Crosstalk between VP and TPA* (X_{PA*})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	50 Ω	In	50 Ω	50 Ω	0 Ω
Crosstalk between VP and TPB (X_{PB})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	50 Ω	50 Ω	In	50 Ω	0 Ω
Crosstalk between VP and TPB* (X_{PB*})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	In	0 Ω
Crosstalk between VG and TPA (X_{PA})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	In	50 Ω	50 Ω	50 Ω	50 Ω
Crosstalk between VG and TPA* (X_{PA*})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	50 Ω	In	50 Ω	50 Ω	50 Ω
Crosstalk between VG and TPB (X_{PB})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	50 Ω	50 Ω	In	50 Ω	50 Ω
Crosstalk between VG and TPB* (X_{PB*})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	50 Ω	50 Ω	50 Ω	In	50 Ω

Background

In addition to the change to the frequency domain, IEEE Std 1394a-2000 modifies the procedure to measure crosstalk between the signal pairs.

Insert the following after K.8.2:

K.8.2A Crosstalk test procedure (between signal pairs)

To evaluate crosstalk between the signal pairs, use the differential test fixture specified by figure K-3B. Then perform the tests described by K.8.2 but use the connection matrix shown in table K-5.

Table K-5 — Connection matrix for crosstalk tests between signal pairs

Measured value	Fixture 1					Fixture 2				
	VP	**TPA**	**TPA***	**TPB**	**TPB***	**VP**	**TPA**	**TPA***	**TPB**	**TPB***
Crosstalk between TPA and TPB (X_{AB})	0 Ω	Out	50 Ω	50 Ω	50 Ω	0 Ω	In	50 Ω	50 Ω	50 Ω
Crosstalk between TPA and TPB* (X_{AB*})	0 Ω	Out	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	In	50 Ω	50 Ω
Crosstalk between TPA* and TPB (X_{A*B})	0 Ω	50 Ω	Out	50 Ω	50 Ω	0 Ω	In	50 Ω	50 Ω	50 Ω
Crosstalk between TPA* and TPB* (X_{A*B*})	0 Ω	50 Ω	Out	50 Ω	50 Ω	0 Ω	50 Ω	In	50 Ω	50 Ω

Insert the following after K.8.3:

K.8.3A Crosstalk limits (between signal pairs)

The test limits for crosstalk between the signal pairs are

$X_{AB} \leq -26$ dB
$X_{AB*} \leq -26$ dB
$X_{A*B} \leq -26$ dB
$X_{A*B*} \leq -26$ dB

Insert the following after Annex L:

Annex M

(informative)

Serial Bus topology considerations for power distribution (cable environment)

This annex provides recommendations for the practical management of Serial Bus topologies with respect to power distribution. The capabilities of Serial Bus devices, cables used to interconnect them, and the topology of their arrangement can all affect a cable-powered device's ability to obtain and use power.

— The path between a cable-powered device and a power source cannot be blocked by devices that do not repeat power.
— The electrical resistance of the path cannot be so large as to reduce voltage to unusable levels.
— A correlation between power user(s) and power provider(s) is necessary in order to budget available power.

Analysis of power distribution for a particular bus topology is based upon the electrical characteristics of the cables, connectors, and devices. These characteristics are normatively specified by this standard; for convenience of reference they are summarized as follows:

a) *Power pair dc resistance.* Subclause 4.2.1.4.6 specifies that the dc resistance of the power wires, VP and VG, is less than or equal to 0.333 Ω. This annex assumes that connector plug to socket resistance is less than or equal to 0.06 Ω for the assembly (both connectors) and that this is included in the 0.333 Ω total.

b) *Output current per port.* Subclause 4.2.2.7 limits output current to a maximum of 1.5 A.

c) *Voltage drop through cable assembly.* The product of 1.5 A and 0.333 Ω yields a maximum voltage drop of 0.5 V for a mated cable assembly.

d) *Voltage drop through node.* Subclause 4.2.2.7 limits the resistance between any two of a node's connector sockets to a maximum of 0.5 Ω. In combination with 1.5 A current per port, the maximum voltage drop through a node is 0.75 V.

e) *Device power requirements.* The minimum power needed on VP for a cable-powered PHY to be operable is 3 W at 8 V.

