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

ISBN x-xxxxx-xxx-x

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.

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

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. "Al" Kiener, *Chair* **Donald C. Loughry,** *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril	Jim Isaak	Marco W. Migliaro
Clyde R. Camp	Ben C. Johnson	Mary Lou Padgett
Joseph A. Cannatelli	Sonny Kasturi	John W. Pope
Stephen L. Diamond	Lorraine C. Kevra	Arthur K. Reilly
Harold E. Epstein	Ivor N. Knight	Gary S. Robinson
Donald C. Fleckenstein	Joseph L. Koepfinger*	Ingo Rüsch
Jay Forster*	D. N. "Jim" Logothetis	Chee Kiow Tan
Donald N. Heirman	L. Bruce McClung	Leonard L. Tripp
Richard J. Holleman		Howard L. Wolfman

*Member Emeritus

Other candiates 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 Initialization and reset 37
5.2 Link-on indication 39
5.3 Operation 39
5.4 Link requests 40
5.5 Status 46
5.6 Transmit 47
5.7 Receive 49
5.8 Electrical characteristics 50

6. PHY register map 59

6.1 PHY register map (cable environment) 59
6.2 PHY register map (backplane environment) 64
6.3 Integrated link and PHY 65

7. Cable physical layer performance enhancement specifications 67

7.1 Cable topology 67
7.2 Cable power and ground 68
7.3 Data signal rise and fall times 69
7.4 Cable PHY packets 70
7.5 Cable PHY line states 73
7.6 Cable PHY timing constants 74
7.7 Node variables 75
7.8 Port variables 76
7.9 Cable physical layer operation 76
7.10 Port disable 99

8. Asynchronous streams 101

8.1 Asynchronous stream packet format 102
8.2 Loose vs. strict isochronous 103

9. Clarifications and corrigenda 105

9.1 Cycle start 105
9.2 Read response for data block 105
9.3 Maximum isochronous data payload 106
9.4 Transaction codes (tcode) 106
9.5 Response codes (rcode) 107
9.6 Tag 108
9.7 Acknowledge codes (ack_code) 109
9.8 Priority arbitration for PHY packets and response packets 110
9.9 Transaction layer services 110
9.10 Serial Bus control request (SB_CONTROL.request) 112
9.11 Serial Bus event indication (SB_EVENT.indication) 112
9.12 NODE_IDS register 112
9.13 SPLIT_TIMEOUT register 113
9.14 Command reset effects 114
9.15 PRIORITY_BUDGET register 114
9.16 Unit registers 116
9.17 Configuration ROM Bus_Info_Block 116
9.18 Node_Unique_ID 118
9.19 Determination of the bus manager 118
9.20 Gap count optimization 118
9.21 Automatic activation of the cycle master 119
9.22 Abdication by the bus manager 119
9.23 Internal device physical interface 121
9.24 Transaction integrity safeguards 121

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 — Digital differentiator signal transformation 36
Figure 5-3 — LPS waveform when differentiated 37
Figure 5-4 — PHY/link interface reset via LPS 38
Figure 5-5 — LReq and Ctl timings 40
Figure 5-6 — Status timing 46
Figure 5-7 — Transmit timing 48
Figure 5-8 — Receive timing 49
Figure 5-9 — Signal levels for rise and fall times 51
Figure 5-10 — PHY to link transfer waveform at the PHY 51
Figure 5-11 — Link to PHY transfer waveform at the PHY 52
Figure 5-12 — PHY to link transfer waveform at the link 52
Figure 5-13 — Link to PHY transfer waveform at the link 52
Figure 5-14 — Link to PHY delay timing 54
Figure 5-15 — Ground coupling circuit example 55
Figure 5-16 — Capacitive isolation barrier circuit example for Ctl[0:1] and D[0:n] 55
Figure 5-17 — Capacitive isolation barrier circuit example for LinkOn 56
Figure 5-18 — Capacitive isolation barrier circuit example for LPS 56
Figure 5-19 — Capacitive isolation barrier circuit example for LReq 56
Figure 5-20 — Capacitive isolation barrier circuit example for SClk 57
Figure 6-1 — Extended PHY register map for the cable environment 59
Figure 6-2 — PHY register page 0: Port Status page 61
Figure 6-3 — PHY register page 1: Vendor Identification page 63
Figure 6-4 — PHY register map for the backplane environment 64
Figure 7-1 — Node power interface for POWER_CLASS one, two or three 68
Figure 7-2 — Self-ID packet formats 70
Figure 7-3 — Link-on packet format 72
Figure 7-4 — PHY configuration packet format 72
Figure 7-5 — Ping packet format 73
Figure 7-6 — Cable physical layer architecture 77
Figure 7-7 — Bus reset state machine 85
Figure 7-8 — Self-ID state machine 88
Figure 7-9 — Cable arbitration state machine 93
Figure 7-10 — Port disable logic 100
Figure 8-1 — Asynchronous stream packet format 102
Figure 9-1— Cycle start packet format 105

Figure 9-2— NODE_IDS format 113
Figure 9-3— SPLIT_TIMEOUT format 114
Figure 9-4— PRIORITY_BUDGET format 115
Figure 9-5— Bus_Info_Block format 117
Figure 9-6— STATE_CLEAR.bus_depend field 120
Figure A-1 — AC power supply with ground 123

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 25
Table 4-3 — Performance group B 26
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 37
Table 5-3 — Initialization of the PHY/link interface 38
Table 5-4 — LinkOn timing parameters 39
Table 5-5 — Ctl[0:1] when PHY is driving 39
Table 5-6 — Ctl[0:1] when the link is driving (upon a grant from the PHY) 40
Table 5-7 — Bus request format for cable environment 40
Table 5-8 — Bus request format for backplane environment 41
Table 5-9 — Register read request format 41
Table 5-10 — Register write request format 41
Table 5-11 — Acceleration control request format 41
Table 5-12 — Request type field 42
Table 5-13 — Request speed field 42
Table 5-14 — Link request effects on PHY variables 42
Table 5-15 — Link rules to initiate a request on LReq 44
Table 5-16 — PHY disposition of link request 45
Table 5-17 — Status bits 47
Table 5-18 — Speed code signaling 49
Table 5-19 — DC specifications for PHY/link interface 50
Table 5-20 — AC timing parameters 51
Table 5-21 — AC timing parameters at the PHY 52
Table 5-22 — AC timing parameters at the link 53
Table 5-23 — Link to PHY delay timing parameters 54
Table 6-1 — PHY register fields for the cable environment 59
Table 6-2 — PHY register Port Status page fields 62
Table 6-3 — PHY register Vendor Identification page fields 63
Table 6-4 — PHY register fields for the backplane environment 64
Table 7-1 — Cable power source requirements 68
Table 7-2 — Output rise and fall times 69
Table 7-3 — Self-ID packet fields 71
Table 7-4 — Link-on packet fields 72
Table 7-5 — PHY configuration packet fields 72
Table 7-6 — Ping packet fields 73
Table 7-7 — Cable PHY received arbitration line states 73
Table 7-8 — Cable PHY timing constants 74
Table 7-9 — Node variables 75
Table 7-10 — Port variables 76
Table 7-11 — Cable PHY code definitions 78
Table 7-12 — Cable PHY packet definitions 79
Table 7-13 — Digital speed filtering 80

Table 7-14 — Data transmit actions 81
Table 7-15 — Start data transmit actions 81
Table 7-16 — Stop data transmit actions 82
Table 7-17 — Data reception and repeat actions (Sheet 1 of 2) 82
Table 7-18 — Start data reception and repeat actions (Sheet 1 of 2) 83
Table 7-19 — Bus reset actions and conditions (Sheet 1 of 2) 86
Table 7-20 — Self ID actions and conditions (Sheet 1 of 3) 90
Table 7-21 — Normal arbitration actions and conditions (Sheet 1 of 3) 95
Table 7-22 — Receive actions and conditions (Sheet 1 of 2) 97
Table 7-23 — Transmit actions and conditions (Sheet 1 of 2) 98
Table 8-1— Maximum data payload for asynchronous primary packets 102
Table 9-1— Maximum payload for isochronous stream packets 106
Table 9-2— Transaction code encoding 106
Table 9-3— Response code encoding 107
Table 9-4— Tag field encoding 108
Table 9-5— Acknowledge codes 109
Table 9-6— Request subactions eligible for priority asynchronous arbitration 115
Table 9-7— Serial Bus-dependent registers in initial units space 116
Table 9-8— Encoding of max_rec field 117

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) Performance enhancements to the PHY layer that are interoperable with the existing standard, *e.g.*, a method to shorten the arbitration delay when the last observed Serial Bus activity is an acknowledge packet;
d) A redefinition of the isochronous data packet, transaction code A_{16}, to permit its use in either the asynchronous or isochronous periods;
e) More stringent requirements on the power to be supplied by a cable power source and a clarification of electrical isolation requirements;
f) 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 Varembé, 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:

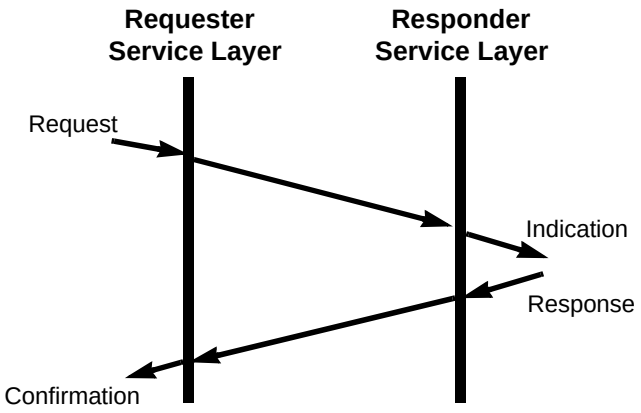


Figure 1-1—Service model

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.

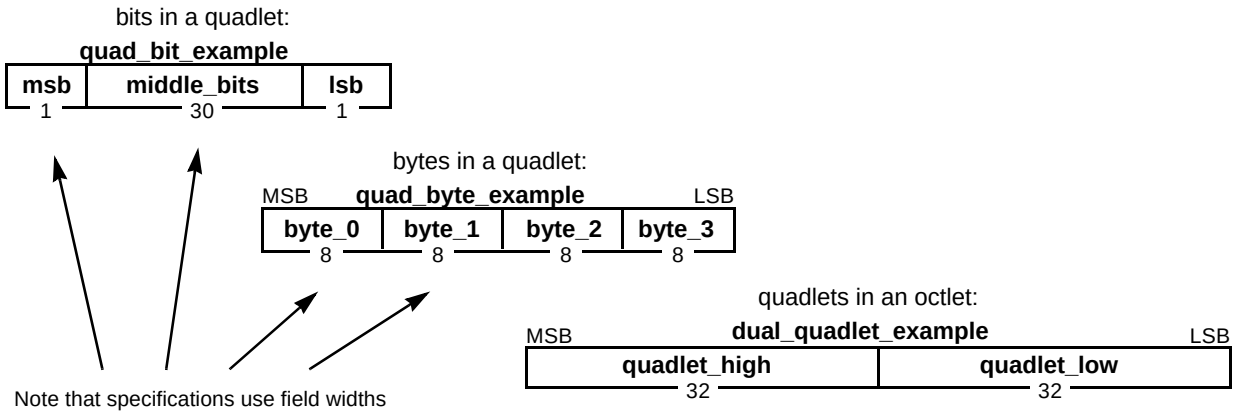


Figure 1-2—Bit and byte ordering

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 “$1A_{16}$” or “11010_2”. In C code examples, hexadecimal numbers have a “`0x`” prefix and binary numbers have a “`0b`” prefix, so the decimal number “26” would be represented by “`0x1A`” or “`0b11010`.”

1.6.5 Packet formats

Most Serial Bus packets consist of a sequence of quadlets. Packet formats are shown using the style given in figure 1-3.

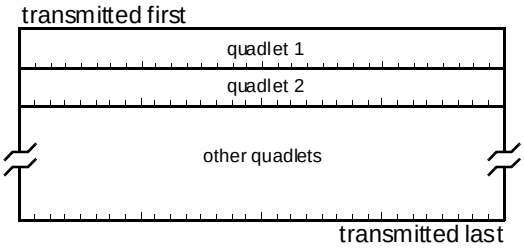


Figure 1-3—Example packet format

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
&=	Logical AND and consequent assignment. The statement q &= FALSE sets the value of q to the expression (q && FALSE). Thevalue of q used in the evaluation of the expression is obtained before the assignment.
<<	Left shift; x << 2 shifts the value of x left by two bit positions and fills the vacated positions with zero
>>	Right shift; vacated bit positions are filled with zero or one according to the data type of the operand but in this 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 + 1 : 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.

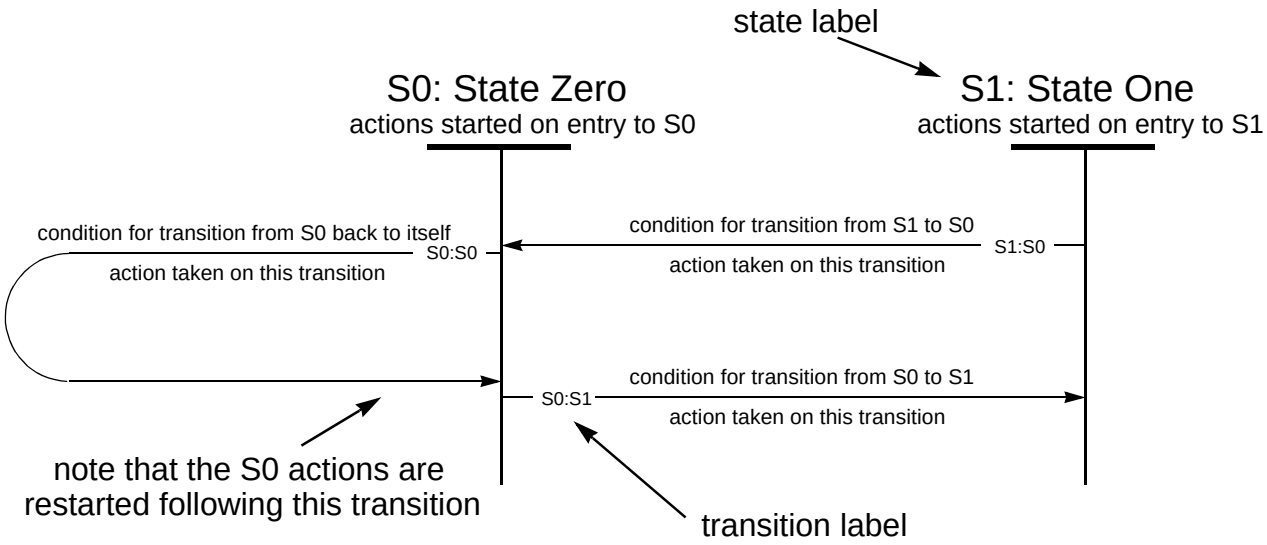


Figure 1-4—State machine example

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.

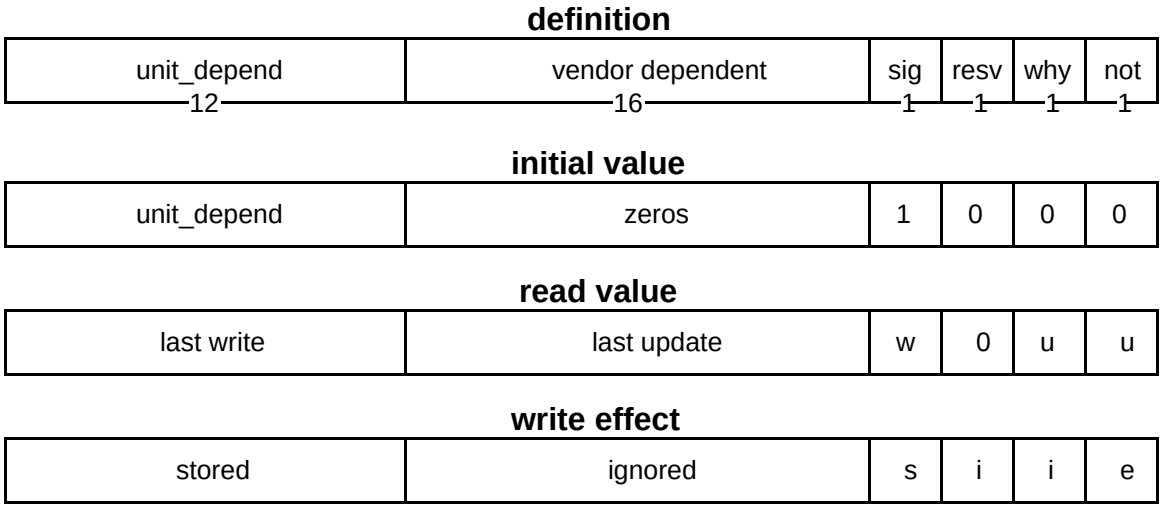


Figure 1-5—CSR format specification (example)

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.

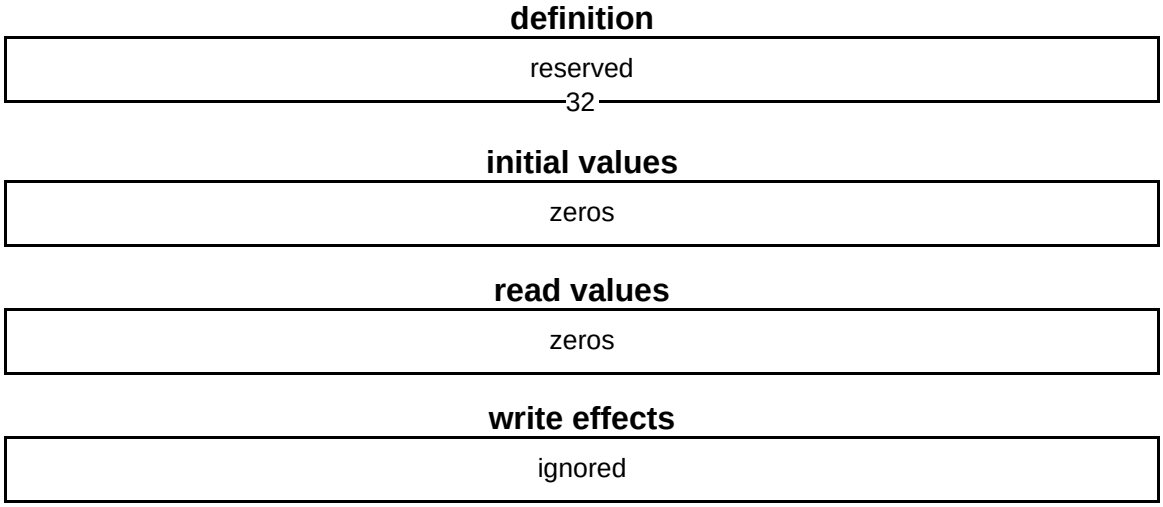


Figure 1-6—Reserved CSR field behavior

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	`new_value = data_value | (old_value & ~arg_value);`
compare_swap	`if (old_value == arg_value) new_value = data_value;`
fetch_add	`new_value = old_value + data value;`
little_add	`(little) new_value = (little) old_value + (little) data_value;`
bounded_add	`if (old_value != arg_value) new_value = old_value + data_value;`
wrap_add	`new_value = (old_value != arg_value) ? old_value + data_value : data_value;`

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 period).

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 ± 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—Microprocessor 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 period.

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 0000_{16}. This address space is reserved for resources accessible immediately after a bus reset. Core registers defined by ISO/IEC 13213:1994 are located within initial register space as are Serial Bus-dependent registers defined by IEEE Std 1394-1995.

2.2.23 initial units space: A portion of initial node space with a base address of FFFF F000 0400_{16}. This places initial units space adjacent to and above initial register space. The CSR's and other facilities defined by unit architectures are expected to lie within this space.

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

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

2.2.26 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 period, 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 originating port:

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

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

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

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

2.2.41 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.

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

2.2.43 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.

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

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

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

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

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

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

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

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

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

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

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

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

2.2.56 transmitting port:

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

2.2.58 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.

2.2.59 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.

