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. This standard follows the IEEE Std 1212-1991 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*
Dick Scheel, *Secretary*

The following is a list of other major participants in the IEEE P1394a working group (those that attended at least three working group meetings in the last four years).

Dave Banks
Max Bassler
Harrison Beasley
Erich Berndlmaier
David Brief
Mike Brown
Joe Chen
Rajiv Choudhary
Richard Churchill
Alistair Coles
Hugh Curley
Bill Duckwall
Brian G. Dugan
Mike Eneboe
Dave Evans
Lou Fasano
Steve Finch
Taka Fujimori
John Fuller
Masamichi Furukawa
Mike Gardner
Ram Gopalan
Gordon Haas
Manish Harpalani
Katsuya Hasegawa
Yasumasa Hasegawa
Shinichi Hatae
Jerry Hauck
Burke Henehan
Jack Hollins
Du Hung Hou
David James
Peter Johansson
Tony Kobayashi
Jim Koser
Takashi Kubo
Steve Kukla
Tadashi Kumihira
Farrukh Latif
Aaron Ludtke
Jerry Marazas
Tetsuya Miyame
Kazayoshi Moriya
Shuhei Moriyoshi
Richard Mourn
Bill Northey
Karen O'Connell
Yasushi Ohtani
Jun Okazaki
Erik Ottem
Alan Perry
Bob Plummer
Matt Pujol
Mehran Ramezani
Dennis Rehm
Todd Roper
Bill Russell
Takashi Sato
Dick Scheel
Andreas Schloissnik
Imran Sharif
Robbie Shergill
Hisato Shima
Michael Sorna
Curtis Stevens
Tom Suters
Shah Talukder
Mike Teener

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

Mary Lynne Nielsen
IEEE Standards Project Editor

P1394a
Draft Standard for a
High Performance Serial Bus (Supplement)

1. Overview

1.1 Scope

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

The following are included in this supplement:

a) Cables and connectors for a 4-pin variant (from the 6-pin already standardized);
b) Standardization of the PHY/LINK interface, which at present is an informative annex to the existing standard;
c) A connection management protocol, and the necessary CSR facilities, for isochronous data;
d) Performance enhancements to the PHY layer that are interoperable with the existing standard, *e.g.*, a method to shorten the arbitration delay when the last observed Serial Bus activity is an acknowledge packet;
e) An incremental reconfiguration protocol, invoked when a device is connected to or disconnected from Serial Bus, which is designed to limit the circumstances under which a Serial Bus reset is necessary;
f) A technique for Serial Bus topologies that form a physical loop to be resolved into a logical tree structure (required by the existing standard) without user intervention;
g) Minor *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 896.2-1991, IEEE Standard for Futurebus+®—Physical Layer and Profile Specification (ANSI).[2]

ISO/IEC 9899: 1990, Programming languages—C.[3]

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

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 communicate with the next higher layer and with the Serial Bus management layer. 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 the higher layer to the lower layer to request the lower layer to perform 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 by a confirmation. A data transfer request usually will trigger a corresponding indication on a 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 the lower layer to the upper layer to indicate a change of state or other event detected by the lower layer. An indication may also communicate parameters that are associated with the change of state or event. An indication may or may not be responded to by a response. A data transfer indication is originally caused by corresponding requests on a peer node.

c) *Response service.* A response service is a communication from the higher layer to the lower layer to respond to an indication. A response may also communicate parameters that indicate the type of response to the indication. A response is always associated with an indication. A data transfer response usually will trigger a corresponding confirmation on a peer node.

d) *Confirmation service.* A confirmation service is a communication from the lower layer to the upper layer to confirm a request service. A confirmation may also communicate parameters that indicate the completion status of the request or any other status. A confirmation is always associated with a request. 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] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[3] 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:

