

P1394a

Draft Standard for a High Performance Serial Bus (Supplement)

Sponsor

**Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society**

Not yet Approved by

IEEE Standards Board

Not yet Approved by

American National Standards Institute

Abstract: Supplemental information for a high-speed serial bus that integrates well with most IEEE standard 32-bit and 64-bit parallel buses is specified. It is intended to extend the usefulness of a low-cost interconnect between external peripherals, **IEEE Std 1394-1995**. This standard 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.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1997 by the Institute of Electrical And Electronics Engineers, Inc.
All rights reserved. Published 1997. Printed in the United States of America.

ISBN x-xxxxx-xxx-x

This is an unapproved IEEE Standards Draft, subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities, including balloting and coordination. If this document is to be submitted to ISO or IEC, notification shall be given to the IEEE Copyright Administrator. Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position. Other entities seeking permission to reproduce this document for these or other uses must contact the IEEE Standards Department for the appropriate license. Use of the information contained in this unapproved draft is at your own risk.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE 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 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 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.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not a part of IEEE Std 1394-1995, IEEE Standard for a High Performance Serial Bus (Supplement).)

This standards effort started in 1996 at the request of...

Patent notice

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 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.

The patent holder has, however, 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 a license. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreement offered by the patent holder. Contact information may be obtained from the IEEE Standards Department.

Committee membership

The following is a list of voting members of the IEEE P1394a working group at the time of publication.

Peter Johansson, *Chair and Editor*
Prashant Kanhere, *Secretary*

The following is a list of other major participants in the IEEE P1394a working group (those that attended at least X working group meetings since its inception).

Kazuyuki Abe	John Fuller	Takashi Matsui	Roy Yasoshima
Richard Baker	Nobuo Furuya	Cyrus Momeni	Takao Yasuda
Steven Bard	James Gay	Ganesh Murthy	Phil Young
Max Bassler	John Grant	Karl Nakamura	
Joe Bennett	Eric Hannah	Bill Northey	
Vilas Bhade	Yasumasa Hasegawa	Takayuki Nyu	
Mike Brown	Jerry Hauck	Farrell Ostler	
Jim Busse	Joe Herbst	Kugao Ouchi	
Carissa Cheung	Jack Hollins	Bill Prouty	
Richard Churchill	Al Kelley	Bradley Saunders	
Claude Cruz	Mark Knecht	David Scott	
David Doman	David LaFollette	James Skidmore	
Firooz Farhoomand	Thang Le	Peter Teng	
Lou Fasano	Paul Levy	Colin Whitby-Strevens	
Takahiro Fujimori	Hirokazu Mamezaki	David Wooten	

The following persons served on the ballot response committee:

The following persons were members of the balloting group:

If the IEEE Standards Board approves this draft standard, it might have the following membership:

E. G. “AP” Kiener, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril
Clyde R. Camp
Joseph A. Cannatelli
Stephen L. Diamond
Harold E. Epstein
Donald C. Fleckenstein
Jay Forster*
Donald N. Heirman
Richard J. Holleman

Jim Isaak
Ben C. Johnson
Sonny Kasturi
Lorraine C. Kevra
Ivor N. Knight
Joseph L. Koepfinger*
D. N. “Jim” Logothetis
L. Bruce McClung

Marco W. Migliaro
Mary Lou Padgett
John W. Pope
Arthur K. Reilly
Gary S. Robinson
Ingo Rüsçh
Chee Kiow Tan
Leonard L. Tripp
Howard L. Wolfman

*Member Emeritus

Other candidates for inclusion might be the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Steve Sharkey
Robert E. Hebner
Chester C. Taylor

Rochelle Stern
IEEE Standards Project Editor

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 References	1
1.4 Document organization	2
1.5 Service model	2
1.6 Document notation	3
2. Definitions and abbreviations	11
2.1 Conformance glossary	11
2.2 Technical glossary	11
3. Overview	15
4. Alternative cable media attachment specification	17
4.1 Connectors	17
4.2 Cables	23
4.3 Connector and cable assembly performance criteria	24
4.4 Signal propagation performance criteria	32
5. PHY/Link interface specification	35
5.1 Operation	37
5.2 Link requests	37
5.3 Status	44
5.4 Transmit	45
5.5 Receive	46
5.6 Electrical characteristics	47
6. PHY register map	53
6.1 PHY register map (cable environment)	53
6.2 PHY register map (backplane environment)	57
6.3 Integrated link and PHY	58
7. Cable physical layer performance enhancement specifications	59
7.1 Cable topology	59
7.2 Cable power and ground	60
7.3 Cable PHY packets	61
7.4 Cable PHY line states	65
7.5 Cable PHY timing constants	66
7.6 Node variables	67
7.7 Port variables	67
7.8 Cable physical layer operation	68
7.9 Port disable	90
8. Asynchronous streams	93
8.1 Asynchronous stream packet format	93
8.2 Loose vs. strict isochronous	94
8.3 Serial Bus event indication (SB_EVENT.indication)	94

9. Clarifications and corrigenda	95
9.1 Cycle start.....	95
9.2 Read response for data block	95
9.3 Maximum isochronous data payload	96
9.4 Transaction codes (tcode)	96
9.5 Response codes (rcode)	97
9.6 Tag	98
9.7 Acknowledge codes (ack_code)	99
9.8 Priority arbitration for response packets	100
9.9 Transaction layer services.....	100
9.10 Serial Bus control request (SB_CONTROL.request).....	102
9.11 Serial Bus event indication (SB_EVENT.indication).....	102
9.12 Command reset effects.....	102
9.13 FAIRNESS_BUDGET register	102
9.14 Unit registers	104
9.15 Configuration ROM Bus_Info_Block.....	104
9.16 Node_Unique_ID.....	106
9.17 Determination of the bus manager.....	106
9.18 Gap count optimization.....	106
9.19 Automatic activation of the cycle master.....	107
9.20 Isochronous period too long	107
9.21 Abdication by the bus manager.....	107
9.22 Transaction integrity safeguards	109

Figure 1-1— Service model	3
Figure 1-2— Bit and byte ordering	4
Figure 1-3— Example packet format	5
Figure 1-4— State machine example	6
Figure 1-5— CSR format specification (example)	7
Figure 1-6— Reserved CSR field behavior	9
Figure 4-1 — Plug body	18
Figure 4-2 — Plug section details	18
Figure 4-3 — Connector socket interface	19
Figure 4-4 — Socket cross-section A-A	20
Figure 4-5 — Cross-section of plug and socket contacts	20
Figure 4-6 — Socket position when mounted on a printed circuit board	21
Figure 4-7 — Flat surface mount printed circuit board connector footprint	22
Figure 4-8 — Flat through-hole mount printed circuit board connector footprint	22
Figure 4-9 — Cable material construction example (for reference only)	23
Figure 4-10 — Cable assembly and schematic (standard to alternate connector)	24
Figure 4-11— Cable assembly and schematic (alternate connectors)	24
Figure 4-12 — Shield and contact resistance measuring points	29
Figure 4-13 — Fixture for cable flex test	32
Figure 5-1 — Discrete PHY/Link interface	35
Figure 5-2 — LPS timing (isolated interface)	36
Figure 5-3 — PHY/Link interface reset via LPS	36
Figure 5-4 — LReq and Ctl timings	38
Figure 5-5 — Status timing	44
Figure 5-6 — Transmit timing	46
Figure 5-7 — Receive timing	47
Figure 5-8 — Valid signal levels for AC measurements	48
Figure 5-9 — Rise and fall time definition	49
Figure 5-10 — PHY to link transfer waveform	49
Figure 5-11 — Link to PHY transfer waveform at the link	49
Figure 5-12 — Link to PHY transfer waveform at the PHY	49
Figure 5-13 — Link to PHY delay timing	51
Figure 6-1 — Extended PHY register map for the cable environment	53
Figure 6-2 — PHY register page 0: Port Status page	55
Figure 6-3 — PHY register page 1: Vendor Identification page	56
Figure 6-4 — PHY register map for the backplane environment	57
Figure 7-1 — Node power interface for POWER_CLASS one, two or three	60
Figure 7-2 — Self-ID packet format	62
Figure 7-3 — Link-on packet format	64
Figure 7-4 — PHY configuration packet format	64
Figure 7-5 — Ping packet format	65
Figure 7-6 — Cable physical layer architecture	69
Figure 7-7 — Bus reset state machine	76
Figure 7-8 — Self-ID state machine	79
Figure 7-9 — Cable arbitration state machine	84
Figure 7-10 — Port disable logic	91
Figure 8-1 — Asynchronous stream packet format	93
Figure 9-1— Cycle start packet format	95
Figure 9-2— FAIRNESS_BUDGET format	103
Figure 9-3— Bus_Info_Block format	105
Figure 9-4— STATE_CLEAR.bus_depend field	108
Figure A-1 — AC power supply with ground	111

Table 1-1— Size notation examples	3
Table 1-2— C code operators summary	5
Table 1-3— Additional C data types	6
Table 1-4— Register definition fields	8
Table 1-5— Read value fields	8
Table 1-6— Write value fields	8
Table 1-7— Summary of lock functions	9
Table 4-1 — Connector socket signal assignment	19
Table 4-2 — Performance group A	26
Table 4-3 — Performance group B	27
Table 4-4 — Performance group C	27
Table 4-5 — Performance group D	28
Table 4-6 — Performance group E	30
Table 4-7 — Performance group F	31
Table 4-8 — Performance group G	31
Table 5-1 — PHY/Link signal description	35
Table 5-2 — LPS timing parameters	36
Table 5-3 — Ctl[0:1] when PHY is driving	37
Table 5-4 — Ctl[0:1] when the link is driving (upon a grant from the PHY)	37
Table 5-5 — Bus request format for cable environment	38
Table 5-6 — Bus request format for backplane environment	38
Table 5-7 — Register read request format	38
Table 5-8 — Register write request format	39
Table 5-9 — Acceleration control request format	39
Table 5-10 — Request type field	39
Table 5-11 — Request speed field	40
Table 5-12 — Link request effects on PHY variables	40
Table 5-13 — Link rules to initiate a request on LReq	41
Table 5-14 — PHY disposition of link request	42
Table 5-15 — Status bits	44
Table 5-16 — Speed code signaling	47
Table 5-17 — TTL DC specifications	48
Table 5-18 — AC timing parameters	50
Table 5-19 — AC timing parameters at the PHY	50
Table 5-20 — AC timing parameters at the link	50
Table 5-21 — Link to PHY delay timing parameters	51
Table 6-1 — PHY register fields for the cable environment	53
Table 6-2 — PHY register Port Status page fields	55
Table 6-3 — PHY register Vendor Identification page fields	56
Table 6-4 — PHY register fields for the backplane environment	57
Table 7-1 — Cable power source requirements	60
Table 7-2 — Self-ID packet fields	62
Table 7-3 — Link-on packet fields	64
Table 7-4 — PHY configuration packet fields	64
Table 7-5 — Ping packet fields	65
Table 7-6 — Cable PHY received arbitration line states	65
Table 7-7 — Cable PHY timing constants	66
Table 7-8 — Node variables	67
Table 7-9 — Port variables	67
Table 7-10 — Cable PHY code definitions	70
Table 7-11 — Cable PHY packet definitions	71
Table 7-12 — Data transmit actions	72
Table 7-13 — Start data transmit actions	72
Table 7-14 — Stop data transmit actions	73
Table 7-15 — Data reception and repeat actions	73

Table 7-16 — Start data reception and repeat actions	74
Table 7-17 — Stop data reception and repeat actions	75
Table 7-18 — Bus reset actions and conditions	77
Table 7-19 — Self ID actions and conditions	81
Table 7-20 — Normal arbitration actions and conditions	86
Table 7-21 — Receive actions and conditions	88
Table 7-22 — Transmit actions and conditions	89
Table 8-1— Maximum data block payload for asynchronous primary packets	94
Table 9-1— Maximum payload for isochronous stream packets	96
Table 9-2— Transaction code encoding	96
Table 9-3— Response code encoding	97
Table 9-4— Tag field encoding	98
Table 9-5— Acknowledge codes	99
Table 9-6— Transaction codes eligible for priority asynchronous arbitration	103
Table 9-7— Serial Bus-dependent registers in initial units space	104
Table 9-8— Encoding of max_rec field	105

P1394a

Draft Standard for a High Performance Serial Bus (Supplement)

1. Overview

1.1 Scope

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

The following are included in this supplement:

- a) Cables and connectors for a 4-pin variant (from the 6-pin already standardized);
- b) Standardization of the PHY/LINK interface, which at present is an informative annex to the existing standard;
- c) ~~A connection management protocol, and the necessary CSR facilities, for isochronous data;~~
- d) 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;
- e) A redefinition of the isochronous data packet, transaction code A₁₆, to permit its use in either the asynchronous or isochronous periods;
- f) More stringent requirements on the power to be supplied by a cable power source and a clarification of electrical isolation requirements;
- g) Miscellaneous *corrigenda* to the existing standard.

The preceding are arranged in no particular order.

1.2 Purpose

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

1.3 References

This standard shall be used in conjunction with the following publications. When the following publications are superseded by an approved revision, the revision shall apply.

ANSI/EIA-364-B-90, Electrical Connector Test Procedures Including Environmental Classifications.¹

IEEE Std 1394-1995, Standard for a High Performance Serial Bus

ISO/IEC 9899: 1990, Programming languages—C.²

ISO/IEC 13213: 1994 [ANSI/IEEE Std 1212, 1994 Edition], Information technology—Microprocessor systems—Control and Status Registers (CSR) Architecture for microcomputer buses.

This standard shall also be used in conjunction with the following publications under development. When approved as a standard, the approved version shall apply.

IEEE P1394b, Draft Standard for a High Performance Serial Bus (Supplement)

IEC 61883/FDIS, Digital Interface for Consumer Electronic AV Equipment

1.4 Document organization

This standard contains this overview, a list of definitions, an informative summary description, sections of technical specification and application annexes. The new reader should read the informative summary and the sections that precede it before the remainder of the document.

1.5 Service model

IEEE Std 1394-1995 and this supplement both use 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. Four types of service are defined by this standard:

- a) *Request service.* A request service is a communication from a layer to an adjacent layer 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 the 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 an adjacent layer 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 an 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 an adjacent layer 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 status. For data transfer requests, the confirmation may be caused by a corresponding response on a peer node.

¹ EIA publications are available from Global Engineering, 1990 M Street NW, Suite 400, Washington, DC, 20036, USA.

² ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse. ISO 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.

If all four service types exist, they are related as shown by the following figure:

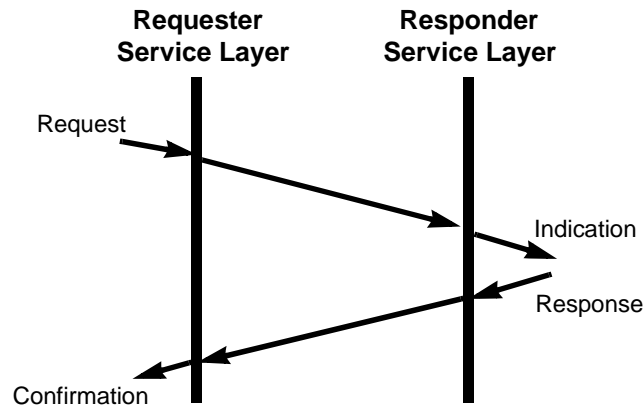


Figure 1-1—Service model

1.6 Document notation

1.6.1 Mechanical notation

All mechanical drawings in this document use millimeters as the standard unit and follow ANSI Y14.2 and ANSI Y14.5-1982 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

The Serial Bus description 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, the Serial Bus description uses terms established in previous IEEE bus standards, which are independent of the processor. 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

The 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 (leftmost) bit is also transmitted first. This ordering convention is illustrated in figure 1-2.

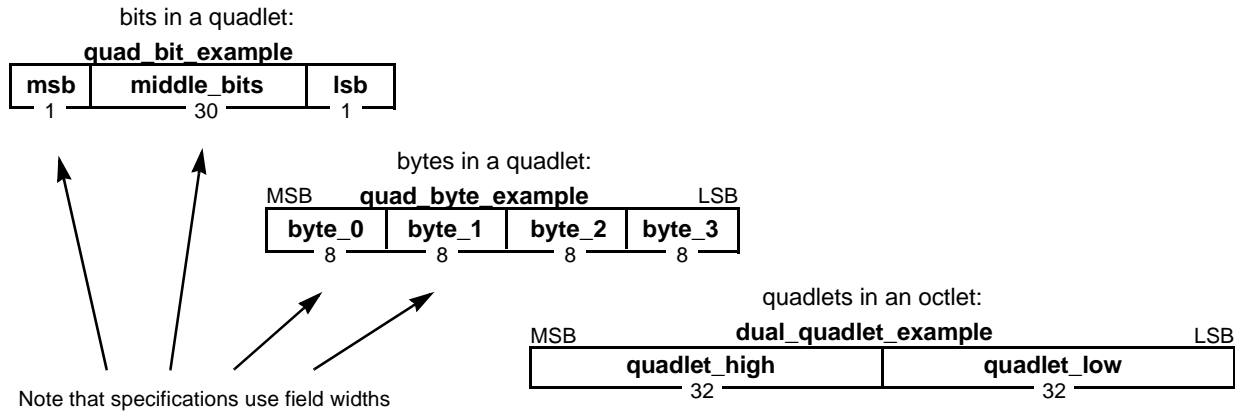


Figure 1-2—Bit and byte ordering

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 1-2.

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 1-2) will be labeled “quad_bit_example[0],” the most significant byte of a quadlet (“byte_0”) will be labeled “quad_byte_example[0:7],” and the most significant quadlet in an octlet (“quadlet_high”) will be 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).

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 “1A₁₆” or “11010₂”. 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 1-3.

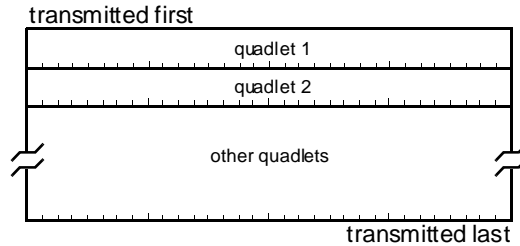


Figure 1-3—Example packet format

Fields appear in packet formats with their correct position and widths. Field widths are also stated explicitly in field descriptions. 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

Table 1-2—C code operators summary (Continued)

Operator	Description
<<	Left shift; $x \ll 2$ 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 supplement are always filled with zero
~	One's complement (unary)

A common construction in C is conditional evaluation, in the form $(expr) ? expr1 : expr2$. This indicates that if the logical expression $expr$ evaluates to nonzero value then $expr1$ is evaluated, otherwise $expr2$ is evaluated. For example, $x = (q > 5) ? x++ : 14;$ first evaluates $q > 5$. If 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. Time elapses 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 1-4.

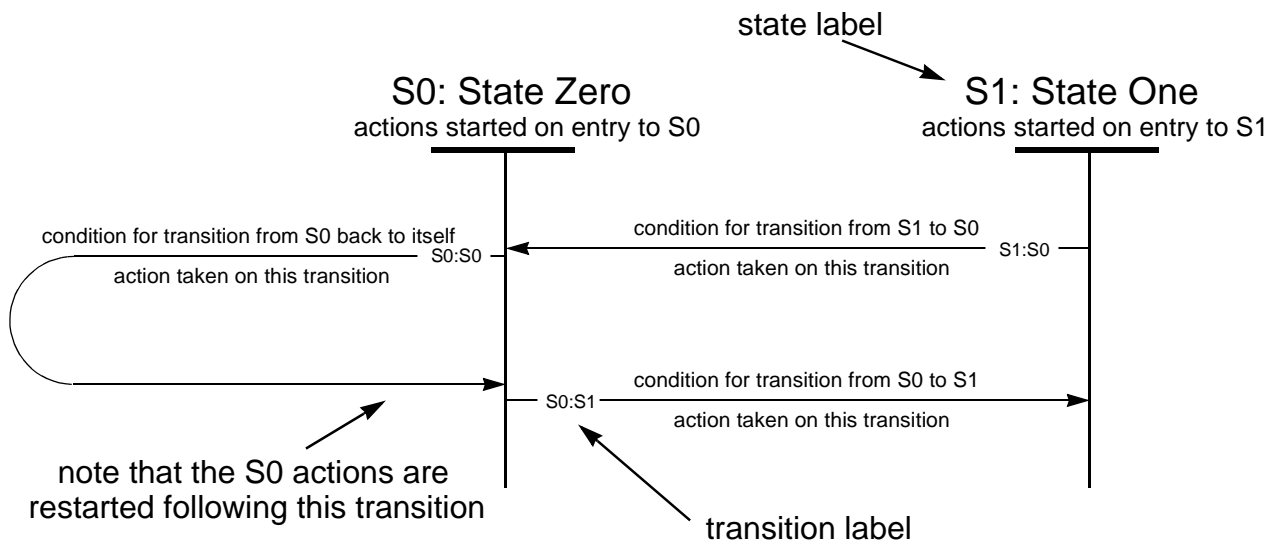


Figure 1-4—State machine example

These state machines make three assumptions:

- a) Time elapses only within discrete states.
- b) 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.
- c) 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 will repeat the actions 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 4. 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 1-5.

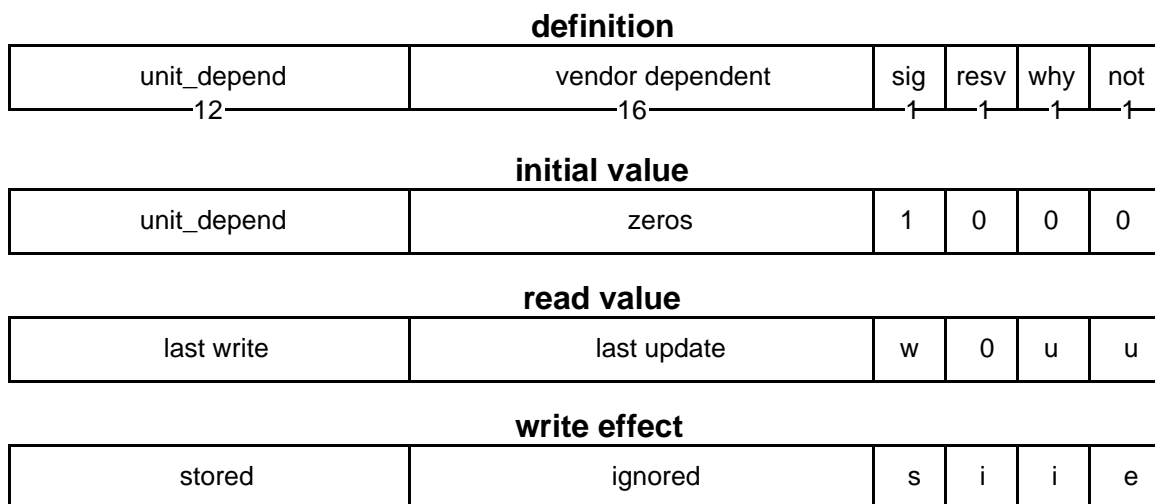


Figure 1-5—CSR format specification (example)

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 1-5 have the following meanings.

Table 1-4—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) or when a quadlet is written to the node’s RESET_START register (command_reset). If a CSR’s power_reset or command_reset values differ from its initial values, all values are explicitly specified.

The read value fields in figure 1-5 have the following meanings.

Table 1-5—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 1-5 have the following meanings.

Table 1-6—Write value 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 is not 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 clause 2.1). Within a CSR, such a field that is labeled *reserved* (sometimes abbreviated as lowercase *r* or *resv*). Reserved fields behave as specified by figure 1-6: they shall be zero and any attempt to write to them shall be ignored.

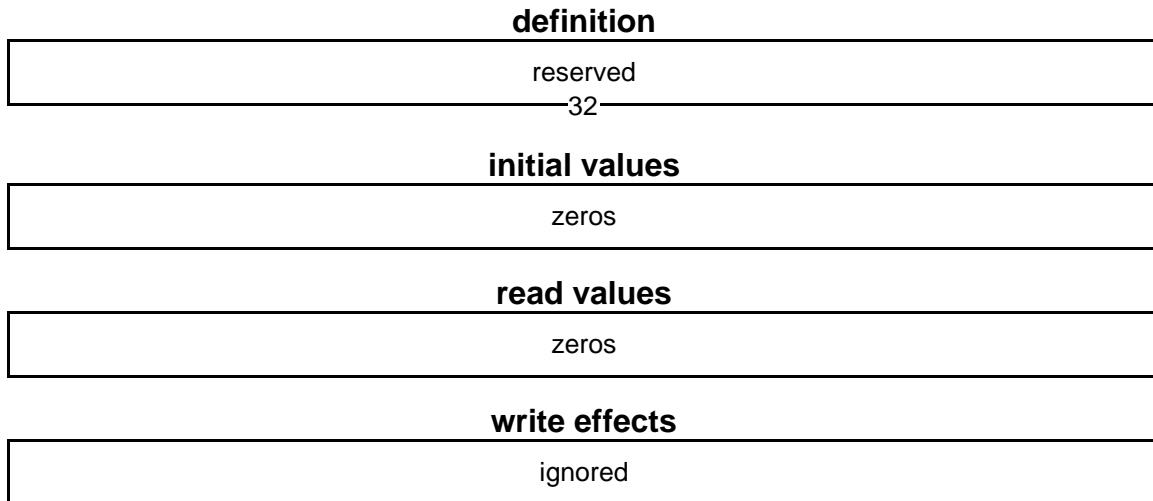


Figure 1-6—Reserved CSR field behavior

This is straight-forward as it applies to read and write requests. The same rules apply to lock requests, but the behaviors are less obvious. Table 1-7 summarizes the different lock functions specified by IEEE Std 1394-1995.

Table 1-7—Summary of lock functions

Lock function	Action
mask_swap	<code>new_value = data_value (old_value & ~arg_value);</code>
compare_swap	<code>if (old_value == arg_value) new_value = data_value;</code>
fetch_add	<code>new_value = old_value + data_value;</code>
little_add	<code>(little) new_value = (little) old_value + (little) data_value;</code>
bounded_add	<code>if (old_value != arg_value) new_value = old_value + data_value;</code>
wrap_add	<code>new_value = (old_value != arg_value) ? old_value + data_value : data_value;</code>

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.

The behavior of a particular lock function is determined by applying rules for reserved fields in order, as follows:

- a) The CSR's *old_value* is obtained as if *via* a read request and returned in the lock response; reserved fields are read as zeros;
- b) An intermediate value is calculated according to the C code above (this is not explicitly shown but is the right-hand part of each of the assignment statements in the table); and
- c) The intermediate value is stored in the CSR as if *via* a write request; reserved fields are ignored and remain zero in the CSR. The contents of the CSR after this operation are the *new_value*.