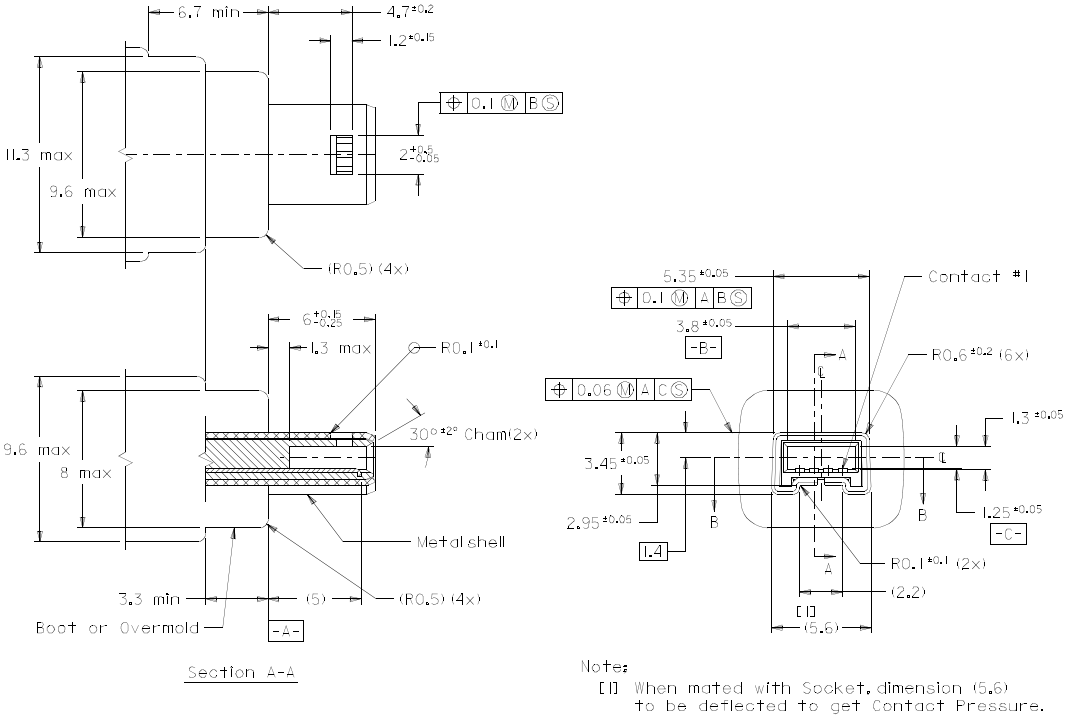


Figure 4-1 — Plug body

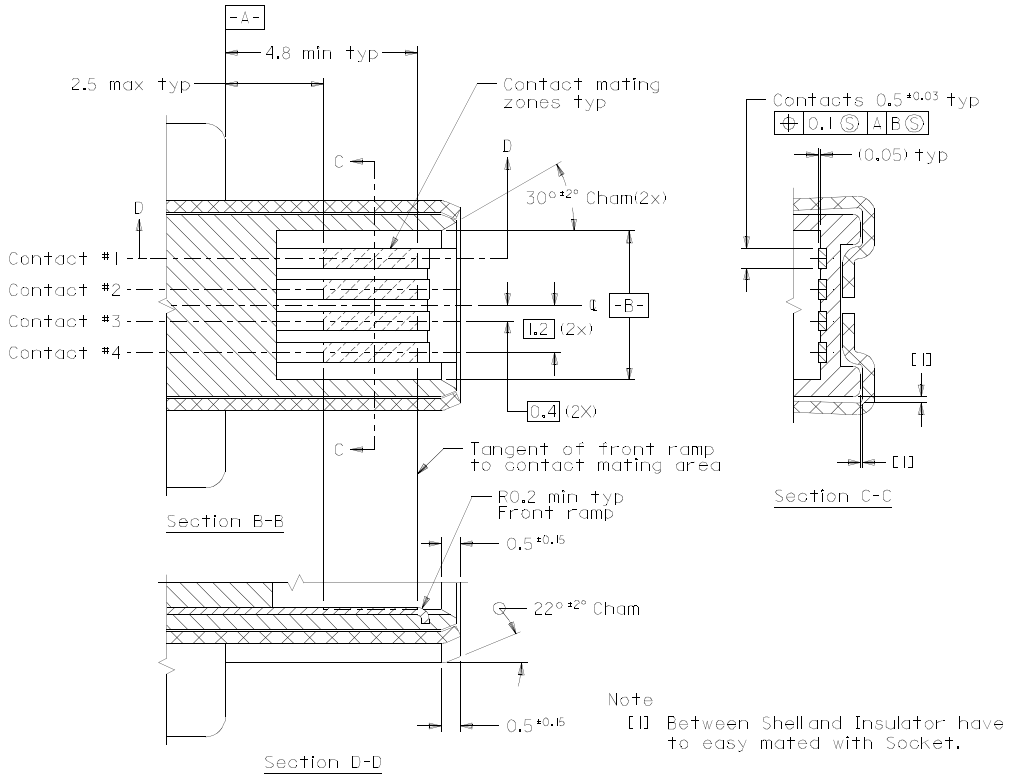


Figure 4-2 — Plug section details

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.

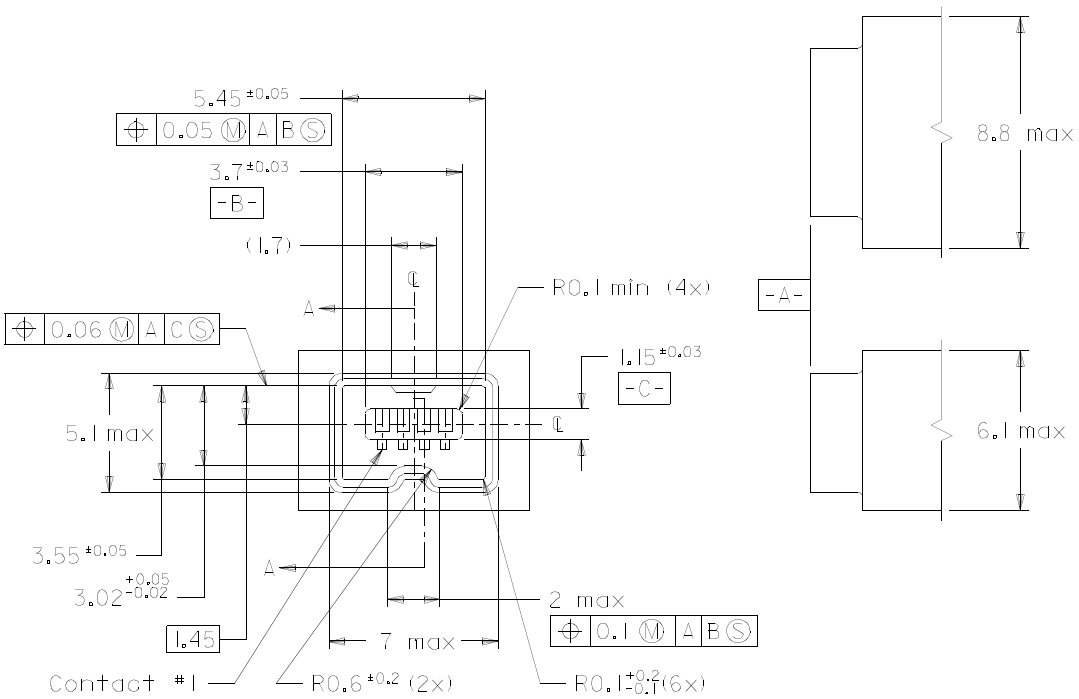


Figure 4-3 — Connector socket interface

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.

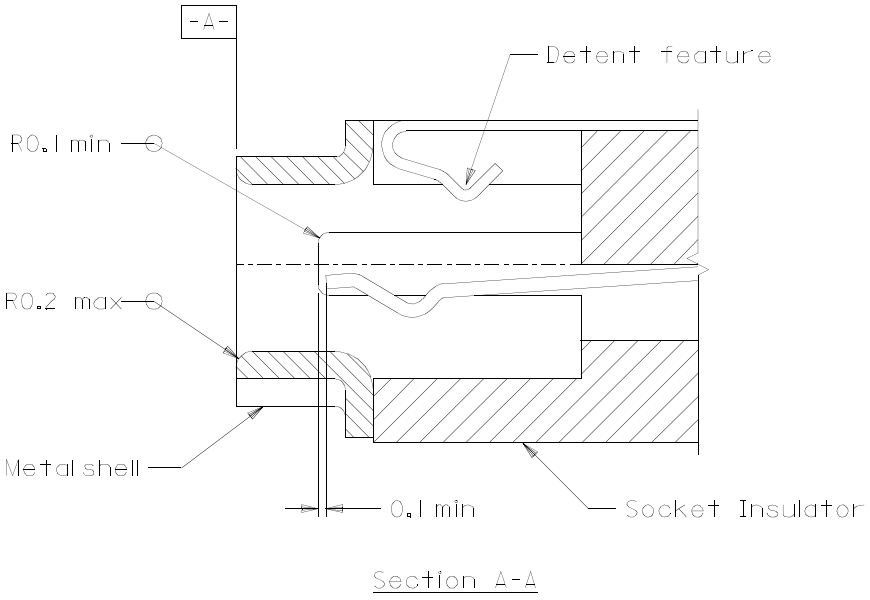


Figure 4-4 — Socket cross-section A–A

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

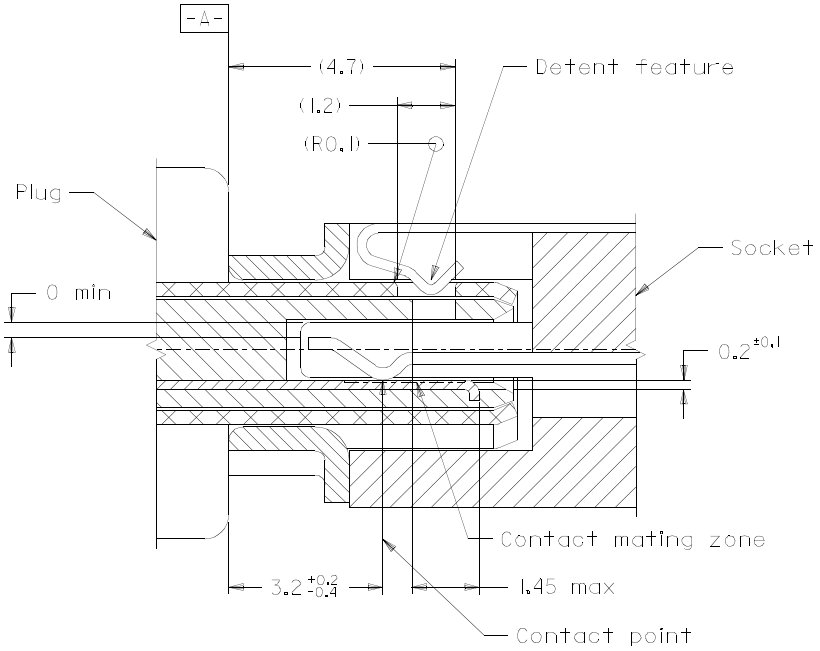


Figure 4-5 — Cross-section of plug and socket contacts

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

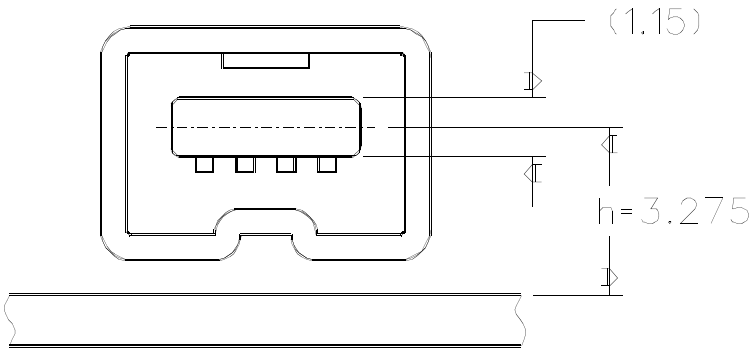


Figure 4-6 — Socket position when mounted on a printed circuit board

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.

a) 0.76 µm (30µin), minimum, gold, over 1.27 µm (50 µin), minimum, nickel.
b) 0.05 µm (2 µin), minimum, gold, over 0.76 µm (30 µin), minimum, palladium-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.

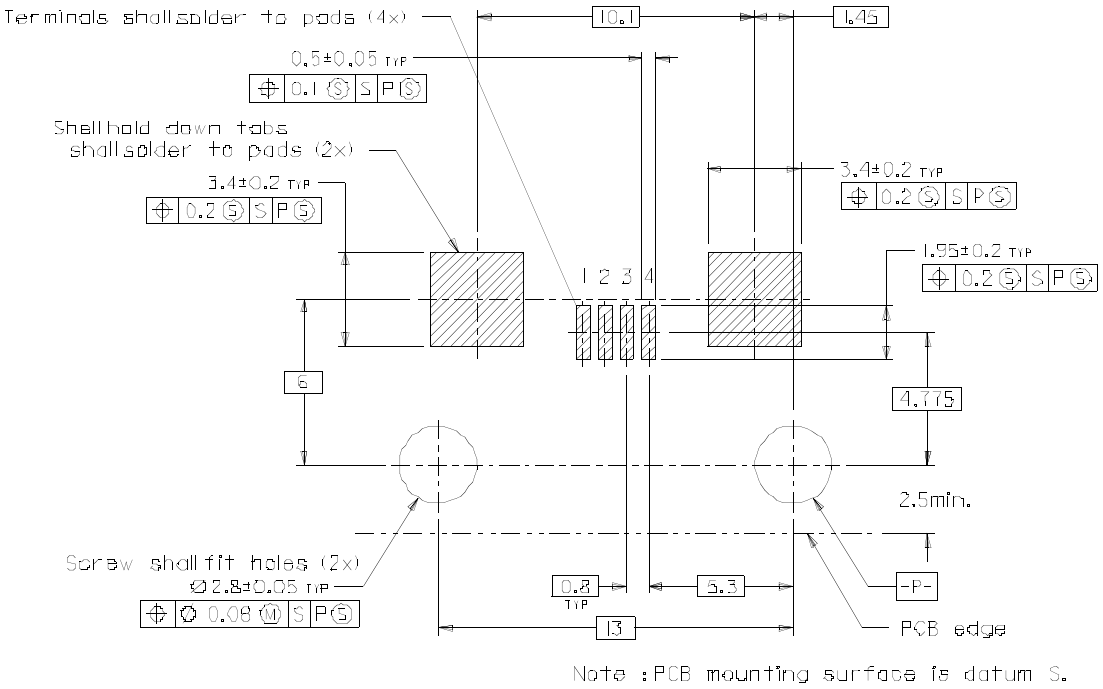


Figure 4-7 — Flat surface mount printed circuit board connector footprint

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

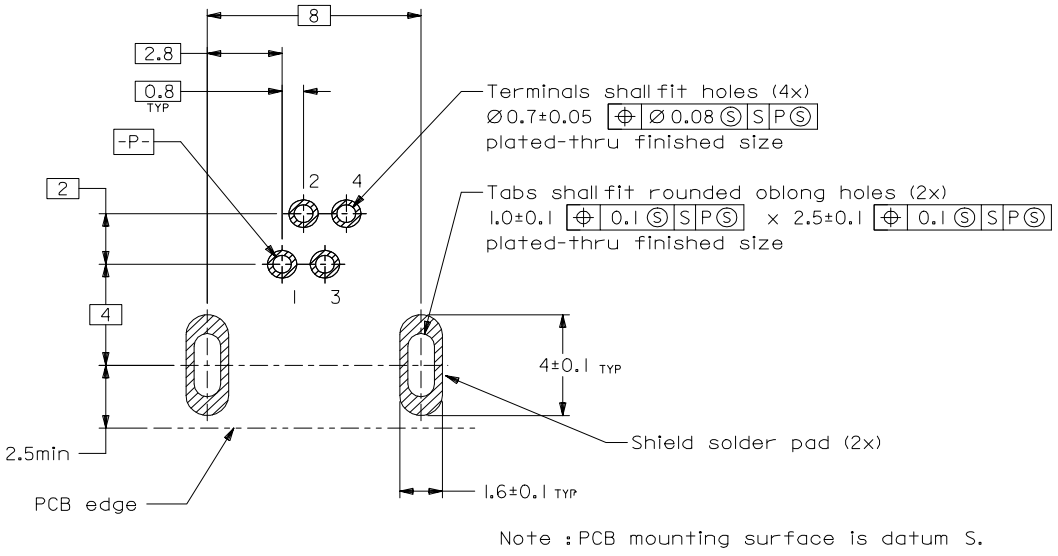


Figure 4-8 — Flat through-hole mount printed circuit board connector footprint

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.

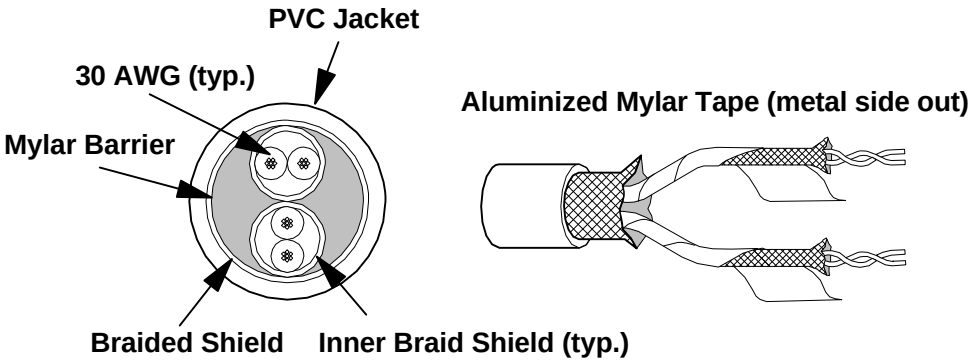


Figure 4-9 — Cable material construction example (for reference only)

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.

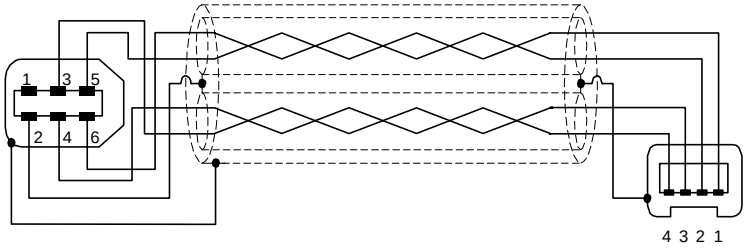
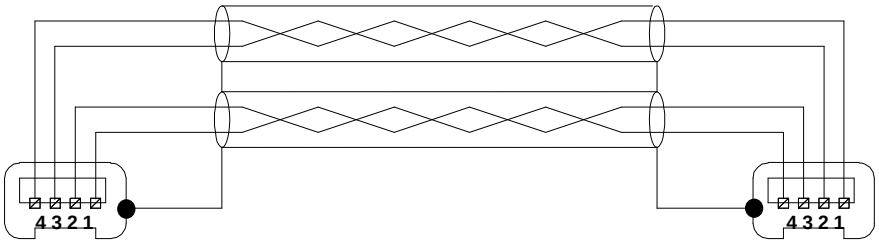


Figure 4-10 — Cable assembly and schematic (standard to alternate connector)



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

Signal Names

Both ends identical; for reference only

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

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

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; + l5 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

Table 4-2 — Performance group A

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance Level
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.

Table 4-3 — Performance group B

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
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:

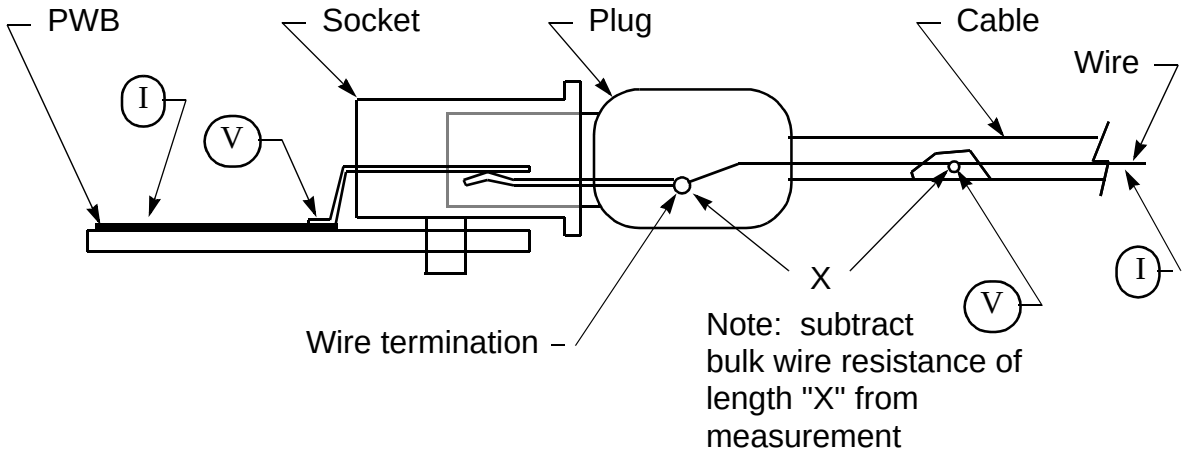
[0] Sockets, unassembled to printed circuit board
[4] Sockets, assembled to printed circuit board
[0] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one).
[4] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-5 — Performance group D

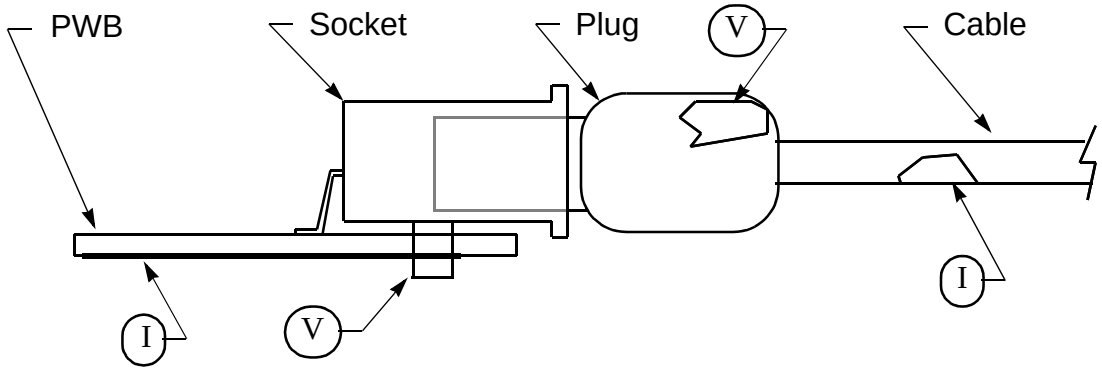
Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
D2	Continuity-housing (shell)		See figure 4-12 for measurement points	Contact 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 ±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 µs under load. No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.
G3	Cable flexing [2 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 µs during flexing.
				(d) Visual	-	No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.