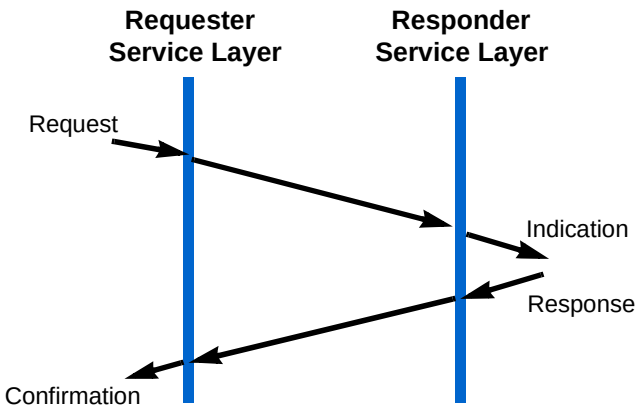


Figure 1-1—Service model

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

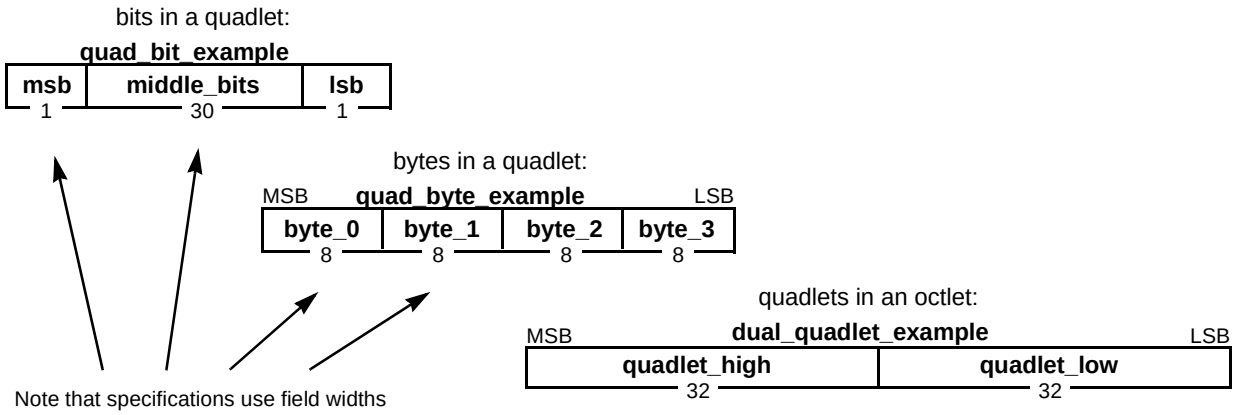


Figure 1-2—Bit and byte ordering

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

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

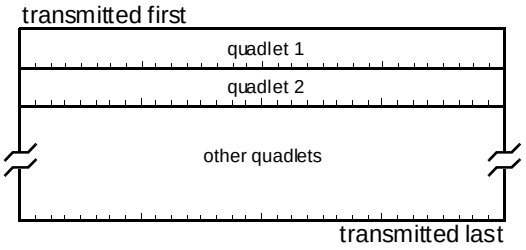


Figure 1-3—Example packet format

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. Since many C code operators are not obvious to the casual reader, their meanings are summarized in table 1-2.

Table 1-2—Specific expression summary

Expression	Description
`~I`	Bitwise complement of integer I
`++I`	Pre-increment of integer I (I is incremented, then used in the expression)
`--I`	Pre-decrement of integer I (I is decremented, then used in the expression)
`I ^ J`	Bitwise XOR of integers I and J
`I & J`	Bitwise AND of integer values I and J
`I | J`	Bitwise OR of integer values I and J
`I << J`	Value of I, shifted left by J bits, zero fill
`I >> J`	Value of I, shifted right by J bits, zero fill if I is an unsigned number, sign extension if I is signed
`I == J`	Equality test, true if I is equal to J
`I != J`	Inequality test, true if I is not equal to J
`!B`	Logical negation of boolean variable B
`A && B`	Logical AND of boolean values A and B
`A || B`	Logical OR of boolean values A and B

In addition, the nonstandard data types (actually, object classes) listed in table 1-3 are supported.

Table 1-3—Serial Bus data types

Data type	Description
`timer`	real value (units of seconds) that autonomously increments at a defined rate
`boolean`	One bit value where 1 is true and 0 is false

Other, more specific data types are defined in the clauses where they are relevant.

All C code is executed as if it takes zero time. Time only elapses when the following function is called (“time” is in units of seconds):

```
void wait (real time);    // wait for “time” to elapse
```

1.6.8 State machine notation

All state machines in this standard use the style shown in figure 1-4.