1.6.12 Operation description priorities

The description of operations in this standard are done in three ways: state machines, C code segments and English language. If more than one description is present, then priority shall be given first to the state machines, then the C code and finally to the English text (including the state machine notes).

2. Definitions and abbreviations

2.1 Conformance glossary

Several keywords are used to differentiate between different levels of requirements and optionality, as defined below. This clause replaces existing clause 2.1 of IEEE Std 1394-1995, "Conformance glossary," in its entirety; the following definitions shall apply to both IEEE Std 1394-1995 and this supplement.

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 in cases where either the object or the code value is set aside for future standardization. Usage and interpretation may be specified by future extensions to this or other IEEE standards. A reserved object shall be zeroed 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 a defined object 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."

2.2 Technical glossary

The following are terms that are used within this standard:

2.2.1 acknowledge: An acknowledge packet.

2.2.2 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.

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

2.2.4 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.

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

2.2.6 arbitration signalling: A protocol for the exchange of bidirectional, unclocked signals between nodes during arbitration.

2.2.7 asynchronous packet: A primary packet transmitted in accordance with asynchronous arbitration rules (outside of the isochronous interval).

2.2.8 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 \pm 100 ppm.

2.2.9 bus ID: A 10-bit number that uniquely identifies a particular bus within a group of interconnected buses.

2.2.10 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.

2.2.11 byte: Eight bits of data.

2.2.12 cable PHY: Abbreviation for the cable physical layer.

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

2.2.14 connected PHY: A peer cable PHY at the other end of a particular physical connection from the local PHY.

2.2.15 CSR Architecture: ISO/IEC 13213:1994 [ANSI/IEEE Std 1212, 1994 Edition], Information technology—Micro-processor systems—Control and Status Registers (CSR) Architecture for microcomputer buses.

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

2.2.17 cycle start packet: A primary packet sent by the cycle master that indicates the start of an isochronous **interval**.

2.2.18 doublet: Two bytes, or 16 bits, of data.

2.2.19 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.

2.2.20 gap: A period of idle bus.

2.2.21 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. See either ISO/IEC 13213:1994 or IEEE Std 1394-1995 for more information on address spaces.

2.2.22 initial register space: A two kilobyte portion of initial node space with a base address of FFFF F000 0000₁₆. 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 IEEE Std 1394-1995.

2.2.23 initial units space: A portion of initial node space with a base address of FFFF F000 0400₁₆. 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.

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

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

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

2.2.27 isochronous resource manager: The node that contains CSRs (BUS_MANAGER_ID, BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE) that permit the cooperative allocation of isochronous resources.

2.2.28 isochronous subaction: Within the isochronous interval, either a concatenated packet or a packet and the gap that preceded it.

2.2.29 kilobyte: A quantity of data equal to 2^{10} bytes.

2.2.30 link layer (LINK): The Serial Bus protocol layer that provides confirmed and unconfirmed transmission or reception of primary packets.

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

2.2.32 nibble: Four bits of data.

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

2.2.34 node ID: A 16-bit number that uniquely differentiates a node from all other nodes within a group of interconnected buses. The 10 most significant bits of node ID are the same for all nodes on the same bus; this is the bus ID. The six least-significant bits of node ID are unique for each node on the same bus; this is called the physical ID. The physical ID is assigned as a consequence of bus initialization.

2.2.35 octlet: Eight bytes, or 64 bits, of data.

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

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

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

2.2.39 physical ID: The least-significant 6 bits of the node ID. On a particular bus, each node's physical ID is unique.

2.2.40 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.

2.2.41 port: The part of the PHY that allows connection to one other node.

2.2.42 primary packet: Any packet that is not an acknowledge or a PHY packet. A primary packet is an integral number of quadlets and contains a transaction code in the first quadlet.

2.2.43 quadlet: Four bytes, or 32 bits, of data.

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

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

2.2.46 self-ID packet: A PHY packet transmitted by a cable PHY during the self-ID phase or in response to a PHY ping packet.

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

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

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

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

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

2.2.52 terabyte: A quantity of data equal to 2^{40} bytes.

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

2.2.54 transaction: A request and the optional, corresponding response.

2.2.55 unified transaction: A transaction completed in a single subaction.

2.2.56 unit: A component of a Serial Bus node that provides processing, memory, I/O or some other functionality. Once the node is initialized, the unit provides a CSR interface. A node may have multiple units, which normally operate independently of each other.

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

3. Overview

Proposed for inclusion by the editorial review session participants:

- Overview of arbitration enhancements (rationale, *etc.*)
- Asynchronous streams
- Performance optimization (PHY pinging and how we use it)
- Connection management protocol
- Fairness modifications (budget for requests, exemptions for one or more responses)
- Power and electrical

4. Alternative cable media attachment specification

The facilities of Serial Bus, 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 standard cable and connectors specified by the existing 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 power conductors of the standard cable 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 standard cable and connectors are relatively bulky. A more compact design would be better suited to these products.

The alternative cables and conductors specified by this supplement enable backwards compatibility with the standard cables specified by IEEE Std 1394-1995. 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.)

With respect to these alternative cables and connectors, only, this section entirely replaces clause 4.2.1 of IEEE Std 1394-1995. Except as superseded by other sections in this supplement, all other clauses in section 4 of the existing standard, “Cable physical layer specification,” continue to apply to alternative cables and connectors.

4.1 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. IEEE Std 1394-1995 specifies standard connectors that have six contacts; this supplement specifies alternative connectors that have four contacts.

All dimensions, tolerances and descriptions of features which affect the intermateability of the alternative shielded connector plugs and sockets are specified within this clause. Features of connector plugs and sockets which do not affect intermateability are not specified and may vary at the option of the manufacturer. Connector features which are not directly controlled within this clause shall be indirectly controlled by performance requirements in clauses 4.3 and 4.4.

The holes and patterns (footprint) for the mounting of some of the possible versions of connectors to the printed circuit board are recommended in clause 4.1.8

4.1.1 Connector plug

The mating features of the connector plug are specified in figures 4-1 and 4-2. They will assure the intermateability of the plug with the alternative sockets specified by this supplement.

It is recommended that the plug contacts have a cylindrical section in the contact area which makes contact at a right angle to the cylindrical section of the socket contacts, thus creating a “crossed cylinders” configuration. The contacts should be designed to create a Hertzian stress, (combination of cylindrical radius, normal force and base and surface material hardnesses) of 225,000-275,000 psi in the mating area. This is to assure that the low-energy signals used in this physical layer are transmitted through the non conductive films which are typically adsorbed on connector contacts.

NOTE—When a cable assembly plug is mated with a socket connector, there shall be 1.0 mm clearance, minimum, between the overmold on the assembly plug and the shield flange on the socket. This clearance is designed into the system to allow proper mating of both passive and latching cable plug assemblies. Deviation of this clearance may affect the performance of the connector interface.

Figures 4-1 and 4-2 describe a plug intended to be used when only detent retention with the socket is required.

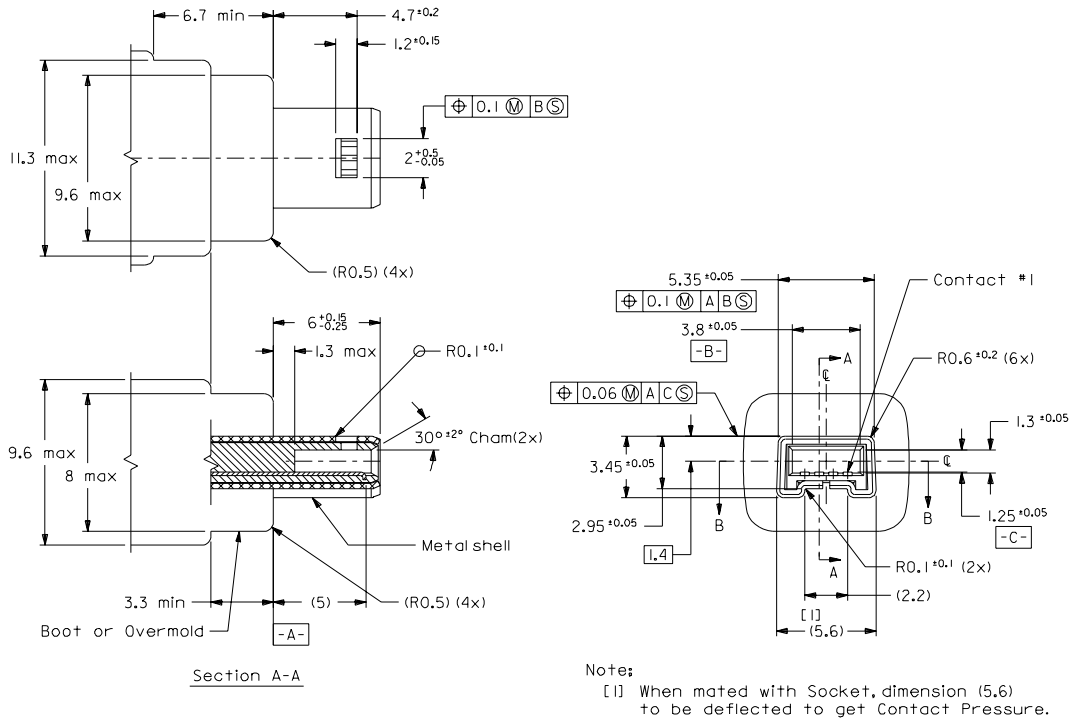


Figure 4-1 — Plug body

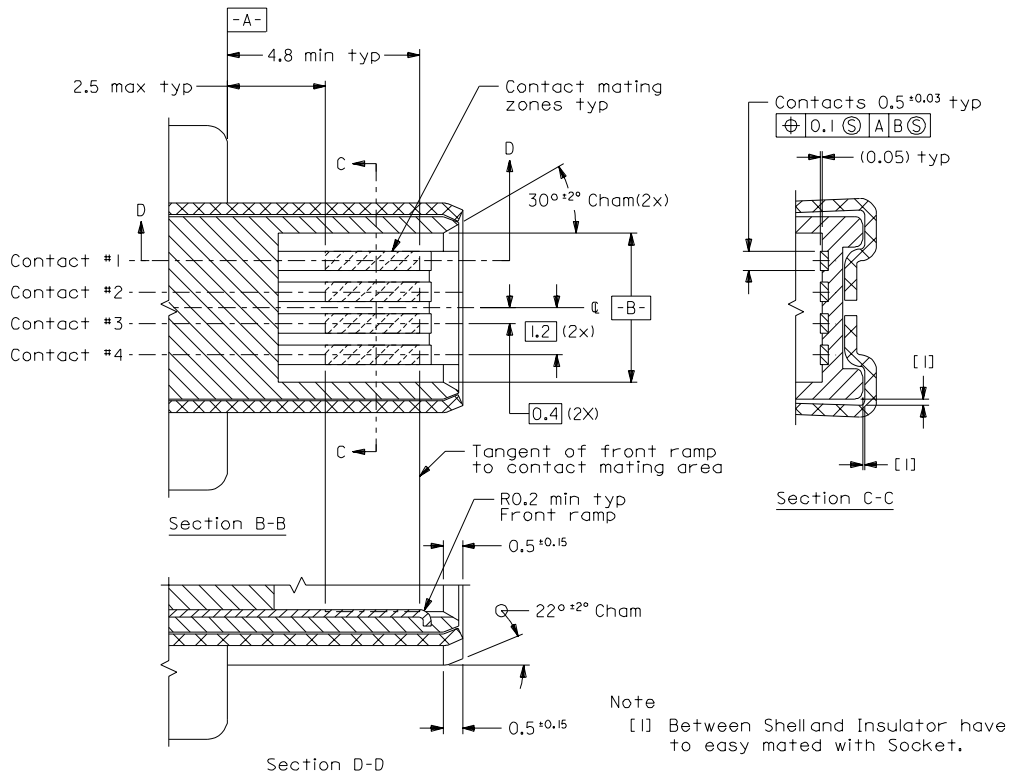


Figure 4-2 — Plug section details

4.1.2 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

4.1.3 Connector socket

The mating features of the connector socket are described in figures 4-3 through 4-5. They will assure the intermateability of the socket with the alternative plugs specified by this supplement.

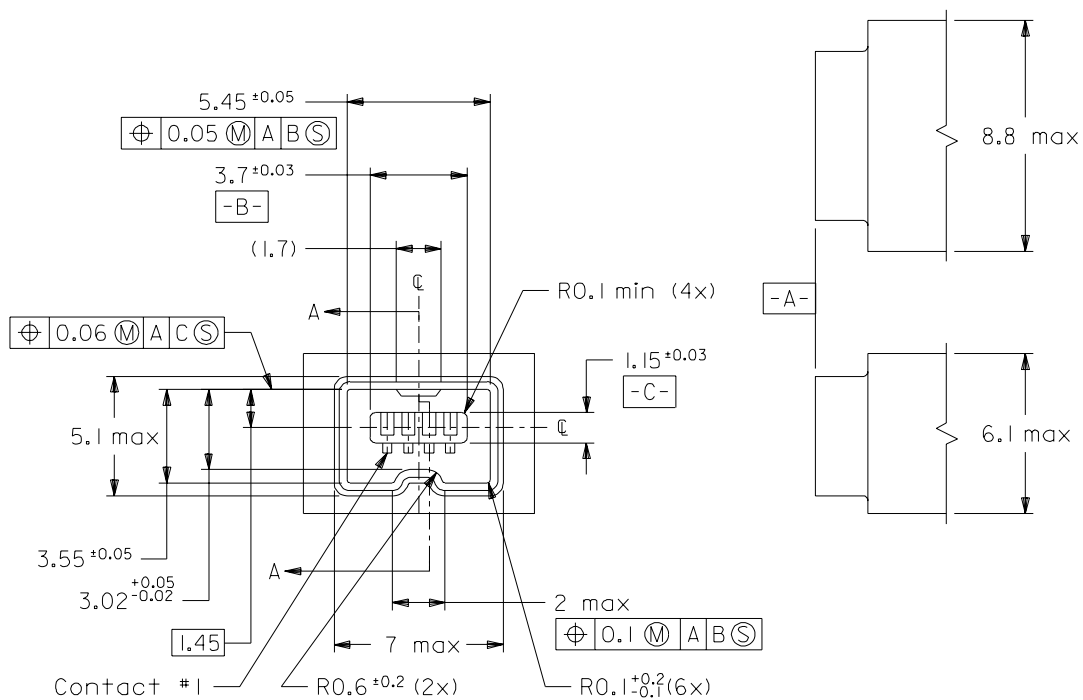


Figure 4-3 — Connector socket interface

The contacts are attached to the signals using the guidance in table 4-1.

Table 4-1 — Connector socket signal assignment

Contact number	Signal name	Comment
1	TPB*	Strobe on receive, data on transmit (differential pair)
2	TPB	
3	TPA*	Strobe on receive, data on transmit (differential pair)
4	TPA	

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

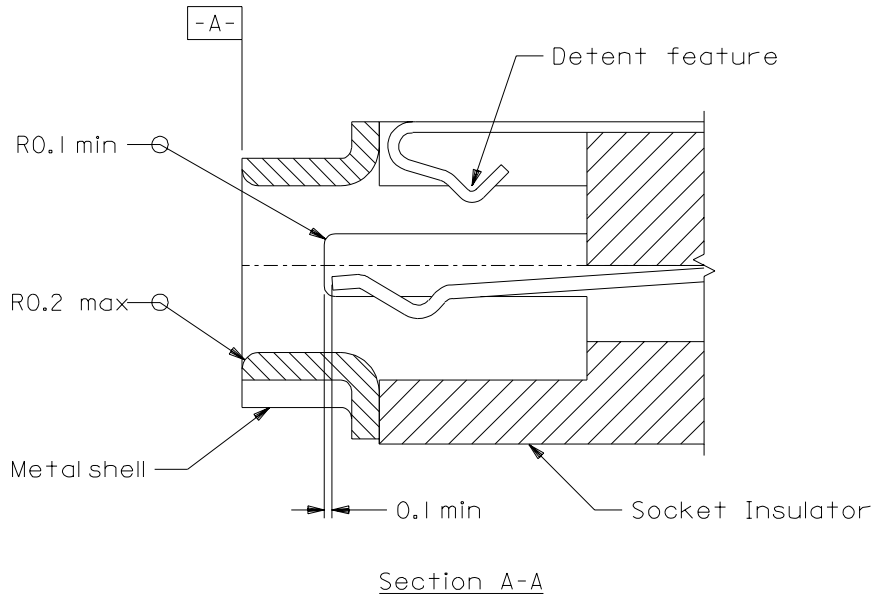


Figure 4-4 — Socket cross-section A-A

Figure 4-5 shows the mated cross section of the plug and socket contacts.

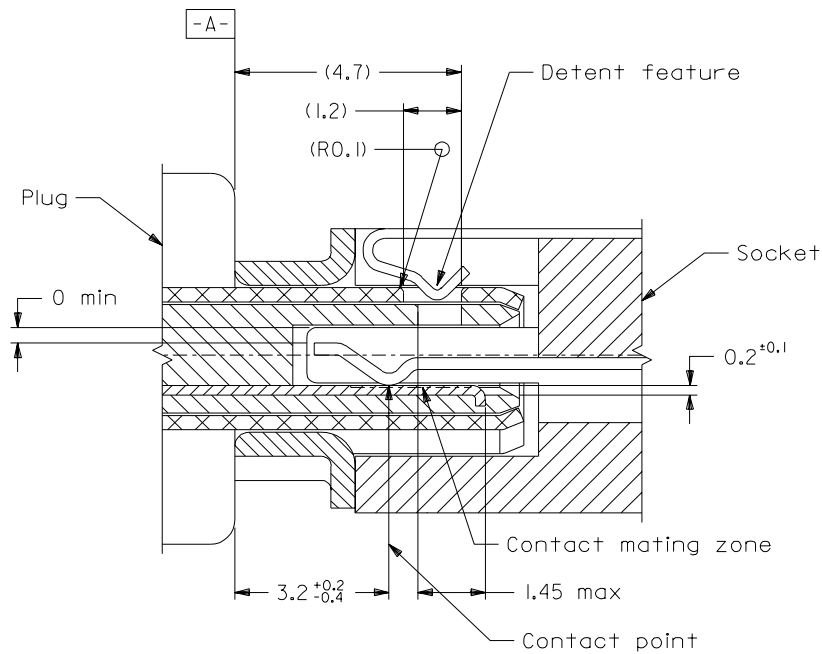


Figure 4-5 — Cross-section of plug and socket contacts

When mounted on a printed circuit board, the socket shall be at a fixed height as illustrated by figure 4-6.

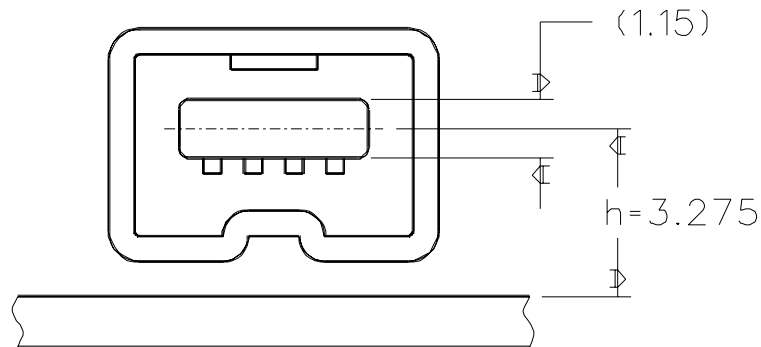


Figure 4-6 — Socket position when mounted on a printed circuit board

4.1.4 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.

- 0.76 μm (30 μin), minimum, gold, over 1.27 μm (50 μin), minimum, nickel.
- 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—A copper strike is acceptable, under the nickel electroplate.

4.1.5 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.

4.1.6 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.

4.1.7 Connector durability

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

4.1.8 Printed circuit board footprints

The footprint of a surface-mount printed circuit board connector shall conform to the dimensional specifications illustrated by figure 4-7 below.

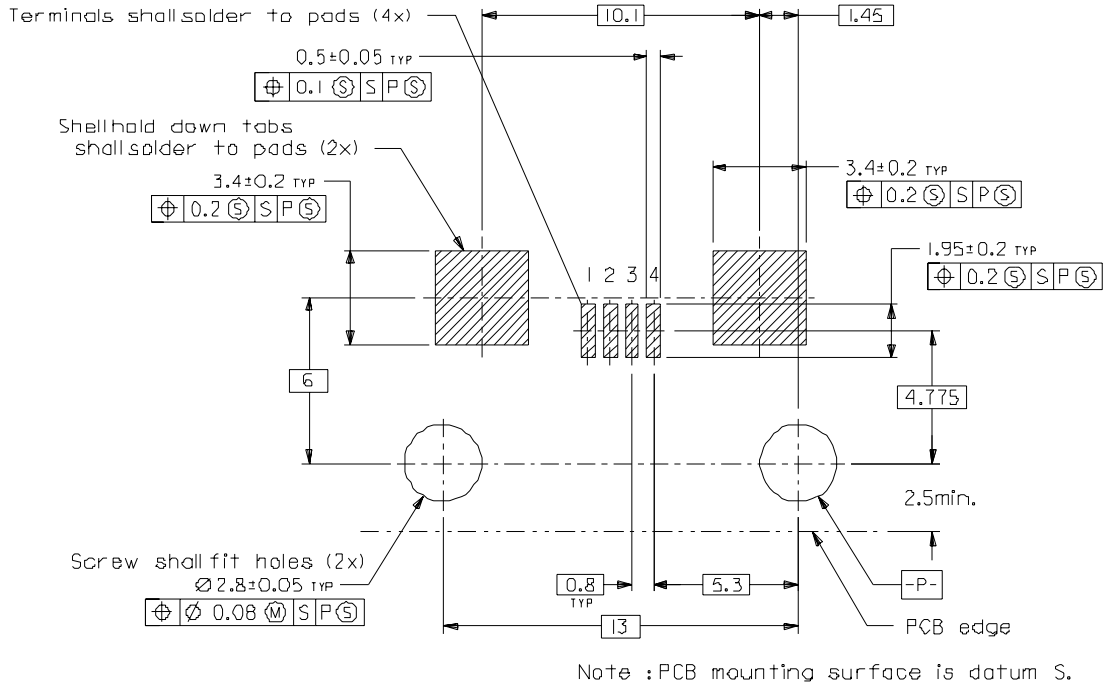


Figure 4-7 — Flat surface mount printed circuit board connector footprint

The footprint of a through-hole printed circuit board connector shall conform to the dimensional specifications illustrated below.

Figure 4-8 — Flat through-hole mount printed circuit board connector footprint

4.2 Cables

All cables and cable assemblies shall meet assembly criteria and test performance found in this supplement.

4.2.1 Cable material (reference)

Linear cable material typically consists of two twisted pair conductors. The two twisted pairs carry the balanced differential data signals. Figure 4-9 illustrates a reference design adequate for a 4.5 m cable. Clause 4.4 describes the performance requirements for the cable assembly.

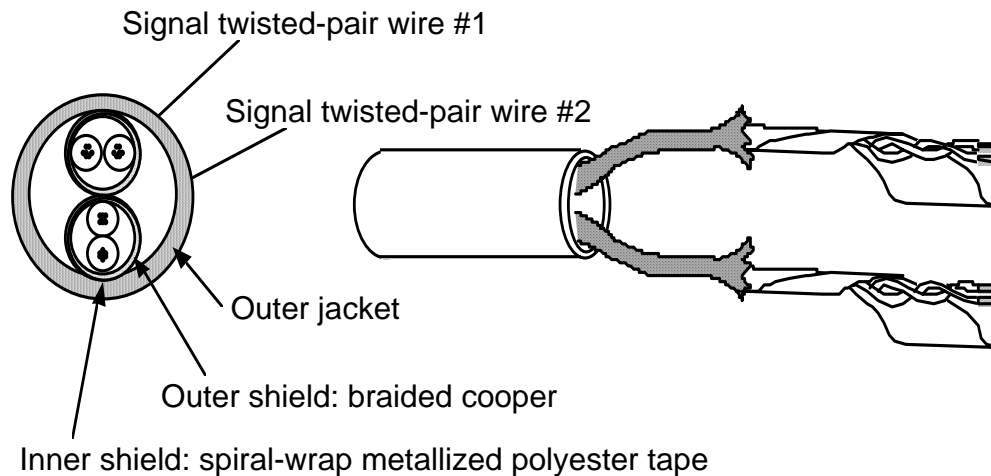


Figure 4-9 — Cable material construction example (for reference only)

CAUTION—Figure 4-9 above does not reflect action taken at the September 25, 1997, IEEE P1394a Working Group meeting in Natick, MA, at which the provisions of document 97-060r0 were adopted unanimously. Please consult the minutes and the FTP site, <ftp://ftp.symbios.com/pub/standards/io/1394/P1394a>, for the most recent information.

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

4.2.2 Cable assemblies

Cable assemblies consist of two plug connectors, either the standard connector specified by IEEE Std 1394-1995 or the alternative connector defined by this supplement, joined by a length of cable material. 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 considerations is given to the actual Serial Bus system topology to be used, as discussed in greater detail in annex A of IEEE Std 1394-1995.

Both cable configurations, standard connector to standard connector and alternative connector to standard connector, are illustrated in the figures below. The connector pins are terminated as shown by figures 4-10 and 4-11. The two signal pairs “cross” in the cable to effect a transmit-to-receive interconnection.