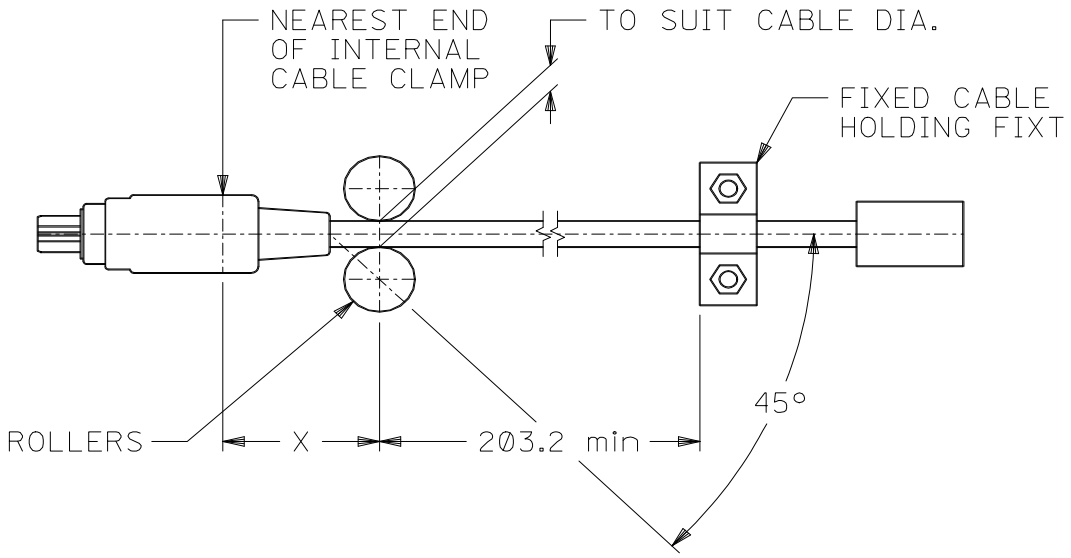


Figure 4-13 — Fixture for cable flex test

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 ± 6) Ω (differential)
Z_{TPB} = (110 ± 6) Ω (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 ± 6) Ω (common mode)
Z_{TPBCM} = (33 ± 6) Ω (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:

$V_{TPA} \leq 5.05$ ns/meter
$V_{TPB} \leq 5.05$ 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:

$V_{TPACM} \leq 5.05$ ns/meter
$V_{TPBCM} \leq 5.05$ 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:

$S \leq 400$ 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:

$X \leq -26$ 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 specifies the protocol and signal timing. It does not describe specific operation of the PHY except for behavior with respect to this interface.

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

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

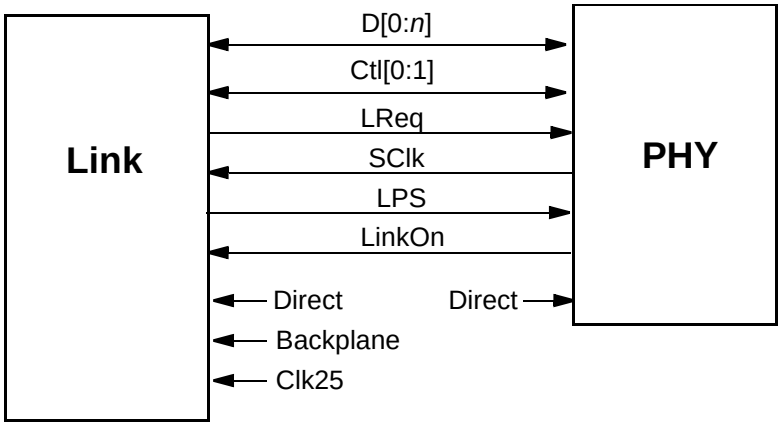


Figure 5-1 — Discrete PHY/link interface

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

Table 5-1 — PHY/link signal description

Name	Driven by	Description
D[0:*n*]	Link or PHY	Data
Ctl[0:1]	Link or PHY	Control
LReq	Link	Link request
SClk	PHY	12.288, 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	Occurrence of a link-on event.

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

Name	Driven by	Description
Direct	Neither	Set high to disable differentiator outputs for the Ctl[0:1], D[0:*n*] and LReq signals. ~~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	Meaningful only if Backplane is high. Set high to indicate a 24.576 MHz SClk; otherwise 12.288 MHz.

Data is transferred between the PHY and link on D[0:*n*]. The implemented width of D[0:*n*] depends on the maximum speed of the ~~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[0:*n*] signals shall be driven low by the device that has control of the interface.

~~The Ctl bus signals control information and is two bits wide.~~

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

The presence of a stable SClk signal generated by the PHY is necessary for the PHY/link interface to be operational. When SClk is not shown in the timing diagrams in this section, each Ctl[0:1], D[0:*n*] or LReq bit cell represents a single clock sample time. The specific timing relationships ~~clock-to-data timing relationships~~ are described in clauses 5.8.2 and 5.8.3.

The LPS signal may be used by the link to disable SClk or reset the interface, as specified in clause 5.1.

The LinkOn signal permits the PHY to indicate an interrupt to the link when either LPS is logically false or the PHY register Link_active bit is zero. The details are specified in clause 5.2.

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

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

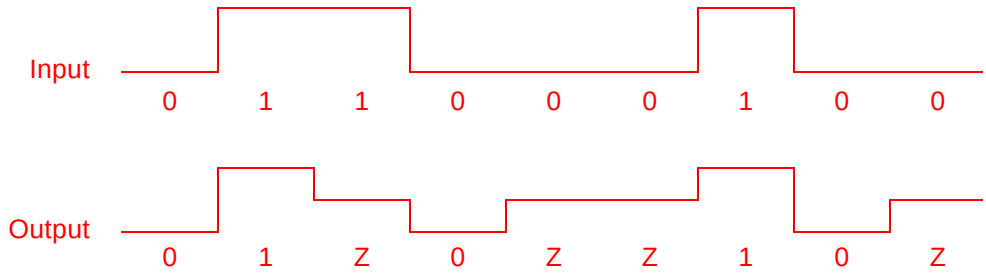


Figure 5-2 — Digital differentiator signal transformation

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

~~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 placing those signals in a high-impedance state. 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 an optional differentiator circuit can operate properly.~~

5.1 Initialization and reset

The LPS input requests the PHY to disable or enable the PHY/link interface. The output characteristics of LPS, if provided by the link, depend upon the interface mode, differentiated or undifferentiated. When the interface mode is differentiated, LPS shall be a pulsed output while logically asserted. When logically deasserted, LPS shall be driven low in either interface mode. The characteristics of LPS are specified by figure 5-3 and table 5-2.

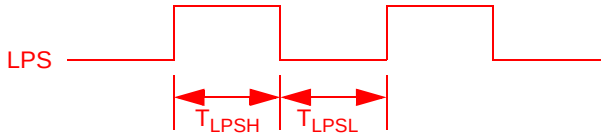


Figure 5-3 — LPS waveform when differentiated

Table 5-2 — LPS timing parameters

Parameter	Description	Unit	Minimum	Maximum
T_{LPSL}	LPS low time (when pulsed)	µs	0.09	1.00
T_{LPSH}	LPS high time (when pulsed)	µs	0.09	1.00
T_{LPS_RESET}	Time for PHY to recognize LPS logically deasserted	µs	1.2	2.75
T_{LPS_WAIT}	Time after PHY removes SClk until link may reassert LPS (when differentiated)	µs	10	

The link requests the PHY to disable and reset the interface by deasserting LPS. Within 1.2 µs after it deasserts LPS, the link shall place Ctl[0:1] and D[0:*n*] in a high-impedance state and condition LReq according to the interface mode: if undifferentiated, LReq shall be driven zero otherwise it shall be placed in a high-impedance state.

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

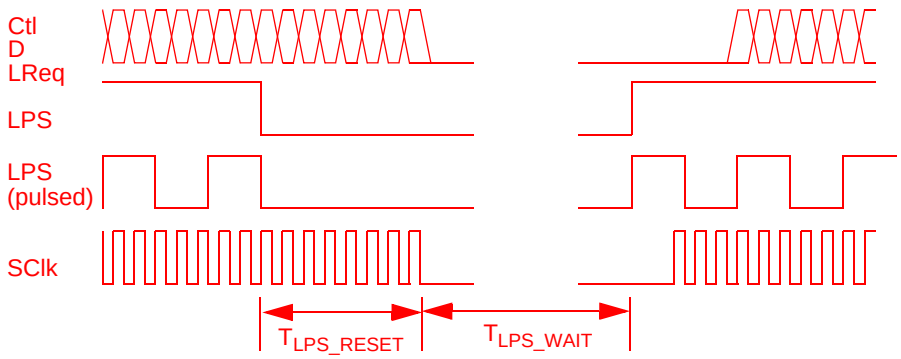


Figure 5-4 — PHY/link interface reset *via* LPS

When the PHY/link interface is reset the PHY shall cancel any outstanding bus request or register read request. Although the cancellation of bus requests may affect PHY arbitration states in ways not described in section 7, the PHY's behaviors (as observable from Serial Bus) shall be consistent with that section. For example, the PHY may have initiated arbitration in response to a bus request but reset of the PHY/link interface might cancel the request before it is granted. Appropriate PHY behavior would be either the transmission of a null packet or the removal of the arbitration request before it is granted.

The C code and state machines in section 7 describe the PHY's operation as if the interface to the link is always operational. If the PHY/link interface is reset while the link is transmitting a packet, the PHY shall behave as if the link had signaled *Idle* and terminated the packet. Similarly, any status information generated by the PHY while the interface is disabled shall be discarded and shall not cause a status transfer upon restoration of the interface.

The handshake just described resets the interface when the link deasserts LPS for a minimum of 2.75 µs. Normal operations may be restored if the link reasserts LPS. After observing LPS, the PHY shall resume SClk as soon as possible. Once SClk resumes the PHY and link shall condition their Ctl[0:1] and D[0:*n*] outputs in accordance with table 5-3.

Table 5-3 — Initialization of the PHY/link interface

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

Upon the eighth and subsequent SClk cycles the PHY shall drive Ctl[0:1] and D[0:*n*] as follows:

- if the PHY is in the idle arbitration state, it shall assert *Receive* on Ctl[0:1] while simultaneously asserting data prefix on D[0:*n*] for at least one SClk cycle;
- until the PHY indicates data prefix as described above, it shall not assert any state other than *Idle* on Ctl[0:1];
- once the PHY indicates data prefix it shall continue to do so until it is in the idle arbitration state, at which time it shall assert *Idle* on Ctl[0:1];
- no status information shall be transferred that commences part way through the status bits; and
- no partial packets shall be transferred on D[0:*n*].

The link may examine Ctl[0:1] once it has driven Ctl[0:1], D[0:*n*] and LReq to zero for one cycle subsequent to the availability of SClk. When the link simultaneously observes *Receive* on Ctl[0:1] and data prefix on D[0:*n*] and subsequently observes *Idle* on Ctl[0:1], the reset of the PHY/link interface is complete. The link shall not assert LReq until the reset is complete.

5.2 Link-on indication

The PHY LinkOn output provides a method to signal the link at times when the link is not active. The link is inactive when either the LPS signal is logically deasserted (see clause 5.1) or the PHY register Link_active bit is zero. The characteristics of the LinkOn signal, specified by table 5-4, permit the link to detect LinkOn in the absence of SClk and also permit the signal to cross an optional isolation barrier. When LinkOn is logically deasserted it shall be driven low.

Table 5-4 — LinkOn timing parameters

Description	Unit	Minimum	Maximum
Frequency	MHz	4	8
Duty cycle	%	40	60
Persistence. Time, measured from the point at which both LPS is active and Link_active is one, after which the PHY shall not signal LinkOn.	ns		500

When either LPS is logically false or the PHY register Link-active bit is zero, a PH_EVENT.*indication* of LINK_ON shall cause the assertion of LinkOn. This signal shall persist so long as the logical AND of the LPS signal and Link_active is zero.

At other times (when the link is active), a PH_EVENT.*indication* of LINK_ON shall be communicated to the link by the transfer of the link-on packet that caused the event. The PHY shall not assert LinkOn if the link is already active.

5.3 Operation

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

The Ctl bus is 2 bits wide. The encoding of these signals is shown in tables 5-5 and 5-6.

Table 5-5 — 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 5-6 — 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.

5.4 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 5-5.

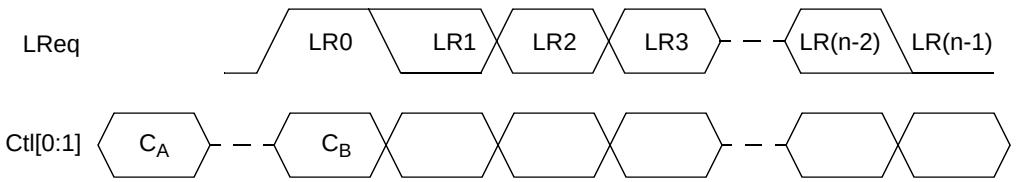


Figure 5-5 — LReq and Ctl timings

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 5-7. 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 5-7 — 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 5-12 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 5-13 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 5-8.

Table 5-8 — 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 5-12 for the encoding of this field.
4-5		Reserved.
6-9	Request Priority	Indicates priority of urgent requests. (Only used with FairReq request type.) All zeros indicates fair request. All ones is reserved (this priority is implied by a PriReq). Other values are used to indicate the priority of an urgent request.
10	Stop Bit	Indicates end of transfer. Always 0.

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

Table 5-9 — 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 5-12 for the encoding of this field.
4-7	Address	The internal PHY address to be read.
8	Stop Bit	Indicates end of transfer. Always 0.

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

Table 5-10 — 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 5-12 for the encoding of this field.
4-7	Address	The internal PHY address to be written.
8-15	Data	For a write transfer, the data to be written to the specified address.
16	Stop Bit	Indicates end of transfer. Always 0.

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

Table 5-11 — 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 5-12 for the encoding of this field.
4	Accelerate	When zero, instructs the PHY to disable arbitration accelerations. A value of one requests the PHY to enable arbitration accelerations.
5	Stop Bit	Indicates end of transfer. Always 0.

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

Table 5-12 — Request type field

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

The request speed field is encoded as shown in table 5-13. 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 5-13 — Request speed field

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

The PHY continuously monitors LReq for link requests and sets internal variables in response to the parameters of the request. These actions occur independently of the state of the PHY arbitration control; the effects upon PHY arbitration, if any, are a consequence of the values of the internal variables, as specified in clause 7.9. Table 5-14 summarizes the effects of the various link requests.

Table 5-14 — Link request effects on PHY variables

Request	PHY variables affected	Note
ImmReq, IsoReq, PriReq, FairReq	`breq`, `speed` `accelerating` (see note)	The `breq` variable is set to IMMED_REQ, ISOCH_REQ, PRIORITY_REQ or FAIR_REQ according to the type of request. The `speed` variable is set to S100, S200, *etc*. according to the encodings specified by table 5-13. 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.

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

Request	PHY variables affected	Note
WrReg	See table 6-1	The PHY updates the addressed register with the data field value from the request and sets the value of any PHY variables that correspond to register bits or fields.
AccCtrl	`accelerating`	If the Accelerate bit in the request is zero `accelerating` is cleared to FALSE; otherwise it is set 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 5-16). The link may reissue a cancelled request when the interface is subsequently idle.

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

NOTE—In order to meet timing requirements, a link may issue an isochronous request after observing *tcode* 8 in a putative cycle start packet but before verifying the CRC. If the CRC fails, the link should not transmit isochronous packet(s) but drop the isochronous request as soon as possible.

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—Although unlikely, more than one node may perceive (one correctly, the others mistakenly) that an incoming packet is intended for it and issue an immediate request before checking the CRC. The PHYs of all nodes would grab control of the bus immediately after the packet is complete. This condition would cause a temporary, localized collision of DATA_PREFIX somewhere between the PHYs intending to acknowledge while all the other PHYs on the bus would see DATA_PREFIX. This collision would appear as “ZZ” line state and would not be interpreted as a bus reset. The mistaken node(s) should drop their request(s) as soon as they check the CRC; the spurious “ZZ” line state would vanish. The only side-effect of such a collision might be the loss of the intended acknowledge packet, which would be handled by the higher layer protocol.

A bus reset causes the PHY to cancel any pending bus request.

In response to register write requests, the PHY takes the value from the data field of the transfer and updates the addressed register. For register read requests, the PHY returns the contents of the addressed register at the next opportunity through a status transfer. If the status transfer is interrupted by 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.

5.4.1 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 5-5. The rules are specified in terms of the values of the Ctl[0:1] lines during these cycles. The sample point at which the link decides whether or not to initiate a request is C_A, which is one or more SClk cycles before C_B, the cycle in which the link sends the request's start bit. The disposition of the request is determined by the value of Ctl[0:1] from C_B 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 or write 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 5-15.

Table 5-15 — 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, Priority	Idle, Status	No	No fair or priority request shall be issued until any outstanding bus request completes.
Immediate	Receive, Idle	No	Sent after *destination_ID* decode during packet reception when the link is ready to transmit an acknowledge packet. The start bit of an immediate request shall be transmitted no later than the fourth cycle subsequent to that in which Ctl[0:1]went Idle following packet reception.
Isochronous	Any	Yes	Sent during an isochronous period when the link is ready to transmit an isochronous packet. The start bit of an isochronous request shall be transmitted no later than a) the eighth cycle subsequent to that in which Ctl[0:1] went from Transmit to Idle or b) the fourth cycle subsequent to that in which Ctl[0:1] went from Receive to Idle. The link shall not issue an isochronous request if it intends to concatenate a packet after the current transmission.
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 `enab_accel` is TRUE and shall be issued once every isochronous period, as soon as possible after the local clock indicates the start of a new isochronous period.
Accelerate	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 of the link's isochronous requests (if any).

In general, the PHY behavior varies dependent on whether another Serial Bus packet is detected before it has successfully completed processing the LReq.

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

Table 5-16 — 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 enab_accel is FALSE, the request is cancelled. Otherwise, if arbitration accelerations are globally enabled for the PHY, then the request is cancelled only if the received packet is 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.
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 then monitor for the next change on Ctl[0:1]
	Grant	Request retained. Bus request LReq was previously issued, and now takes priority.	Service previous bus request, then monitor for next change on Ctl[0:1]
	Receive	Request retained.	Receive packet, then monitor for next change on Ctl [0:1]
	Status	Request is retained by the PHY until corresponding register data is returned.	If unrelated status is received or the desired status is interrupted, monitor Ctl[0:1] for desired status.
	Idle		Monitor Ctl[0:1]
Register write, Acceleration control	Any	Request completed	

The PHY shall guarantee that neither subaction nor arbitration reset gap status information is lost because of a response to a register read request. During some period prior to the anticipated detection of a gap, it may be necessary for the PHY to defer completion of a register read request in order to avoid the loss of status information.

5.4.2 Acceleration control

The ack-accelerated and fly-by arbitration enhancements specified in clause 7.9 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 link avoids this problem by selectively disabling and enabling these arbitration enhancements. The acceleration control request permits the link to disable and enable ack-accelerated and fly-by arbitration enhancements while leaving the other arbitration enhancements unaffected. The cycle master does not issue the acceleration control request.

The time period in which ~~neither~~ ack-accelerated and fly-by accelerations shall not be used extends from the time of the local cycle synchronization event until a cycle start packet is observed. During this period, the link at any node that is not the cycle master shall use the acceleration control request as follows:

a) The link shall not make a fair or priority request unless an acceleration control request with a zero Accelerate bit has been transmitted since the most recent local cycle synchronization event;

b) The link shall not use the *Hold* protocol to concatenate an asynchronous primary packet after an acknowledge packet transmitted by another node but may use the *Hold* protocol after its own *ack_pending* only to complete a concatenated transaction;

c) Upon conclusion of this time period, the link may reenable ack-accelerated and fly-by acceleration by transmitting an acceleration control request whose Accelerate bit is set to one.

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

5.5 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* 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 5-6.

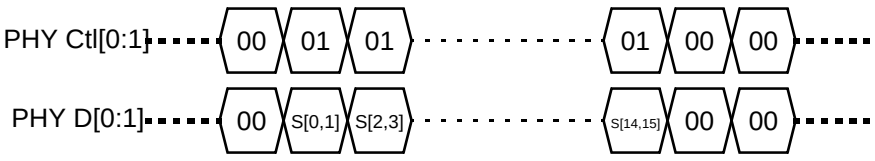


Figure 5-6 — Status timing

The structure of the status data is specified by table 5-17 below.

Table 5-17 — 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. Status transfers shall commence with S[0:1] in all cases.