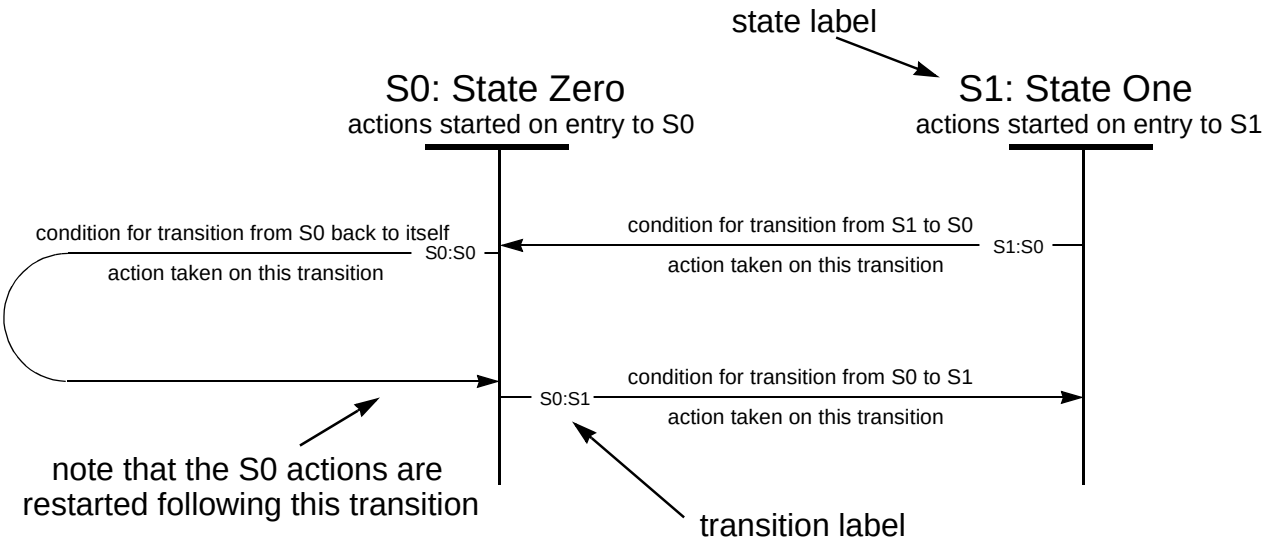


Figure 1-4—State machine example

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.
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 restart the actions from the beginning.

1.6.9 CSR, ROM and field notation

This standard describes a number of CSRs and fields within these registers. To distinguish register and field names from node states or descriptive text, the register name is always capitalized. Thus, the notation STATE_CLEAR.*lost* is used to describe the lost bit within the STATE_CLEAR register. In this standard, a bold type font is used to emphasize a term, particularly on its first usage.

All CSRs are quadlets and are quadlet aligned. The address of a register (which is always a multiple of 4) is specified as the byte offset from the beginning of the initial register space. When a range of register addresses is described, the ending address is the address of the last register, which is also a multiple of 4. These addressing conventions are illustrated in table 1-4.

Table 1-4—Example CSR addressing conventions

Offset	Register Name	Description
0	STATE_CLEAR	First CSR location
4-12	OTHER_REGISTERS	Next three CSR locations

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 precisely defines the format and function of Serial Bus-specific CSRs. Some of these registers are read-only, some are read-write and some have special side-effects on writes. To define the content and function of these CSRs wholly, their specification includes the format (the sizes and names of bit field locations), the initial value of the register (if not zero), 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.

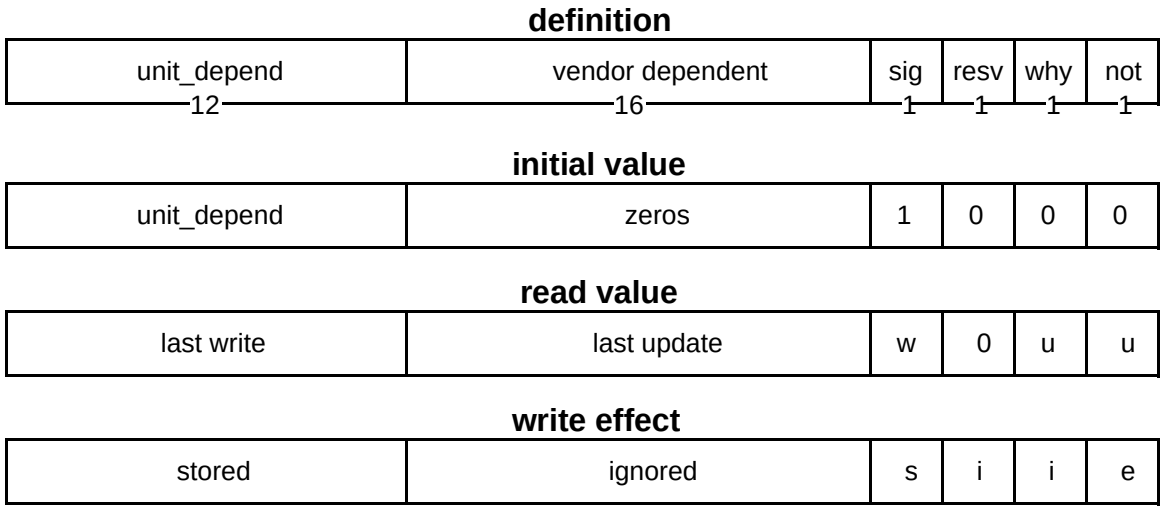


Figure 1-5—CSR format specification (example)

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 table 1-5 have generic meanings.