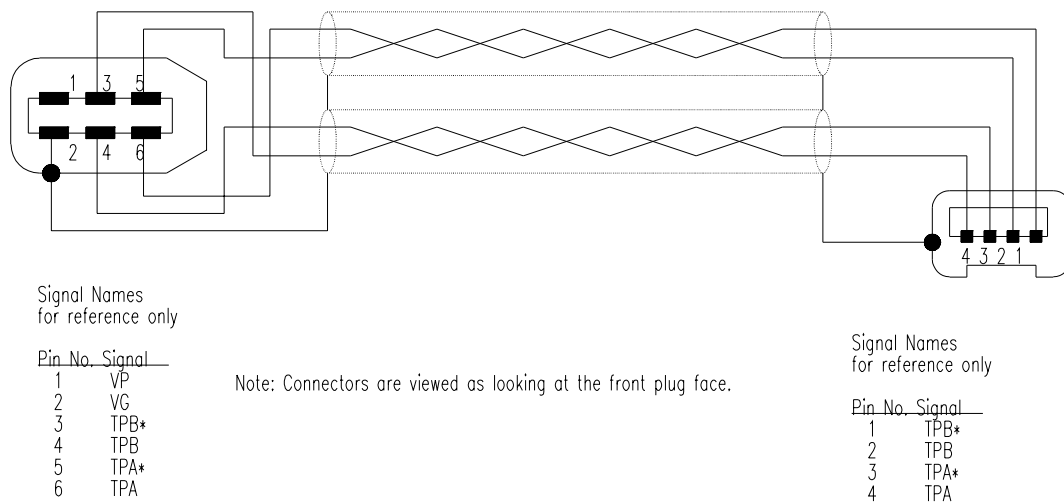


Figure 4-10 — Cable assembly and schematic (standard to alternate connector)

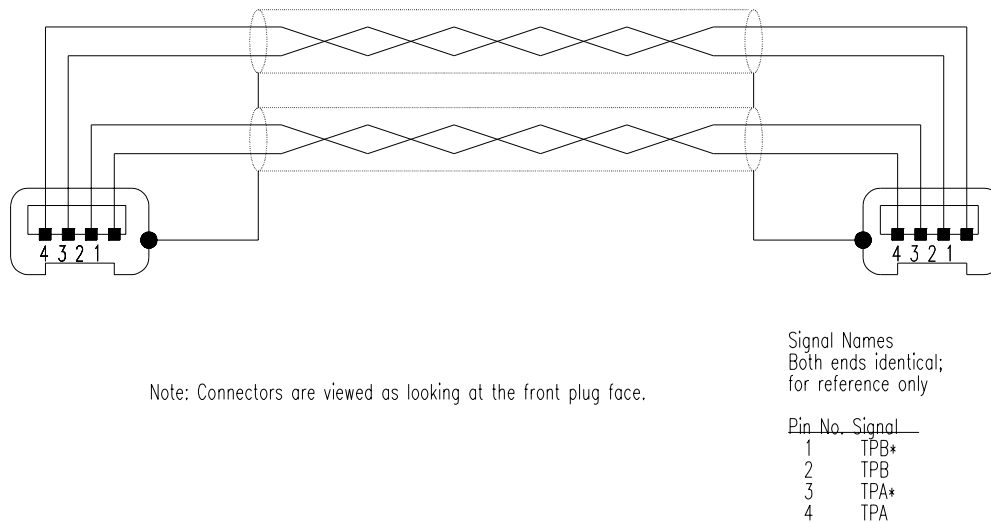


Figure 4-11—Cable assembly and schematic (alternate connectors)

CAUTION—The figures above do not reflect action taken at the September 25 IEEE P1394a Working Group meeting in Natick, MA, at which the provisions of document 97-060r0 were adopted unanimously. Please consult the minutes and the FTP site, <ftp://ftp.symbios.com/pub/standards/io/1394/P1394a>, for the most recent information.

4.3 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-B-90. Table 1 of ANSI/EIA 364-B-90 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 follows: “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; + 15 degree C to + 85 degrees 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, 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 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 will be 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 which 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 which 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 which conforms to this specification. In order to test to these performance groups, ANSI/EIA tests require that the cable construction used be specified.

2—All resistance values shown in the following performance groups 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.

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.

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

Number of samples:

- [2] Sockets, unassembled to printed circuit board used for Phase 1, A1 and A2 (one each).
- [2] Sockets, assembled to printed circuit board
- [2] Plugs, unassembled 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.

Table 4-2 — 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 figures 4-1 through 4-5	No defects that would impair normal operations. No deviation from dimensional tolerances.
A2	Plating thickness measurement					Record thickness; see 4.1.4
A3	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated pair.
A4	Vibration	ANSI/EIA 364-28A-83	Condition I (See note)	Continuity	ANSI/EIA 364-46-84	No discontinuity at 1 μs or longer. (Each contact)
A5	None			Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.
A6	Mechanical shock (specified pulse)	ANSI/EIA 364-27A-83	Condition A (See note)	Continuity	ANSI/EIA 364-46	No discontinuity at 1 μs or longer. (Each contact)
A7	None			Low-level contact resistance	ANSI/EIA 364-23	20 mΩ maximum change from initial per mated contact.

NOTE—Connectors are to be mounted on a fixture which simulates typical usage. The socket shall be mounted to a panel which 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 accord with the pattern shown in figure? 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 in IEEE Std 1394-1995 for details.

4.3.2 Performance group B: Low-level contact resistance when subjected to thermal shock and humidity stress

Number of samples:

- [0] Sockets, unassembled to printed circuit board
- [2] Sockets, assembled to printed circuit board
- [0] Plugs, unassembled to cable.
- [2] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-3 — 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	IEC 68-2-14	10 cycles (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.
B3	Humidity	ANSI/EIA 364-31A-83	Condition A (96 h.) Method II (cycling) nonenergized Omit steps 7a and 7b. (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.

4.3.3 Performance group C: Insulator integrity when subjected to thermal shock and humidity stress

Number of samples:

- [2] Sockets, unassembled to printed circuit board
- [0] Sockets, assembled to printed circuit board
- [2] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one).
- [0] Cable assemblies with a plug assembled to one end, 2 m long.

Table 4-4 — 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 Vdc ± 10 Vdc Method C (unmated and unmounted)	Withstanding voltage	ANSI/EIA 364-20A-83	No flashover. No sparkover. No excess leakage. No breakdown.
C2	Thermal shock	IEC 68-2-14	10 cycles (unmated)	Withstanding voltage (same conditions as C1)	ANSI/EIA 364-20A-83	No flashover. No sparkover. No excess leakage. No breakdown.
C3	Insulation resistance	ANSI/EIA 364-21A-83	Test voltage 100 Vdc ± 10 Vdc (unmated and unmounted)	Insulation resistance	ANSI/EIA 364-21A-83	1 GΩ, minimum, between adjacent contacts and contacts and shell.
C4	Humidity (cyclic)	ANSI/EIA 364-31A-83	Condition A (96 h.) Method III nonenergized Omit steps 7a and 7b	Insulation resistance (same conditions as C3)	ANSI/EIA 364-21A-83	1 GΩ, minimum.

4.3.4 Performance group D: Contact life and durability when subjected to mechanical cycling and corrosive gas exposure

Number of samples:

[0] Sockets, unassembled to printed circuit board

[4] Sockets, assembled to printed circuit board

[0] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one).

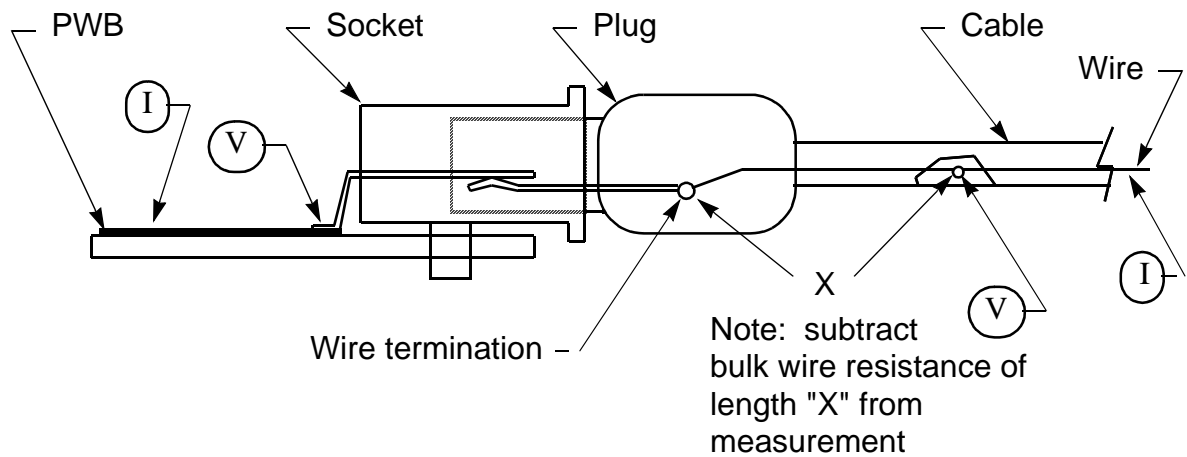
[4] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-5 — Performance group D

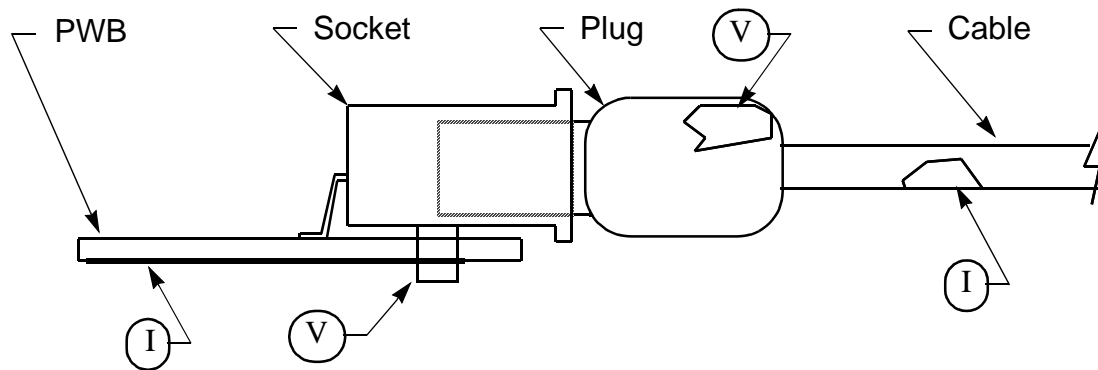
Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
D2	Continuity-housing (shell)		See figure 4-12 for measurement points	Contact resistance, braid to socket shell	ANSI/EIA 364-06A-83	50 mΩ maximum, initial from braid to socket shell at 100 mA, 5 Vdc open circuit max.
D3	Durability	ANSI/EIA 364-09B-91	(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	20 mΩ maximum change from initial per mated contact.
D5	Continuity-housing (shell)		See figure 4-12 for measurement points	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell.
D6	Mixed flowing gas	ANSI/EIA 364-65-92	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	20 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			

Table 4-5 — Performance group D (Continued)

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D8	Mixed flowing gas	ANSI/EIA 364-65-92	Class II Exposures: Expose mated for 10 day	Low level contact resistance at end of exposure	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.
D9	Continuity-housing (shell)		See figure 4-12 for measurement points	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell.



a) contact resistance



b) shield resistance

Figure 4-12 — Shield and contact resistance measuring points

4.3.5 Performance group E: Contact resistance and unmating force when subjected to temperature life stress

Number of samples:

- [0] Sockets, unassembled to printed circuit board
- [2] Sockets, assembled to printed circuit board
- [0] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one).
- [2] Cable assemblies with a plug assembled to one end, 2 m long.

Table 4-6 — 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 4-12	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum initial from braid to socket shell.
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	20 mΩ maximum change from initial per mated contact.
E5	Continuity-housing (shell)			Contact resistance	ANSI/EIA 364-06A-83	50 mΩ maximum change from initial from braid to socket shell.
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

4.3.6 Performance group F: Mechanical retention and durability

Number of samples:

- [0] Sockets, unassembled to printed circuit board
- [2] Sockets, assembled to printed circuit board
- [0] Plugs, unassembled to cable.
- [2] Plugs, assembled to cable, one end only, 25 cm long.

Table 4-7 — 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 \pm 50 cycles	Unmating only	ANSI/EIA 364-13A-83	Unmating force at end of durability cycles: 4.9 N minimum 39.0 N maximum

4.3.7 Performance group G: General tests

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

Since the tests listed below 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 4-8 — 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 801-2	1 to 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 plugs]		Fix plug housing and apply a 49.0 N load for one minute on cable axis.	Continuity, visual	ANSI/EIA 364-46	No discontinuity on contacts or shield greater than 1 μ s 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 plugs]	ANSI/EIA 364-41B-89	Condition I, dimension X=5.5 x cable diameter; 100 cycles in each of two planes	(a) Withstanding voltage	Per C1	Per C1
				(b) Insulation resistance	Per C3	Per C3
				(c) Continuity	ANSI/EIA 364-46-84	No discontinuity on contacts or shield greater than 1 μ s 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 4-13 — Fixture for cable flex test

4.4 Signal propagation performance criteria

The test procedures for all parameters listed in this clause are described in Annex K of IEEE Std 1394-1995.

4.4.1 Signal impedance

The differential mode characteristic impedance of the signal pairs shall be measured by time domain reflectometry at < 100 ps rise time using the procedure described in annex K.3 of IEEE Std 1394-1995:

$$Z_{TPA} = (110 \pm 6) \Omega \text{ (differential)}$$

$$Z_{TPB} = (110 \pm 6) \Omega \text{ (differential)}$$

The common mode characteristic impedance of the signal pairs shall be measured by time domain reflectometry at < 100 ps rise time using the procedure described in annex K.3 of IEEE Std 1394-1995:

$$Z_{TPACM} = (33 \pm 6) \Omega \text{ (common mode)}$$

$$Z_{TPBCM} = (33 \pm 6) \Omega \text{ (common mode)}$$

4.4.2 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 annex K.4 of IEEE Std 1394-1995.

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

4.4.3 Signal pairs propagation delay

The differential propagation delay of the signal pairs through the cable shall be measured in the frequency domain using the procedure described in annex K.5 of IEEE Std 1394-1995:

$$V_{\text{TPA}} \leq 5.05 \text{ ns/meter}$$

$$V_{\text{TPB}} \leq 5.05 \text{ ns/meter}$$

NOTE—The common mode propagation delay of the signal pairs should be the same or less than the differential propagation delay. No test procedure is described for this in annex K.5 since this will be the case for all but the most exotic cable constructions:

$$V_{\text{TPACM}} \leq 5.05 \text{ ns/meter}$$

$$V_{\text{TPBCM}} \leq 5.05 \text{ ns/meter}$$

4.4.4 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 using the procedure described in annex K.6.1 of IEEE Std 1394-1995:

$$S \leq 400 \text{ ps}$$

4.4.5 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 MHz to 500 MHz using the procedure described in annex K.8 of IEEE Std 1394-1995:

$$X \leq -26 \text{ dB}$$

5. PHY/Link interface specification

This section standardizes the PHY/Link interface previously described in an informative annex of IEEE Std 1394-1995. It covers the protocol and signal timing. It does not cover specific operation of the PHY except for behavior with respect to this interface.

The interface specified in this section is a scalable method to connect one Serial Bus link chip to one Serial Bus PHY chip. The width of the data bus scales with the highest speed both chips can support, using two signals per 100 Mbps. The clock rate of the signals at this interface remains constant, independent of speed, and supports isolation for implementations where it is desirable.

The interface may be used to transmit data, receive data or status, or issue requests. The link makes requests of the PHY *via* the dedicated LReq signal. In response, the PHY may transfer control of the bidirectional signals to the link. At all other times the PHY controls the bidirectional signals.

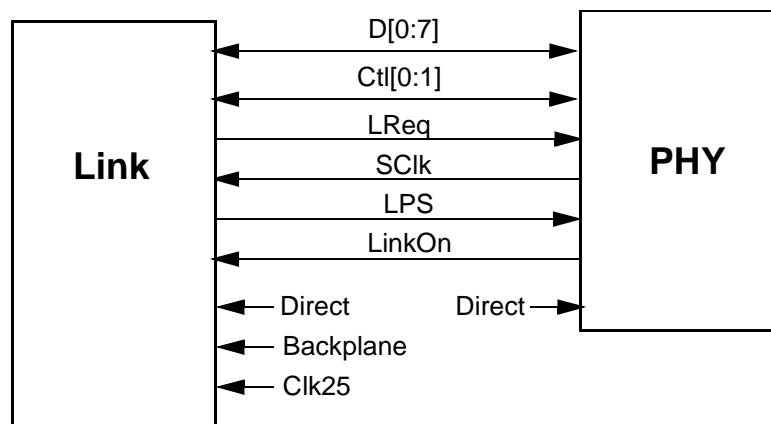


Figure 5-1 — Discrete PHY/Link interface

With the exception of D[0:7], discrete PHY implementations shall support all of the PHY signals shown in figure 5-1. The number of data bits implemented depends upon the maximum speed supported by the PHY. Discrete link implementations shall implement D[0:7], Ctl[0:1], LReq and SClk. Link support for the other signals is optional. These signals are described by table 5-1.

Table 5-1 — PHY/Link signal description

Name	Driven by	Description
D[0:7]	Link or PHY	Data
Ctl[0:1]	Link or PHY	Control
LReq	Link	Link request port
SClk	PHY	24.576 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	Receipt of a link-on packet. Once asserted it shall remain asserted until LPS is asserted or the PHY register L bit is set to one.
Direct	Neither	Set high to indicate a direct connection or low to indicate an isolation barrier between the link and PHY
Backplane	Neither	Set high if backplane PHY
Clk25	Neither	Set high to notify the link that SClk is 24.576 MHz

NOTE—The Clk25 signal is meaningful only if the link is configured for backplane operations.

The LPS signal disables and enables the PHY/Link interface. The PHY shall enable the interface if LPS either is continuously asserted or, in the presence of an optional isolation barrier, is a nominal 300 KHz pulsed output that conforms to figure 5-2 and table 5-2. Otherwise, if LPS is not observed for T_{LPS_RESET} the PHY disables the interface by taking Ctl[0:1], Data[0:n] and Sclk to zero.

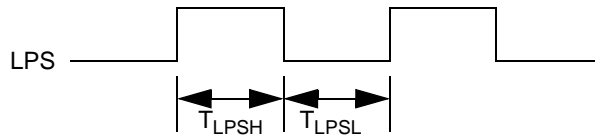


Figure 5-2 — LPS timing (isolated interface)

Table 5-2 — LPS timing parameters

Parameter	Description	Unit	Minimum	Maximum
T_{LPSL}	LPS low time (isolated interface)	μs	0.09	2.00
T_{LPSH}	LPS high time (isolated interface)	μs	0.09	2.00
T_{LPS_RESET}	Time for PHY to recognize absence of LPS	μs	2.25	2.75
T_{LPS_WAIT}	Time after PHY removes SClk until link may reassert LPS (isolated interface)	μs	10	

The interface may be reset by deasserting LPS for a minimum of 2.75 μs . When LPS is reasserted, the PHY shall resume SClk as soon as possible. The PHY shall assert zeros on Ctl[0:1] and D[0:n] for the first cycle of the resumed SClk and on the third and subsequent SClk cycles shall drive control and data in accordance with the arbitration state machines. When an isolated interface is present, the link shall assert zeros on Ctl[0:1], D[0:n] and LReq for the SClk cycle that immediately follows the resumption of SClk. The link shall ignore Ctl[0:1] prior to the third SClk cycle and thereafter shall ignore Ctl[0:1] until at least one Idle is observed. These timings are illustrated by figure 5-3.

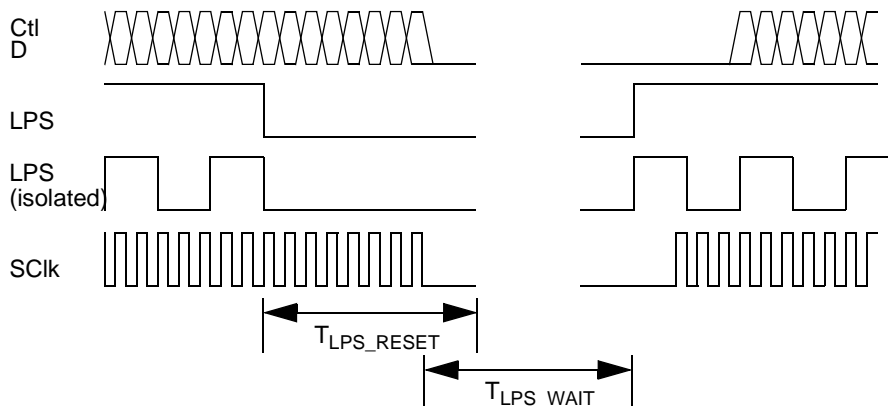


Figure 5-3 — PHY/Link interface reset via LPS

The interface described in this section supports data rates of S100, S200 and S400 in the cable environment and S25 and S50 in the backplane environment. In the timing diagrams in this section, each bit cell represents one clock sample time. The specific clock-to-data timing relationships are described in clauses 5.6.2 and 5.6.3.

The Ctl bus carries control information and is two bits wide. The LReq signal is used by the link to request access to Serial Bus, to read or write PHY registers or to control arbitration acceleration. The Direct signal indicates whether the two chips are connected directly or through an isolation barrier. When the Direct signal is asserted it shall disable the differentiator on the D, Ctl, SClk, LPS, LinkOn and LReq signals.

NOTE—The nominal DC signal LinkOn shall be pulsed in order that it may cross an optional isolation barrier.

Data is transferred between the PHY and link on D[n]. The implemented width of D[n] depends on the maximum speed of the connected PHY device: 2 bits for S100 or slower, 4 bits for S200 and 8 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[n] signals shall be driven low.

NOTE—In the backplane environment, transfers use D[0:1], but SClk is used to clock the transfers at either 24.756 MHz (for BTL and ECL applications) or 12.288 MHz (for TTL applications).

Whenever control is transferred between the PHY and the link, the side relinquishing control always drives the control and data buses to logic zero levels for one clock before tri-stating its output buffers. An additional clock with zero on the control and data signals is necessary for the link when it is transferring control to the PHY without a Hold request. This is necessary to ensure that the differentiator circuit can operate properly.

5.1 Operation

There are four operations that occur on the interface: link request, status transfer, data transmit and data receive. The link uses the link request operation to read or write an internal PHY register, to ask the PHY to initiate a transmit action **or to control arbitration acceleration**. The PHY may initiate a status transfer autonomously and shall initiate a receive action whenever a packet is received from Serial Bus.

The Ctl bus is 2 bits wide. The encoding of these signals is shown in tables 5-3 and 5-4.

Table 5-3 — 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	An incoming packet is being transferred from the PHY to the link.
11 ₂	Grant	The link is granted the bus to send a packet.

Table 5-4 — 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.

5.2 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 which 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 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 5-4.

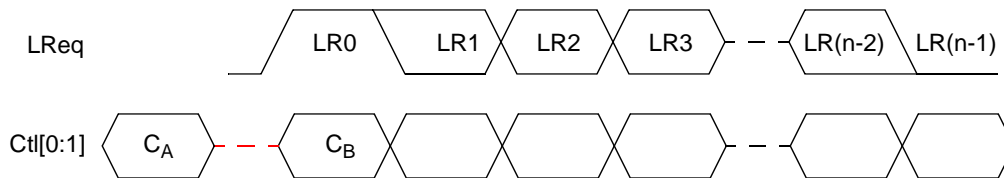


Figure 5-4 — LReq and Ctl timings

If the LReq transfer is a bus request in the cable environment, it is 7 or 8 bits long and has the format given in table 5-5. Links compliant with this supplement shall send all 8 bits.

NOTE—PHYs shall accept bus requests in both the format specified by IEEE Std 1394-1995 and the format shown below.

Table 5-5 — Bus request format for cable environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always 1.
1-3	Request Type	Indicates type of bus request—immediate, isochronous, priority or fair. See table 5-10 for the encoding of this field.
4-6	Request Speed	The speed at which the PHY will 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 5-11 for the encoding of this field.
7	Stop Bit	Indicates end of transfer. Always 0. 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 5-6.

Table 5-6 — Bus request format for backplane environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always 1.
1-3	Request Type	Indicates type of bus request—immediate, isochronous, priority or fair. See table 5-10 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 0.

If the transfer is a register read request, it is 9 bits long and has the format given in table 5-7.

Table 5-7 — Register read request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always 1.
1-3	Request Type	Indicates that this is a register read. See table 5-10 for the encoding of this field.

Table 5-7 — Register read request format

Bit(s)	Name	Description
4-7	Address	The internal PHY address to be read.
8	Stop Bit	Indicates end of transfer. Always 0.

If the transfer is a register write request, it is 17 bits long and has the format given in table 5-8.

Table 5-8 — Register write request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always 1.
1-3	Request Type	Indicates that this is a register write. See table 5-10 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 0.

If the transfer is an acceleration control request, it is 6 bits long and has the format given in table 5-9.

Table 5-9 — Acceleration control request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of request. Always 1.
1-3	Request Type	Indicates that this is an acceleration control request. See table 5-10 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 0.

The request type field is encoded as shown in table 5-10.

Table 5-10 — 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 a 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 5-11. Although encoding for speeds up to S3200 is specified below, the PHY/link interface defined by this supplement does not support speeds in excess of S400.

Table 5-11 — 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 clause 7.8. Table 5-12 summarizes the effects of the various link requests.

Table 5-12 — 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 5-11. The accelerating variable is affected only by an IsoReq, which sets it to TRUE.
RdReg	—	The values returned by a register read are unspecified after the PHY gives indication of a bus reset until the PHY successfully transfers register zero to the link.
WrReg	See note	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. A one written to the IBR register bits sets ibr TRUE, sets reset_time to RESET_TIME and initiates a bus reset. A one written to the ISBR register bit sets isbr TRUE, reset_time to SHORT_RESET_TIME and initiates an arbitrated (short) bus reset.
AccCtrl	accelerating	If the Accelerate bit in the request is zero accelerating is cleared to FALSE; otherwise it is set 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 5-14). 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 packet or immediately thereafter but before the next subaction gap. The PHY cancels an isochronous request when a subaction gap or bus reset is observed

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 two SClk **cycles** after observing *grant* on Ctl[0:1].

NOTE—Though highly unlikely, it is conceivable that more than one nodes may perceive (one correctly, the others mistakenly) that an incoming packet is intended for them and all issue acknowledge requests before checking the CRC. The PHYs of all nodes would grab control of the bus immediately after the packet is complete. This condition causes 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) drops their request(s) as soon as they have checked the CRC and the spurious “ZZ” line state vanishes. 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 an incoming packet, 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 packet transmission is complete or the request is cancelled. The PHY shall ignore bus requests issued while a previous request is pending.

5.2.1 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 clause 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, labelled C_A and C_B in figure 5-4. The rules are specified in terms of the values of the Ctl[0:1] lines during these cycles. C_A is one or more SClk cycles before the link sends the request’s start bit and C_B is 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 onwards.

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) until any outstanding bus request has been granted or the link has been able to determine that it has been cancelled;
- the link should not issue a register read request when a previous register read **or write** request is outstanding. PHY behavior in this case is undefined; and
- 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 5-13.