5.6 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* followed by *idle* on Ctl[0:1], the link takes control of the interface by asserting *idle*, *hold* or *transmit* on Ctl[0:1] one cycle after sampling *idle* from the PHY. The link should assert *idle* for one cycle before changing the state of Ctl[0:1] to either *hold* or *transmit* but shall not assert *idle* for more than one cycle. PHY implementations shall tolerate *idle* for one cycle prior to *hold* or *transmit*. The link asserts *hold* to keep ownership of the bus while preparing data. The PHY asserts DATA_PREFIX on Serial Bus during this time. When it is ready to begin transmitting a packet, the link asserts *transmit* on Ctl[0:1] along with the first bits of the packet. After sending the last bits of the packet, the link asserts either *idle* or *hold* on Ctl[0:1] for one cycle and then it asserts *idle* for one additional cycle before placing those signals in a high-impedance state.

Whenever control of the bidirectional signals is transferred between the PHY and link, the device relinquishing control shall drive Ctl[0:1] and D[0:*n*] to logic zero levels for one clock before releasing the interface. This permits both devices to act upon registered versions of the interface signals while allowing the new owner a clock cycle in which to sample and respond. Note that when the link transfers control to the PHY without a *Hold* request, an additional clock with logic zero on the control and data signals is necessary so as not to place the signal lines in a high impedance state before the PHY takes control.

An assertion of *hold* after the last bits of a packet indicates to the PHY that the link needs to send another packet without releasing the bus. This function is used by the link to concatenate a packet after an acknowledge or to concatenate isochronous packets. With this assertion of *hold* the link simultaneously signals the speed of the next packet on the data lines, as encoded by table 5-18. Once *hold* is asserted, the PHY waits a MIN_PACKET_SEPARATION time and then asserts *grant* as before. After observing *grant* on Ctl[0:1], the link resumes control of the interface by asserting *idle*, *hold* or

transmit on Ctl[0:1]. The link should assert *idle* for one SClk cycle, but shall not assert *idle* for more than one cycle, before changing Ctl[0:1] to *hold* or *transmit*. 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 less the delay between the PHY's transmission of TX_DATA_PREFIX and its assertion of *grant* on Ctl[0:1].

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. When the link wishes to send an S100 packet after any packet of a higher speed, it shall make a separate isochronous request.

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 5-7. In the diagram, D0 through D_n are the data symbols of the packet, SP represents the speed code for the packet (encoded according to the values specified in table 5-18) and ZZ represents high impedance state. The link should assert the signals indicated by the shaded SClk cycles (this may be necessary in the presence of an isolation barrier).

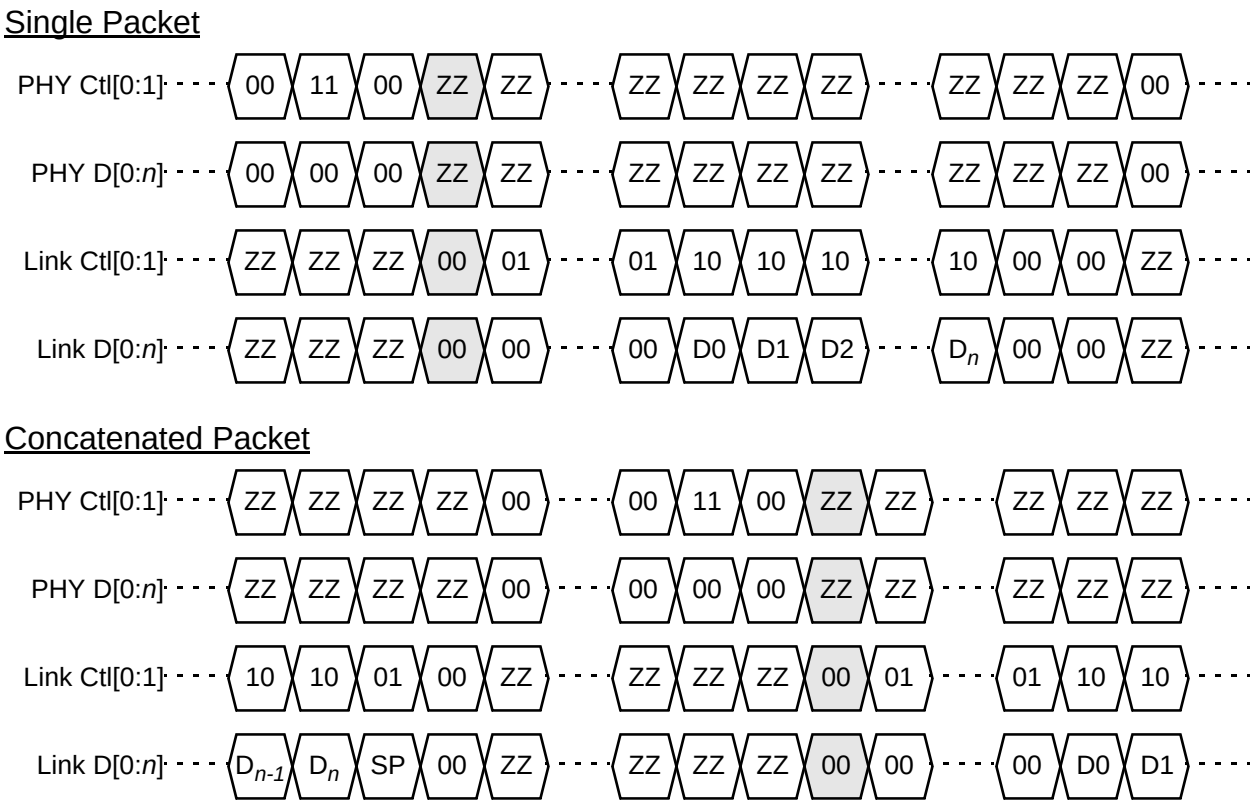


Figure 5-7 — Transmit timing

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.

5.7 Receive

Whenever the PHY sees data prefix on Serial Bus, it initiates a receive operation by asserting *receive* on Ctl[0:1] and 1's on D[0:*n*]. The PHY indicates the start of a packet by placing the speed code (encoding shown in table 5-18) on D[0:*n*], followed by the contents of the packet. The PHY holds Ctl[0:1] in *receive* until the last symbol of the packet has been transferred. The PHY indicates the end of the packet by asserting *idle* on Ctl[0:1]. Note that signaling the speed code is a PHY/link protocol and not a data symbol to be included in the calculation of the CRC.

It is possible that a PHY can see data prefix appear and then disappear on Serial Bus without seeing a packet. This is the case when a packet of a higher speed than the PHY can receive is being transmitted. In this case, the PHY will end the packet by asserting *idle* when data prefix goes away.

If the PHY is capable of a higher data rate than the link, the link detects the speed code as such and ignores the packet until it sees the *idle* state again.

The timing for the receive operation is shown in figure 5-8. In the diagram, SP refers to the speed code and D0 through D_n are the data symbols of the packet.

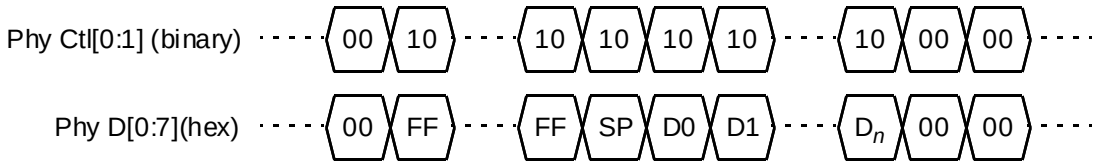


Figure 5-8 — Receive timing

The speed code for the receive operation is defined as shown in table 5-18. This is also the same speed encoding used by the link to signal speed to the PHY during concatenated packet transmission.

Table 5-18 — Speed code signaling

D[0:*n*]		Data rate
Transmitted	**Observed**	
00000000_2	$00xxxxxx_2$[a]	S100
01000000_2	$0100xxxx_2$	S200
01010000_2	01010000_2	S400
01010001_2	01010001_2	S800
01010010_2	01010010_2	S1600
01010011_2	01010011_2	S3200
11111111_2	$11xxxxxx_2$	Data prefix indication

[a] An "x" indicates ignored on receive.

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

5.8 Electrical characteristics

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

5.8.1 DC signal levels and waveforms

DC parametric attributes of the PHY/link interface signals~~, in the case of direct connection,~~ are specified by table 5-19. 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. V_{CC} is obtained from the vendor's specifications.

Table 5-19 — DC specifications for PHY/link interface

Name	Description	Conditions	Unit	Minimum	Maximum
V_{OH}	Output high voltage (undifferentiated)	I_{OH} = -1 mA ~~V_{CC} = min~~	V	2.8	
V_{OHD}	Output high voltage (differentiated)	I_{OH} = -9 mA at V_{CC} = 3 V I_{OH} = -11 mA at V_{CC} = 4.5 V	V	V_{CC} - 0.4	
V_{OL}	Output low voltage (undifferentiated)	I_{OL} = 1 mA ~~V_{CC} = min~~	V		0.4
V_{OLD}	Output low voltage (differentiated)	I_{OL} = 9 mA at V_{CC} = 3 V I_{OL} = 11 mA at V_{CC} = 4.5 V	V		0.4
V_{IH}	Input high voltage (Direct, Backplane and Clk25 and undifferentiated Ctl, D, LReq and SClk)		V	2.6	V_{CC}[a]+10%
V_{IL}	Input low voltage (undifferentiated)		V		0.7
V_{LIH}	Input high voltage (LinkOn and LPS)		V		V_{LREF} + 1[b]
V_{LIL}	Input low voltage (LinkOn and LPS)		V	V_{LREF} + 0.2[b]	
V_{IT+}	Hysteresis input rising threshold (differentiated)		V	V_{REF} + 0.3	V_{REF} + 0.9[c]
V_{IT-}	Hysteresis input falling threshold (differentiated)		V	V_{REF} - 0.9[c]	V_{REF} - 0.3
V_{REF}	Reference voltage		V	V_{CC}/2.25	V_{CC}/2
V_{LREF}	Reference voltage[d] (LinkOn and LPS inputs)		V	0.5	1.6
~~I_{IH}~~	~~Input high current~~	~~V_{IN} = V_{CC}~~ ~~V_{CC} = max~~	~~µA~~		~~40~~
~~I_{IL}~~	~~Input low current~~	~~V_{IN} = GND~~ ~~V_{CC} = max~~	~~µA~~		~~600~~
C_{IN}	Input capacitance		pF		7.5

[a] Refers to driving device's power supply

[b] The LinkOn and LPS receiver parameters are based on a swing of 2.4 V for the received signal. Links which only depend on receiving the initial edge of LinkOn may be capable of operating with less constrained values.

c. When designing a device capable of both undifferentiated and differentiated operation, V_{IH} and V_{IL} effectively constrain these V_{IT+} and V_{IT-} values to V_{REF} + 0.8 V and V_{REF} - 0.8 V, respectively.

d. For a particular application, the nominl bias point V_{LREF} should be chosen in conjunction with the receiver parameters so that a loss of power by the transmitting device is perceived as zero by the receiving device.

5.8.2 AC timing

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

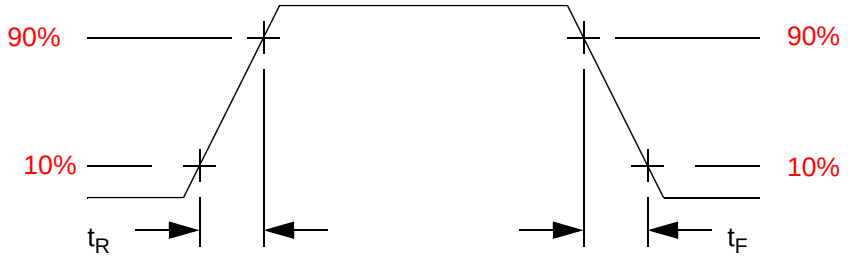


Figure 5-9 — Signal levels for rise and fall times

Other signal characteristics of the PHY/link interface are specified by table 5-20. If an isolation barrier is implemented it shall cause neither delay nor skew in excess of the values specified. AC measurements shall be taken from the 1.575 V level of SClk to the input or output Ctl[0:1], D[0:*n*] or LReq levels and shall assume an output load of 10 pF.

Table 5-20 — AC timing parameters

Name	Description	Unit	Minimum	Maximum
	SClk frequency	MHz	49.152 ± 100 ppm	
	SClk duty cycle	%	40	60
t_R	Rise time ~~from 0.8 V to 2.35 V~~	ns	0.7	2.4
t_F	Fall time ~~from 2.35 V to 0.8 V~~	ns	0.7	2.4
idel	Delay through isolation barrier	ns	0	2
	Skew through isolation barrier	ns	0	0.5
	~~Hysteresis input rising threshold~~	~~V~~	~~Vcc/2 + 0.2~~	~~Vcc/2 + 1.3~~
	~~Hysteresis input falling threshold~~	~~V~~	~~V_{cc}/2 - 1.3~~	~~Vcc/2 - 0.2~~
	Isolation barrier recovery time	µs	0	10

Figures 5-10 and 5-11 below illustrate the transfer waveforms as observed at the PHY. A PHY shall implement values for tpd1, tpd2 and tpd3 within the limits specified in table 5-21 and shall not depend upon values for tpsu and tph greater than the minimums specified.

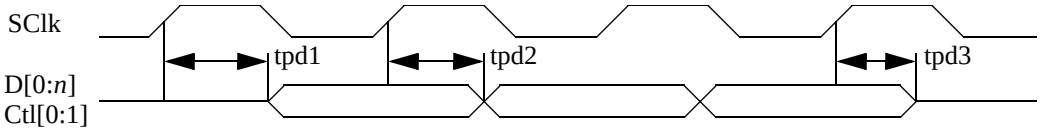


Figure 5-10 — PHY to link transfer waveform at the PHY

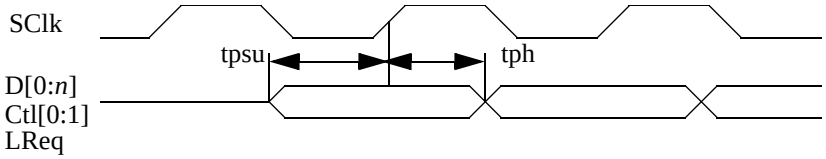


Figure 5-11 — Link to PHY transfer waveform at the PHY

The values for the timing parameters illustrated above are specified below.

Table 5-21 — AC timing parameters at the PHY

Name	Description	Unit	Minimum	Maximum
tpd1	Delay time, SClk input high to initial instance of D[0:*n*] and Ctl[0:1] outputs valid	ns	0.5	13.5
tpd2	Delay time, SClk input high to subsequent instance(s) of D[0:*n*] and Ctl[0:1] outputs valid	ns	0.5	13.5
tpd3	Delay time, SClk input high to D[0:*n*]] and Ctl[0:1] invalid (high-impedance)	ns	0.5	13.5
tpsu	Setup time D[0:*n*], Ctl[0:1] and LReq inputs before SClk	ns	6	
tph	Hold time D[0:*n*], Ctl[0:1] and LReq inputs after SClk	ns	0	

Figures 5-12 and 5-13 below illustrate the transfer waveforms as observed at the link. A link shall implement values for tld1, tld2 and tld3 within the limits specified in table 5-22 and shall not depend upon values for tlsu and tlh greater than the minimums specified.

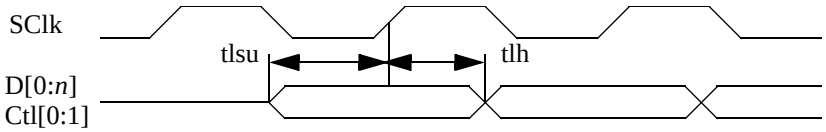


Figure 5-12 — PHY to link transfer waveform at the link

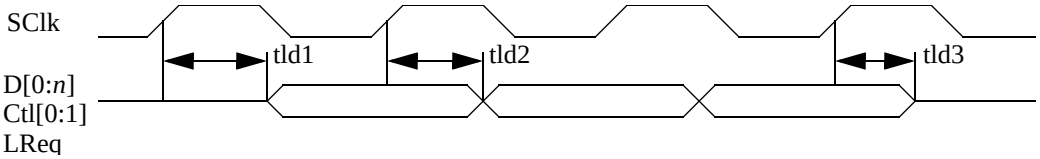


Figure 5-13 — Link to PHY transfer waveform at the link

The values for the timing parameters illustrated above are specified below.

Table 5-22 — AC timing parameters at the link

Name	Description	Unit	Minimum	Maximum
tld1	Delay time, SClk input high to initial instance of D[0:*n*], Ctl[0:1] and LReq outputs valid	ns	1	10
tld2	Delay time, SClk input high to subsequent instance(s) of D[0:*n*], Ctl[0:1] and LReq outputs valid	ns	1	10
tld3	Delay time, SClk input high to D[0:*n*], Ctl[0:1] and LReq invalid (high-impedance)	ns	1	10
tlsu	Setup time, D[0:*n*] and Ctl[0:1] inputs before SClk	ns	6	
tlh	Hold time, D[0:*n*] and Ctl[0:1] inputs after SClk	ns	0	

.

5.8.3 AC timing (informative)

The protocol of this interface is designed such that all inputs and outputs at this interface can be registered immediately before or after the I/O pad and buffer. No state transitions need be made that depend directly on the chip inputs; chip outputs can come directly from registers without combinational delay or additional loading. This configuration provides generous margins on setup and hold time.

In the direction from the PHY to the link, timing follows normal source-clocked signal conventions. A 0.5 ns allowance is made for skew through an (optional) isolation barrier.

In the direction from the link to the PHY, the data is timed at the PHY in reference to SClk, whose frequency allows a nominal budget of 20 ns for delay, inclusive of the PHY input setup time. Possible sources of delay are an isolation barrier or internal SClk delay at the link caused by a clock tree. Figure 5-14 illustrates the relationship of these delays. Note that the maximum round-trip delay of 14 ns (calculated as $tdrt1_{max} = idel_{max} + tld1_{max} + idel_{max}$) provides generous delays for both the link and the PHY. The link, after the receipt of SClk, has 10ns to assert valid data while at the PHY the minimum input setup for the subsequent SClk cycle is 6 ns (calculated as $tpsu_{min} = 20\ ns - tdrt1_{max}$). Also note that

the minimum round-trip delay until the next change in data of 21 ns (calculated as $tdrt2_{min}$ = 20 ns + $idel_{min}$ + $tld2_{min}$ + $idel_{min}$) limits the hold time at the PHY to 1 ns (calculated as tlh_{min} = $tdrt2_{min}$ - 20 ns); the hold time is further reduced to zero to provide a guard band of 1 ns.

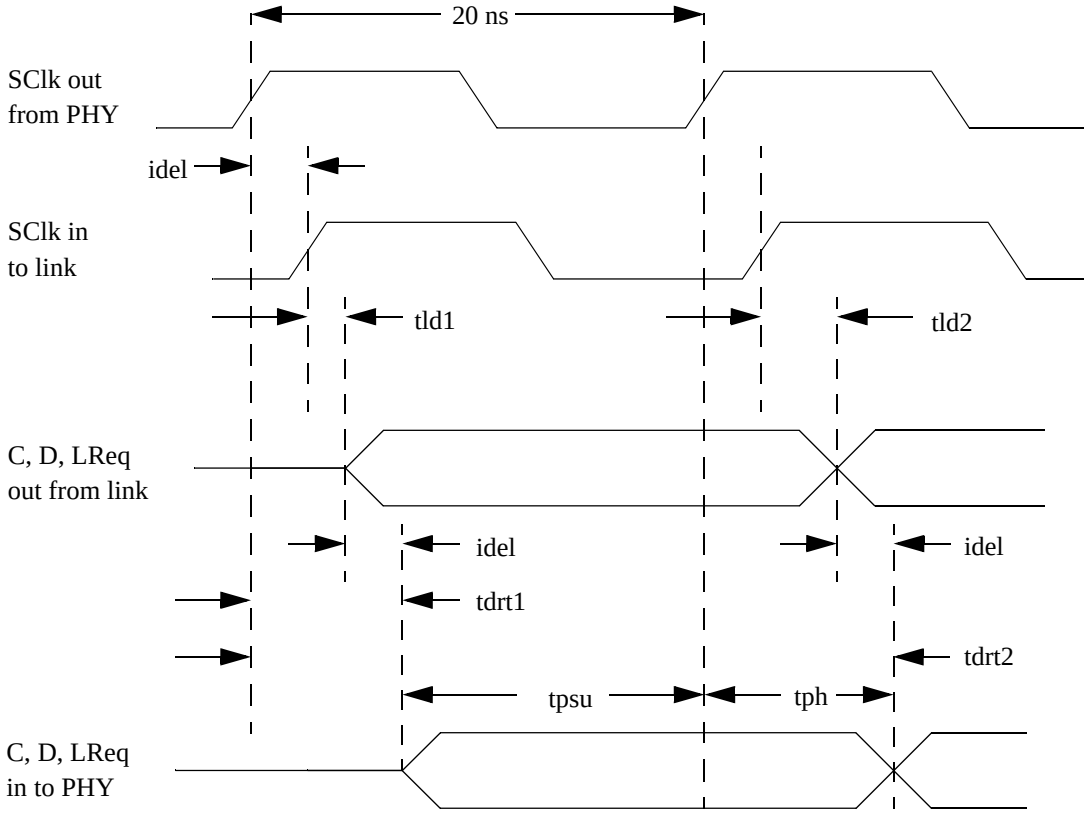


Figure 5-14 — Link to PHY delay timing

The values for the delays illustrated above are given by table 5-23 below.