The cable material performance requirements can be met by the reference design illustrated in 4.2.1.2.1. The reference design assumes a maximum cable assembly length of 4.5 m and the use of 22 AWG (7×30) for power and ground. It may be possible to construct longer cables with a larger wire gauge, so long as the power pair dc resistance criterion of 0.333 Ω for the cable assembly is met.

NOTE—Cable assembly requirements of 4.2.1.2.2 that specify the connection of VG to the inner cable shields at both ends effectively lower the resistance of VG to 0.167 Ω.

Ground difference potential was not addressed previously, but may be of concern for safety reasons. Ground difference potential is measured in the same way as power—through a mated cable assembly with the measurements taken at the printed circuit board side of the connector sockets. Ground difference potential in excess of 0.5 V may be reason for concern.

Table M-1 is derived based upon the preceding characteristics. For each of the three common classes of power provider (as identified by POWER_CLASS in the self-ID packets), the table shows the greatest hop count at which a minimum of 3 W are available at a minimum of 8 V (assuming that there are no other power consumers on the path from the power provider to the intended power consumer). The last column in the table shows the maximum current available to the power consuming device at the wattage provided by the power source.

Table M-1 — Power provider ranges by POWER_CLASS and launch voltage

POWER_CLASS	Launch voltage (V)	Maximum hops	Maximum current available (A)
001_2 (15 W)	20	19	0.750
	24	23	0.625
	26		0.577
	30		0.500
010_2 (30 W)	20	9	1.500
	24	15	1.250
	26	18	1.150
	30	23	1.000
011_2 (45 W)	30	17	1.500

The maximum hops in table M-1 are limited to 23 because this is the largest bus topology that can operate within a maximum gap count of 63, if 4.5 m cables and a PHY delay of 0.144 μs are assumed. Given the characteristics of a particular power provider, wattage, and launch voltage, it is possible to calculate the power available to a power consumer according to the number of hops that separate the two. The aggregate resistance, R, between the power provider and consumer may be calculated as follows:

$$R = (\text{Hops} \times 0.833 - 0.5)\Omega$$

and the result used in the following equations:

$$P = I^2R$$

$$E = IR$$

where

E is the voltage drop,
I is the current,
R is the resistance of the wire and connector,
P is the power available.

An example power analysis is provided for the configuration illustrated by figure M-1, which shows a power provider separated by three hops from the power consumer. Kirchoff's law is used to determine the voltage available at VP for each node.

Assume the power provider is POWER_CLASS two (30 W) with a launch voltage of 26 V The VP measurements are taken at the printed circuit board side of the connector socket. All the cables are assumed to be 4.5 m and constructed per the reference designs in this standard. The power consumer is assumed to draw 1.15 A.