Table 5-13 — Link rules to initiate a request on LReq

Request	Permitted when PHY has control of the interface and Ctl[0:1] at C_A is	Permitted when link controls the interface	Additional requirements
Fair	Idle, Status	No	No fair or priority request shall be issued until any outstanding bus request completes.
Priority	Idle, Status	No	

Table 5-13 — Link rules to initiate a request on LReq (Continued)

Request	Permitted when PHY has control of the interface and Ctl[0:1] at C _A is	Permitted when link controls the interface	Additional requirements
Immediate	Receive, Idle	No	Sent after <i>destination_ID</i> 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 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 packet 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 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.
Register read	Any	Yes	Shall not be issued while there are pending register read requests.
Register write	Any	Yes	
Decelerate	Any	Yes	Decelerate is issued by cycle slaves if <i>enab_accel</i> 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	Any	Yes	Accelerate may only be issued by cycle slaves once every isochronous period, after an incoming cycle start packet has been recognized and after all isochronous requests.

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 treatment of the LReq by monitoring the value of Ctl[0:1] from C_B onwards, as shown in the following table:

Table 5-14 — PHY disposition of link request

Request	Ctl[0:1] (at C _B or later)	PHY behavior	Link action
Fair, Priority	Receive	If arbitration acceleration is enabled, and the incoming packet is null or has no more than 8 bits, then request retained, otherwise request discarded as soon as the PHY determines that the incoming packet has more than 8 bits. Request always discarded if arbitration acceleration is not enabled.	Continue to monitor for the next change on Ctl[0:1] if request 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 enhanced arbitration is not enabled in the PHY, the request is cancelled. If enhanced arbitration is enabled in the PHY, then the request is cancelled only if the received packet is not greater than 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.

Table 5-14 — PHY disposition of link request (Continued)

Request	Ctl[0:1] (at C _B or later)	PHY behavior	Link action
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.
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 retained unless status reports a bus reset	Continue to monitor for the next change on Ctl[0:1] if request 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, and 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	

5.2.2 Acceleration control

The **ack-accelerated** and fly-by arbitration enhancements specified in clause 7.8 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** or fly-by acceleration by **node(s) other than the root** can prolong asynchronous traffic on the bus indefinitely and disrupt isochronous operations.

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, but the link at any node that is not the cycle master shall use the acceleration control request as follows:

- a) The time period in which **neither ack-accelerated nor** fly-by accelerations shall not be requested extends from the time of the local cycle synchronization event until one of an arbitration reset gap, cycle start packet or two subaction gaps are observed;
- b) **During this period, a 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;**
- c) Upon conclusion of the time period defined in a), the link may reenable **ack-accelerated** and fly-by acceleration by transmitting an acceleration control request whose Accelerate bit is set to one.

NOTE—Link pattern-matching to detect a cycle start packet should examine the entire first quadlet of received primary packets for the value FFFF 008F₁₆ and not rely solely upon an observed transaction code of 8. The values of the cycle start packet fields are specified by clause 6.2.2.2.3 of IEEE Std 1394-1995.

A link that makes one or more bus requests to transmit an isochronous packet need not use the acceleration control request to reenable 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.

5.3 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 to perform the transfer. The PHY initiates the transfer by asserting `status` (01₂) on Ctl[0:1], along with the first two bits of status information on D[0:1]. The PHY asserts `status` on Ctl[0:1] for the duration of the status transfer. The PHY may prematurely end a status 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 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, four bits of status are sent to the link. The timing for a status transfer is shown in figure 5-5.

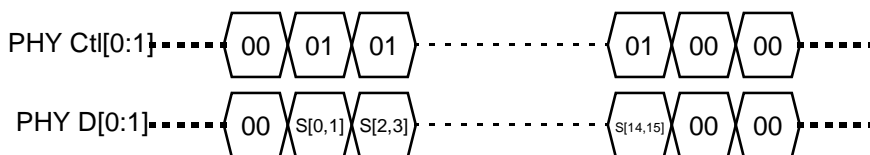


Figure 5-5 — Status timing

The structure of the status data is specified by table 5-15 below.

Table 5-15 — 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	PHY_interrupt	This indicates one or more of the following interrupt conditions: - Loop detect interrupt - Cable power fail interrupt - Arbitration state machine time-out - Bias change detect (only on a disabled port) 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.

NOTE—The PHY clears ARB_RESET_GAP and SUBACTION_GAP upon any transition out of state A0: Idle (see figure 7-9)—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 zero whichever of the four status bits 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.

5.4 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]. The link should assert *idle* for at least 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. 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 tri-stating those signals.

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 5-16. 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 at least one SClk cycle, but shall not assert *idle* for more than one cycle, before changing Ctl[0:1] to *hold* or *transmit*. The link shall ensure that the time from when it asserts *hold* on Ctl[0:1] (at the end of a packet) to when it asserts *transmit* on Ctl[0:1] (and starts to provide data for the concatenated packet on D[0:n]) does not exceed MAX_BUS_HOLD.

The link may transmit concatenated packets at a different speeds, with one exception: the link shall not concatenate an S100 packet after any packet of a higher speed.

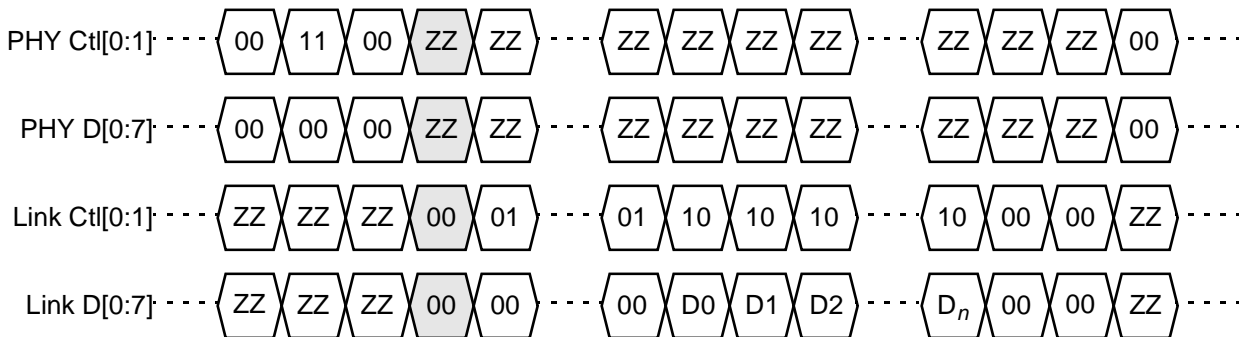
NOTE—If the multi-speed capabilities of the PHY have not been enabled (see clause 6.1), all concatenated packets shall be transmitted at the speed originally specified as part of the bus request. This requirement provides for backward compatibility when a PHY compliant with this specification is interfaced 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.

NOTE—Whenever the link and PHY exchange ownership of D[0:n] and Ctl[0:1], the entity relinquishing control shall refrain from sampling the interface for one cycle. This permits both link and PHY to act upon registered versions of the interface signals and also permits the new owner one cycle in which to sample and respond.

The timings for both a single and a concatenated packet transmit operation are illustrated in figure 5-6. In the diagram, D_0 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 5-16) and ZZ represents high impedance state. The link should assert the signals indicated by the shaded SC1k cycles (this may be necessary in the presence of an isolation barrier).

Single Packet



Concatenated Packet

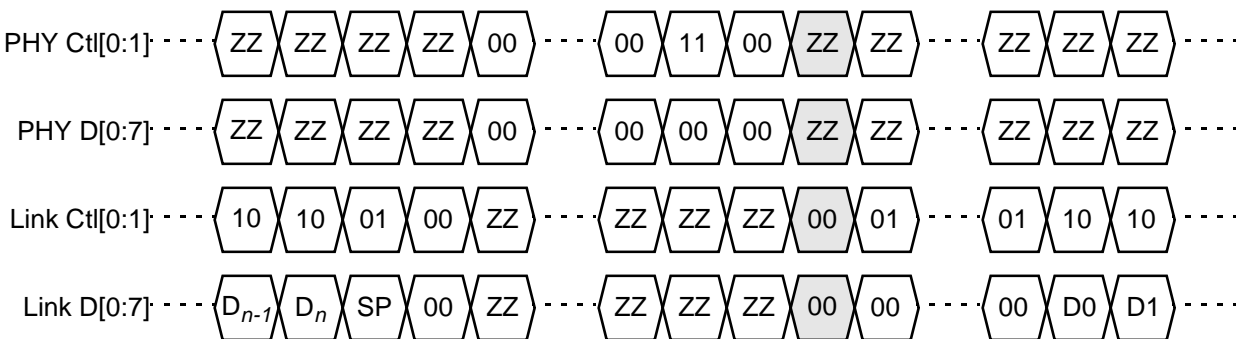


Figure 5-6 — Transmit timing

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.

5.5 Receive

Whenever the PHY sees the “data-on” state on Serial Bus, it initiates a receive operation by asserting *receive* on Ctl[0:1] and 1’s on $D[n]$. The PHY indicates the start of a packet by placing the speed code (encoding shown in table 5-16) on $D[n]$, 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 the speed code is a PHY/Link protocol and is not included in the calculation of the CRC or other data protection mechanisms.

It is possible that a PHY can see data-on 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 will end the packet by asserting *idle* when the data-on state 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 5-7. In the diagram, SP refers to the speed code and D0 through D_n are the data symbols of the packet.

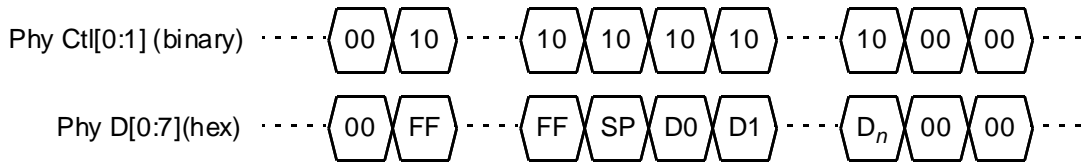


Figure 5-7 — Receive timing

The speed code for the receive operation is defined as shown in table 5-16. This is also the same speed encoding used by the link to signal speed to the PHY during concatenated packet transmission.

Table 5-16 — Speed code signaling

D[0:7]	Data rate
00xxxxx ₂ ^a	S100
0100xxx ₂	S200
0101000 ₂	S400
0101000 ₁ ₂	S800
0101001 ₀ ₂	S1600
0101001 ₁ ₂	S3200
1111111 ₂	“data-on” indication

^a. The “x” means transmitted as 0, ignored on receive.

NOTE—The speed code is only applicable for cable applications. For backplane applications, the speed code is set to 00xxxxx.

5.6 Electrical characteristics

This clause specifies the signal and timing characteristics of the interface between a discrete PHY and link.

5.6.1 Signal levels and waveforms

All AC measurements are made from the 1.4 V level of the clock to the valid input or output data levels as shown in figure 5-8.

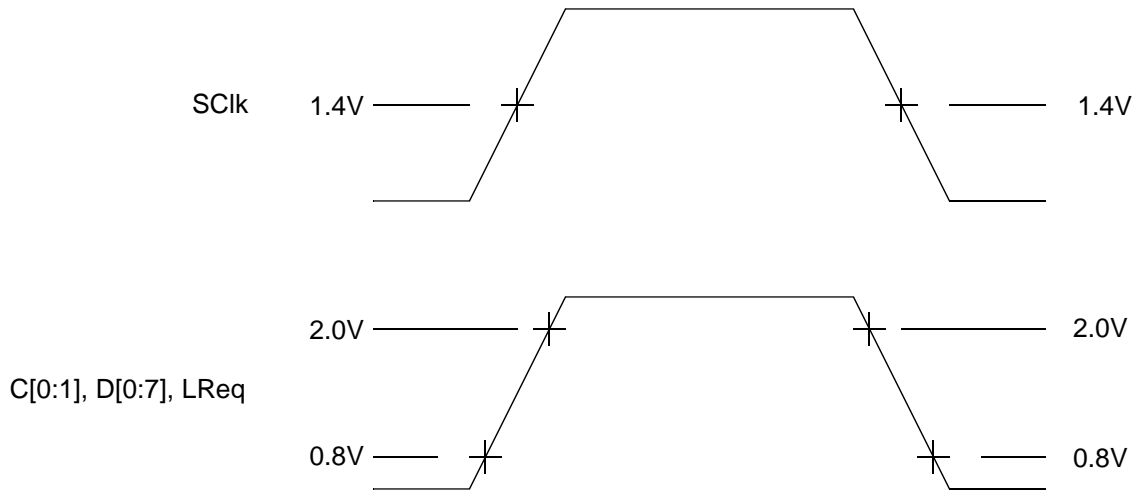


Figure 5-8 — Valid signal levels for AC measurements

DC parametric attributes of all TTL inputs on the PHY/Link interface are specified by table 5-17. Input levels may be greater than the power supply level (e.g., a 5 V output driving 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 below are compliant with this standard.

Table 5-17 — TTL DC specifications

Name	Description	Conditions		Unit	Minimum	Nominal	Maximum
V_{OH}	Output high voltage	$I_{OH} = -400 \mu A$	$V_{CC} = \text{Min}$	V	2.2	3.0	V_{CC}
V_{OL}	Output low voltage	$I_{OL} = 1 \text{ mA}$	$V_{CC} = \text{Min}$	V	GND	0.25	0.6
V_{IH}	Input high voltage			V	2.0		$V_{CC}^a + 10\%$
V_{IL}	Input low voltage			V	GND		0.8
I_{IH}	Input high current	$V_{CC} = \text{Max}$	$V_{IN} = 2.4 \text{ V}$	μA			40
I_{IL}	Input low current	$V_{CC} = \text{Max}$	$V_{IN} = 0.4 \text{ V}$	μA			600
C_{IN}	Input capacitance			pf			4.0
t_R	Clock rise time	0.8 V to 2.0 V		ns	0.7		2.4
t_F	Clock fall time	2.0 V to 0.8 V		ns	0.7		2.4
t_R	Data rise time	0.8 V to 2.0 V		ns	0.7		
t_F	Data fall time	2.0 V to 0.8 V		ns	0.7		

^a. Refers to driving device power supply

The rise and fall time measurement definitions, t_R and t_F , for SClk, C[0:1], D[0:7] and LReq are shown in figure 5-9.

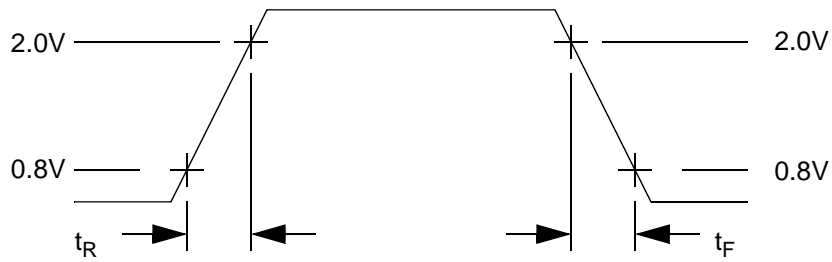


Figure 5-9 — Rise and fall time definition

All AC measurements assume an output load of 10 pF.

5.6.2 AC timing

Figures 5-10 through 5-12 below illustrate the transfer waveforms as observed at the link and the PHY. A PHY shall implement values for t_{lsu} and t_{lh} of at least the minimums specified in table 5-19 and shall not depend upon values for t_{psu} and t_{ph} less than the minimums specified; the measurements shall be taken at the PHY. A link shall implement values for t_{d1} , t_{d2} and t_{d3} within the limits specified in table 5-20 and shall not depend upon values for t_{lsu} and t_{lh} less than the minimums specified; the measurements shall be taken at the link. If an isolation barrier is implemented it shall not cause neither delay nor skew in excess of the values specified in table 5-18.

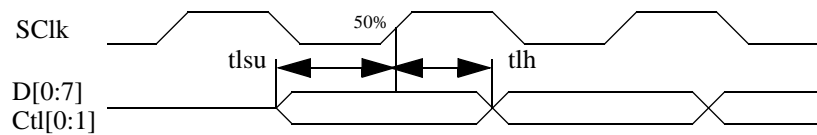


Figure 5-10 — PHY to link transfer waveform

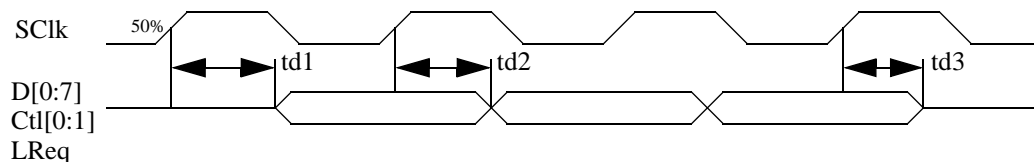


Figure 5-11 — Link to PHY transfer waveform at the link

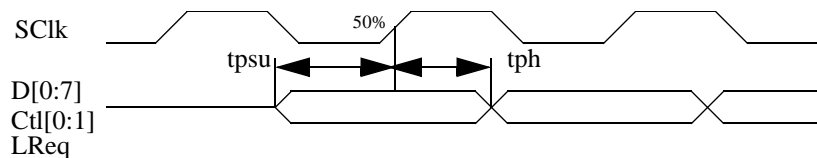


Figure 5-12 — Link to PHY transfer waveform at the PHY

The values for the timing parameters illustrated above are specified in the following tables.

Table 5-18 — AC timing parameters

Name	Description	Unit	Minimum	Nominal	Maximum
	SCLk frequency	MHz		49.152 ± 100 ppm	
	SCLk duty cycle	%	40		60
idel	Delay through isolation barrier	ns	0		2
	Skew through isolation barrier	ns	0		0.5
	Hysteresis input rising threshold	V	$V_{cc}/2 + 0.2$		$V_{cc}/2 + 1.3$
	Hysteresis input falling threshold	V	$V_{cc}/2 - 1.3$		$V_{cc}/2 - 0.2$

Table 5-19 — AC timing parameters at the PHY

Name	Description	Unit	Minimum	Nominal	Maximum
tlsu	Setup time, D[0:7] and Ctl[0:1] outputs before SCLk	ns	6.5		
tlh	Hold time D[0:7] and Ctl[0:1] outputs after SCLk	ns	0.5		
tpsu	Setup time D[0:7], Ctl[0:1] and LReq inputs before SCLk	ns	6		
tph	Hold time D[0:7], Ctl[0:1] and LReq inputs after SCLk	ns	0		

Table 5-20 — AC timing parameters at the link

Name	Description	Unit	Minimum	Nominal	Maximum
td1	Delay time, SCLk input high to initial instance of D[0:7], Ctl[0:1] and LReq outputs valid	ns	1		10
td2	Delay time, SCLk input high to subsequent instance(s) of D[0:7], Ctl[0:1] and LReq outputs valid	ns	1		10
td3	Delay time, SCLk input high to D[0:7], Ctl[0:1] and LReq invalid (tri-state)	ns	1		10
tlsu	Setup time, D[0:7] and Ctl[0:1] inputs before SCLk	ns	6		
tlh	Hold time, D[0:7] and Ctl[0:1] inputs after SCLk	ns	0		

5.6.3 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, exclusive 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 5-13 illustrates the relationship of these delays. Note that the maximum round-trip delay of 14 ns (calculated as $tdrt1_{max} = idel_{max} + td1_{max} + idel_{max}$) provides generous delays for both the link and the PHY. The link, after the receipt of SClk, has 10ns to assert valid data while at the PHY the minimum input setup for the subsequent SClk cycle is 6 ns (calculated as $tpsu_{min} = 20\text{ ns} - tdrt1_{max}$). Also note that the minimum round-trip delay until the next change in data of 21 ns (calculated as $tdrt2_{min} = 20\text{ ns} + idel_{min} + td2_{min} + idel_{min}$) limits the hold time at the PHY to 1 ns (calculated as $tlh_{min} = tdrt2_{min} - 20\text{ ns}$); the hold time is further reduced to zero to provide a guard band of 1 ns.

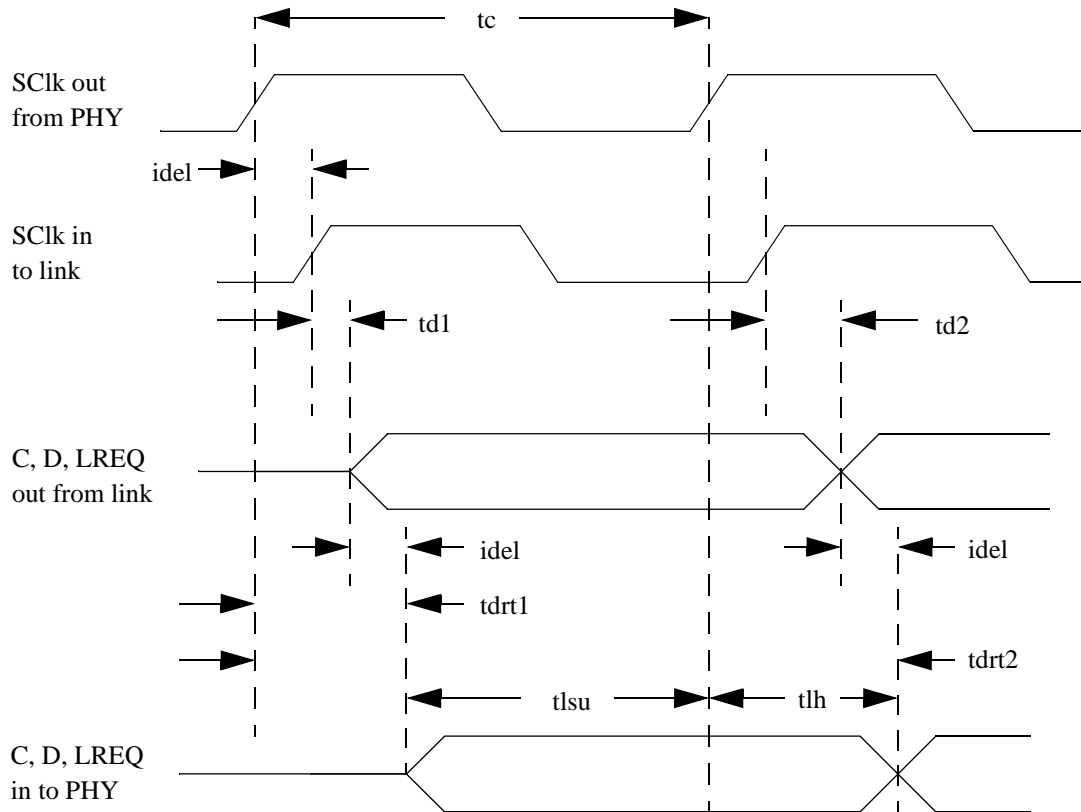


Figure 5-13 — Link to PHY delay timing

The values for the delays illustrated above are given by table 5-21 below.

Table 5-21 — Link to PHY delay timing parameters

Name	Description	Unit	Minimum	Nominal	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

6. PHY register map

Although Annex J of IEEE Std 1394-1995, from which this section is derived, originally described an interface to a discrete PHY, the material that follows 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 the clauses that follow.

6.1 PHY register map (cable environment)

In the cable environment, the extended PHY register map illustrated by figure 6-1 shall be implemented by all designs compliant with this supplement. Reserved fields are shown shaded in grey.

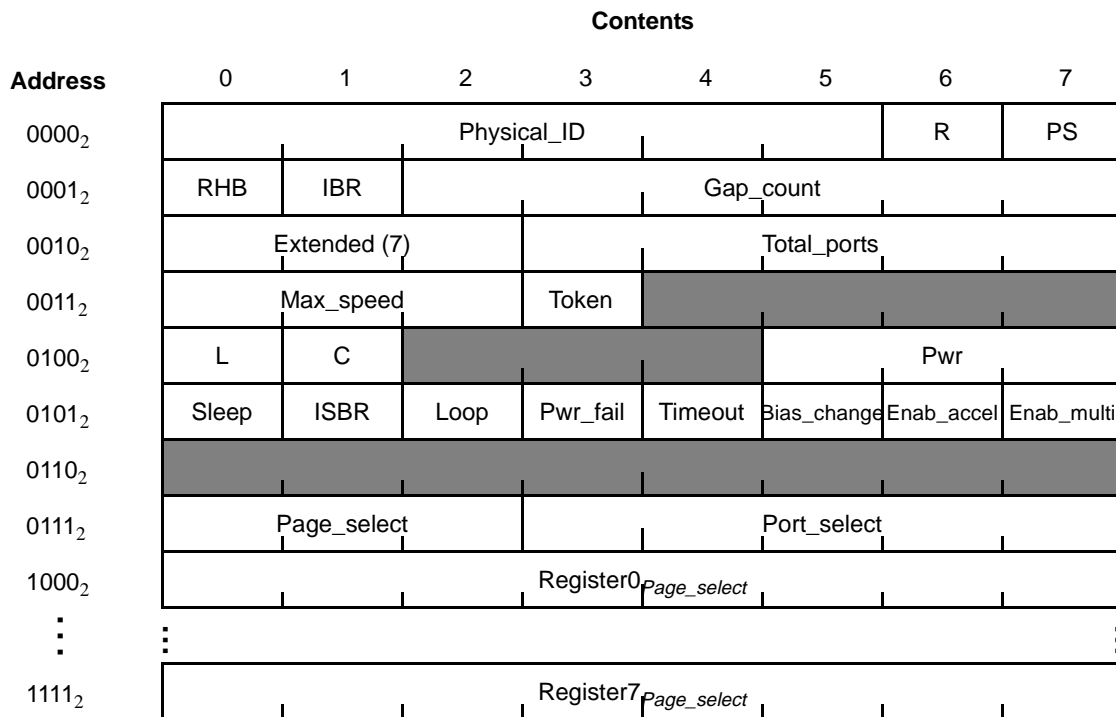


Figure 6-1 — Extended PHY register map for the cable environment

The meaning, encoding and usage of all the fields in the extended PHY register map are summarized by table 6-1. Power reset values not specified are resolved by the operation of the PHY state machines subsequent to a power reset.

Table 6-1 — PHY register fields for the cable environment

Field	Size	Type	Power reset value	Description
Physical_ID	6	r		The address of this node determined during self-identification.
R	1	r		When set to one, indicates that this node is the root.
PS	1	r		Cable power status (see clause 7.2).
RHB	1	rw	0	Root hold-off bit. When set to one, instructs the PHY to attempt to become the root during the next tree identify process.
IBR	1	rw	0	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 166 μs and is self-clearing.