Table 5-23 — Link to PHY delay timing parameters

Name	Description	Unit	Minimum	Maximum
tdrt1	Round-trip delay from SClk output at the PHY to valid Ctl[0:1], D:[0:7] and LReq at the PHY	ns	1	14
tdrt2	Round-trip delay from SClk output at the PHY to changed or invalid Ctl[0:1], D:[0:7] and LReq at the PHY	ns	21	34

5.8.4 Isolation barrier (informative)

The example circuits shown in this clause demonstrate how to achieve galvanic isolation between a discrete PHY and link by means of a capacitive isolation barrier. For applications that require isolation, other methods may be used. When capacitive isolation is used between the PHY and the link, the grounds of both devices must be coupled as shown in figure 5-15. The details of this ground coupling are omitted from figures 5-16 through 5-20.

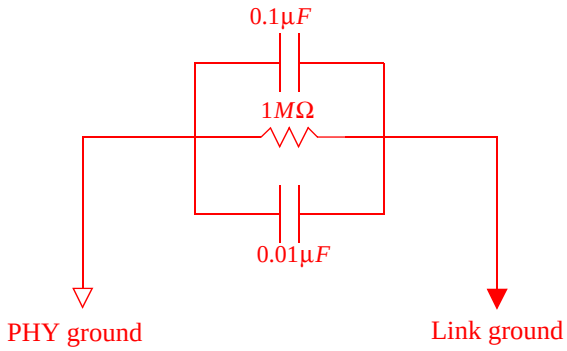


Figure 5-15 — Ground coupling circuit example

The example circuite that follow illustrate different requirements of the various signals of the PHY/link interface.

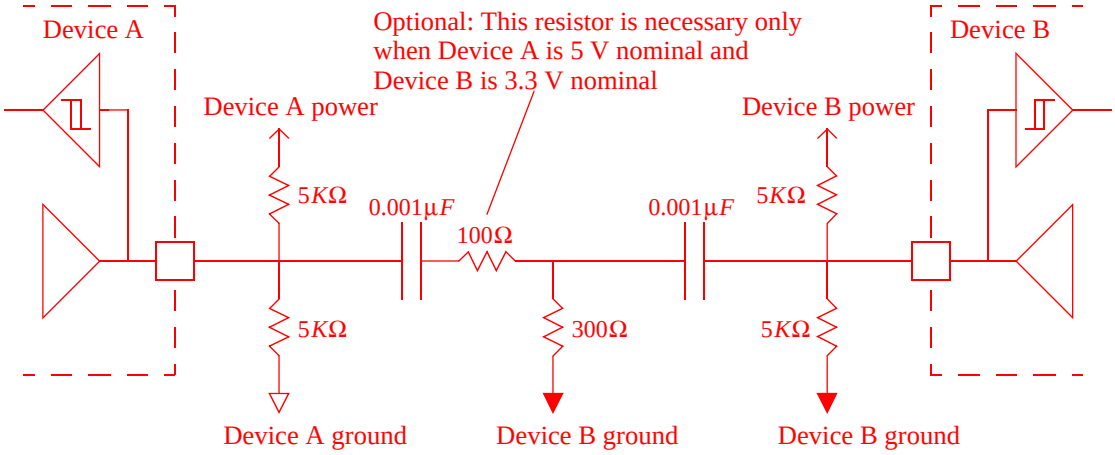


Figure 5-16 — Capacitive isolation barrier circuit example for Ctl[0:1] and D[0:*n*]

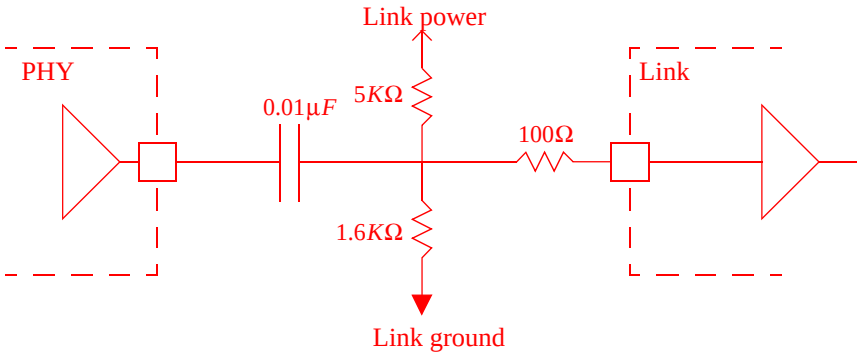


Figure 5-17 — Capacitive isolation barrier circuit example for LinkOn

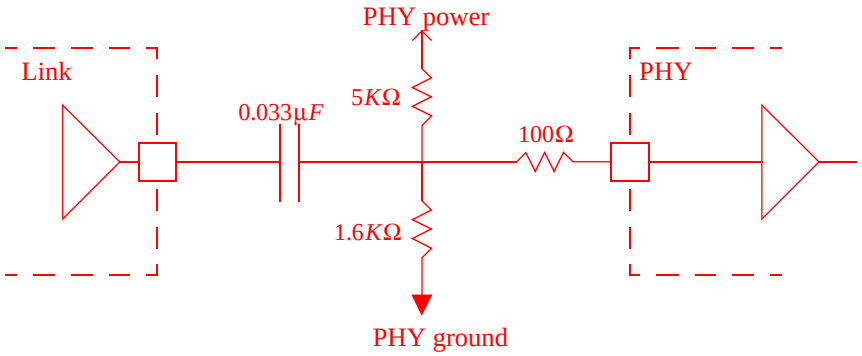


Figure 5-18 — Capacitive isolation barrier circuit example for LPS

NOTE—In figures 5-17 and 5-18, the values of the resistors between signal and ground or signal and power should be chosen to suit the implemented value of V_{LREF}. The values shown are appropriate when V_{LREF} is nominally 0.8 V.

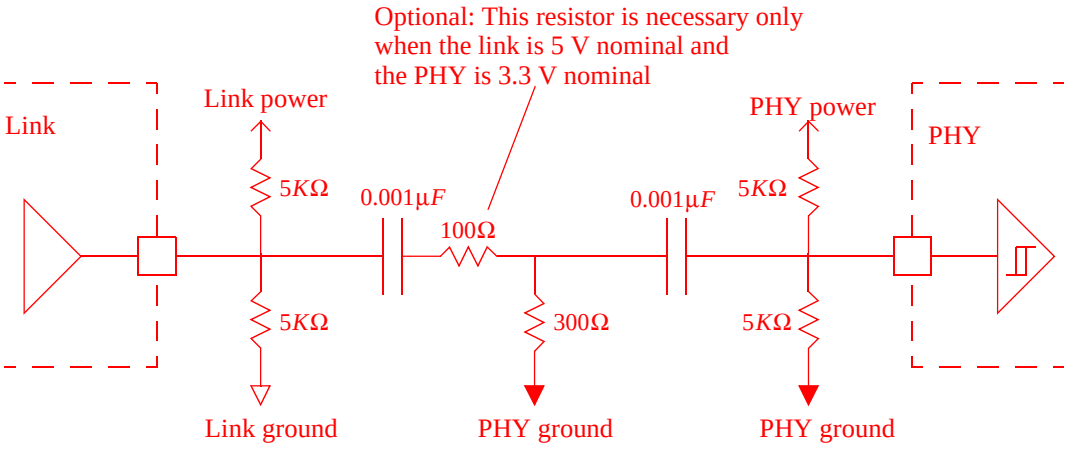


Figure 5-19 — Capacitive isolation barrier circuit example for LReq

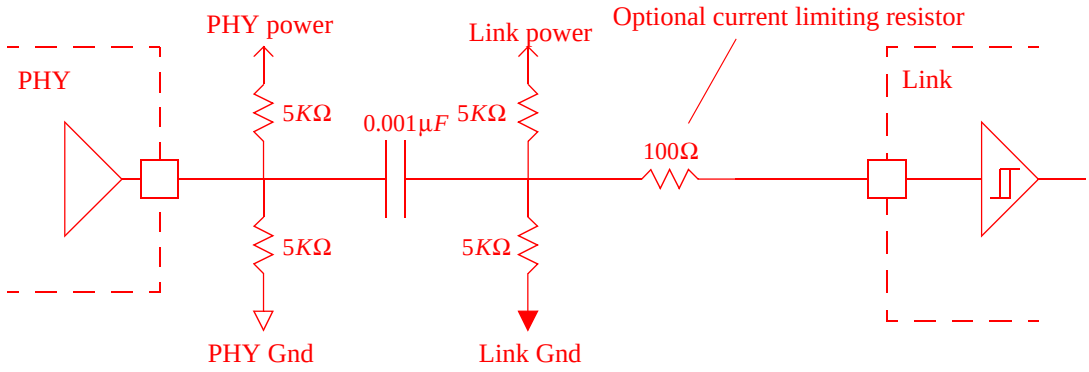


Figure 5-20 — Capacitive isolation barrier circuit example for SClk

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.

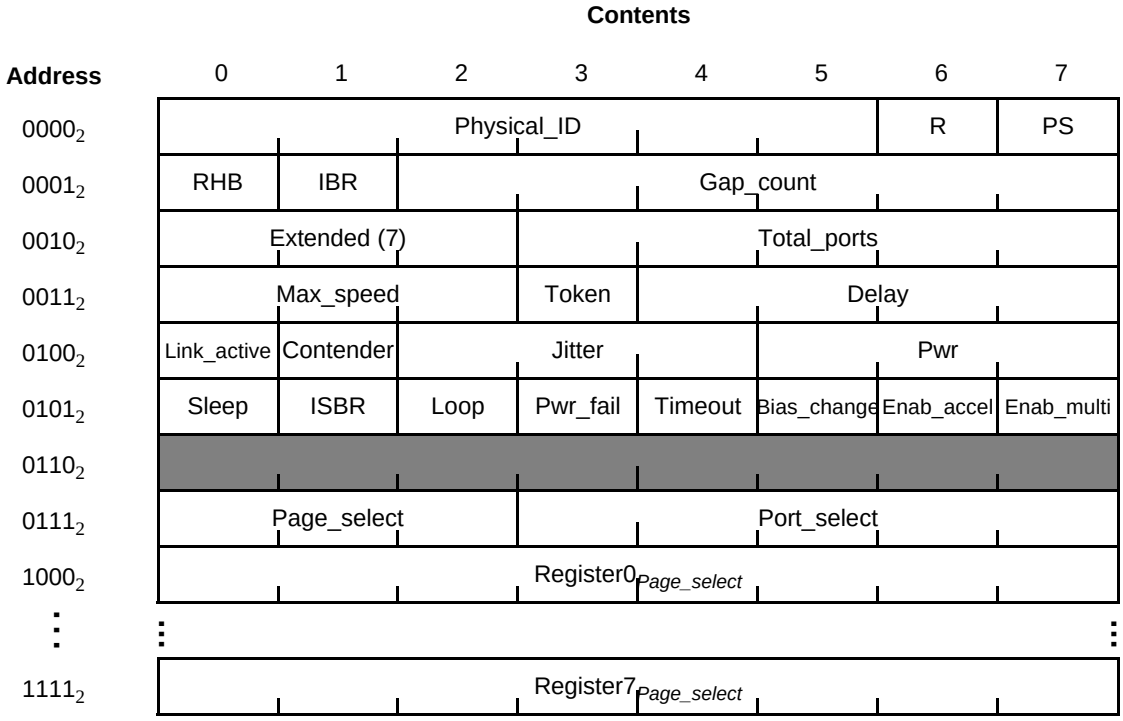


Figure 6-1 — Extended PHY register map for the cable environment

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. A value of 63 indicates a malconfigured bus; the link shall not transmit any packets.
R	1	r		When set to one, indicates that this node is the root.
PS	1	r		Cable power 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.

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

Field	Size	Type	Power reset value	Description
IBR	1	rw	0	Initiate bus reset. When set to one, instructs the PHY to set `ibr` TRUE and `reset_time` to `RESET_TIME`. These values in turn cause the PHY to initiate a bus reset without arbitration; the reset signal is asserted for 166 µs. This bit is self-clearing.
Gap_count	6	rw	$3F_{16}$	Used to 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 speed(s) this PHY supports: 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
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 *enab_token* bit).
Delay	4	r	vendor-dependent	Worst-case repeater delay, expressed as 144 + (delay * 20) ns.
Link_active	1	rw	See description	Link active. Cleared or set by software to control the value of the L bit transmitted in the node's self-ID packet 0, which shall be the logical AND of this bit and LPS active. If hardware implementation-dependent means are not available to configure the power reset value of the Link_active bit, the power reset value shall be one.
Contender	1	rw	See description	Cleared or set by software to control the value of the C bit transmitted in the self-ID packet. If hardware implementation-dependent means are not available to configure the power reset value of this bit, the power reset value shall be zero.
Jitter	3	r	vendor-dependent	The difference between the fastest and slowest repeater data delay, expressed as (jitter + 1) * 20 ns.
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 instructs the PHY to set `isbr` TRUE and `reset_time` to `SHORT_RESET_TIME`. These values in turn cause the PHY to arbitrate and issue a short bus reset. This bit is self-clearing.
Loop	1	rw	0	Loop detect. A write of one to this bit clears it to zero.
Pwr_fail	1	rw	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	See description	Enable arbitration acceleration. When set to one, the PHY shall use the enhancements specified in clause 7.9. For a discrete PHY, the power reset value of the Enab_accel bit shall be zero unless hardware implementation-dependent means are available to configure the power reset value. Integrated PHY/link implementations may implement Enab_accel as a read-only bit.

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

Field	Size	Type	Power reset value	Description
Enab_multi	1	rw	See description	Enable multi-speed packet concatenation. When set to one, the link shall signal the speed of all packets to the PHY. For a discrete PHY, the power reset value of the Enab_multi bit shall be zero unless hardware implementation-dependent means are available to configure the power reset value. Integrated PHY/link implementations may implement Enab_multi as a read-only bit.
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 *Page_select* presents *per* port information, this field selects which port's registers are accessible through the window at PHY register addresses 1000_2 through 1111_2, inclusive. Ports are numbered monotonically starting at zero, p0.

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 5.5. 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 1000_2 through 1111_2, 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. Other pages are reserved.

The Port Status page is used to access configuration and status information for each of the PHY's ports. The port is selected by writing zero to *Page_select* and the desired port number to *Port_select* in the PHY register at address 0111_2. The format of the Port Status page is illustrated by figure 6-2 below; reserved fields are shown shaded in grey.

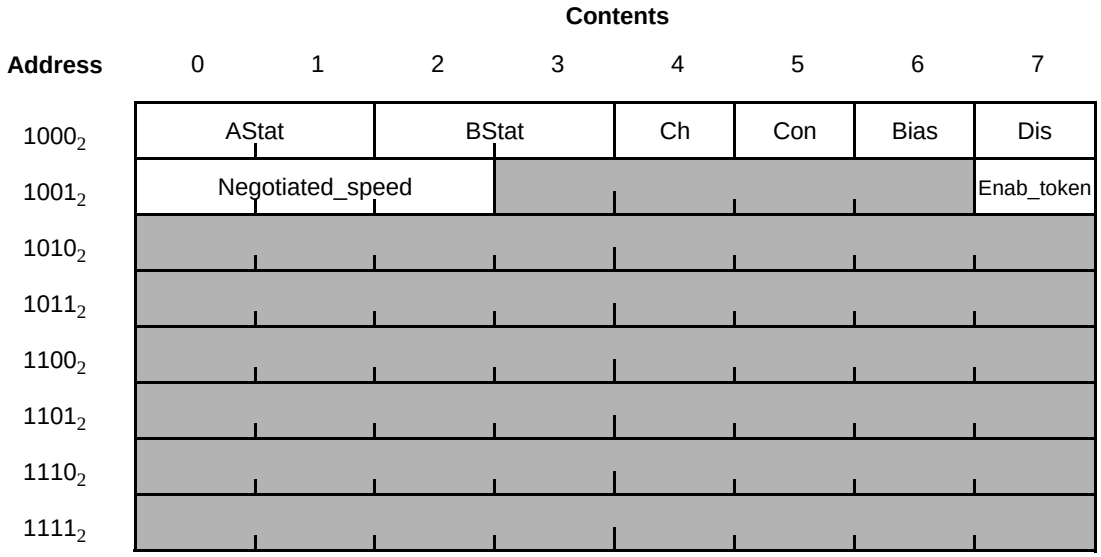


Figure 6-2 — PHY register page 0: Port Status page

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 7-19).
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.
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 Max_speed field in PHY register 3.
Enab_token	1	rw	Enable token-style arbitration. When set to one, the enhancements specified in clause 7.9 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 0111_2. The format of the Vendor Identification page is illustrated by figure 6-3 below; reserved fields are shown shaded in grey.

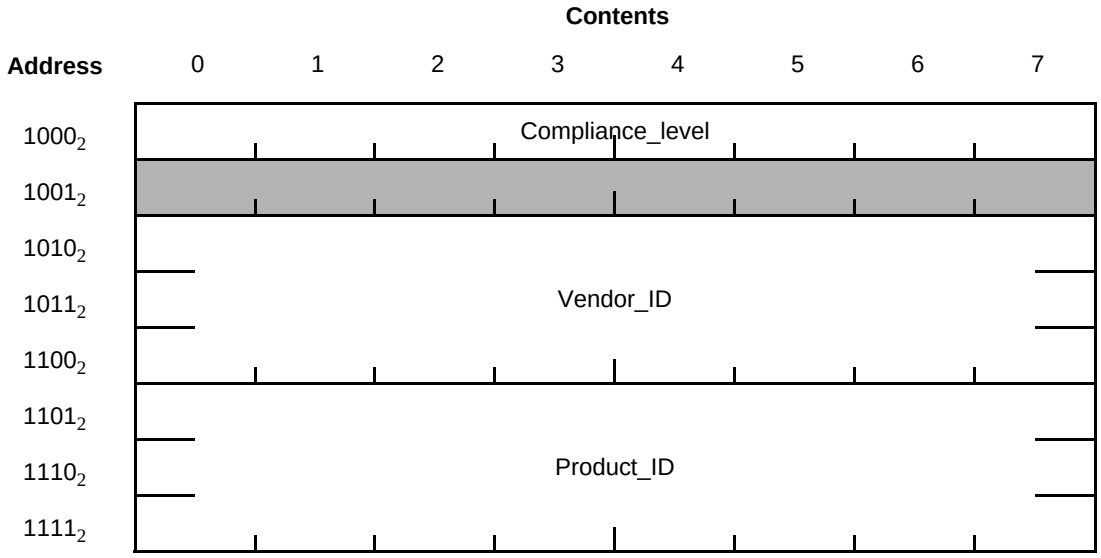


Figure 6-3 — PHY register page 1: Vendor Identification page

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
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 0111_2. The PHY vendor shall determine the meaning of *Port_select* when *Page_select* equals seven. PHY vendors may implement and specify the format of up to eight vendor-dependent registers, at addresses 1000_2 through 1111_2, inclusive.

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.

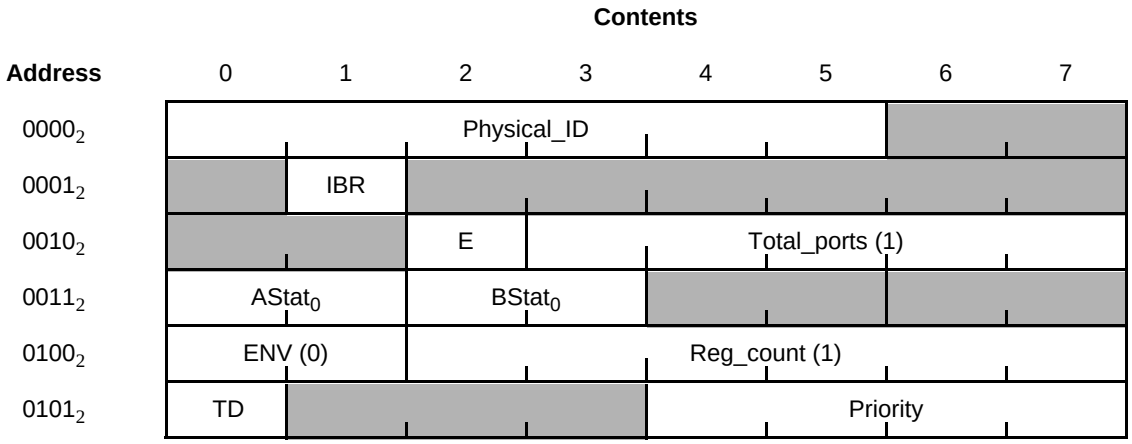


Figure 6-4 — PHY register map for the backplane environment

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.
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 pinging). The ability to transmit a “ping” packet to a 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 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 7.4.1). There may be other facilities, for example in a node’s configuration ROM, that identify the power characteristics of a node in more detail than is possible in the self-ID packet; these are beyond the scope of this standard.

Serial Bus may be unpowered or powered; in the latter case, there may be more than one power source. The possibility of multiple sources requires that 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.