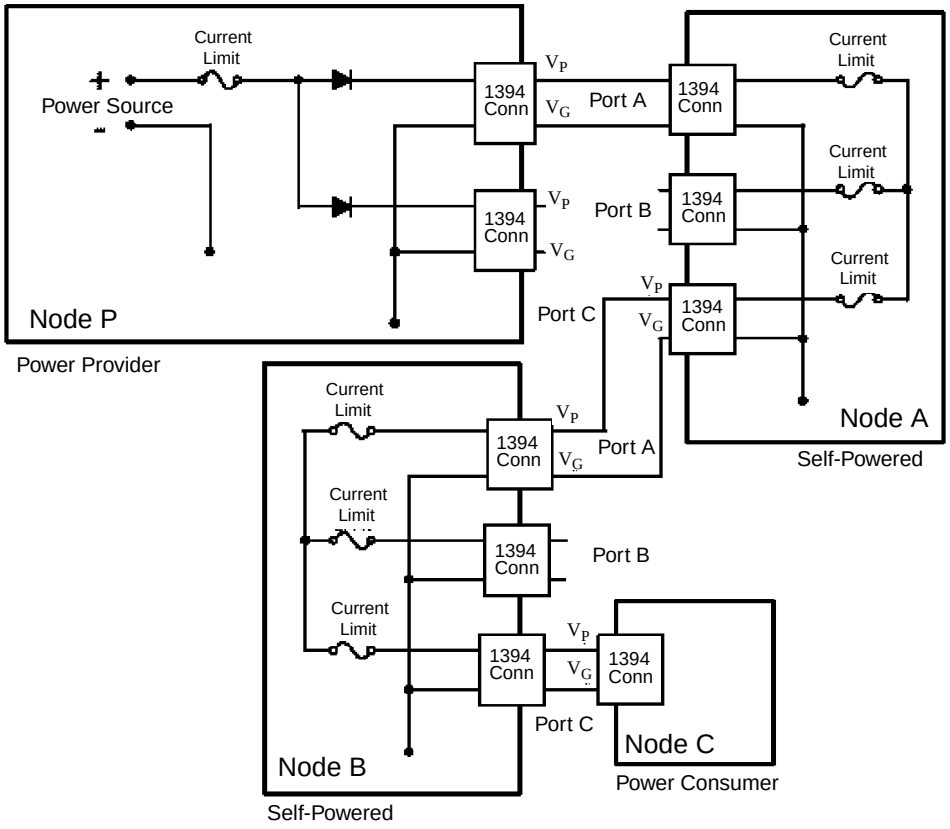


Figure M-1 — Power distribution example

The voltage drop across VP in the cable that connects nodes P and A is equal to the current multiplied by the resistance of the wire and connector ($E = IR$). Given a current of 1.15 A, a mated cable assembly resistance of 0.33 Ω, the voltage drop across this cable is 0.383 V.

After the cable, the current passes through node A. The maximum port to port resistance of node A, or X, yields a voltage drop of 0.575 V.

The voltage drops in the remaining cables, from node A to node B and from node B to node C, are identical to the voltage drop in the first cable. The voltage drop through node B is also the same as through node A. The aggregate voltage drop for these two cables and node B is 1.341 V. The cumulative voltage drop from the power provider to the power consumer is 2.299 V.

Thus, the net voltage available to the power consumer at the printed circuit board side of the connector socket is the launch voltage, 26 V, less all of the intermediate voltage drops, 2.299 V. In other words, 23.7 V. It is left to the designer to calculate the losses in printed circuit board traces within the power consumer and arrive at the net usable voltage available to the device's circuitry.

In a like fashion, the power available to the power consumer may be obtained from $P = I^2R$. Calculate the power drop for each cable assembly and for each node through which the power passes. Subtract the aggregate power loss from the power provided by the source to yield 27.4 W available to the power consumer.

Insert the following after annex M:

Annex N

(informative)

Bibliography

[B1] ANSI Y14.2M-1992 (Reaff 1998), Line conventions and lettering.

[B2] ANSI Y14.5M-1994 (Reaff 1999), Dimensioning and tolerancing—includes inch and metric.

[B3] IEEE Std 100-1996, The IEEE Standard Dictionary of Electrical and Electronics Terms.

[B4] IEEE P1394b, Draft Standard for a High Performance Serial Bus—Amendment 2.[12]

[B5] Japanese Ministry of Posts and Telecom Law for Electric Equipment, Wired Electric Communication Detailed Law 17.

[B6] Japanese Ministry of Trading and Industry, Dentori Law.

[B7] Japanese Ministry of Construction, Fire Law.

[B8] JEIDA-37, Japanese Ministry of Trading and Industry, Electronic Equipment Technology Criteria.

[B9] NFPA 70-1999, National Electric Code® (NEC®).

[B10] NFPA 75-1999, Standard for the Protection of Electronic Computer/Data-Processing Equipment.

[B11] The EEC CE Marking Directive.

[B12] The EEC EMC Directive.

[B13] The EEC Low Voltage Directive (as amended by the CE Marking Directive).

[B14] The EEC Telecommunications Terminal Equipment Directive.

[B15] UL 478-1984 (Reaff 1986), Information Processing and Business Equipment.

[12]This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE (http://standards.ieee.org/).