Table 6-1 — PHY register fields for the cable environment (Continued)

Field	Size	Type	Power reset value	Description
Gap_count	6	rw	3F ₁₆	Used to configure the arbitration timer setting in order to optimize gap times according to the topology of the bus. See 4.3.6 of IEEE Std 1394-1995 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 maximum speed this PHY supports; the encoding is the same as for the <i>xspd</i> bit in self-ID packet 7 (see clause 7.3.1).
Token	1	r	vendor-dependent	When set to one, indicates that the PHY is capable of token-style arbitration (which shall be separately enabled for each port by the <i>enab_token</i> bit).
L	1	rw	1	Link enabled. Cleared or set by software to control the value of the L bit transmitted in the node's self-ID packet 0. The transmitted L bit shall be the logical AND of this bit and the LPS signal.
C	1	rw	See note	Contender. Cleared or set by software to control the value of the C bit transmitted in the 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.
Pwr	3	rw	vendor-dependent	Power class. Controls the value of the pwr field transmitted in the self-ID packet. See 4.3.4.1 of IEEE Std 1394-1995 for the encoding of this field.
Sleep	1	rw	?	Sleep mode. When set to one, places the PHY in the as-yet-to-be-defined sleep mode. If sleep mode is not supported a write to this bit has no effect.
ISBR	1	rw	0	Initiate short (arbitrated) bus reset. A write of one to this bit requests 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	0	Cable power failure detect. Set to one when the PS bit changes from one to zero. A write of one to this bit clears it to zero.
Timeout	1	rw	0	Arbitration state machine timeout. A write of one to this bit clears it to zero.
Bias_change	1	rw	0	Bias change detect. Set to one when TP bias changes on any disabled port. The state of TP bias for enabled ports does not affect this bit. A write of one to this bit clears it to zero.
Enab_accel	1	rw	0	Enable arbitration acceleration. When set to one, the PHY shall use the enhancements specified in clause 7.8.
Enab_multi	1	rw	0	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	5	rw	vendor-dependent	If the page selected by <i>Page_select</i> presents <i>per</i> 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.

The *RHB* bit should be zero unless it is necessary to establish a particular node as the cycle master. In particular, bus manager- and isochronous resource manager-capable nodes should not set their *RHB* bit(s) to one and should not attempt to become the root unless there is no cycle master. This recommendation is made in anticipation of a requirement for Serial Bus to Serial Bus bridges to become root to distribute the cycle clock.

When any one of the *Loop*, *Pwr_fail*, *Timeout* or *Bias_change* bits transitions from zero to one, *PHY_interrupt* shall be set to one. *PHY_interrupt* is reported as S[3] in a PHY status transfer, as specified by clause 5.3. These bits in PHY register five are unaffected by writes to the register if the corresponding bit position is zero. When the bit written to the PHY register is one, the corresponding bit is zeroed.

The upper half of the PHY register space, addresses 1000₂ through 1111₂, inclusive, provides a windows through which additional pages of PHY registers may be accessed. This supplement defines pages zero, one and seven, the Port Status page, the Vendor Identification page and a vendor-dependent page.

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 0111₂. The format of the Port Status page is illustrated by figure 6-2 below.

Address	Contents							
	0	1	2	3	4	5	6	7
1000 ₂	AStat		BStat		Ch	Con	Bias	Dis
1001 ₂	Negotiated_speed							Enab_token
1010 ₂								
1011 ₂								
1100 ₂								
1101 ₂								
1110 ₂								
1111 ₂								

Figure 6-2 — PHY register page 0: Port Status page

The meanings of the register fields with the Port Status page are defined by the table below.

Table 6-2 — PHY register Port Status page fields

Field	Size	Type	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)
Ch	1	r	If equal to one, the port is a child, else 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 in IEEE Std 1394-1995).
Con	1	r	If equal to one, the port is connected, else disconnected. The power reset value is zero. This bit reports the value of the connected variable for the port (see the connection_status() function in table 7-18). The value reported by this bit is filtered by hysteresis logic, as specified by figure 7-10, to reduce multiple status changes caused by contact scrape when a connector is inserted or removed.
Bias	1	r	If equal to one, bias voltage is detected (possible connection). The value reported by this bit is filtered by hysteresis logic, with a time of CONNECT_TIMEOUT, to reduce multiple status changes caused by contact scrape when a connector is inserted or removed.

Table 6-2 — PHY register Port Status page fields (Continued)

Field	Size	Type	Description
Dis	1	rw	When set to one, the port shall be disabled. The value of this bit subsequent to a power reset is implementation-dependent, but should be a hardware configurable option.
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 <i>xspd</i> bit in self-ID packet 7 (see clause 7.3.1).
Enab_token	1	rw	Enable token-style arbitration. When set to one, the enhancements specified in clause 7.8 shall be enabled for this port. The power reset value is zero.

The *AStat*, *BStat*, *Ch* and *Con* fields are present in both the legacy and extended PHY registers and have identical meanings, defined by table 6-2 above, in both cases.

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 0111₂. The format of the Vendor Identification page is illustrated by figure 6-3 below.

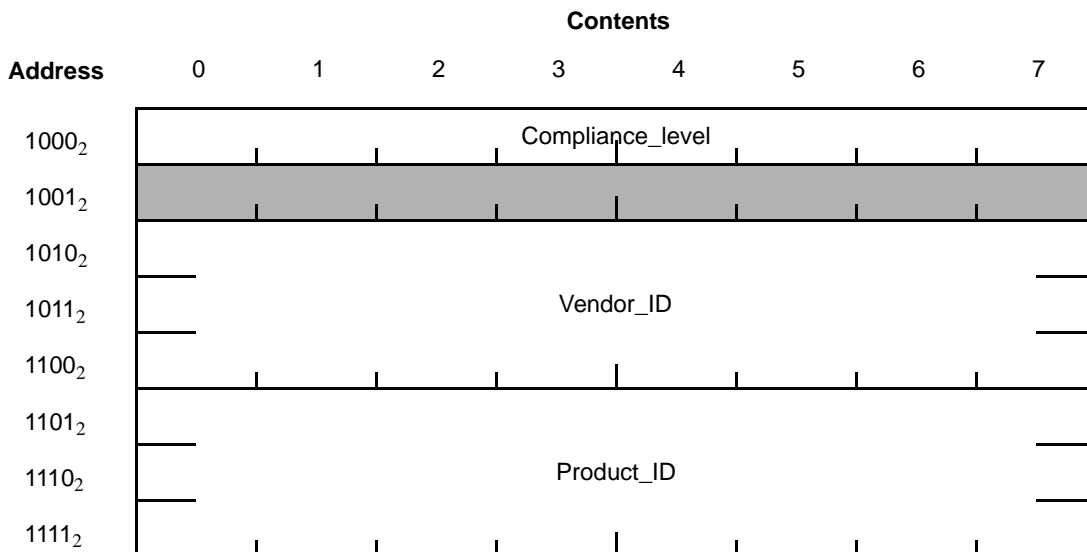


Figure 6-3 — PHY register page 1: Vendor Identification page

The meanings of the register fields within the Vendor Identification page are defined by the table below.

Table 6-3 — 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

Table 6-3 — PHY register Vendor Identification page fields (Continued)

Field	Size	Type	Description
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 Committee (RAC). 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 0111₂. 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 1000₂ through 1111₂, inclusive.

6.2 PHY register map (backplane environment)

The backplane environment has a PHY register map similar to that of the cable environment, except that certain fields are not used and that other fields shall always be set to a particular value. In addition, the backplane environment may make use of the enhanced register map to indicate an enhanced register that contains the transceiver disable (TD) and Priority fields.

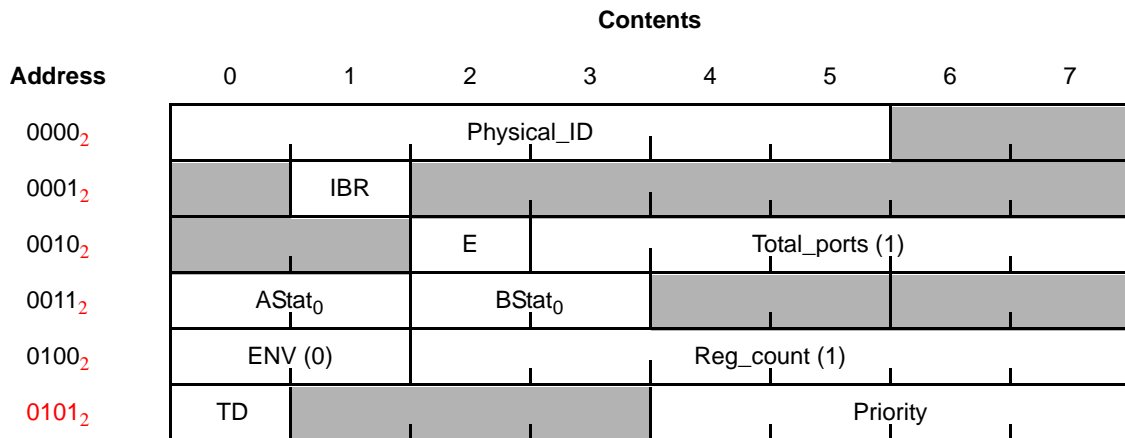


Figure 6-4 — PHY register map for the backplane environment

With the exception of the fields specified by the table below, the meaning of the backplane PHY register fields is the same as for the cable environment (see table 6-1).

Table 6-4 — 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.
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.
E	1	r	If equal to zero, no enhanced registers are used. If equal to one, enhanced registers at address 0100 ₂ and 0101 ₂ are present.

Table 6-4 — PHY register fields for the backplane environment

Field	Size	Type	Description
Total_ports	5	r	The number of ports on this PHY. In the backplane environment there is one port per PHY.
AStat ₀	2	r	Data line state (uses the same encoding as for cable).
BStat ₀	2	r	Strobe line state (uses the same encoding as for cable).
ENV	2	r	Present if the E bit is one. ENV shall be equal to zero in the backplane environment; other values are reserved.
Reg_count	6	r	Present if the E bit is one, in which case it shall be greater than or equal to one. When Reg_count is greater than one, the format of additional enhanced registers at addresses 0110 ₂ and above are vendor-dependent.
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.
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.

6.3 Integrated link and PHY

The register map described in the preceding clauses 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 clauses. The status that may be read from the registers and the behavior caused by a write to the registers shall be identical with that of a discrete link and PHY combination.

7. Cable physical layer performance enhancement specifications

This section of the supplement specifies 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. This supplement 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 this supplement correct the defect and promote interoperability between existing PHYs and those that conform to this supplement.
- 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 a primary packet is observed and then propagated toward the root: fly-by arbitration. The other occurs subsequent to the observation of an acknowledge packet: ack-accelerated arbitration.
- Token-style arbitration. A group of cooperating nodes may take advantage of inherent Serial Bus characteristics by coordinating bus arbitration requests such that the node closest to the root arbitrates before the others within the group but then yields the arbitration grant to the node furthest from the root. After this furthest node transmits one or more packets, the nodes successively closer to the root employ ack-accelerated or fly-by arbitration to concatenate their transmissions without the necessity for subaction gaps.
- Transmission delay calculation (PHY ping). The ability for a PHY to transmit a “ping” packet to another PHY and time its return permits the inclusion of cables longer than 4.5 meters or PHYs with delays longer than 144 ns into Serial Bus topologies.
- Extended speed encoding. Although speeds in excess of S400 are not specified by this supplement, the coding infrastructure in the self-ID packets is established for future use.

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; the clauses that follow replace existing clauses of IEEE Std 1394-1995 in their entirety and are so identified.

7.1 Cable topology

Informative sections of IEEE Std 1394-1995 assume that Serial Bus cable topologies are limited by individual cable lengths less than or equal to 4.5 meters 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 this supplement, 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 to time the worst-case Serial Bus round trip delay, cable lengths less than or equal to 4.5 meters and a maximum hop count of 16 are still recommended.

7.2 Cable power and ground

This clause replaces 4.2.2.7 of IEEE Std 1394-1995, “Power and ground,” in its entirety.

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 clause 7.3.1). There may be other facilities, for example in a node’s configuration ROM, that identify the power characteristics of a node in more detail that 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 power sources be manufactured such that current from a node providing higher voltage does not flow into sources of lower output voltage. Power sources that identify themselves with POWER_CLASS of one, two or three in their self-ID packet(s) shall implement, for each of their ports, the diode and current limiting scheme illustrated by figure 7-1.

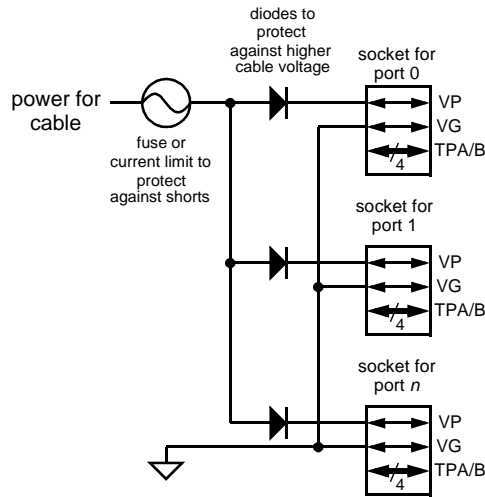


Figure 7-1 — Node power interface for POWER_CLASS one, two or three

Cable power sources that do not identify themselves as such in their self-ID packets shall not permit the inflow of power from a higher voltage power source, but the implementation details may differ from figure 7-1. All cable power sources shall meet the following requirements:

Table 7-1 — Cable power source requirements

Condition	Limit	Units
Maximum output current per port	1.5	Amp
Minimum output voltage (POWER_CLASS one, two or three)	20	Vdc
Minimum output voltage (all other POWER_CLASS values)	8	Vdc
Maximum output voltage	33	Vdc
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.

When cable power is available, it is in the nominal range 7.5 V to 33 V. A PHY that detects cable power of at least 7.5 V shall set the *PS* bit (cable power status) 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.

NOTE—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 above.

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

- 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 shall enable the node to consume additional power up to the limit specified by the node's self-ID packet(s);
- Inrush energy shall not exceed 18 mJ in 3 ms; and
- 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 $(I_{load} / 1.5 \text{ A}) * 100 \text{ mA}$; and
 - 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, shall be less than 50 μA .

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.

7.3 Cable PHY packets

This clause extends the definition of PHY packets and completely replaces IEEE Std 1394-1995 clause 4.3.4, “Cable 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. With the exception of the expansion of the speed code to four bits in the self-ID packet, there are no substantive changes to the previously defined packet formats.

The cable physical layer sends and receives 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.

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 configuration packets (which family includes, at present, only the ping packet)

NOTE—The PHY packets can be distinguished from the null-data isochronous packet (the only link 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.

7.3.1 Self-ID packet

The cable PHY sends one to four self-ID packets at the base rate during the self-ID phase of arbitration. The number of self-ID packets sent depends on the maximum number of ports it has. The cable PHY self-ID packets have the following format:

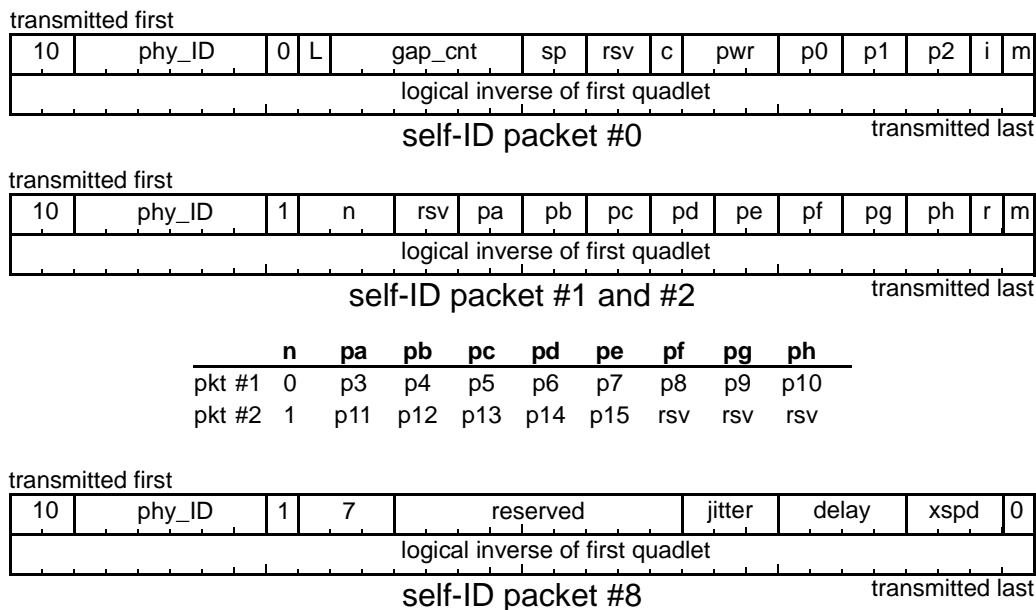


Figure 7-2 — Self-ID packet format

Self-ID packets with sequence numbers, *n*, between 2 and 6, inclusive, are reserved for future standardization.

Table 7-2 — Self-ID packet fields

Field	Derived from	Comment
phy_ID	physical_ID	Physical node identifier of the sender of this packet
L	LPS PHY register L bit	If set, this node has active link and transaction layers
gap_cnt	gap_count	Current value for this nodes' PHY_CONFIGURATION.gap_count field.
sp	PHY_SPEED	Speed capabilities: 00 ₂ 98.304 Mbit/s 01 ₂ 98.304 and 196.608 Mbit/s 10 ₂ 98.304, 196.608 and 393.216 Mbit/s 11 ₂ Extended speed capabilities reported in self-ID packet 8
c	CONTENDER	If set and the link_active flag is set, this node is a contender for the bus or isochronous resource manager as described in clause 8.4.2 of IEEE Std 1394-1995.

Table 7-2 — Self-ID packet fields (Continued)

Field	Derived from	Comment
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. 101 ₂ Node is powered from the bus and is using up to 3 W. No additional power is needed to enable the link^a. 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 and higher layers^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 and higher layers^a.
p0 ... p15	NPORT, child[n], connected[n]	Port connection status: 11 ₂ Connected to child node 10 ₂ Connected to parent node 01 ₂ Not connected to any other PHY 00 ₂ Not present on this PHY
i	initiated_reset	If set, this node initiated the current bus reset (<i>i.e.</i> , it started sending a bus_reset signal before it received one) ^b (Optional. If not implemented, this bit shall be returned as a zero)
m	more_packets	If set, another self-ID packet for this node will immediately follow (<i>i.e.</i> , 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
jitter	PHY_DELAY_JITTER	The difference between the fastest and slowest repeater data delay, expressed as (jitter + 1) * 20 ns.
delay	PHY_DELAY	Worst-case repeater delay, expressed as 144 + (delay * 20) ns.
xspd	PHY_SPEED	Extended speed capabilities: 000 ₂ 98.304 Mbit/s 001 ₂ 98.304 and 196.608 Mbit/s 010 ₂ 98.304, 196.608 and 393.216 Mbit/s 011 ₂ 98.304, 196.608, 393.216 and 786.43 Mbit/s 100 ₂ 98.304, 196.608, 393.216, 786.432 and 1,572.864 Mbit/s 101 ₂ 98.304, 196.608, 393.216, 786.432, 1,572.864 and 3,145.728 Mbit/s All other values are reserved for future definition
r, rsv		reserved for future definition, set to zeros

^a. The link **and higher layers** is enabled by the link-on PHY packet described in clause 7.3.2; **this packet may also enable application layers.**

^b. There is no guarantee that exactly one node will have this bit set. More than one node may request a bus reset at the same time.

7.3.2 Link-on packet

For those nodes that do not automatically power-on their link layer circuitry, reception of the following cable PHY packet shall cause a PH_EVENT.indication of LINK_ON. (See clause 8.4.4 of IEEE Std 1394-1995, "Power management (cable environment)," for further information.)

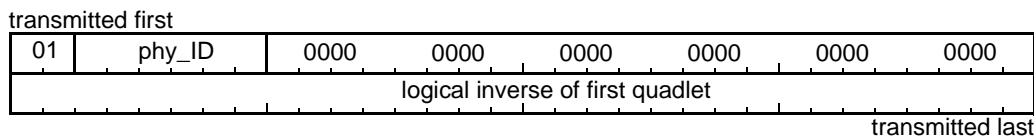


Figure 7-3 — Link-on packet format

Table 7-3 — Link-on packet fields

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

The PH_EVENT.indication of LINK_ON shall persist so long as the logical and of the LPS signal and the PHY register L bit is zero. In the case of a discrete PHY implementation, an externally accessible signal shall be asserted as long as this indication persists. A typical method for software to respond to (and clear) the indication is to write a value of one to the L bit.

7.3.3 PHY configuration packet

It is possible to optimize Serial Bus performance for particular configurations the following ways:

- a) setting the gap_count used by all nodes to a smaller value (appropriate to the actual worst-case number of hops between any two nodes)
- b) forcing a particular node to be the root after the next bus initialization (for example, in isochronous systems, the root shall be cycle-master capable)

Both of these actions shall be effected for all nodes by means of the PHY configuration packet shown below. (The PH_CONTROL.request service affects only the local node and is not recommended for changes to gap_count) The procedures for using this PHY packet are described in section 8 of IEEE Std 1394-1995 and in clause clause 9.18 of this supplement.

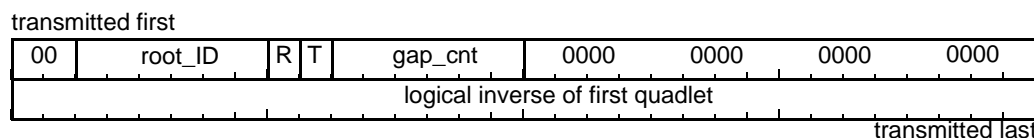


Figure 7-4 — PHY configuration packet format

Table 7-4 — 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.

Table 7-4 — PHY configuration packet fields (Continued)

Field	Affects	Comment
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 variable. This value goes into effect immediately on receipt and stays valid after the next bus reset (gap_count is set to 63 after a second bus reset without an intervening PHY configuration packet as described in reset_start_actions() in clause 7.8.2.1.2)

A PHY that transmits a configuration packet as the result of a Link request shall set its force_root bit and gap_count variable as if the configuration packet had been received from the bus.

7.3.4 Extended PHY configuration packets

A PHY configuration packet with R=0 and T=0 is utilized to define extended PHY configuration packets according to the value in the gap_cnt field (this is renamed the type field in the figures that follow). The extended PHY configuration packets have no effect upon either the force_root bit or gap_count field of any node.

7.3.5 Ping packet

The reception of the cable PHY packet shown in figure 7-5 shall cause the node identified by phy_ID to transmit a self-ID packet that reflects 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 transmitted may differ from that of the most recent self-identify process.

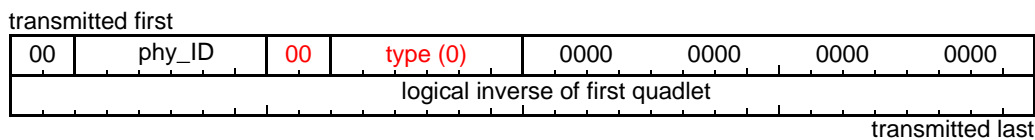


Figure 7-5 — Ping packet format

Table 7-5 — Ping packet fields

Field	Derived from	Comment
phy_ID	physical_ID	Physical node identifier of the destination of this packet
type		Extended PHY configuration packet type (zero indicates ping packet)

7.4 Cable PHY line states

This clause defines a new rule by which a PHY decodes the interpreted arbitration signals (Arb_A and Arb_B) into a line state; it is in addition to IEEE Std 1394-1995 clause 4.3.3, "Cable PHY line states."

Table 7-6 — Cable PHY received arbitration line states

Interpreted arbitration signals		Line state name	Comment
Arb_A	Arb_B		
0	Z	RX_TOKEN_GRANT	The parent PHY is granting the bus (although no TX_REQUEST was sent by the child)

The RX_TOKEN_GRANT line state is recognized when received by a parent port during the normal arbitration phase.

7.5 Cable PHY timing constants

This clause 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; it is in addition to IEEE Std 1394-1995 clause 4.3.5, “Cable PHY timing constants.”

Table 7-7 — Cable PHY timing constants

Timing constant	Minimum	Maximum	Comment
ACK_RESPONSE_TIME	0.05 μ s	0.17 μ s	Measured at the cable connector from the last bit of DATA_END that follows the primary packet to the first bit of DATA_PREFIX that precedes the acknowledge packet.
ARB_RESPONSE_DELAY	33.3 ns	PHY_DELAY	Delay between an RX_REQUEST signal arriving at the receiving port and the TX_REQUEST signal being sent at the transmit port(s).
BUS_TO_LINK_DELAY	PHY_DELAY		Data propagation time measured from the Serial Bus connector to the PHY/link interface.
CONNECT_TIMEOUT	336.0 ms	341.3 ms	Connection debounce time
LINK_TO_BUS_DELAY	40 ns	100 ns	Data propagation time measured from the PHY/link interface to the Serial Bus connector.
MAX_ARB_STATE_TIME	200 μ s	400 μ s	Maximum time in any state (before generating a bus reset) except the idle state or a state that exits after an explicit time-out.
MAX_BUS_HOLD		1.63 μ s	Maximum time a node may transmit a TX_DATA_PREFIX signal between the request acknowledge and data packet of concatenated asynchronous subactions or between data packets of concatenated isochronous subactions. 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_TIME		84.31 μ s	The maximum time that clocked data may be transmitted continuously.
PHY_DELAY	80 ns	See self-ID packet 8	Best-case repeater data delay has a fixed minimum.
PING_RESPONSE_TIME	50 ns	170 ns	Time permitted a PHY to respond to a ping packet (see clause 7.3.5), measured at the connector from the last bit of DATA_END to the first bit of DATA_PREFIX for the self-ID packet(s).
RESET_DETECT	80.0 ms	85.3 ms	Time for a connected node to confirm a reset signal
ROOT_CONTEND_FAST	0.76 μ s	0.80 μ s	Time to wait in state T3: Root Contention if the random bit is zero, as described in clause 4.4.2.2 of IEEE Std 1394-1995. ($\sim 80 / \text{BASE_RATE}$)
ROOT_CONTEND_SLOW	1.60 μ s	1.64 μ s	Time to wait in state T3: Root Contention if the random bit is one, as described in clause 4.4.2.2 of IEEE Std 1394-1995. ($\sim 160 / \text{BASE_RATE}$)
SHORT_RESET_TIME	1.30 μ s	1.40 μ s	Short reset hold time. ($\sim 128 / \text{BASE_RATE}$)
RESET_WAIT	0.16 μ s	0.16 μ s	Reset wait delta time. ($\sim 16 / \text{BASE_RATE}$)

Note that the constant RESET_WAIT is redefined by this supplement 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 now expressed as RESET_TIME + RESET_WAIT; the corresponding arbitrated (short) reset wait time is SHORT_RESET_TIME + RESET_WAIT.