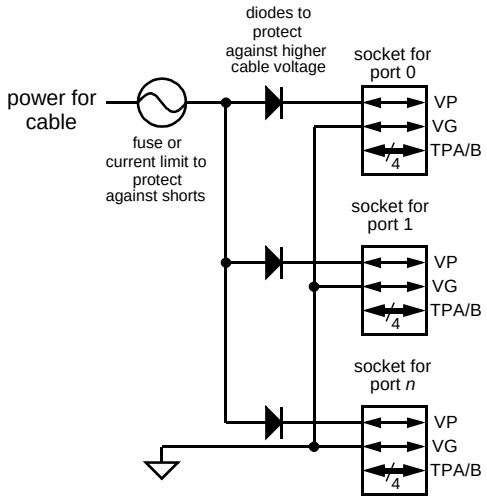


Figure 7-1 — Node power interface for POWER_CLASS one, two or three

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 enables 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 les than or equal to (I_{load} / 1.5 A) * 100 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 Data signal rise and fall times

Table 7-2 below replaces table 4-22 in IEEE Std 1394-1995. The output rise and fall times for data signals are measured from 10% to 90% at the connector and are dependent on the data rate. This supplement adds minimum rise and fall times to the specification.

Table 7-2 — Output rise and fall times

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

NOTE—The differential received signal amplitude specification in table 4-13 of IEEE Std 1394-1995 is not a receiver sensitivity specification for PHY inputs. Designers should consider factors such as the worst-case received waveform (*e.g.*, slow rise or fall times near the signal thresholds) and board design characteristics when choosing receiver sensitivity.

7.4 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.

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

The cable physical layer packet types are

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

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

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

7.4.1 Self-ID packets

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

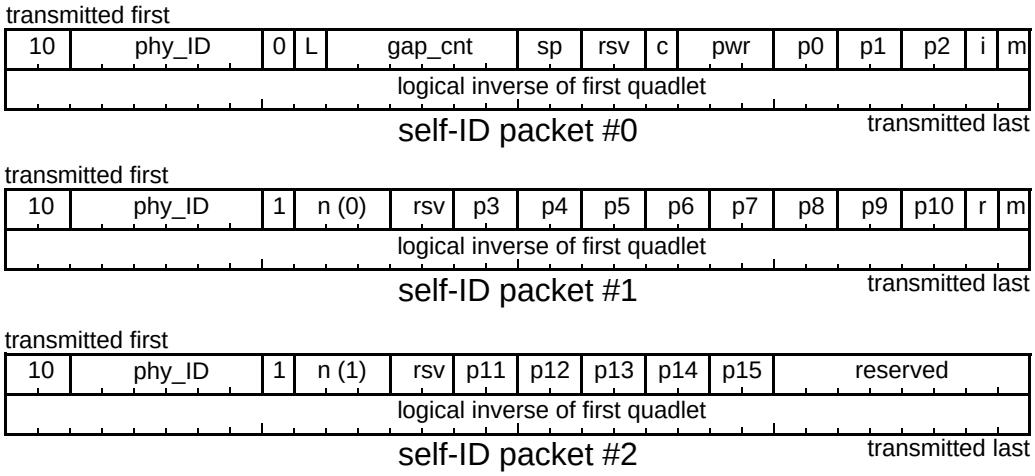


Figure 7-2 — Self-ID packet formats

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

NOTE—IEEE Std 1394-1995 defines self-ID packet formats that permit a maximum of four self-ID packets to be generated by a PHY. Although this supplement defines only three self-ID packets, link designers should accommodate the reception of up to 252 self-ID packets during the self-identify process. Such a link design both supports IEEE Std 1394-1995 and permits future Serial Bus standards to define an additional self-ID packet without adverse impact on contemporary links.

Table 7-3 — Self-ID packet fields

Field	Derived from	Comment
phy_ID	`physical_ID`	Physical node identifier of the sender of this packet
L	`LPS` `Link_active`	If set, this node has active link and transaction layers. In discrete PHY implementations, this shall be the logical AND of Link_active and LPS active.
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 PHY register 3
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.
pwr	`POWER_CLASS`	Power consumption and source characteristics: 000_2 Node does not need power and does not repeat power. 001_2 Node is self-powered and provides a minimum of 15 W to the bus. 010_2 Node is self-powered and provides a minimum of 30 W to the bus. 011_2 Node is self-powered and provides a minimum of 45 W to the bus. 100_2 Node may be powered from the bus and is using up to 3 W. No additional power is needed to enable the link[a]. 101_2 Reserved for future standardization. 110_2 Node is powered from the bus and is using up to 3 W. An additional 3 W is needed to enable the link[a]. 111_2 Node is powered from the bus and is using up to 3 W. An additional 7 W is needed to enable the link[a].
p0 … p15	`NPORT`, `child[`*n*`]`, `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.e.*, it started sending a bus_reset signal before it received one)[b] (Optional. If not implemented, this bit shall be zero)
m	`more_packets`	If set, another self-ID packet for this node will immediately follow (*i.e.*, if this bit is set and the next self-ID packet received has a different phy_ID, then a self-ID packet was lost)
n		Extended self-ID packet sequence number
r, rsv		Reserved for future standardization, set to zeros

a. The link is enabled by the link-on PHY packet described in clause 7.4.2; 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.

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

7.4.2 Link-on packet

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 more information.)

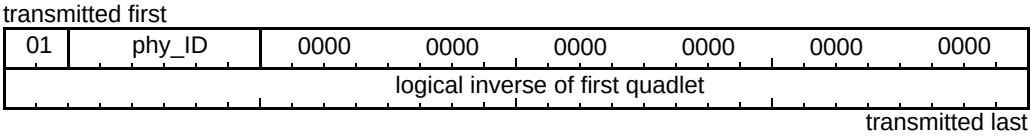


Figure 7-3 — Link-on packet format

Table 7-4 — Link-on packet fields

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

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

7.4.3 PHY configuration packet

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

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

Both of these actions shall be effected for all nodes (including the originator) by means of the PHY configuration packet shown below. The PH_CONTROL.*request* service affects only the local node and is not recommended for changes to either gap_count or force_root. The procedures for using this PHY packet are described in section 8 of IEEE Std 1394-1995 and in clause 9.20 of this supplement.

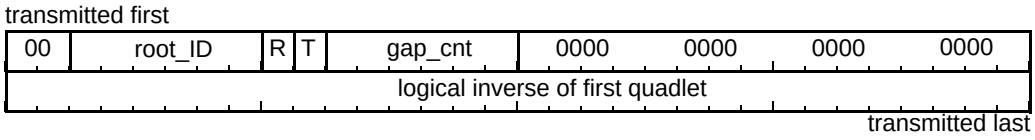


Figure 7-4 — PHY configuration packet format

Table 7-5 — 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 7-5 — 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 remains valid through the next bus reset. A second bus reset without an intervening PHY configuration packet resets gap_count to 63, as described in `reset_start_actions()` in clause 7.9.3.1.2)

7.4.4 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.

7.4.5 Ping packet

The reception of the cable PHY packet shown in figure 7-5 shall cause the node identified by phy_ID to transmit self-ID packet(s) that reflect the current configuration and status of the PHY. Because of other actions, such as the receipt of a PHY configuration packet, the self-ID packet transmitted may differ from that of the most recent self-identify process.

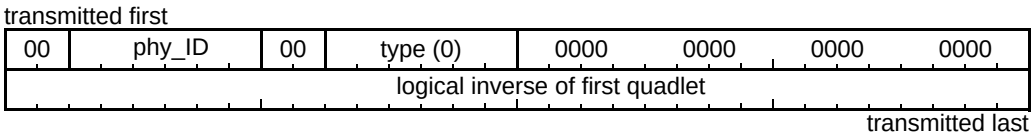


Figure 7-5 — Ping packet format

Table 7-6 — 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)

7.5 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 7-7 — 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.

7.6 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 7-8 — Cable PHY timing constants

Timing constant	Minimum	Maximum	Comment
ACK_RESPONSE_TIME	40 ns	240 ns	Idle time measured at the cable connector from the end of DATA_END that follows a primary packet to the start of DATA_PREFIX that precedes the acknowledge packet.Time permitted a PHY to respond to a ping packet (see clause 7.4.5), measured at the connector from the end of DATA_END to the start of DATA_PREFIX for the first self-ID 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.
CONCATENATION_PREFIX_TIME	160 ns		At a transmitting port, the time between the end of clocked data and the start of speed signaling time ~~(when present)~~ for the concatenated packet that follows.
CONNECT_TIMEOUT	336.0 ms	341.3 ms	Connection debounce time
DATA_PREFIX_HOLD	40 ns		At a transmitting port, the time between the end of speed signalling (when present) and the start of clocked data.
DATA_PREFIX_TIME			This timing constant is no longer defined; see DATA_PREFIX_HOLD and MIN_DATA_PREFIX.
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 µs	400 µs	Maximum time in any state (before a bus reset shall be initiated) except the idle state or a state that exits after an explicit time-out.
MAX_BUS_HOLD		1.63 µs	Maximum time a node may transmit TX_DATA_PREFIX between concatenated packets. ~~the request acknowledge and data packets 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 µs	The maximum time that clocked data may be transmitted continuously. If this limit is exceeded, unpredictable behavior may result.
MIN_DATA_PREFIX	140 ns		The total time an originating port transmits a TX_DATA_PREFIX signal prior to clocked data. This constant is not applicable to data prefix that precedes a concatenated packet (see CONCATENATION_PREFIX_TIME).
PHY_DELAY	60 ns	See PHY registers	Best-case repeater data delay has a fixed minimum.
~~PING_RESPONSE_TIME~~	~~50 ns~~	~~240 ns~~	~~Time permitted a PHY to respond to a ping packet (see clause 7.4.5), measured at the connector from the end of DATA_END to the start of DATA_PREFIX for the first self-ID packet.~~

Table 7-8 — Cable PHY timing constants (Continued)

Timing constant	Minimum	Maximum	Comment
RESET_DETECT	80.0 ms	85.3 ms	Time for a connected node to confirm a reset signal
RESET_WAIT	0.16 µs		Reset wait delta time. (~16/BASE_RATE)
ROOT_CONTEND_FAST	0.76 µs	0.80 µs	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. (~80/BASE_RATE)
ROOT_CONTEND_SLOW	1.60 µs	1.64 µs	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. (~160/BASE_RATE)
SHORT_RESET_TIME	1.30 µs	1.40 µs	Short reset hold time. (~128/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.

7.7 Node variables

Each node's PHY has a set of variables that are referenced in the C code and state machines in clause 7.9. The values of these variables may be affected by writes to PHY registers, the transmission or reception of PHY configuration packets or by arbitration state actions—including bus reset. A reset of the PHY/link interface affects none of these variables. The definitions in table 7-9 entirely replace clause 4.3.8 of IEEE Std 1394-1995, "Node variables."

Table 7-9 — Node variables

Variable name	Power reset value	Comment
accelerating	TRUE	Set TRUE or FALSE by accelerate or decelerate requests issued by the link *via* LReq (see clause 5.4) and used by the arbitration state machines. See also enab_accel below.
arb_enable	—	TRUE if the PHY may arbitrate on behalf of a fair request within the current fairness interval.
cable_power_active	—	TRUE if cable power is within normal operating range (see clause 7.2).
enab_accel	FALSE	Globally enables or disables all PHY accelerations specified by clause 7.9. 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.

7.8 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. A reset of the PHY/link interface affects none of these variables. The definitions in table 7-10 entirely replace clause 4.3.9 of IEEE Std 1394-1995, "Port variables."

Table 7-10 — 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.

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

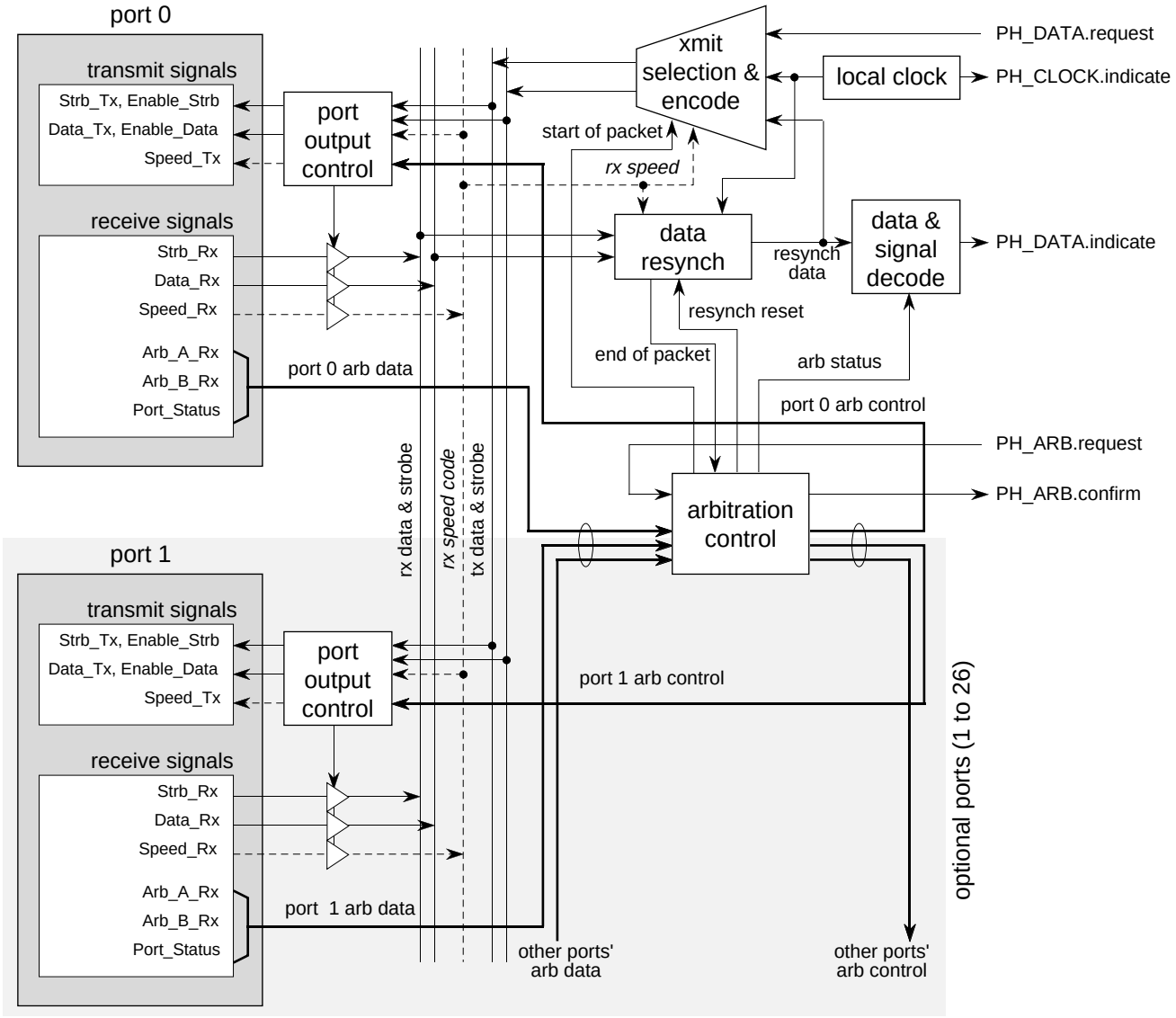


Figure 7-6 — Cable physical layer architecture

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 7.9.3

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 7-9, 7-10, 7-11 and 7-12.

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

The C language code and state machines are not normative descriptions of implementations; they are normative descriptions of externally apparent PHY behaviors. Different implementations are possible. In particular, the C language code and state machines do not contain provisions to enforce the LReq rules specified by clause 5.4.1; if the link issues bus requests that do not follow the rules the PHY behavior is unspecified.

Table 7-11 — Cable PHY code definitions

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

boolean ack;                        // Set if last packet observed was exactly 8 bits
boolean 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[NPORT];        // Filtered and latched receive speed for indicated port
void portT(int port_number, portData txData);   // Transmit txData on indicated port
void portTspeed (int port_number, speedCode speed); // Set transmit speed on indicated port
boolean random_bool();              // Returns a random TRUE or FALSE value
int 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
boolean rx_S200, rx_S400;           // Outputs from speed signal comparators
speedCode rx_speed, tx_speed;       // Current packet speeds
```


Table 7-12 — 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 :2;
            quadlet c:1;            // Isochronous resource manager contender
            quadlet pwr:3;          // Power class
            quadlet p0:2;           // Port 0 connection status
            quadlet p1:2;           // Port 1 connection status
            quadlet p2:2;           // Port 2 connection status
            quadlet i:1;            // Initiated reset
            quadlet m:1;            // More self-ID packets...
         };
         struct {           // Subsequent self-ID packets
            quadlet :8;
            quadlet ext:1;          // Nonzero for second and subsequent self-ID packets
            quadlet n:3;            // Sequence number
            quadlet :2;
            quadlet pa:2;           // Port connection status...
            quadlet pb:2;
            quadlet pc:2;
            quadlet pd:2;
            quadlet pe:2;
            quadlet pf:2;
            quadlet pg:2;
            quadlet ph:2;
            quadlet :2;
         };
         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;
```

7.9.1 Speed signal sampling and filtering

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

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

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

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

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

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

Table 7-13 — Digital speed filtering

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

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

NOTE—The C code above is not intended to preclude other implementations. For example, some implementation may require that the outputs remain stable for three consecutive samples. Others might implement different sampling algorithms for S200 and S400. The behavior of any implementation shall conform to the signal and timing requirements of IEEE Std 1394-1995 and this supplement.

7.9.2 Data transmission and reception

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

7.9.2.1 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 7-14 — 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(PH_CLOCK.indication);    // Wait for clock
   if (bit == tx_data)                 // If no change in data
      tx_strobe = ~tx_strobe;          // Invert strobe
   tx_data = bit;
   for (i = 0; i < NPORT; i++)
      if (connected[i] && 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 7-15 — Start data transmit actions

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

   for (i = 0; i < NPORT; i++) {
      if (!connected[i])
         speed_OK[i] = FALSE;
      else {
         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++)
      if (connected[i])
         portTspeed(i, S100);          // Go back to normal signal levels
   wait_time(DATA_PREFIX_HOLD);        // 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 7-16 — 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 (connected[i] && i != receive_port)
            portT(i, TX_DATA_PREFIX);             // ...and the last dribble bit
      wait_time(CONCATENATION_PREFIX_TIME);       // Speed signal after this time
   } else if (ending_status == DATA_END) {
      for (i = 0; i < NPORT; i++)
         if (connected[i] && 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.~~

7.9.2.2 Data reception and repeat

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

Table 7-17 — 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);
         speed_signalled = TRUE;
         signal(SPEED_SIGNAL_RECEIVED);      // Notify start_rx_packet
      }
      new_signal = tpSignals();              // Get signal
```


Table 7-17 — Data reception and repeat actions (Sheet 2 of 2)

```
      if (new_signal == IDLE)
         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_strobe != old_strobe) || (new_data != old_data)) {
                                             // Either data or strobe changed
            FIFO[fifo_wr_ptr] = new_data;    // Put data in FIFO
            fifo_wr_ptr = ++fifo_wr_ptr % FIFO_DEPTH; // Advance or wrap FIFO pointer
            signal(DATA_STARTED);            // Signal rx_bit to start
         }
         old_strobe = new_strobe;
         old_data = new_data;
      }
   }
}

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

void rx_bit(dataBit *rx_data, boolean *end_of_data) {
   int i;

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

Starting data reception requires initializing the data resynchronizer and 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 7-18 — 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
   portT(receive_port, IDLE);                // Turn off grant, get ready to receive
   for (i = 0; i < NPORT; i++)
      if (connected[i] && i != receive_port)
         portT(i, TX_DATA_PREFIX);           // Send data prefix out repeat ports
   wait_event(SPEED_SIGNAL_RECEIVED | DATA_STARTED | IDLE_DETECTED);
   tx_speed = rx_speed;                      // Get speed of packet to repeat
   if (rx_speed == S100)
      rx_dribble_bits = 2;                   // Need for FIFO empty test
   else
      rx_dribble_bits = (rx_speed == S200) ? 4 : 8;
   if (speed_signalled) {                    // Repeat the speed signal...
      for (i = 0; i < NPORT; i++)
         if (connected[i] && i != receive_port) {
            speed_OK[i] = (tx_speed <= max_peer_speed[i]);
            if (speed_OK[i])
               portTspeed(i, tx_speed);      // Receiver can accept, send speed intentions
         }
      wait_time(SPEED_SIGNAL_LENGTH);
      for (i = 0; i < NPORT; i++)
```

Table 7-18 — Start data reception and repeat actions (Sheet 2 of 2)

```
         if (connected[i] && i != receive_port)
            portTspeed(i, S100);          // Go back to normal signal levels
      wait_time(DATA_PREFIX_HOLD);     // Finish data prefix
      wait_event(DATA_STARTED | IDLE_DETECTED); // Wait for decoder to start
   }
   speed_signalled = FALSE;                  // Reset for each packet
   for (i = 0; i < 2 * rx_dribble_bits - 1; i++)   // Buffer enough bits to allow for variation
      wait_event(PH_CLOCK.indication);       // in clock frequencies (same value as for dribble
                                             // bits) and for the dribble bits themselves
}
```

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

NOTE—The description of the FIFO buffer in tables 7-15 and 7-16 assumes that the PHY strips dribble bits from incoming packets and regenerates them for repeated packets. Alternate implementations, *e.g.*, one which repeats dribble bits, may be valid so long as no dribble bits are transferred to the link.

7.9.3 Arbitration

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

Some of these arbitration classes may be enhanced as defined by this 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 phases[1] 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.

7.9.3.1 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.

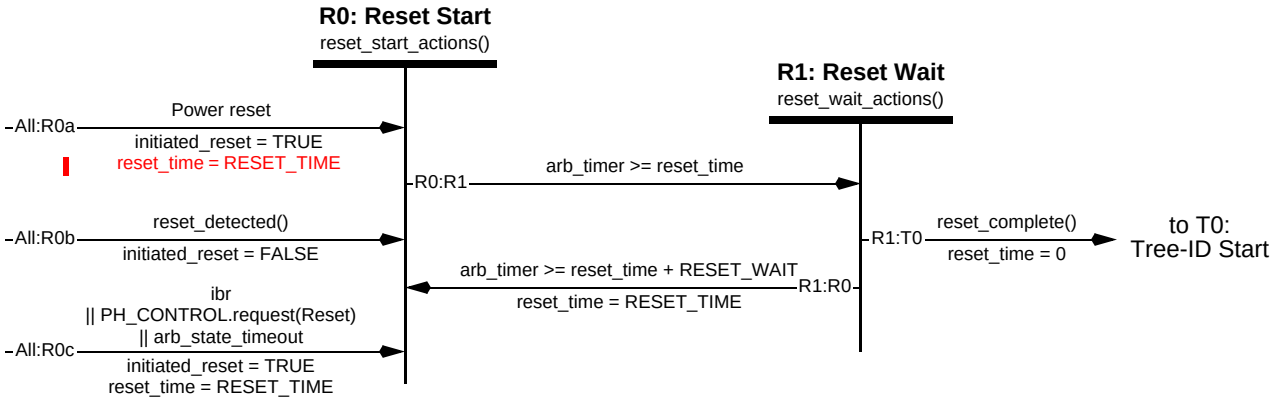


Figure 7-7 — Bus reset state machine

7.9.3.1.1 Bus reset state machine notes

Transition All:R0a. This is the entry point to the bus reset process if the PHY experiences a power reset. On power reset, PHY register values and internal variables are set as specified in this 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.

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

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).

7.9.3.1.2 Bus reset actions and conditions

Table 7-19 — 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);
   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);
            }
```


Table 7-19 — Bus reset actions and conditions (Sheet 2 of 2)

```
         } else if (connected[i]) {
            reset_time = (PHY_state == RX) ? SHORT_RESET_TIME : RESET_TIME;
            return(TRUE);
         }
   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
}
```

7.9.3.2 Self identify

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

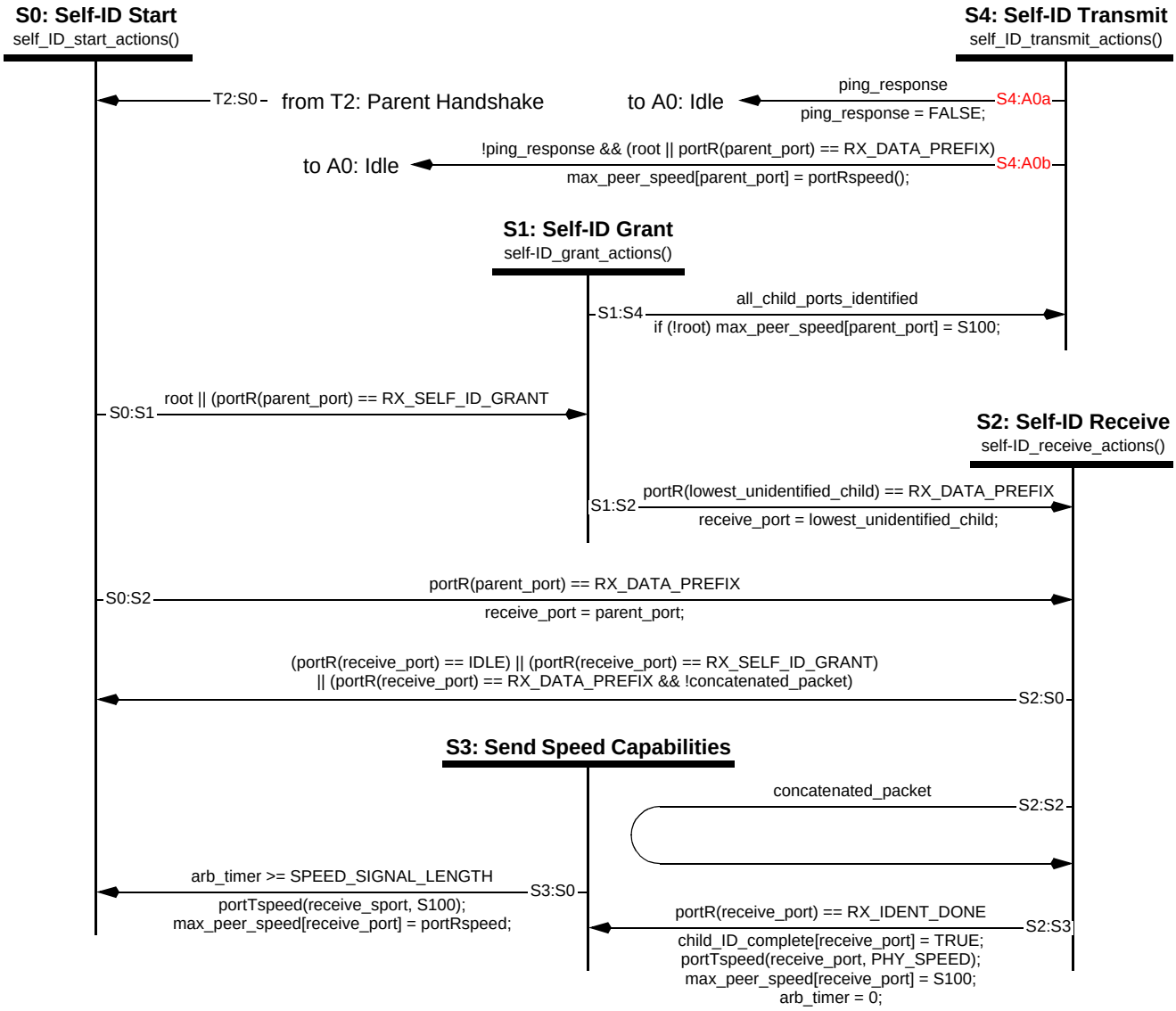


Figure 7-8 — Self-ID state machine

7.9.3.2.1 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. The parent PHY must also monitor the received speed signal whenever RX_IDENT_DONE is received from the child. Because of resynchronization delays in repeating the packet, the parent PHY may not complete retransmission of the packet data and data end signal for up to 144 ns after the start of the RX_IDENT_DONE signal. Since the child sends its speed signal for no more than 120 ns from the start of the RX_IDENT_DONE signal, the parent could miss the speed signal from the child if it entered S3 before completing a speed signal sample.

Transition S2:S0. When the receive port goes IDLE (ZZ), gets a RX_SELF_ID_GRANT (0Z) or observes RX_DATA_PREFIX (10) for a unconcatenated packet it enters the Self-ID Start state to continue the self-ID process for the next child. The last case guards against a possible failure to observe IDLE.

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

Transition S2:S3. If the PHY gets an RX_IDENT_DONE (Z1) signal from the receiving port, it flags that port as identified and starts sending the speed capabilities signal. It also starts the speed signaling timer and sets the port speed to the S100 rate.

State S3: Send Speed Capabilities. If a node is capable of sending data at a higher rate that S100, it transmits on the receiving child port its speed capability signals as defined in clause 4.2.2.3 of IEEE Std 1394-1995 for a fixed duration SPEED_SIGNAL_LENGTH. The parent PHY must also monitor the received speed-signal whenever RX_IDENT_DONE is received from the child.

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

State S4: Self-ID Transmit. At this point, all child ports have been flagged as identified, so the PHY can now send its own self-ID packet (see clause 7.4) 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) while simultaneously transmitting 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 highest indicated speed is recorded as the speed capability of 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.

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

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

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.

7.9.3.2.2 Self-ID actions and conditions

Table 7-20 — 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

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_unidentifed_child))
         portT(i, TX_GRANT);  // Send grant to lowest unidentified child (if any)
      else if (connected[i])
         portT(i, TX_DATA_PREFIX);// Otherwise, tell others to prepare for packet
}

void self_ID_receive_actions() {
   int i;

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

   receive_port = NPORT;            // Indicate that we are transmitting (no port has this number)
```


Table 7-20 — Self ID actions and conditions (Sheet 2 of 3)

```
   start_tx_packet(S100);           // Send data prefix and 98.304 Mbit/sec speed code
   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 = LPS && Link_active;   // 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);
         breq = NO_REQ;         // Cancel pending requests (only fair and priority possible here)
      } 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
         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
   }
}
```

Table 7-20 — Self ID actions and conditions (Sheet 3 of 3)

```
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
   }
}
```

7.9.3.3 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).

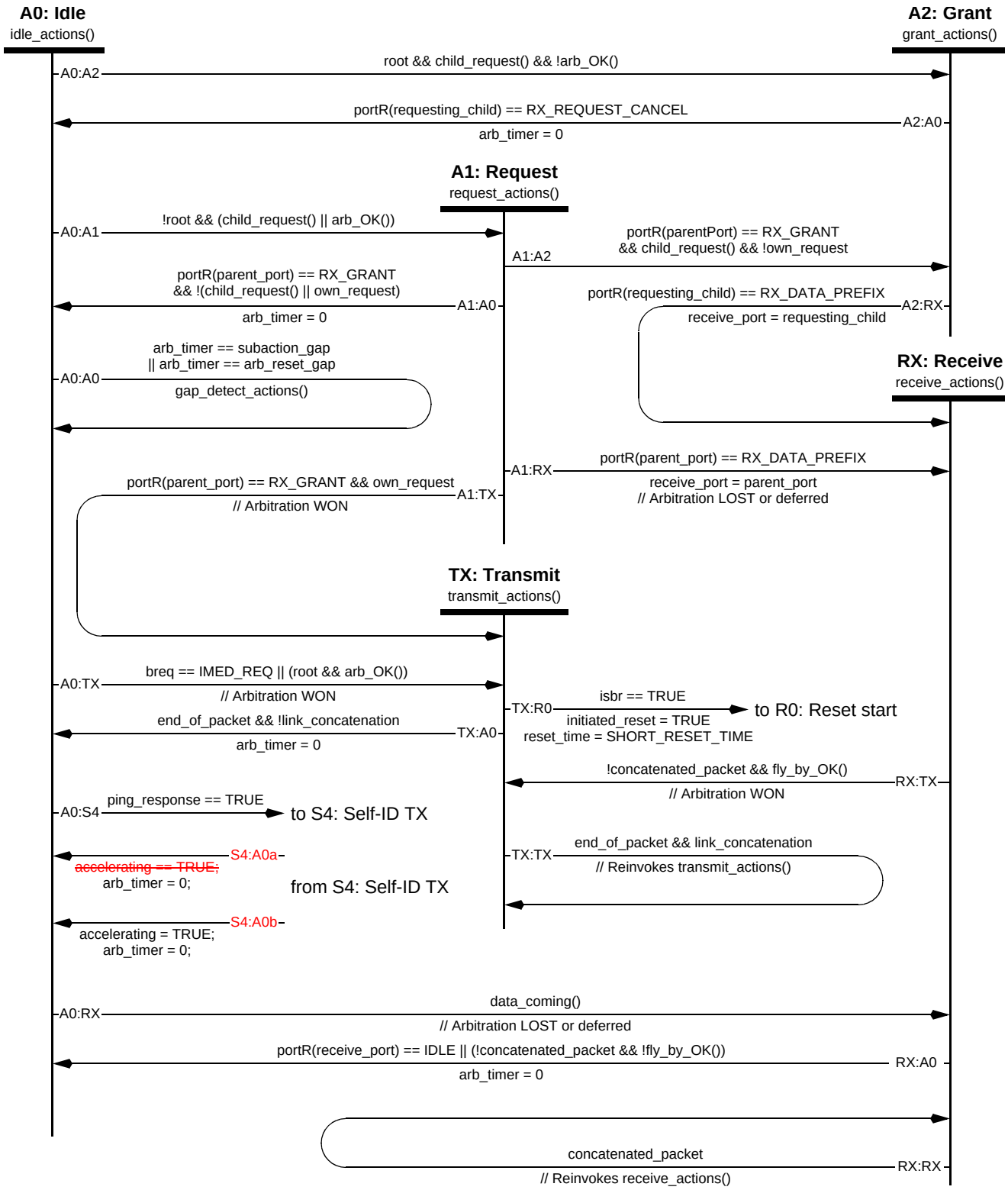


Figure 7-9 — Cable arbitration state machine

7.9.3.3.1 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 notifies the node controller. The detection of an arbitration reset gap marks the end of a fairness interval; the PHY sets the arbitration enable flag.

Transition A0:A1. If the PHY has a queued request (other than an immediate request) from its own link or receives an RX_REQUEST signal (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 no 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.

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

State A1: Request. At this point, the PHY sends a TX_REQUEST signal (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. Outstanding fair and priority requests are cancelled—immediately if arbitration enhancements are globally enabled, otherwise by the receipt of any packet other than an acknowledgement—and the link will have to reissue them later. Note that the packet received could be a PHY packet (self-ID, link-on or PHY configuration), acknowledge, or normal data packet. PHY configuration and link-on packets are interpreted by the PHY, as well as being passed on to the link layer.

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

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

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

State TX: Transmit. Unless an arbitrated (short) bus reset has been requested, the transmission of a packet starts by the node sending a TX_DATA_PREFIX and speed signal as described in 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.

Transition TX:TX. The link is using the *Hold* protocol to send a concatenated packet. Remain in the transmit state and restart the transmit process for the next packet.

7.9.3.3.2 Normal arbitration actions and conditions

Table 7-21 — 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++)
```

Table 7-21 — Normal arbitration actions and conditions (Sheet 2 of 3)

```
      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;

   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 for whole bus?
        PH_EVENT.indication(BUS_RESET_COMPLETE);
        bus_initialize_active = FALSE;
     }
  }
}

void idle_actions() {
   int i;

   rx_speed = S100;                 // Default in anticipation of no explicit receive speed code
   for (i = 0; i < NPORT; i++)      // Turn off all transmitters
      portT(i, IDLE);
}


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 + arb_delay)
      async_arb_OK = enab_accel && accelerating && ack;
```


Table 7-21 — Normal arbitration actions and conditions (Sheet 3 of 3)

```
   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;
   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
}
```

7.9.3.3.3 Receive actions and conditions

Table 7-22 — 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);
```

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

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

      case RX_DATA_END:
         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;
      }
   }
}
```

7.9.3.3.4 Transmit actions and conditions

Table 7-23 — 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;
```


Table 7-23 — Transmit actions and conditions (Sheet 2 of 2)