7.6 Node variables

Each node's PHY has a set of variables that are referenced in the C code and state machines in clause 7.8. 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. The definitions in table 7-8 entirely replace clause 4.3.8 of IEEE Std 1394-1995, "Node variables."

Table 7-8 — Node variables

Variable name	Power reset value	Comment
accelerating	TRUE	Set TRUE or FALSE by accelerate or decelerate requests issued by the link <i>via</i> LReq (see clause 5.2) and used by the arbitration state machines. See also enab_accel below.
arb_enable	—	TRUE when the PHY may arbitrate as soon as the next subaction gap is observed.
cable_power_active	—	TRUE is cable power is within normal operating range (see clause 7.2).
enab_accel	FALSE	Globally enables or disables all PHY accelerations specified by clause 7.8. 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 clause 4.4.2.2 of IEEE Std 1394-1995). If only one node on a bus has force_root set TRUE, that node is guaranteed to become the root.
gap_count	63	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.
initiated_reset	TRUE	TRUE if this node initiated the bus reset in progress. Cleared to FALSE upon completion of the self-identify process.
link_active	TRUE	TRUE if the node's link is present and enabled.
more_packets	—	Flag which 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.

7.7 Port variables

In addition to the variables described in the preceding clause, each node's PHY has a set of variables replicated for each port. The definitions in table 7-9 entirely replace clause 4.3.9 of IEEE Std 1394-1995, "Port variables."

Table 7-9 — Port variables

Variable name	Power reset value	Comment
child	—	TRUE if this port is connected to a child node.
connected	FALSE	TRUE if there is a peer PHY connected to this port.
child_ID_complete	—	TRUE when the child node connected to this port has finished its self-ID.
max_peer_speed	—	Maximum speed capability of the peer PHY connected to this port.
port_status	—	TRUE if TP bias is present. This is not filtered by any hysteresis circuitry.
speed_OK	—	The connected port can accept a packet at the requested speed.

7.8 Cable physical layer operation

With the exception noted below, this clause replaces 4.4 of IEEE Std 1394-1995, "Cable PHY operation," in its entirety. The subclause for which no change has been made from the existing standard is as follows:

- 4.4.2.2, "Tree identify"

The changes specified in this supplement affect the bus reset, self-identify and normal arbitration phases of cable PHY operation and incorporate the following enhancements:

- The duration of the bus reset signal may be reduced to approximately 1.3 μs if the port that initiates the reset first arbitrates for the bus; and
- The probability of a multitude of bus resets during and subsequent to a physical cable insertion or removal is reduced.

The operation of the cable physical layer can best be understood with reference to the architectural diagram shown in figure 7-6:

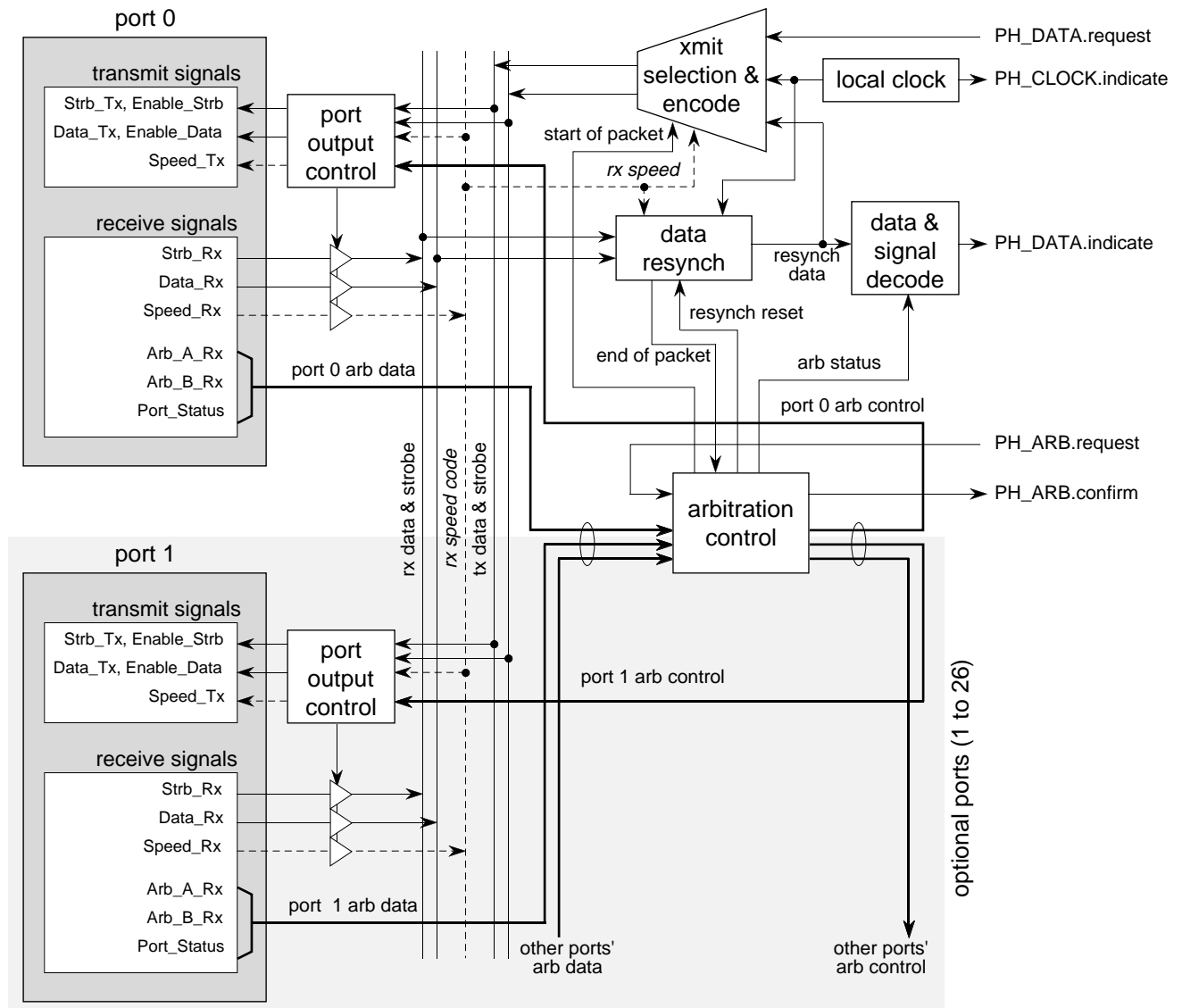


Figure 7-6 — Cable physical layer architecture

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 clause 7.8.2

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 98.304 Mbit/s). The operation of this block is described in clause 4.4.1.2 of IEEE Std 1394-1995.

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 clause 4.4.1.1 of IEEE Std 1394-1995.

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 clause use the syntax specified in clause 1.6.7 and the definitions in tables 7-8, 7-9, 7-10 and 7-11.

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 used to describe PHY behavior does not contain provisions to enforce the LReq rules specified by clause 5.2.1; if the link issues bus requests that do not follow the rules the PHY behavior is unspecified.

Table 7-10 — Cable PHY code definitions

```

const int FIFO_DEPTH = 8;           // IMPLEMENTATION-DEPENDENT!
enum PHY_state {R0, R1,           // Tracks the PHY state (names per state diagrams)
                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 arb_enable;                // Set if a node may arbitrate upon detection of a subaction gap
timer arb_timer();                 // Timer for arbitration state machines
boolean bus_initialize_active;     // Set while the PHY is initializing the bus
int child_count;                   // Number of child ports
int contend_time;                  // Amount of time to wait during root contention
boolean DS_clock;                  // FALSE unless encoded DS clock available on the receive port
// (data or strobe transition observed within the last 20 ns)

boolean end_of_reception;          // Set when reception of packet is complete
boolean force_root;                // Set to delay start of tree-ID process for this node
dataBit fifo[FIFO_DEPTH];         // Data resynch buffer
unsigned fifo_rd_ptr, fifo_wr_ptr; // Data resynch buffer pointers
boolean gap_count_reset_disable;   // If set, a bus reset will not force the gap_count to the maximum
boolean ibr;                       // Set when a long bus reset is needed
boolean isbr;                      // Set when an arbitrated (short) bus reset should be attempted
boolean isolated_node;             // Set if no ports connected
boolean own_request;               // Latch the value of arb_OK() at the time it is evaluated
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(int port_number); // Return current speed from 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 reset_time;                    // Duration to assert bus reset signal
boolean root_test;                 // Flag that is randomly set during root contention
int rx_dribble_bits;               // Keep track of dribble bits in FIFO
speedCode rx_speed, tx_speed;      // Current packet speeds
boolean waiting_for_data_start;    // First data bit not yet received

```

Table 7-11 — Cable PHY packet definitions

```

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 del:2; // Worst case PHY delay
                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;
                quadlet pc:2;
                quadlet pd:2;
                quadlet pe:2;
                quadlet pf:2;
                quadlet pg:2;
                quadlet ph:2;
                quadlet :2;
            };
            struct { // "Caboose" packet
                quadlet :21;
                quadlet jitter:3; // PHY_DELAY jitter
                quadlet delay:4; // Worst-case repeater delay
                quadlet xspd:3; // Extended speed code
                quadlet :1; // Constant zero
            };
            struct { // PHY configuration packet (includes ping 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;
            };
        };
    };
    union {
        quadlet checkQuadlet;
        dataBit checkBits[32];
    };
};
} PHY_PKT;
    
```

7.8.1 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.

7.8.1.1 Cable environment data transmission

Data transmission 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 no data is sent, which leaves the drivers in the "01" data prefix condition.

Table 7-12 — Data transmit actions

```
static dataBit tx_data, tx_strobe; // Memory of last signal sent

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

    wait_event(PHY_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 (i != receivePort)
            if (speed_OK[i]) {
                portData pd = {phyData(tx_strobe), phyData(tx_data)};
                portT(i, pd);
            } else
                portT(i, TX_DATA_PREFIX);
}
```

The edge rates and jitter specifications for the transmitted signal are given in clause 4.2.3 of IEEE Std 1394-1995.

Starting data transmission requires sending a special data prefix signal and a speed code. The `speed_OK[i]` flag for each port is `TRUE` if the **connected PHY** has the capabilities to receive the data:

Table 7-13 — Start data transmit actions

```
void start_tx_packet(speed) // Send data prefix and speed code
    int i;

    for (i = 0; i < NPORT; i++) {
        portT(i, TX_DATA_PREFIX); // Send data prefix
        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++)
        portTspeed(i, S100); // Go back to normal signal levels
    wait_time(DATA_PREFIX_TIME); // Finish data prefix
}
```

Ending a data transmission requires sending extra bits (known as “dribble bits”) which 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(DATA_PREFIX) or not (PH_DATA.request(DATA_END))):

Table 7-14 — Stop data transmit actions

```
void stop_tx_packet (phyData ending_status, speedCode tx_speed) {
    switch (tx_speed) {
        case S400:          // Pad with six dribble bits
            tx_bit(1);
            tx_bit(1);
            tx_bit(1);
            tx_bit(1);
        case S200:          // Pad with two dribble bits
            tx_bit(1);
            tx_bit(1);
        default:
            break;
    }
    tx_bit((ending_status == DATA_PREFIX) ? 1 : 0); // Penultimate bit...
    wait_event(PH_CLOCK.indication());              // Wait for clock
    if (ending_status == DATA_PREFIX) {
        for (i = 0; i < NPORT; i++)
            if (i != receive_port)
                portT(i, TX_DATA_PREFIX);           // ...and the last dribble bit
        wait_time(CONCATENATION_PREFIX_TIME);       // Speed signal after this time (200 ns)
    } else if (ending_status == DATA_END) {
        for (i = 0; i < NPORT; i++)
            if (i != receive_port)
                portT(i, TX_DATA_END);
        wait_time(DATA_END_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) and the current port state is 01, this algorithm sets port state to 11 for one bit time, then back to 01. If the desired ending state is 10 (TX_DATA_END) and the current port state is 01, the port state sequence is 00 followed by 10. The process is similar if the current port state is 10.

7.8.1.2 Cable environment 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. This process can be described as two routines communicating *via* a small FIFO:

Table 7-15 — Data reception and repeat actions (Sheet 1 of 2)

```
static tpSig old_data, old_strobe;                // Memory of last signal sent

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

void decode_bit (void) {
    repeat {
        if (portRspeed(receive_port) > S100) {
            rx_speed = portRspeed(receive_port);
            signal(SPEED_SIGNAL_RECEIVED);         // Notify start_rx_packet
        }
        new_signal = tpSignals();                 // Get signal
        if (new_signal == IDLE)
```

Table 7-15 — Data reception and repeat actions (Sheet 2 of 2)

```

        signal(IDLE_DETECTED);
    else {
        new_data = new_signal.TPA;           // Received data is on TPA
        new_strobe = new_signal.TPB;        // Received strobe is on TPB
        if ((new_signal.TPA != 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) {
    int i;

    wait_event(PHY_CLOCK_indication);      // Wait for clock
    if ((fifo_rd_ptr - fifo_wr_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 doing the speed signaling with the sender of the data. At the same time, the node must start up 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:

Table 7-16 — Start data reception and repeat actions (Sheet 1 of 2)

```

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

    fifo_rd_ptr = fifo_wr_ptr = 0;         // Reset data resynch buffer
    waiting_for_data_start = TRUE;         // First data bit not yet received
    portT(receive_port, IDLE);             // Turn off grant, get ready to receive
    wait_event(SPEED_SIGNAL_RECEIVED | DATA_STARTED | IDLE_DETECTED);
    if (rx_speed == S100)
        rx_dribble_bits = 2;
    else {
        // If speed signalling, then must repeat it
        rx_dribble_bits = (rx_speed == S200) ? 4 : 8; // Need for FIFO empty test
        tx_speed = rx_speed;               // Get speed of packet to repeat
        for (i = 0; i < NPORT; i++)
            if (i != receive_port) {
                portT(i, TX_DATA_PREFIX);   // Send data prefix out repeat ports
                speed_OK[i] = (tx_speed <= max_peer_speed[i]);
                if (speed_OK[i])
                    portTspeed(i, tx_speed); // Receiver can accept, send speed intentions
            }
    }
}

```

Table 7-16 — Start data reception and repeat actions (Sheet 2 of 2)

```

wait_time(SPEED_SIGNAL_LENGTH);
for (i = 0; i < NPORT; i++)
    if (i != receive_port)
        portTspeed(i, S100);          // Go back to normal signal levels
        wait_time(DATA_PREFIX_TIME);  // Finish data prefix
        wait_event(DATA_STARTED | IDLE_DETECTED); // Wait for decoder to start
}
for (i = 0; i < FIFO_DEPTH/2 - 1; i++)
    wait_event(PHY_CLOCK_indication); // Make sure FIFO is centered
}
    
```

Ending data reception is basically the same as ending transmission with the primary difference being that the ending state for transmitting ports is determined by the ending state of the receiving port:

Table 7-17 — Stop data reception and repeat actions.

```

void stop_rx_packet_() { // Receiving done, clean up ending
    int i;
    .
    portData_ending_data;
    ending_data = portR(receive_port); // Get ending state on receive port
    stop_tx_packet(ending_data, rx_speed); // Finish repeat
}
    
```

7.8.2 Cable environment 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 interval; 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 supplement. 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 phases¹ 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.

¹ Clause 4.4.2.2 of IEEE Std 1394-1995, which describes the tree identify process, is unchanged and is not reproduced in this supplement.

7.8.2.1 Bus reset

The bus reset process starts when a bus reset signal is recognized on a connected port or generated locally. Its purpose is to guarantee that all nodes propagate the reset signal. This supplement defines two types of bus reset, long bus reset (identical to that specified by IEEE Std 1394-1995) and arbitrated (short) bus reset. The PHY variable `reset_time` controls the length of the bus reset generated or propagated.

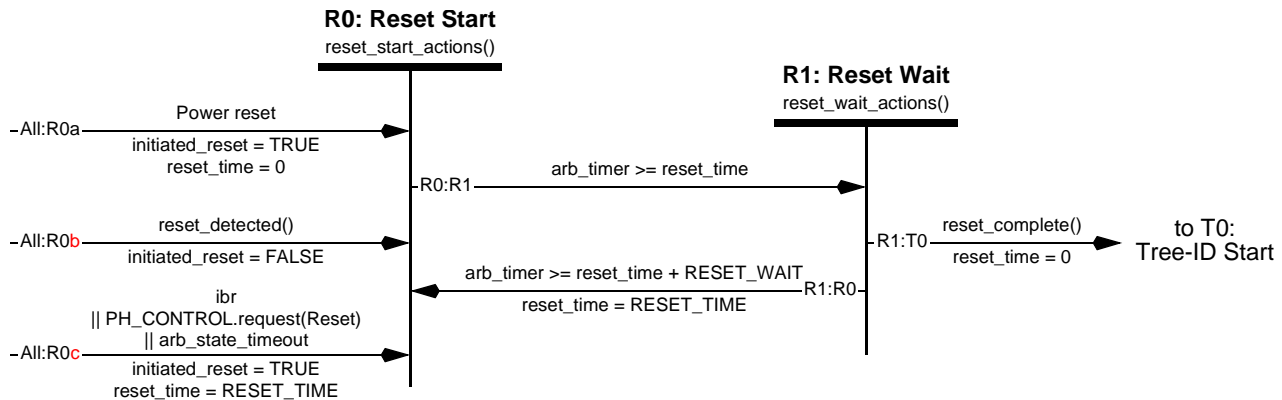


Figure 7-7 — Bus reset state machine

7.8.2.1.1 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 section; 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 connected port’s arbitration signal lines (see table 4-28 in IEEE Std 1394-1995).

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; or
- 3) The PHY stays in any state (except the idle state or a state that has an explicit time-out) for longer than `MAX_ARB_STATE_TIME`.

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 IDLEs, 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).

7.8.2.1.2 Bus reset actions and conditions

Table 7-18 — Bus reset actions and conditions (Sheet 1 of 2)

```

boolean connection_in_progress[NPORT]; // Not needed outside of the reset state machines
timer connect_timer(); // Timer for connection status monitor

void connection_status() { // Continuously monitor port status in all states
    int i;

    isolated_node = TRUE; // Assume true until first connected port found
    for (i = 0; i < NPORT; i++) {
        isolated_node &= !connected[i];
        if (connection_in_progress[i]) {
            if (!port_status[i])
                connection_in_progress[i] = FALSE; // Lost attempted connection
            else if (connect_timer >= (isolated_node) ? 2 * CONNECT_TIMEOUT : CONNECT_TIMEOUT) {
                connection_in_progress[i] = FALSE;
                connected[i] = TRUE; // Confirmed connection
                if (isolated_node) // Can we arbitrate?
                    ibr = TRUE; // No, transition to R0 for reset
                else
                    isbr = TRUE; // Yes, arbitrate for short reset
            }
        } else if (!connected[i]) {
            if (port_status[i]) { // Possible new connection?
                connect_timer = 0; // Start connect timer
                connection_in_progress[i] = TRUE;
            }
        } else if (!port_status[i]) { // Disconnect?
            connected[i] = FALSE; // Effective immediately!
            if (child[i]) // Parent still connected?
                isbr = TRUE; // Yes, arbitrate for short reset
            else
                ibr = TRUE; // No, transition to R0 for reset
        }
    }
}

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

    if (PHY_state == R0 || PHY_state == R1) // Ignore while in reset states themselves
        return(FALSE);
    if (DS_clock) // RX data makes it impossible to detect reset
        return(FALSE);
    for (i = 0; i < NPORT; i++)
        if (portR(i) == BUS_RESET) // More than 20 ns (transient DS == 11)
            if (connection_in_progress[i]) {
                reset_time = 0;
                if (isolated_node)
                    reset_time = SHORT_RESET_TIME;
                else if (connect_timer >= RESET_DETECT)
                    reset_time = RESET_TIME;
                if (reset_time != 0) {
                    connection_in_progress[i] = FALSE;
                    connected[i] = TRUE
                    return(TRUE);
                }
            }
        } else if (connected[i]) {
            reset_time = (PHY_state == RX) ? SHORT_RESET_TIME : RESET_TIME;
            return(TRUE);
        }
}

```

Table 7-18 — Bus reset actions and conditions (Sheet 2 of 2)

```

    }
    return(FALSE);
}

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

    PH_EVENT.indication(BUS_RESET_START);
    ibr = isbr = FALSE; // Don't replicate resets!
    breq = NO_REQ; // Discard any and all link requests
    child_count = physical_ID = 0;
    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 (connected[i])
            portT(i, BUS_RESET); // Propagate reset signal
        else
            portT(i, IDLE); // But only on connected ports
        child[i] = FALSE;
        child_ID_complete[i] = FALSE;
    }
    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
}

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

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

```

7.8.2.2 Self identify

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

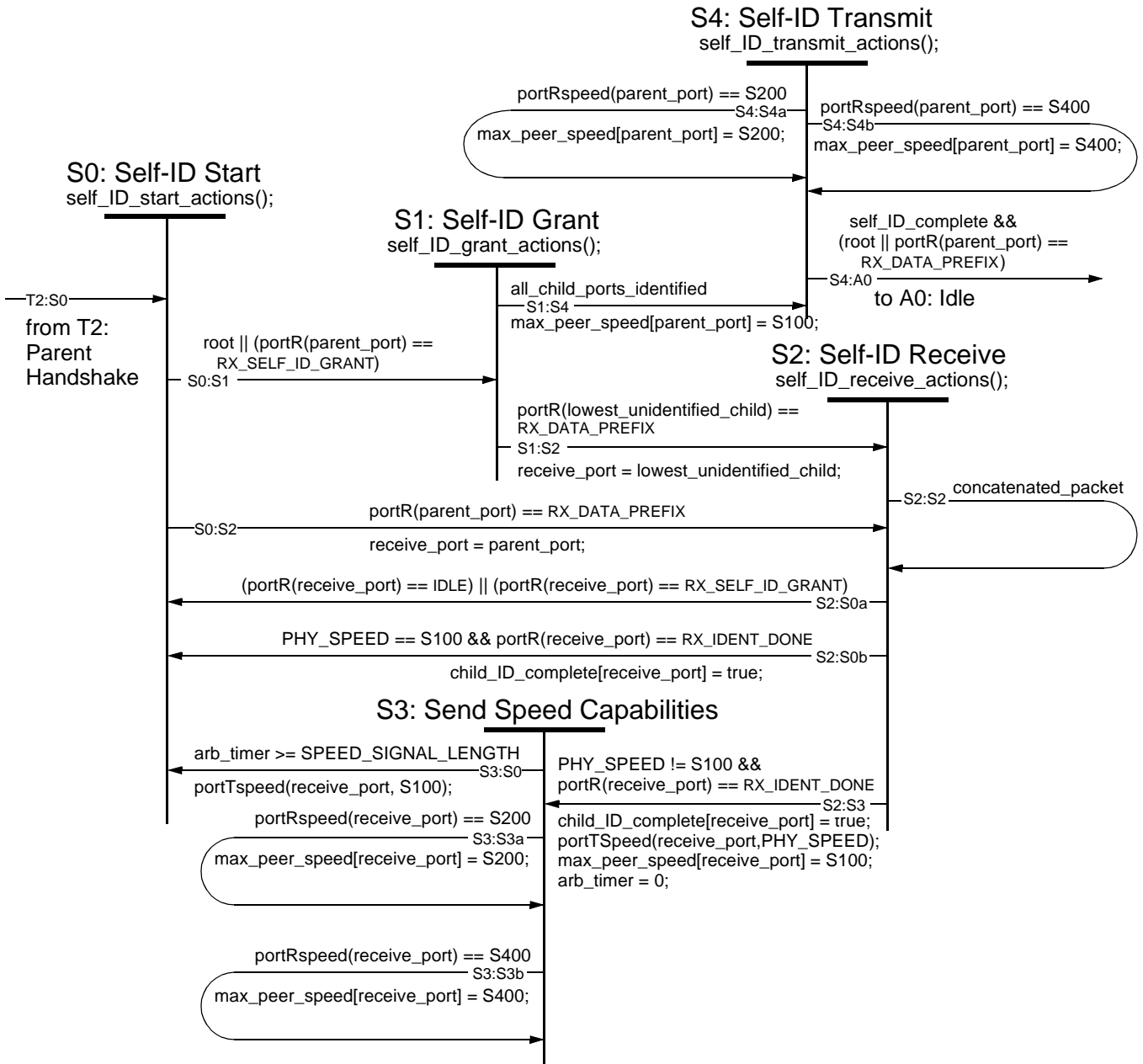


Figure 7-8 — Self-ID state machine

NOTE—This state machine does not include the reset conditions, since that would result in an overly-complex figure. See the All:R0a and All:R0b transition statements on page 76.

7.8.2.2.1 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.

Transition S0:S1. If a node is the root, or if it receives a RX_SELF_ID_GRANT signal (0Z) from its parent, it enters the Self-ID Grant state.

Transition S0:S2. If a node receives a RX_DATA_PREFIX signal (10) 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 (Z0) to the lowest numbered of those. All other connected ports are sent a TX_DATA_PREFIX signal (01) to warn them of the start of a self-ID packet.

Transition S1:S2. When the PHY receives a RX_DATA_PREFIX signal (10) 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 clause 4.4.1.2 of IEEE Std 1394-1995. Note that multiple self-ID packets may be received in this state.

Transition S2:S0a. When the receive port goes IDLE (ZZ) or gets a RX_SELF_ID_GRANT (0Z) it enters the Self-ID Start state to continue the self-ID process for the next child.

Transition S2:S0b. If the PHY gets an RX_IDENT_DONE (Z1) signal from the receiving port and the node is only capable of running at the S100 data rate, it flags that port as identified.

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

Transition S2:S3. If the PHY gets an RX_IDENT_DONE (Z1) signal from the receiving port and the node is capable of running at the S200 or S400 data rates, 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 than S100, it transmits on the receiving child port its speed capability signals as defined in clause 4.2.2.3 of IEEE Std 1394-1995 for a fixed duration SPEED_SIGNAL_LENGTH.

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.

Transition S3:S3a. If the child port signals S200 capabilities, it is recorded in the max_peer_speed variable for that port.

Transition S3:S3b. If the child port signals S400 capabilities, it is recorded in the max_peer_speed variable for that port.

State S4: Self-ID Transmit. At this point, all child ports have been flagged as identified, so the PHY can now send its own self-ID packet (see clause 7.3) using the process described in clause 4.4.1.1 of IEEE Std 1394-1995. When a non-root node is finished, it sends a TX_IDENT_DONE signal (1Z) and a speed capability signal as defined in clause 4.2.2.3 of IEEE Std 1394-1995 to its parent and IDLE (ZZ) to its children. The speed capability signal is transmitted for a fixed time duration (SPEED_SIGNAL_LENGTH). Simultaneously it monitors the bus for a speed capability transmission from

the parent. The root node just sends IDLE (ZZ) to its children. Note that the children will then enter the Idle state described in the next clause, but they will never start arbitration since an adequate arbitration gap will never open up until the Self-ID process is completed for all nodes.