```
   tx_speed = speed;             // Assume speed has been set correctly...
                                 // (from PH_ARB.request or concatenated packet speed code)
   receive_port = NPORT;         // Impossible port number ==> PHY transmitting
   start_tx_packet(tx_speed);    // Send data prefix & speed signal
   if (isbr)                     // 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
      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 needs to be
            break;      // guaranteed by 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);    // Used elsewhere to (conditionally) accelerate
   if (bit_count == 64) {     // We have transmitted a PHY packet
      for (i = 0; i < 32; i++)          // Check PHY packet for good format
         if (tx_phy_pkt.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;
            }
         }
   }
}
```

7.10 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 placed in a high-impedance state;

— 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.

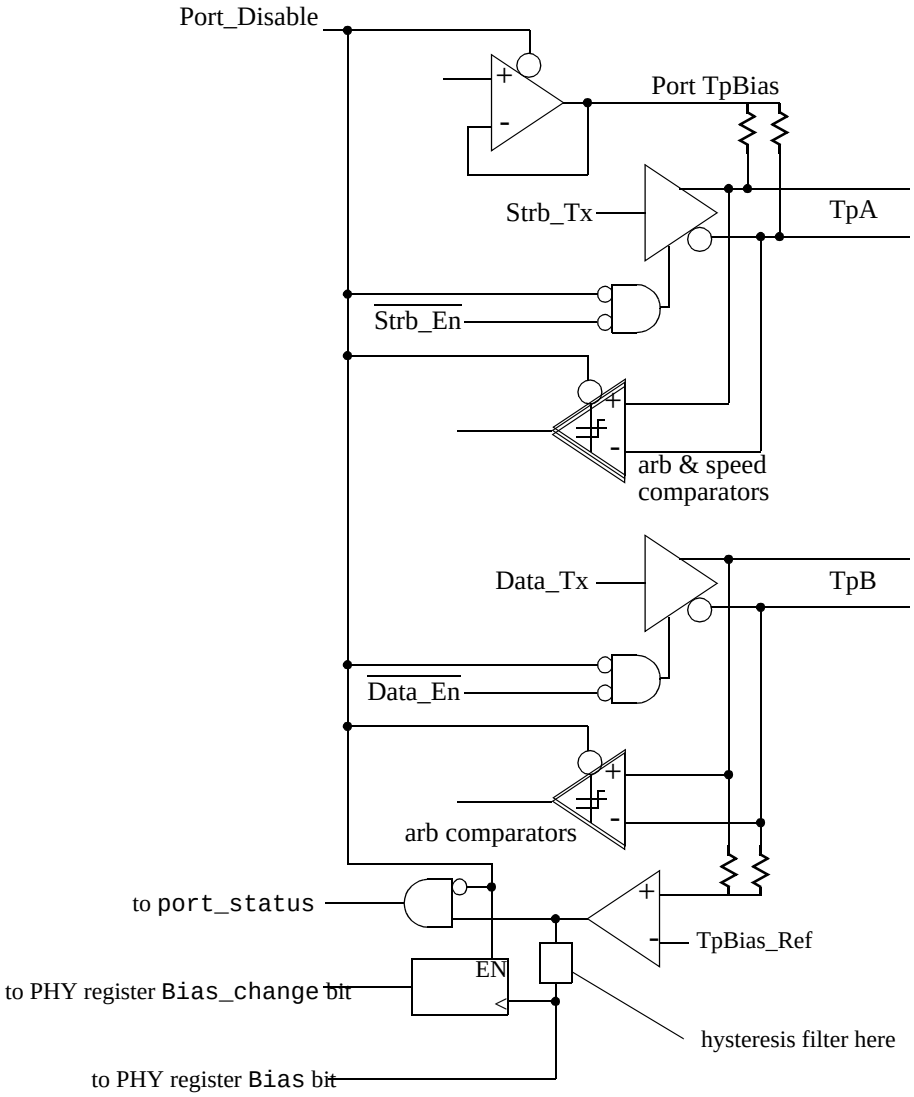


Figure 7-10 — Port disable logic

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 previously defined, packets with a transaction code of A_{16} were called isochronous data block packets[1] and are subject to the following restrictions:

— An isochronous stream packet is transmitted only during the isochronous period. The isochronous period is controlled by the cycle master, which signals the start of the period with a cycle start packet. The period ends when a subaction gap is observed, which happens after all isochronous talkers have had a chance to transmit;
— Two resources, bandwidth and a channel number, are allocated from the isochronous resource manager registers BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE, respectively; and
— For a given channel number, no more than one talker may transmit an isochronous stream 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 during the asynchronous period, subject to the same arbitration requirements, including fairness, as other request subactions;
— 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.

[1] Throughout this supplement, primary packets with a transaction code of A_{16} are referred to as stream packets; the arbitration mode determines whether they are asynnchronous or isochronous stream packets. The name “isochronous stream packet” is equivalent to the IEEE Std 1394-1995 name “isochronous data block packet.”

8.1 Asynchronous stream packet format

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

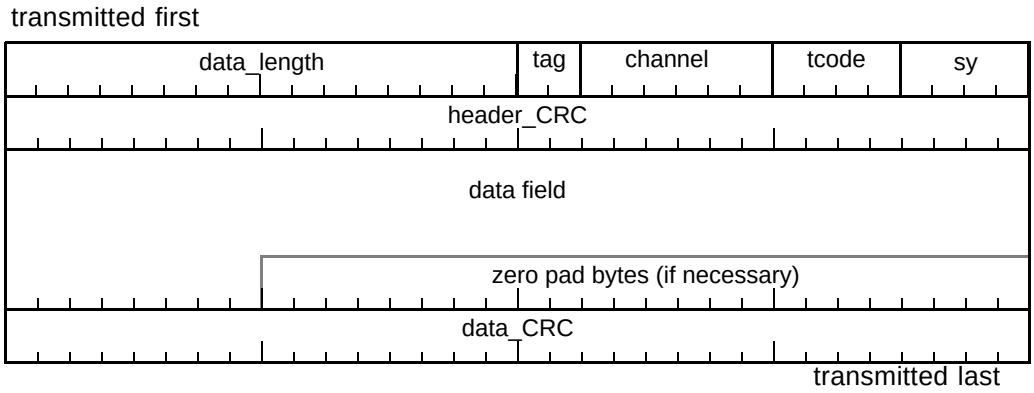


Figure 8-1 — Asynchronous stream packet format

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 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 stream packet outside of the isochronous period (strict isochronous), a significant number of contemporary Serial Bus link designs relax this requirement and permit the reception of an isochronous stream packet at any time (loose isochronous).

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 are able to make valid use of isochronous data that follows so long as the link permits its reception.

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.

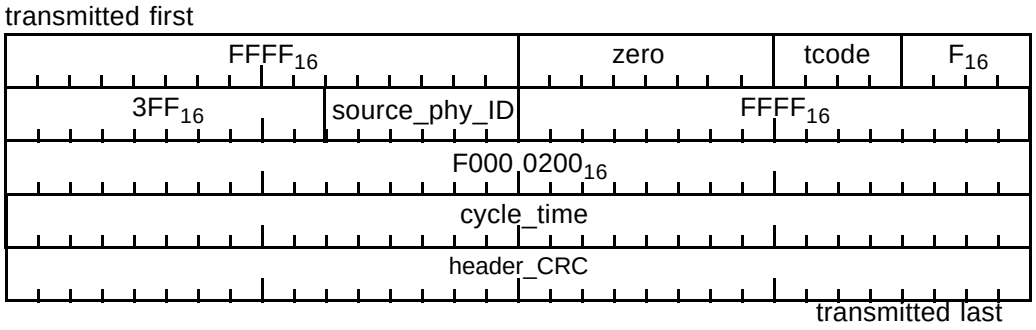


Figure 9-1—Cycle start packet format

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 may 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. If *data_length* is zero no data CRC shall be calculated.

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	Header size (quadlets)	Name	Comment
0	5	Write request for data quadlet	Request subaction, quadlet payload
1	5	Write request for data block	Request subaction, block payload
2	4	Write response	Response subaction for both write requests types, no payload
3		Reserved	
4	4	Read request for data quadlet	Request subaction, no payload
5	4	Read request for data block	Request subaction, quadlet (*data_length*) payload
6	5	Read response for data quadlet	Response subaction to read request for quadlet, quadlet payload
7	5	Read response for data block	Response subaction to read request for block, block payload
8	5	Cycle start	Request to start isochronous period, quadlet payload
9	5	Lock request	Request subaction, block payload
A_{16}	2	Stream data	Asynchronous or isochronous subaction, block payload
B_{16}	5	Lock response	Response subaction for lock request, block payload
C_{16}		—	Reserved for future standardization.
D_{16}		—	Reserved for future standardization.
E_{16}		—	Utilized internally by some link designs; not to be standardized
F_{16}		—	Reserved for future standardization.

9.5 Response codes (*rcode*)

Clause 6.2.4.10 of IEEE Std 1394–1995, “Response code (*rcode*)”, defines the meaning of the *rcode* 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 (*e.g.*, 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. Deadlocks 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.

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 stream 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 packet.

Table 9-4—Tag field encoding

Value	Meaning
0	Data field format unspecified
1	Data format and *sy* 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 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 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 *destination_offset* 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 *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 PHY packets and response packets

When transmitting a PHY packet or a response packet, a link may use priority arbitration requests. If the node implements the PRIORITY_BUDGET register (see clause 9.15), priority requests for neither PHY packets nor response packets shall count against the node's priority arbitration budget. A PHY packet is any 64-bit packet whose least significant 32 bits are the one's complement of the most significant 32 bits. A response packet is an asynchronous primary packet with a *tcode* of 2, 6, 7 or B_{16}.

If an *ack_busy_X*, *ack_busy_A* or *ack_busy_B* is received in acknowledgment of a response packet, the node shall not retransmit the response packet until the next fairness interval.

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. 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 is 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, 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:

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 equilibrum is reestablished may be more vulnerable to disruption. Any additional bus resets that occur during this time increase the likelihood that users perceive an interrupton 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.
— UNEXPECTED CHANNEL DETECTED (Optional, available only at the active isochronous resource manager). The isochronous resource manager observed a stream packet 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 NODE_IDS register

The following specifications replace IEEE Std 1394-1995 clause 8.3.2.2.3 in its entirety.

The NODE_IDS register reports and permits modification of a node's bus ID and physical ID. Together these form a 16-bit node ID used by the link to determine if a primary packet is addressed to the node. Serial Bus reserves the 16-bit bus-dependent field, as indicated by the shaded field within figure 9-2.

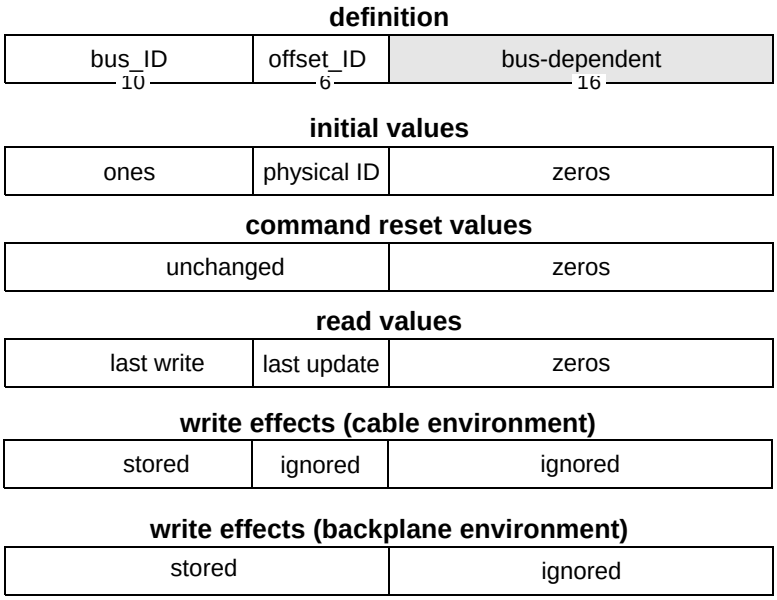


Figure 9-2—NODE_IDS format

The 10-bit read/write ***bus_ID*** field provides software with a mechanism for reconfiguration of the initial node address space. The *bus_ID* field permits node addresses on one bus to be distinguished from those on another. All nodes on a bus shall have identical *bus_ID* values.

NOTE—A bus consists of all physically connected nodes that are within the same arbitration domain, *i.e.*, nodes that receive their arbitration grant(s) from the same root node.

The 6-bit ***offset_ID*** field shall have a value generated as a side-effect of the bus initialization process. Within this standard, the value of NODE_IDS.*offset_ID* is also known as the physical ID of the node. This field is read-only in the cable environment and read/write in the backplane environment.

NOTE—The CSR Architecture requires that if there are any side-effects of a nonbroadcast write transaction to a register, the affected node shall delay the return of a transaction response until all effects of the write are complete. In the case of the NODE_IDS register, a return of *resp_complete* indicates that the node recognizes transactions to the newly assigned NODE_IDS value. The contents of the *source_ID* field of the response packet shall be equal to the most significant 16 bits of the NODE_IDS register.

9.13 SPLIT_TIMEOUT register

Split-transaction error detection requires that all nodes on Serial Bus share the same time-out value and that requester and responder behave in complementary fashion. The following specifications replace IEEE Std 1394-1995 clause 8.3.2.2.6 in its entirety.

The SPLIT_TIMEOUT register establishes the time-out value for the detection of split-transaction errors. The value of SPLIT_TIMEOUT is the maximum time permitted for the receipt of a response subaction after the transmission of a request subaction. After this time, a responder shall not transmit a response for the request subaction and a requester shall terminate the transaction with a request status of TIMEOUT. For a requester the time-out period commences when an *ack_pending* is received in response to a request subaction. A responder starts the time-out period when an *ack_pending* is transmitted. Figure 9-3 illustrates the portions of the SPLIT_TIMEOUT register implemented on Serial Bus.

NOTE—A requester should not reuse the transaction label from an expired request subaction in a subsequent request subaction to the same node unless at least twice the split time-out period has elapsed since the initiation of the expired subaction.

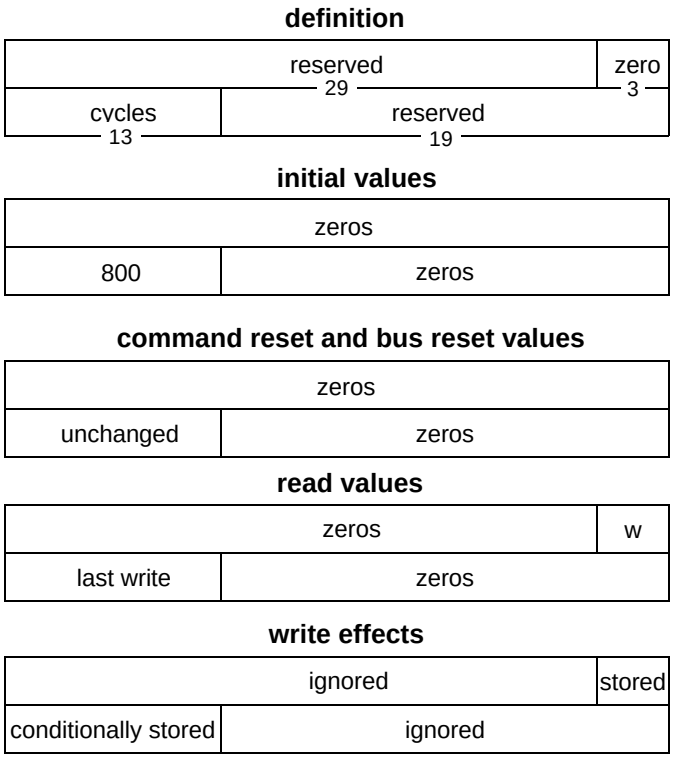


Figure 9-3—SPLIT_TIMEOUT format

The ***sec*** field, in units of seconds, and the ***cycles*** field, in units of 125 µs, together specify the time-out value. The value of *cycles* shall be less than 8000. The bus manager, if present, shall insure that all nodes on the bus have identical values in their SPLIT_TIMEOUT registers.

The minimum timeout value is 0.1 second. If a value smaller than this is written to the SPLIT_TIMEOUT register it may be ignored or rounded up to 0.1 second.

NOTE—The Serial Bus definition of the SPLIT_TIMEOUT register differs from that of the CSR Architecture. Serial Bus interprets the most significant 13 bits of the SPLIT_TIMEOUT_LO register as units of 1/8000 seconds, rather than a true binary fraction of a second with units of 1/8192 seconds. Since precise time-outs are not necessary, the bus manager may ignore this difference when calculating values for use within the SPLIT_TIMEOUT_LO register.

9.14 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 either in this document or in clause 8.3.2.3 of IEEE Std 1394-1995.

9.15 PRIORITY_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 218_{16} within initial register space.

The PRIORITY_BUDGET register permits the bus manager to configure a node's asynchronous arbitration behavior. This register provides a mechanism for a node to be granted permission to use priority arbitration during the asynchronous period. The definition is given by figure 9-4 below.

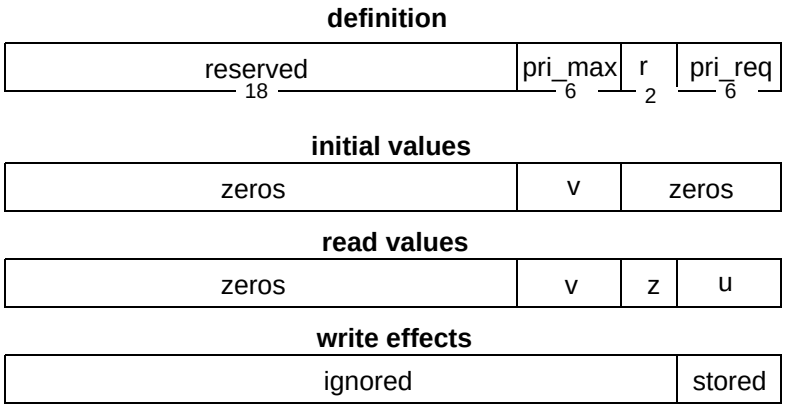


Figure 9-4—PRIORITY_BUDGET format

The *pri_max* field shall specify the maximum value the node expects to be stored in *pri_req*. If a write request attempts to update *pri_req* to a larger value, the result is unspecified.

The *pri_req* field shall specify the maximum number of certain priority arbitration requests the link is permitted to make of the PHY during a fairness interval. The primary packet transaction codes for which priority ~~asynchronous~~ arbitration may be used are specified by table 9-6; PHY packets and response packets may also use priority arbitration (see clause 9.8). A Serial Bus fairness interval exists between the occurrence of an arbitration reset gap and the first subsequent arbitration reset gap. In addition to, 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—Request subactions eligible for priority asynchronous arbitration

tcode	Name	Comment
0	Write request for data quadlet	Request subaction, quadlet payload
1	Write request for data block	Request subaction, block payload
4	Read request for data quadlet	Request subaction, no payload
5	Read request for data block	Request subaction, quadlet (*data_length*) payload
9	Lock request	Request subaction, block payload
A_{16}	Stream data	Asynchronous subaction, block payload

Each time a link receives PHY status of ARB_RESET_GAP, it shall reset an internal variable, *priority_request_count*, to the value of *pri_req*. Except in one case, 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. Even if *priority_request_count* is nonzero, if a node receives an *ack_busy_X*, *ack_busy_A* or *ack_busy_B* in acknowledgment of a request subaction, the node shall not retransmit the request packet until the next fairness interval. Each time a priority arbitration request is granted for one of the transaction codes specified, 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 PRIORITY_BUDGET registers of all nodes on the local bus is less than or equal to 63 minus the number of nodes.

9.16 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} — $FFFC_{16}$		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.

9.16.1 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 PHY register *Max_speed* field defined in table 6-1. The remaining most-significant five bits are reserved for future standardization and may not be relied upon to be read as zeros.

9.16.2 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.

9.17 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 nodes 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 9-5; this replaces existing figure 8-20 in clause 8.3.2.5.4 of IEEE Std 1394-1995.

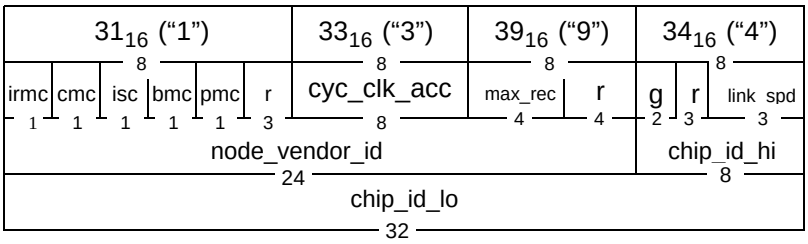


Figure 9-5—Bus_Info_Block format

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 0400_{16} through FFFF F000 $07FC_{16}$, inclusive) but any directories or leaves that are indirectly addressed from the first kilobyte. The CRC in the first quadlet of configuration ROM shall be recalculated each time the *generation* field is updated.

The ***link_spd*** field shall report the maximum speed capability of the node's link layer; the encoding used is the same as for the PHY register *Max_speed* field defined in table 6-1.

9.18 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 obsoleted.

9.19 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 $3F_{16}$ as the only way in which a candidate bus manager is confirmed as the new bus manager. If a candidate bus manager successfully completes a lock (compare 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 $3F_{16}$ 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 $3F_{16}$ or the physical ID of the candidate bus manager, the candidate is confirmed as the new bus manager.

9.20 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 7.4.3) 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.

9.21 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).

9.22 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.

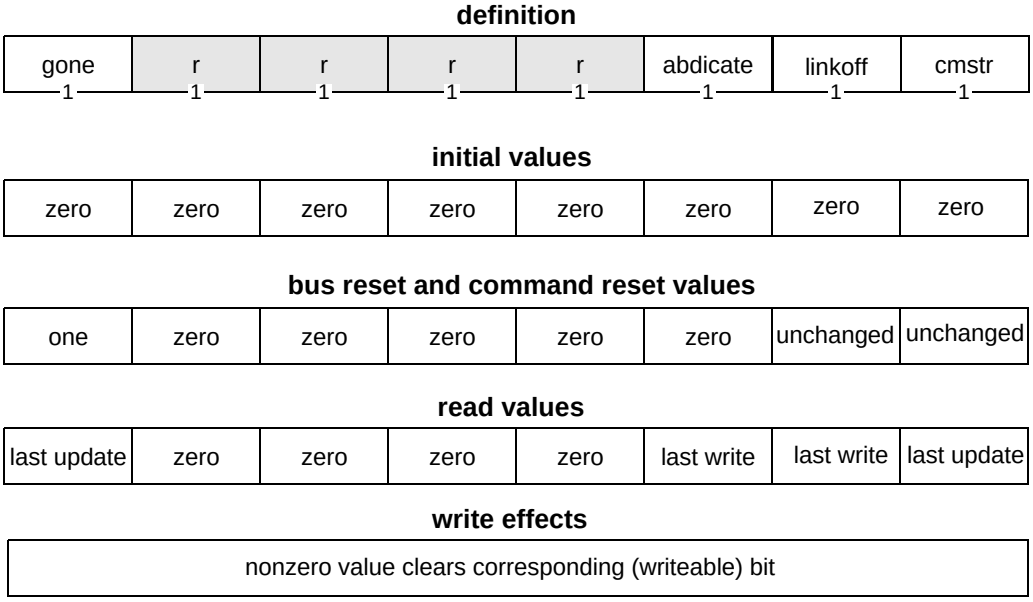


Figure 9-6—STATE_CLEAR.*bus_depend* field

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 9.21 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 it's 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.

9.23 Internal device physical interface

Annex C of IEEE Std 1394-1995 specifies a physical interface suitable for Serial Bus devices mounted internal to a module's enclosure. When this optional interface is utilized, all clauses of annex C are normative with the exception of clause C.1, "Overview". The overview is informative and describes the rationale for the internal device physical interface specified by the remainder of the annex.

This supplement retitles clause C.1 "Overview (informative)" and replaces the clause in its entirety with the text that follows. The other clauses of annex C are not affected.

The cable media attachment specification in clause 4.2.1 of IEEE Std 1394-1995 is suitable to external, box-to-box applications. (An example would be a computer, printer and video camera connected *via* Serial Bus; the computer and printer are powered from different AC outlets while the camera takes power from the Serial Bus cable.) The external cable also provides power to all PHYs on the bus so that they can maintain their bus repeater capability even when their local power is off. When necessary to accommodate different power domains (*i.e.*, from different AC power sources), each node provides isolation between the its local AC power and external cable power. The external environment requires mechanically strong shielded cables and connectors.

Iinternal devices may not have the same design criteria as external, box-to-box applications; they may be optimized for low cost, low power, minimum components and minimum package size (*e.g.*, mass storage devices). Internal devices usually share a common power domain with other devices packaged within the enclosure and may not require mechanically strong or shielded connectors and cables. Internal devices may require other packaging options, such as hot-plug, auto-dock, blindmate; they may need various connector methodologies, such as cable or board attachment with such connector systems as surface mount or card edge.

A goal of the internal device interface is to allow implementation options for both the device vendor and the system integrator. These options enable Serial Bus internal devices to accommodate a wide range of applications in a cost-effective manner. Device options include a second port that can be configured as either as a repeater (bus) or as a second independent port (dual path). Packaging options include cable attachment, board attachment, or a combination of the two. Pins are allocated in the internal device connector to support these options.

9.24 Transaction integrity safeguards

IEEE Std 1394-1995 makes little provision for facilities or implementation constraints that enhance resistance to tampering by malicious agents. Because Serial Bus may be connected to external gateways (such as cable network interface units) which may be reprogrammable from a remote location, there is a desire to provide building blocks upon which more tamper-resistant systems may be constructed. In particular it is important for Serial Bus modules to possess unforgeable identities and to not be able to snoop asynchronous request or response packets addressed to other nodes.

A module compliant with this 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 $3FF_{16}$ 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 $3F_{16}$, or b) the concatenation of $3FF_{16}$ and either the physical ID assigned to the node's PHY during the self-identify process or $3F_{16}$.

All exceptions to these requirements, if any, shall be explicitly specified in future standards developed and approved through the IEEE standards development process. At the time of writing, the only anticipated exceptions are for Serial Bus to Serial Bus bridges, which work is in progress in the IEEE P1394.1 working group.

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.

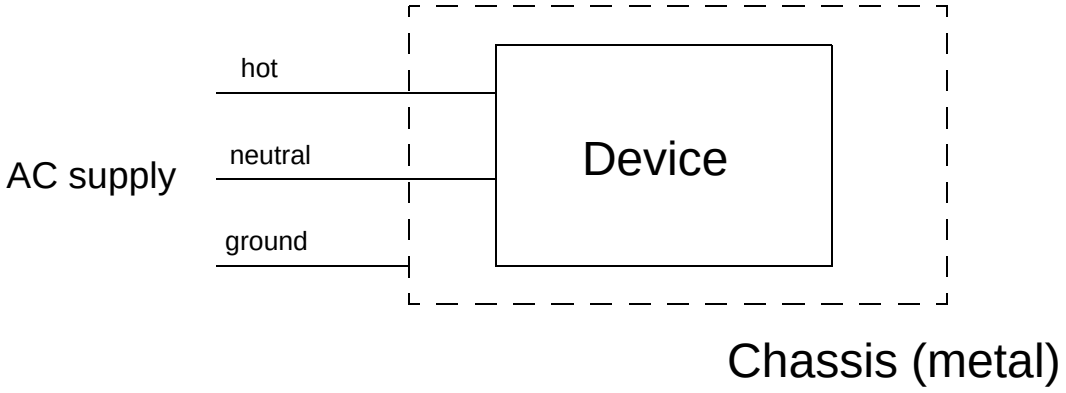


Figure A-1 — AC power supply with ground

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、セキュリティー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.