Transition S4:S4a. If the parent port signals S200 capabilities, it is recorded in the max_peer_speed variable for that port.

Transition S4:S4b. If the parent port signals S400 capabilities, it is recorded in the max_peer_speed variable for that port.

Transition S4:A0. 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:

- 1) The node is the root. When the root enters the Idle state, all nodes are now sending IDLE signals (ZZ) and the gap timers will eventually get large enough to allow normal arbitration to start.
- 2) The node starts to receive a new self-ID packet (RX_DATA_PREFIX – 10). This will be 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.

7.8.2.2.2 Self-ID actions and conditions

Table 7-19 — Self ID actions and conditions (Sheet 1 of 3)

```

boolean all_child_ports_identified; // Set if all child ports have been identified
int lowest_unidentified_child;      // Lowest numbered active child that has not sent its self-ID
boolean self_ID_complete;          // Set if the self_ID transmission is complete

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] && connected[i]) { // If connected 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
            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
    }
}

```

Table 7-19 — Self ID actions and conditions (Sheet 2 of 3)

```

    physical_ID = physical_ID + 1; // Otherwise, take next PHY address
    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_pkt, ps;

    self_ID_complete = FALSE;
    receive_port = NPORT; // Indicate that 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_PREFIX);
    PH_DATA.indication(DATA_START, S100);
    for (SID_pkt_number = 0; SID_pkt_number <= last_SID_pkt; SID_pkt_number++) {
        selfID.dataQuadlet = 0; // Clear all zero fields in self ID packet
        selfID.type = 0b10;
        selfID.phy_ID = physical_ID;
        if (SID_packet_number == 0) { // First self ID packet?
            selfID.L = link_pwr; // Link active or not
            selfID.gap_cnt = gap_count;
            selfID.sp = PHY_SPEED;
            selfID.del = PHY_DELAY;
            selfID.c = CONTENDER;
            selfID.pwr = POWER_CLASS;
            selfID.i = initiated_reset;
        } else {
            selfID.seq = 1; // Indicates second and subsequent packets
            selfID.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 (!connected[port_number])
                ps |= 0b01; // Unconnected
            else if (child[port_number])
                ps |= 0v11; // Connected child
            else
                ps |= 0b10; // Connected parent
            port_number++;
            ps <<= 2; // Make room for next port's status
        }
        selfID |= ps;
        if (SID_pkt_number == last_SID_pkt) { // Last packet?
            tx_quadlet(selfID);
            tx_quadlet(~selfID);
            stop_tx_packet(TX_DATA_END, S100); // Yes, signal data end */
            PH_DATA.indication(DATA_END);
        } else {
            selfID.m = 1; // Other packets follow, set "more" bit
            tx_quadlet(self_ID_pkt);
            tx_quadlet(~self_ID_pkt);
            stop_tx_packet(TX_DATA_PREFIX, S100); // Keep bus for concatenation
            PH_DATA.indication(DATA_PREFIX);
            PH_DATA.indication(DATA_START, S100);
        }
    }
}

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
}

```

Table 7-19 — Self ID actions and conditions (Sheet 3 of 3)

```
else
    portT(port_number, TX_IDENT_DONE); // Notify parent that self-ID is complete
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
}
self_ID_complete = TRUE; // Signal completion
ping_response = FALSE;
}

void tx_quadlet(quadlet quad_data) { // Send a quadlet...
    int i;

    for (i = 0; i < 32; i++) { // ...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
    }
}
```

7.8.2.3 Normal arbitration

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

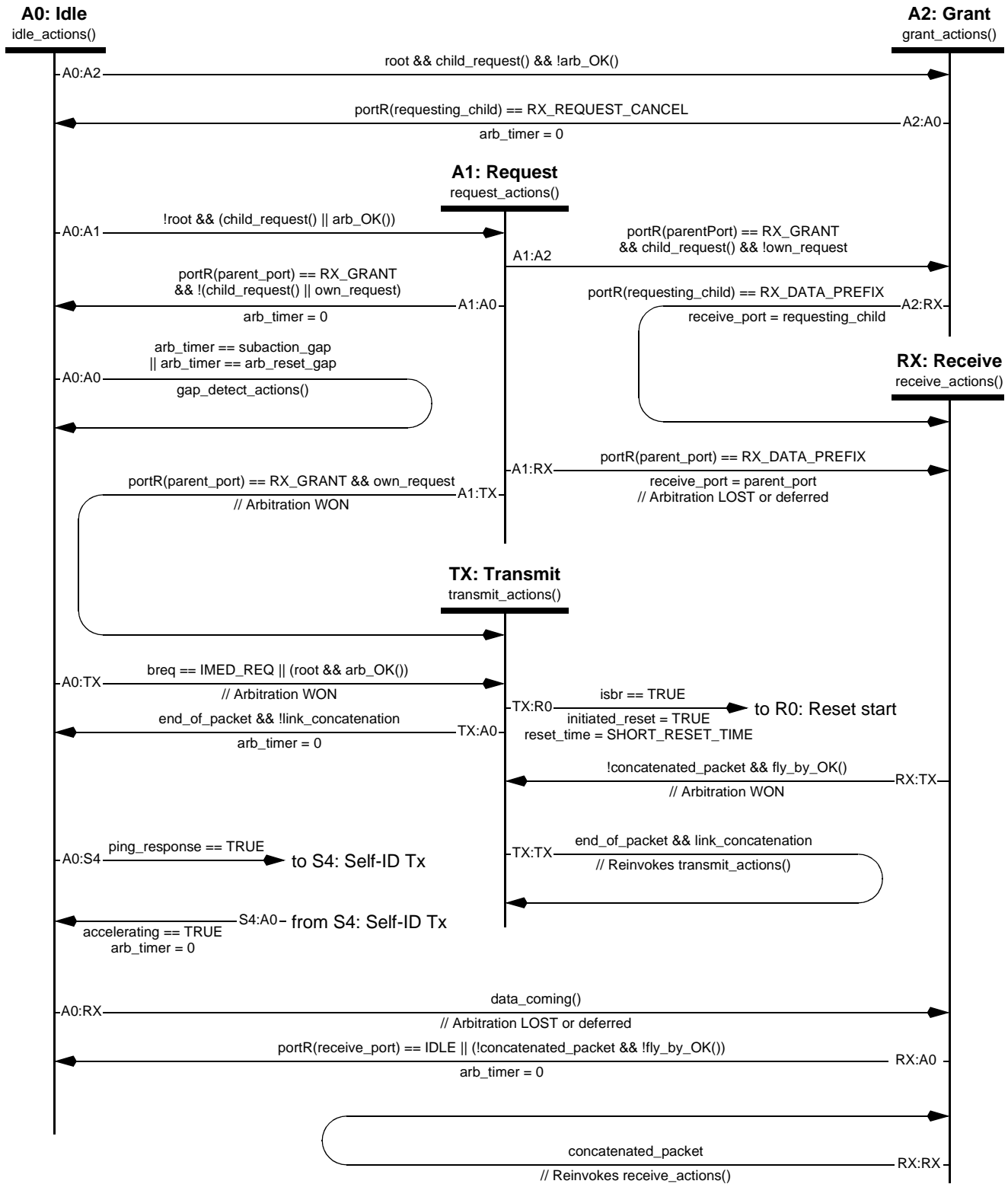


Figure 7-9 — Cable arbitration state machine

7.8.2.3.1 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 (ZZ). 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. In addition, if this is the first subaction gap after a bus reset it signals the completion of the self-identify process and the PHY **also** notifies the node controller. The detection of an arbitration reset gap **also** marks the end of a fairness interval; the PHY sets the arbitration enable flag.

~~**Transition A0:A2.** If the PHY receives a request after an arbitration reset gap, it can start the request process immediately.~~

Transition A0:A1. If the PHY has a queued **isochronous** request **from its own link** or receives an RX_REQUEST signal (0Z) 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 a RX_REQUEST signal (0Z) from one of its children, **has now queued requests from its own link** and is the root, it starts the bus grant process.

Transition A0:RX. If the PHY receives the RX_DATA_PREFIX signal on any of its ports while idle, it shifts into the Receive state and notifies the link layer that any pending arbitration requests have been lost.

Transition A0:TX. If the PHY has a queued isochronous 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.

State A1: Request. At this point, the PHY sends a TX_REQUEST signal (Z0) to its parent and a data prefix (01) to all its connected children. This will signal all children to get ready to receive a packet.

Transition A1:A0. If the PHY receives a RX_GRANT signal (00) from its parent and the requesting child has withdrawn its request, the PHY returns to Idle state.

Transition A1:A2. If the PHY receives a RX_GRANT signal (00) 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 a RX_DATA_PREFIX signal (10) from its parent, then it 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 a RX_GRANT signal (00) 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 (Z0) and the other children are sent a TX_DATA_PREFIX (01) so that they will 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 a RX_REQUEST_CANCEL signal (Z0) 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 RX: Receive. When the node starts the receive process, it clears all its request flags (forcing the link layer to send new requests if there were any queued) notifies the link layer that the bus is busy and starts the packet receive process described below. 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.

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 clause 4.2.2.3 of IEEE Std 1394-1995 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 clause 4.4.1.1 of IEEE Std 1394-1995 and returns to the Idle state.

Transition TX:R0. If arbitration has succeeded and the `reset_time` variable has a nonzero value, there is no packet to transmit. The PHY transition's to the Reset start state to commence a short bus reset.

7.8.2.3.2 Normal arbitration actions and conditions

Table 7-20 — Normal arbitration actions and conditions (Sheet 1 of 3)

```
int requesting_child;           // Lowest numbered requesting child

boolean fly_by_OK() {          // 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 (connected[i] && child[i] && (portR(i) == RX_REQUEST)) {
            requesting_child = i;      // Found a child that is requesting the bus
            return(TRUE);
        }
    return(FALSE);
}

boolean token_request() {      // TRUE if at least one token-enabled child needs a grant
    int i;
```

Table 7-20 — Normal arbitration actions and conditions (Sheet 2 of 3)

```

for (i = 0; i < NPORT; i++)
    if (connected[i] && child[i] && grant_needed[i]) {
        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 (connected[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() {

    if (arb_timer >= reset_gap_time) { // End of fairness interval?
        arb_enable = TRUE; // Reenable fair arbitration
        PH_DATA.indication(ARBITRATION_RESET_GAP); // Alert link
    } else if (arb_timer >= subaction_gap_time) {
        PH_DATA.indication(SUBACTION_GAP); // Notify link
        if (bus_initialize_active) { // End of self-identify process?
            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);
}

void request_actions() {
    int i;

    for (i = 0; i < NPORT; i++)
        if (connected[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_time)
        async_arb_OK = enab_accel && accelerating && ack;
    else if (arb_timer == subaction_gap_time + arb_delay)
        async_arb_OK = TRUE;
    else if (arb_timer >= arb_reset_gap_time + arb_delay)
        async_arb_OK = TRUE;
    if (breq == ISOCH_REQ)
        own_request = !parent_token_enable;
    else if (breq == PRI_REQ)
        own_request = async_arb_OK;
    else if (breq == FAIR_REQ)
        own_request = async_arb_OK && arb_enable;
}

```

Table 7-20 — Normal arbitration actions and conditions (Sheet 3 of 3)

```

else if (isbr)
    own_request = async_arb_OK;
else
    own_request = FALSE;
return(own_request);
}

void grant_actions() {
    int i;

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

```

7.8.2.3.3 Receive actions and conditions

Table 7-21 — Receive actions and conditions (Sheet 1 of 2)

```

void receive_actions() {
    boolean end_of_data;
    unsigned bit_count = 0, i, rx_data, tx_speed;

    ack = concatenated_packet = 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
    tx_speed = rx_speed;
    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.bits[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
            }
        }
    } while (!end_of_data);
    if (portR(receive_port) == IDLE) { // Unexpected end of data...
        if (bit_count > 8 && (breq == FAIR_REQ || breq == PRIORITY_REQ)) {
            breq = NO_REQ; // Discard (unless link believes there was an ACK)
            PH_ARB.confirmation(LOST);
        }
        ack = FALSE; // Disable fly-by acceleration
        return;
    }
    switch(portR(receive_port)) { // Send appropriate end of packet indicator
        case RX_DATA_PREFIX:

```

Table 7-21 — Receive actions and conditions (Sheet 2 of 2)

```

concatenated_packet = TRUE;
PH_DATA.indication(DATA_PREFIX);    // Concatenated packet coming
stop_tx_packet(DATA_PREFIX, rx_speed);
break;

case RX_DATA_END:
rx_speed = S100; // Default when no explicit speed code received
if (fly_by_OK())
    stop_tx_packet(DATA_PREFIX, tx_speed); // Fly-by concatenation
else {
    PH_DATA.indication(DATA_END);    // Normal end of packet
    stop_tx_packet(DATA_END, tx_speed);
}
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.bits[i] == rx_phy_pkt.checkBits[i])
        return;                      // Check bits invalid - ignore packet
switch(rx_phy_pkt.type) {            // Process PHY packets by type
case 0b00:                          // PHY config packet
    if (rx_phy_pkt.ext_type == 0)
        ping_response = (rx_phy_pkt.phy_ID == physical_ID);
    else {
        if (rx_phy_pkt.R)             // Set force_root if address matches
            force_root = (rx_phy_pkt.address == physical_ID)
        if (rx_phy_pkt.T) {          // Set gap_count unconditionally
            gap_count = rx_phy_pkt.gap_count;
            gap_count_reset_disable = TRUE;
        }
    }
}
break;

case 0b01:                          // Link-on packet
    if (rx_phy_pkt.address == physical_ID)
        PH_EVENT.indication(LINK_ON);
    break;
}
}
}

```

7.8.2.3.4 Transmit actions and conditions

Table 7-22 — Transmit actions and conditions (Sheet 1 of 2)

```

void transmit_actions() {

end_of_packet = FALSE;
int bit_count = 0, i;
PHY_packet rx_phy_pkt, tx_phy_pkt;
phyData data_to_transmit;

if (breq == FAIR_REQ)
    arb_enable = FALSE;
breq = NO_REQ;
tx_speed = speed;                // Copy speed from PH_ARB.request
receive_port = NPORT;           // Impossible port number ==> PHY transmitting
start_tx_packet(tx_speed);      // Send data prefix & speed signal
if (isbr)                       // Avoid phantom packets...
    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
}
}

```

Table 7-22 — Transmit actions and conditions (Sheet 2 of 2)

```

switch(data_to_transmit) {
  case DATA_ONE:
  case DATA_ZERO:
    tx_bit(data_to_transmit);
    if (bit_count < 64) // Accumulate possible PHY packet
      rx_phy_pkt.bits[bit_count] = data_to_transmit;
    bit_count++;
    break;

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

  case DATA_END:
    stop_tx_packet(DATA_END, tx_speed);
    end_of_packet = TRUE; // End of packet indicator
    break;
}
}
ack = (bit_count == 8); // For acceleration purposes, any 8-bit packet is an ACK
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.bits[i] == tx_phy_pkt.checkBits[i])
      return; // Check bits invalid - ignore packet
  if (tx_phy_pkt.type == 0b00)
    if (tx_phy_pkt.ext_type == 0)
      ping_response = (tx_phy_pkt.phy_ID == physical_ID);
  else {
    if (tx_phy_pkt.R)
      force_root = (tx_phy_pkt.root_ID == physical_ID);
    if (tx_phy_pkt.T) {
      gap_count = tx_phy_pkt.gap_cnt;
      gap_count_reset_disable = TRUE;
    }
  }
}
}
}
}

```

7.9 Port disable

PHY implementations compliant with this standard that implement optional *per* port disable capabilities support the Disable bit as a writable bit (see clause 6.1 for the specification of the PHY register interface used to control this feature). When the Disable bit is set to one, the affected PHY port shall be disabled. Otherwise the PHY port shall be enabled for normal connection detection, receive and transmit operations.

When a PHY port is disabled, the following shall be in effect:

- TpBias current for the port shall be off;
- The port drivers shall be tri-stated;
- With the sole exception of the common mode connection status receiver, the port line state receivers shall be in a “disconnected” state; and
- The port common mode connection status bit, Con, shall be zero and the associated port variable maintained by the PHY connection state machine shall indicate a disconnected port.

Figure 7-10 below illustrates a possible implementation of the port disable logic.

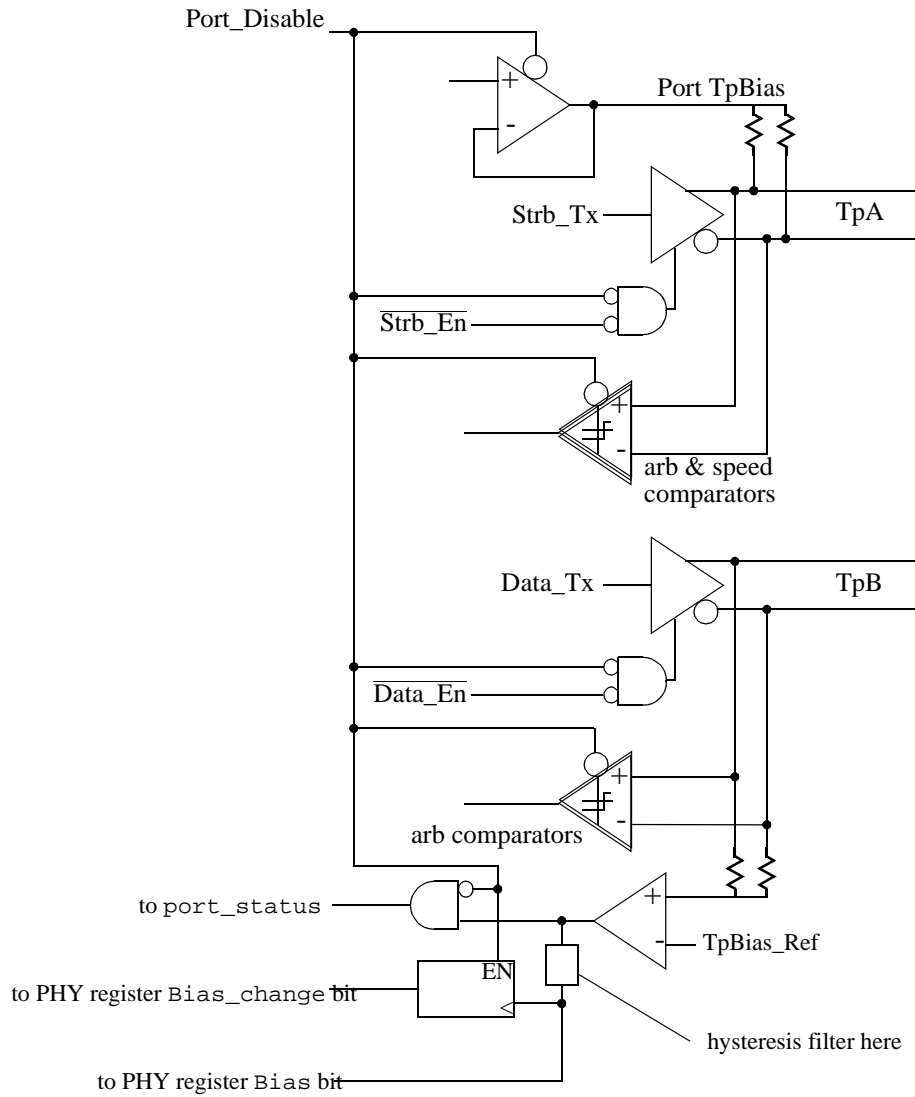


Figure 7-10 — Port disable logic

8. Asynchronous streams

Asynchronous streams are an extension of IEEE Std 1394-1995 facilities to use an existing primary packet type with different arbitration requirements. As defined, packets with a transaction code of A_{16} are called isochronous data block packets and are subject to the following restrictions:

- An isochronous data block 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; and
- For a given channel number, no more than one talker may transmit an isochronous data block packet with that channel number each isochronous period.

This extension to IEEE Std 1394-1995 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 **only** during the asynchronous period, subject to the same arbitration requirements, including fairness, as **all other request subactions asynchronous primary packets**;
- The channel number shall be allocated from the isochronous resource manager register CHANNELS_AVAILABLE; and
- 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.

8.1 Asynchronous stream packet format

The format of an asynchronous stream packet is identical to that of an isochronous data block, as specified by clause 6.2.3.1 of IEEE Std 1394-1995, and illustrated by figure 8-1.

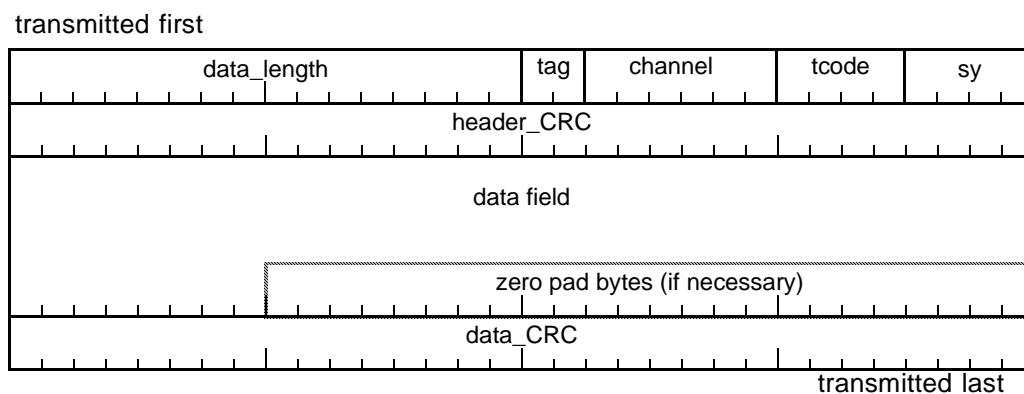


Figure 8-1 — Asynchronous stream packet format

The fields of an asynchronous stream packet shall conform to requirements of this standard and those specified in clause 6.2.4 of IEEE Std 1394-1995.

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 8-1 (which replaces table 6-4 in clause 6.2.2.3 of IEEE Std 1394-1995).

Table 8-1—Maximum data block payload for asynchronous primary packets

Data rate	Maximum payload (bytes)	Comment
S25	128	TTL backplane
S50	256	BTL and ECL backplane
S100	512	Cable base rate
S200	1024	
S400	2048	
S800	4096	
S1600	8192	
S3200	16384	

The *tag* field shall have a value of zero: unformatted data.

The *channel* field shall identify the stream and shall be allocated from the isochronous resource manager CHANNELS_AVAILABLE register.

NOTE—Subsequent to a bus reset, asynchronous stream packets may not be transmitted until the 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 above remains as described by IEEE Std 1394-1995.

8.2 Loose vs. strict isochronous

Although IEEE Std 1394-1995 prohibits the reception of an isochronous data block 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 data block packet at any time (loose isochronous).

This standard removes the IEEE Std 1394-1995 strict isochronous requirement; link designs compliant with this standard shall receive 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—The reception of isochronous packets at any time is sensible, even without consideration of asynchronous streams. If a cycle start packet is corrupted and rendered unrecognizable, many applications function are able to make valid use of isochronous data that follows so long as the link permits its reception.

8.3 Serial Bus event indication (SB_EVENT indication)

In clause 8.2.3 of IEEE Std 1394-1995, the definition of the DUPLICATE_CHANNEL_DETECTED parameter is replaced by the following:

- ~~DUPLICATE_CHANNEL_DETECTED. (Optional) An isochronous stream packet was received with a channel number equal to one of the isochronous channel numbers in use by this node. An isochronous stream packet is a primary packet identified by *tcode* A_{16} that is received after a cycle start packet and before the first subaction gap that follows the cycle start packet.~~

9. Clarifications and *corrigenda*

Since the publication of IEEE Std 1394-1995 a number of ambiguities, technical errors and typographical errors have been identified by implementors and other readers. The impact of most is minor and in many cases a thoughtful reading of the whole of the standard can lead the reader to the correct interpretation.

This section addresses the more important clarifications and *corrigenda* in no particular order. Some errors in IEEE Std 1394-1995 were deemed too minor (or their correction too self-evident) to warrant inclusion here.

9.1 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 following specifications replace IEEE Std 1394-1995 clause 6.2.2.2.3 in its entirety. The format of a cycle start packet is shown by figure 9-1.

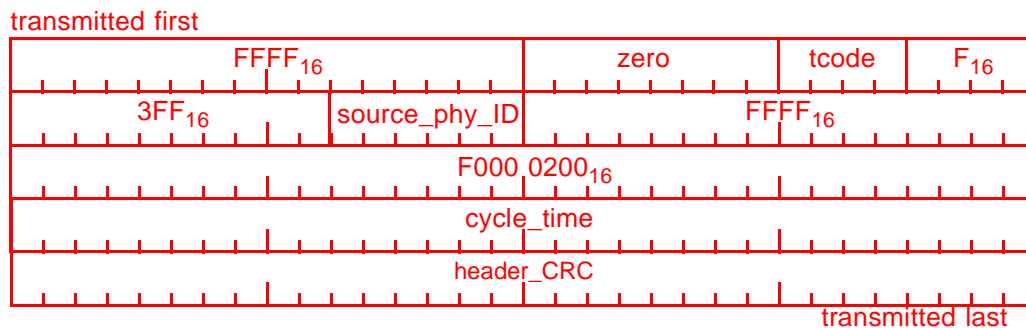


Figure 9-1—Cycle start packet format

The *tcode* field shall be 8.

The *source_phy_ID* field shall be the physical 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 clause 8.3.2.3.1 of IEEE Std 1394-1995).

The *header_CRC* field shall be calculated as specified by clause 6.2.4.15 of IEEE Std 1394-1995.

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* is 8 and the *header_CRC* is valid; recipients of cycle start packets shall not verify the other header fields.

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.

9.2 Read response for data block

This supplement adds an additional requirement to the *data_length* field specified by clause 6.2.2.3.3 of IEEE Std 1394-1995. 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.

9.3 Maximum isochronous data payload

Clause 6.2.3.1 of IEEE Std 1394-1995 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. This supplement adds an additional requirement, that the maximum data payload of an isochronous stream packet is speed-dependent and shall conform to table 9-1.

Table 9-1—Maximum payload for isochronous stream packets

Data rate	Maximum payload (bytes)	Comment
S25	256	TTL backplane
S50	512	BTL or ECL backplane
S100	1024	Cable base rate
S200	2048	
S400	4096	
S800	8192	
S1600	16384	
S3200	32768	

9.4 Transaction codes (*tcode*)

Clause 6.2.4.5 of IEEE Std 1394-1995, “Transaction code (*tcode*)”, defines the meaning of the *tcode* that identifies primary packets. This supplement extends the scope of *tcode* A_{16} in order to support asynchronous streams and also permanently reserves *tcode* E_{16} . Table 9-2 below replaces existing table 6-9 in IEEE Std 1394-1995.

Table 9-2—Transaction code encoding

Code	Name	Comment
0	Write request for data quadlet	Request subaction, quadlet payload
1	Write request for data block	Request subaction, block payload
2	Write response	Response subaction for both write requests types, no payload
3	Reserved	
4	Read request for data quadlet	Request subaction, no payload
5	Read request for data block	Request subaction, quadlet (<i>data_length</i>) payload
6	Read response for data quadlet	Response subaction to read request for quadlet, quadlet payload
7	Read response for data block	Response subaction to read request for block, block payload
8	Cycle start	Request to start isochronous period, quadlet payload
9	Lock request	Request subaction, block payload
A_{16}	Stream data block	Asynchronous or isochronous subaction, block payload
B_{16}	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.

9.5 Response codes (*r*code)

Clause 6.2.4.10 of IEEE Std 1394–1995, “Response code (*r*code)”, defines the meaning of the *r*code returned in a split transaction request. The table, identified as table 6-11 in the current standard, is reproduced below for convenience of reference.

Table 9-3—Response code encoding

Code	Name	Comment
0	resp_complete	The node has successfully completed the command.
1 to 3	—	Reserved for future standardization.
4	resp_conflict_error	A resource conflict was detected. The request may be retried.
5	resp_data_error	Hardware error, data is unavailable.
6	resp_type_error	A field in the request packet was set to an unsupported or incorrect value, or an invalid transaction was attempted (<i>e.g.</i> , a write to a read-only address).
7	resp_address_error	The destination offset in the request was set to an address not accessible in the destination node.
8 to F ₁₆	—	Reserved for future standardization.

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. This supplement clarifies response code usage in ambiguous cases 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.

9.5.1 resp_complete

Nodes shall respond with resp_complete in the circumstances described below (this is not an exhaustive list, just some examples as circumstances 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 IEEE Std 1394–1995 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 below.

9.5.2 resp_conflict_error

Nodes shall respond with resp_conflict_error in the circumstances described below:

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. Potential Deadlocks typically may arise when a request cannot be queued and blocks a node’s transaction resources.

9.5.3 resp_data_error

Nodes shall respond with resp_data_error in the circumstances described below:

An otherwise valid request packet is received but there is a data CRC error for the data payload.

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.

9.5.4 `resp_type_error`

Nodes shall respond with `resp_type_error` in the circumstances described below:

A request packet is received with a valid *tcode* (transaction code) value but the *extended_tcode* field value is reserved by IEEE Std 1394–1995.

NOTE—If a packet is received with a *tcode* value that is reserved by IEEE Std 1394–1995, the node shall not respond. ~~since it is not possible to determine whether the packet is a request, response, isochronous or some other packet.~~

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 one byte data block request.

9.5.5 `resp_address_error`

Nodes shall respond with `resp_address_error` in the circumstances described below:

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.

9.6 Tag

Clause 6.2.4.12 of IEEE Std 1394–1995, “Tag”, defines the meaning of the *tag* field transmitted in an isochronous data packet. This supplement defines, by reference to IEC 61883/FDIS, one of the previously reserved *tag* values. Table 9-4 below replaces existing table 6-12 in IEEE Std 1394-1995.

The *tag* field provides a label, useful to applications, that specifies the format of the payload carried by ~~a stream an isochronous data~~ packet.

Table 9-4—Tag field encoding

Value	Meaning
0	Data field format unspecified
1	Data format and <i>sy</i> field specified by IEC 61883/FDIS
2	Reserved for future standardization
3	Reserved for future standardization

9.7 Acknowledge codes (*ack_code*)

Clause 6.2.5.2.2 of IEEE Std 1394–1995, “Acknowledge code (*ack_code*)”, defines the meaning of the *ack_code* transmitted in immediate response to a nonbroadcast **request or response primary** packet. This supplement defines three new acknowledge codes, *ack_address_error*, *ack_conflict_error* and *ack_tardy*. Table 9-5 below replaces existing table 6-13 in IEEE Std 1394-1995.

Table 9-5—Acknowledge codes

Code	Name	Comment
0	reserved	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 will follow at a later time. This code shall not be returned for a response subaction.
3	reserved	
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 will accept the packet when the node is not busy during the next occurrence of retry phase A (see clause 7.3.5 in IEEE Std 1394-1995).
6	ack_busy_B	The packet could not be accepted. The destination transaction layer will accept the packet when the node is not busy during the next occurrence of retry phase B (see clause 7.3.5 in IEEE Std 1394-1995).
7 — A ₁₆	reserved	
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 block payload did not match the length contained in the data_length field. This code shall not be returned for any packet that does not have a data block payload.
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 <i>destination_offset</i> field in the request was set to an address not accessible in the destination node

An *ack_complete* shall not be sent in response to a read or lock request.

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 new 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 **immediate response of *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 but is expected to be on the order of *min* to *max* milliseconds.

NOTE—Request subactions are completed by either an *ack_pending* and a subsequent response packet or by an acknowledgment other than *ack_pending*. At the transaction layer both methods are equivalent and the same criteria shall be used in either case to select the appropriate acknowledgement or response code. Responses indicated by acknowledge packets are preferred over a separate response packets, since the former utilizes less Serial Bus bandwidth.

9.8 Priority arbitration for response packets

When transmitting a response packet, a link may use priority arbitration requests. If the node implements the FAIRNESS_BUDGET register (see clause 9.13), priority requests for response packet shall not count against the node's budget. A response packet is an asynchronous primary packet with a *tcode* of 2, 6 or 7.

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.

9.9 Transaction layer services

Section 6 of IEEE Std 1394-1995, "Link layer specification", in the descriptions of the different types of primary Serial Bus packets, requires 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. ~~This is a pleasing symmetry but~~ A close examination of clause 7, "Transaction layer specification", reveals that insufficient information is communicated to the transaction layer in order for it to meet this requirement.

The portion of section 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 implementors, namely, that quadlet transactions should be used only if the address is quadlet aligned.

There is also ambiguity in IEEE Std 1394-1995 with respect to the *source_ID* field transmitted in a response packet. A request addressed to a node is identified by the *destination_ID* field and may specify a bus ID of either $3FF_{16}$ (the local bus) or the value present in the most significant ten bits of the addressed node's NODE_IDS register. In order for the requester's transaction label matching between requests and responses to operate correctly, ~~it is necessary for~~ the response's *source_ID* shall be identical to the *destination_ID* field from the request.

Uniform behavior of transaction layer implementations shall be achieved by conformance to the specifications given below. Briefly, the specifications:

- a) Require that the source node ID in a response packet be equal to the destination node ID from the corresponding request;
- b) 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;
- c) 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;
- d) 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; and
- e) Emphasize that support for quadlet transactions is mandatory in all Serial Bus implementations but that support for block transactions is optional.

9.9.1 Transaction data request (TRAN_DATA.request)

In clause 7.1.2.1, two new parameters are added to the list of parameters communicated to the transaction layer *via* this service:

- 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*, quadlet or block, generated by the transaction layer. This parameter shall have a value of BLOCK TCODE or QUADLET TCODE.

9.9.2 Transaction data response (TRAN_DATA.response)

In clause 7.1.2.4, two new parameters are added to the list of parameters communicated to the transaction layer *via* this service:

- 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*, quadlet or block, received by the transaction layer. This parameter shall have a value of BLOCK TCODE or QUADLET TCODE.

9.9.3 Sending a transaction request

Clause 7.3.3.1.2, under the heading “State TX1: Send Transaction Request”, describes how the transaction code parameter (communicated to the link layer *via* LK_DATA.request) is to be selected. The nonprocedural list that follows the first paragraph is replaced with:

- 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.

9.9.4 Sending a transaction response

Clause 7.3.3.1.3, under the heading “State TX2: Send Transaction Response”, describes how the transaction code parameter (communicated to the link layer *via* LK_DATA.request) is to be selected. The nonprocedural list that follows the first paragraph is replaced with:

- 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; or
- Lock response, if the transaction type value in the transaction data request is LOCK.

9.9.5 CSR Architecture transactions mapped to Serial Bus

In IEEE Std 1394-1995 clause 7.4, below Table 7-10, “CSR Architecture / Serial Bus transaction mapping”, a paragraph describes alignment and minimal transaction requirements for Serial Bus nodes. That paragraph is replaced with the following language:

At a minimum, 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.

9.10 Serial Bus control request (SB_CONTROL.request)

A Serial Bus reset has the potential to disrupt isochronous data flow. Isochronous devices may be designed to compensate for anticipated disruptions but until equilibrium is reestablished may be more vulnerable to disruption. 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 or the node controller should not make an SB_CONTROL.request that specifies a Reset action until two seconds have elapsed subsequent to the completion of the self-identify process that follows a bus reset.

9.11 Serial Bus event indication (SB_EVENT.indication)

The definition of the DUPLICATE CHANNEL DETECTED and UNEXPECTED CHANNEL DETECTED bus event parameters in clause 8.2.3 of IEEE Std 1394-1995 are replaced 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. An isochronous stream packet was received during the isochronous period with a channel number equal to a channel number transmitted by this node in an isochronous stream packet during the same isochronous period.~~
- UNEXPECTED CHANNEL DETECTED (Optional, available only at the active isochronous resource manager). The isochronous resource manager observed a **isochronous** stream packet ~~during the last isochronous period~~ whose channel number is not specified in the Expected Channel List.

NOTE—The above events shall not be reported for a stream packet (transaction code A_{16}) observed after the subaction gap that terminates the isochronous period and before the cycle start packet that initiates the next isochronous period.

9.12 Command reset effects

A write to the RESET_START register (command reset) shall have no effects upon any of the Serial Bus-dependent registers defined in clause 8.3.2.3 of IEEE Std 1394-1995. ~~This is of particular relevance to the isochronous resource manager BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers.~~

9.13 FAIRNESS_BUDGET register

Reserved, backplane environment.

Optional, cable environment. This register shall be implemented on nodes capable of using asynchronous priority arbitration for certain primary packets and shall be located at offset 218₁₆ within initial register space.

The FAIRNESS_BUDGET register permits the bus manager to configure a node's asynchronous arbitration behavior **in order to optimize Serial Bus utilization**. 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 9-2 below.

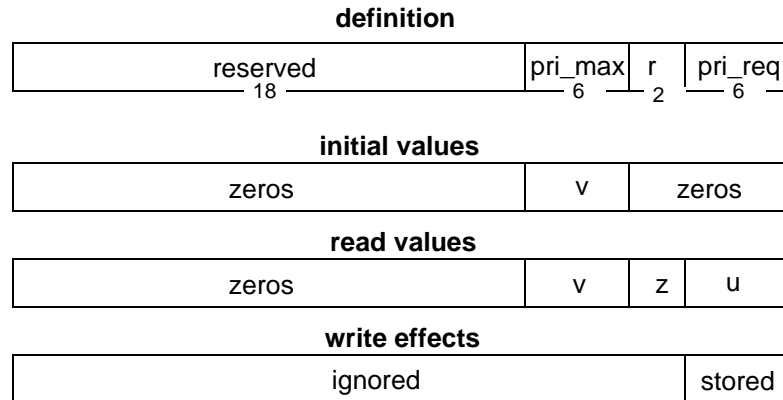


Figure 9-2—FAIRNESS_BUDGET format

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 asynchronous priority arbitration requests the link is permitted to make of the PHY during a fairness interval. A Serial Bus fairness interval exists between the occurrence of an arbitration reset gap and the first subsequent arbitration reset gap. The primary packet transaction codes for which priority asynchronous arbitration may be used are specified by table 9-6. The *pri_req* default value of zero is equivalent to the fair arbitration behavior specified by IEEE Std 1394-1995; any priority requests enabled by a nonzero value of *pri_req* are in addition to the fair arbitration request permitted each node.

Table 9-6—Transaction codes eligible for priority asynchronous arbitration

Code	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 (<i>data_length</i>) payload
9	Lock request	Request subaction, block payload
A ₁₆	Stream data block	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 9-6 so long as *priority_request_count* is nonzero. Each time a priority arbitration request is granted, 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. This supplement does not change that use of 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 FAIRNESS_BUDGET registers of all nodes on the local bus is less than or equal to 63 minus the number of nodes.

9.14 Unit registers

Clause 8.3.2.4 in IEEE Std 1394-1995 reserves a range of addresses in initial units space for Serial Bus-dependent or other uses, notably the TOPOLOGY_MAP and SPEED_MAP registers defined by that standard. Additional portions of that address space have been utilized both by this supplement and by other draft standards. Table 9-7 below replaces existing table 8-4 in IEEE Std 1394-1995.

Table 9-7—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/FDIS
904 ₁₆ — 97C ₁₆	OUTPUT_PLUG	Specified by IEC 61883/FDIS
980 ₁₆	INPUT_MASTER_PLUG	Specified by IEC 61883/FDIS
984 ₁₆ — 9FC ₁₆	INPUT_PLUG	Specified by IEC 61883/FDIS
A00 ₁₆ — AFC ₁₆		Reserved for Serial Bus
B00 ₁₆ — CFC ₁₆	FCP command frame	Specified by IEC 61883/FDIS
D00 ₁₆ — EFC ₁₆	FCP response frame	Specified by IEC 61883/FDIS
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	Present at the bus manager, only
3000 ₁₆ — 3FFC ₁₆		Reserved for Serial Bus

Except as specified by IEEE Std 1394-1995, this supplement or future Serial Bus standards, unit architectures shall not implement any CSRs that fall within the above address space.

9.14.1 SPEED_MAP_REGISTERS (cable environment)

The paragraph in IEEE Std 1394-1995 clause 8.3.2.4.2 that describes the *speed_code* entries is replaced with the following definition:

The three least-significant bits of the *speed_code* bytes specify one of the data transfer speeds S100, S200 or S400, S800, S1600 or S3200. The *speed_code* bytes use the same encoding as the *xspd* field specified in table 7-2. The remaining most-significant **five** bits are reserved for future standardization and may not be relied upon to be read as zeros.

9.14.2 TOPOLOGY_MAP registers (cable environment)

The definition of the TOPOLOGY_MAP is unchanged from the current standard but implementors are advised that PHYs compliant with this supplement transmit a minimum of two self-ID packets during the self-identify process.

9.15 Configuration ROM Bus_Info_Block

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 **most** nodes **are expected to might** generate a flurry of quadlet read requests to ascertain the identity, by means of the EUI-64, 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;
- Outside of IEEE P1394a standardization activities, work is underway to define power management for Serial Bus. A new entity, the power manager, will 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. This supplement 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 is shown in figure 9-3; this replaces existing figure 8-20 in clause 8.3.2.5.4 of IEEE Std 1394-1995.

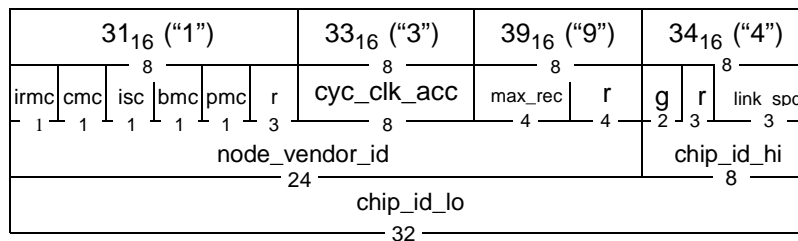


Figure 9-3—Bus_Info_Block format

With the exception of *max_rec*, the definitions of all fields previously specified by IEEE Std 1394-1995 are unchanged.

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. The maximum data payload is equal to 2^{max_rec+1} bytes, as specified by table 9-8.

Table 9-8—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

Table 9-8—Encoding of *max_rec* field (Continued)

Code	Maximum data payload (bytes)
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*.

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.

Upon the detection or initiation of a bus reset, the *generation* field (abbreviated as *g* in the figure above) shall be **modified** if any portion of configuration ROM has changed since the prior bus reset. Configuration ROM includes not only the first kilobyte of ROM (quadlets in the address range FFFF F000 0400₁₆ through FFFF F000 07FC₁₆, inclusive) but any directories or leaves that are indirectly addressed from the first kilobyte. **If the *generation* field overflows when incremented it shall wrap to zero.** The CRC in the first quadlet of configuration ROM shall be recalculated each time the *generation* field is updated.

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 self-ID packet *xspd* field specified in table 7-2.

9.16 Node_Unique_ID

The requirements of clauses 8.3.2.5.5.3 and 8.3.2.5.7.1 of IEEE Std 1394-1995 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 optional.

9.17 Determination of the bus manager

Clause 8.4.2.5 of IEEE Std 1394-1995 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 3F₁₆ 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 and 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 is not the anticipated 3F₁₆ but is instead the physical ID of the bus manager.

If the response to a successful lock (compare and swap) request to the BUS_MANAGER_ID register returns an *old_value* of either 3F₁₆ or the physical ID of the candidate bus manager, the candidate is confirmed as the new bus manager.

9.18 Gap count optimization

A bus manager or, in the absence of a bus manager, an isochronous resource manager may optimize Serial Bus performance by transmitting a PHY configuration packet (see clause 7.3.3) with the *gap_cnt* field set to a value less than 63 and the *T* bit set to 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.

9.19 Automatic activation of the cycle master

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.

Cycle master operations are controlled by the *cmstr* bit in the STATE_SET register defined in clause 8.3.2.2.1 of the current standard. The paragraphs that specify the behavior of *cmstr* are replaced with the following definition:

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 node is not the root.

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; else
- c) Otherwise *cmstr* shall be set to the value of the *cmc* bit (from the bus information block).

9.20 Isochronous period too long

If the cycle master detects an isochronous **period** that exceeds the maximum time, it shall clear its own *cmstr* bit and cease transmission of cycle start packets. The cycle too long condition is defined as the failure to detect a subaction gap within 125 μ s of the transmission of a cycle start packet.

9.21 Abdication by the bus manager

This clause 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 supplement.

In order to support the procedures described here, a new bus-dependent bit is defined for the STATE_CLEAR register, as illustrated below.

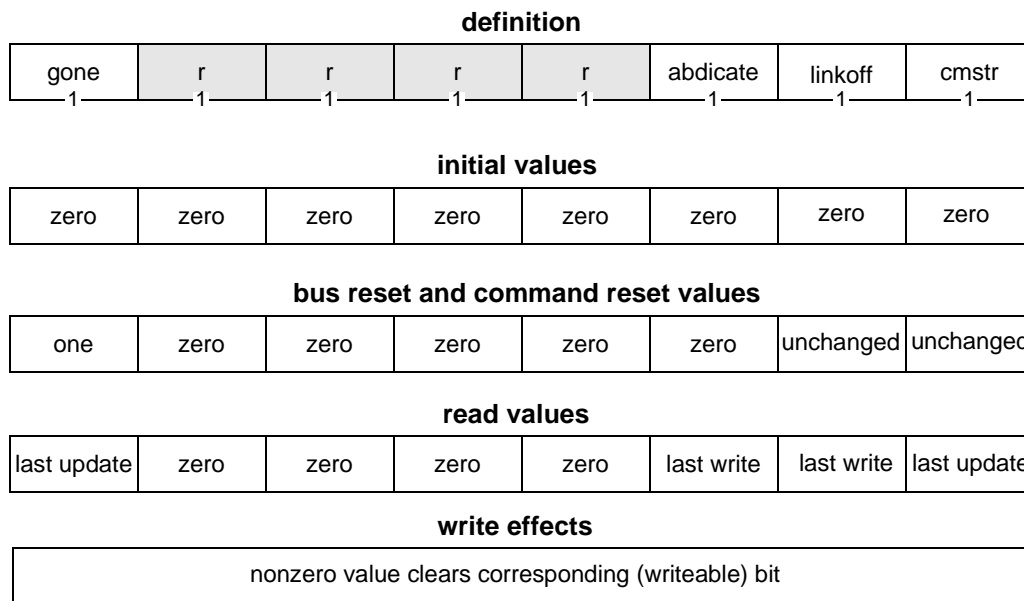


Figure 9-4—STATE_CLEAR.bus_depend field

The four shaded *r* bits are reserved for future definition by Serial Bus.

The *gone* and *linkoff* bits are specified by IEEE Std 1394-1995.

The *cmstr* bit is specified by clause 9.19 above.

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 IEEE Std 1394-1995, the incumbent manager contends immediately after the first subaction gap that follows a bus reset while nonincumbent, bus manager-capable nodes wait one second before contending. When *abdicate* is one, the node shall wait one second before contending—whether incumbent or not.

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 IEEE Std 1394-1995, 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; and
- d) If the candidate bus manager fails to become the bus manager, it shall transmit a PHY configuration packet with the *R* bit set to one, the *root_ID* field set to the value of the candidate’s own physical ID and the *T* bit cleared to zero. The effect of this PHY configuration packet is to clear the *force_root* variable of other nodes to zero while leaving the *gap_count* at its present value. The candidate bus manager shall set its own *force_root* variable to one, initiate a Serial Bus reset and attempt to become the bus manager as described in c) above.

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 supplement. 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.

9.22 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.

With respect to these security considerations, A **module** compliant with this supplement shall meet the following requirements **at the time of manufacture**:

- 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**;
- A node shall not originate an asynchronous request or response packet with a *source_ID* field that is not equal to either a) the most significant 16 bits of the node's `NODE_IDS` register or b) the concatenation of $3FF_{16}$ (**the local bus ID**) and the physical ID assigned to the node's PHY during the self-identify process; and
- 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 a) 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 $3F_{16}$ (**the broadcast physical ID**), or b) the concatenation of $3FF_{16}$ (**the local bus ID**) and either the physical ID assigned to the node's PHY during the self-identify process or $3F_{16}$ (**the broadcast physical ID**).

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, which work is in progress in the IEEE P1394.1 working group. **Only for the purposes of forwarding asynchronous request and response packets may a bridge recognize packets not addressed to it or transmit packets as a proxy for other nodes.**

NOTE—A module that does not conform to these transaction integrity requirements shall not claim full compliance with this specification.

Annex A

(normative)

Cable environment electrical isolation

This annex replaces IEEE Std 1394-1995 annex A, “Cable environment system properties”, in its entirety. That annex defines specifications in the following areas:

- 1) External shielded cable interconnection;
- 2) Internal unshielded interconnection;
- 3) Cable power sourcing and connection;
- 4) Powering PHY integrated circuits (ICs) and devices; and
- 5) 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.* In addition to the elimination of the normative requirements of prior annex A, clause 4.2.1.4.8 of IEEE Std 1394-1995, “Shield ac coupling,” is deleted.

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.

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 below.

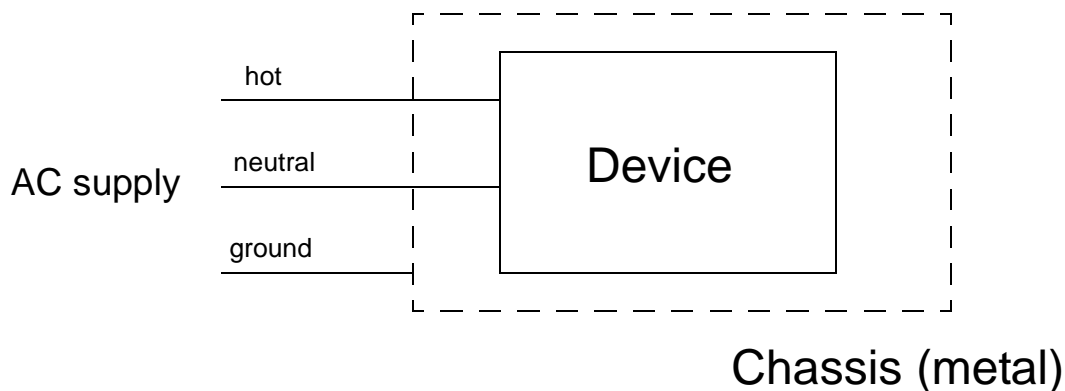


Figure A-1 — AC power supply with ground

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.

A.2 Electrical isolation

The cables and connectors specified by IEEE Std. 1394-1995 provide three ways in which chassis-to-chassis ground currents may flow:

- The ground wire returns ground current to a chassis with a non-isolated 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); or
- The outer shield 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.

The alternate cables and connectors standardized in this supplement, which have no power and ground conductors, pose an additional problem: that of providing a return path to the PHY for common mode speed signalling currents through their shields while still blocking power line currents. There are two solutions:

- Limit alternate cables to S100 operation—in which case common mode speed signalling currents do not arise; or
- Carefully manage the connector shield returns provide adequate RF emissions shielding and no power line frequency conductance at the same time preserving recognizable speed signals for the PHY.

A.3 Agency requirements

The information below 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 the standard permits cable power distribution at voltages greater than 24V 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, ANSI/NFPA 70. The equipment may also be installed according to the Standard for the Protection of Electronic Computer/Data-Processing Equipment, ANSI/NFPA 75.

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 will 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, and other equipment for the detection and signalling of conditions capable of causing damage or injury to persons, fire extinguishing systems and electrical power-supply equipment such as motor-generator sets, and 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:

Information Processing and Business Equipment, UL 478

National Electric Code, ANSI/NFPA 70

Standard for the Protection of Electronic Computer/Data-Processing Equipment, ANSI/NFPA 75

Reference documents applicable in Japan include:

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

A more accurate citation of these references is given by the text below:

- アメリカの 電気工事配線規定 ANSI/NFPA70 ,75に相当する日本の規定は
1. (NFPA70類似) 電気設備技術基準 通産省令 (法律)
 2. (NFPA70類似) 内線線規程 (社) 日本電気協会
(法律ではない、電気設備技術基準の解説)
 3. (NEPA,75類似) 情報システム安全対策基準 通産省機怪情報産業部
(法律ではない、セキュリ
ティ対策の目標を示した
ガイドラン)
- UL478相当の日本の規格は
- 情報処理機器に対して
JEIDA-37 (社) 日本電子工業振興協会 コンピュータ業界自主安全基準
- 家電製品、電源コード、プラグ 他 (政府指定品目) に対しては、
電気用品取締法 通産省令 (法律)

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 above provide reference information for selection and installation of cabling in walls, temporary partitions, under floors, in overhead or suspended ceilings or in adverse atmospheres.