Table 1-5—Register definition fields

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

The CSRs defined in this document shall be initialized when power is restored (a **power_reset**) or when a quadlet is written to its RESET_START register (a **command_reset**). For most registers, the initial value after a power_reset or command_reset is the same. When the initial CSR values differ, the two initial values are explicitly illustrated.

The read value fields in table 1-6 have a generic meaning.

Table 1-6—Read value fields

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

The write-effect fields in table 1-7 have a generic meaning.

Table 1-7—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.

The register description specifies its bus transaction read/write characteristics, as well as whether it is a required register. A read-write register (**RW**) is expected to be read and written via bus transactions; a read-only register (**RO**) is expected to only be read; a write-only register (**WO**) is expected to only be written. Although reads of WO registers and writes of RO registers are not expected, the register definition still defines their results.

1.6.11 Reserved registers and fields

Some CSR addresses correspond to unimplemented registers. This includes optional registers (when the option is not implemented) and reserved registers (which are required to be unimplemented). The capabilities of these unimplemented registers are exactly defined to minimize conflicts between current implementations and future definitions, as illustrated in figure 1-6.

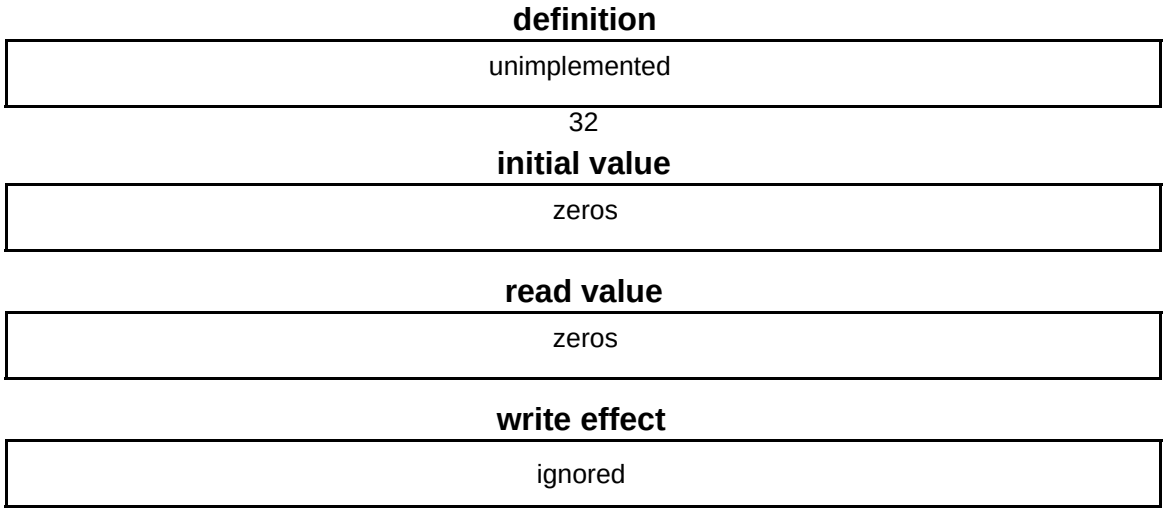


Figure 1-6—Reserved fields

Within an implemented register, a field that is reserved for future revisions of this standard is labeled *reserved* (sometimes abbreviated as *r* or *resv*). For a reserved field within an implemented register, the field is ignored on a write and returns zero on a read, as formally specified below:

reserved:

Required.	Reserved for future definitions.
Initial value:	Zero.
Read4 value:	Shall return zero.
Write4 effect:	Shall be ignored.

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 segments 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 follows:

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 standards. A reserved object shall be zeroed or, upon development of a future 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 gap: The period of idle bus between the end of a packet and the start of an acknowledge.

2.2.3 acknowledge packet: A link-layer packet returned by a destination node back to a source node in response to most primary packets. An acknowledge packet is always exactly 8 bits long.

2.2.4 arbitration: The process by which nodes compete for ownership of the bus. The cable environment uses a hierarchical point-to-point algorithm, while the backplane environment uses the bit-serial process of transmitting an arbitration sequence. At the completion of an arbitration contest, only one node will be able to transmit a data packet.

2.2.5 arbitration clock rate: The rate used to define a number of timing requirements within the backplane physical layer. It is 49.152 MHz ± 100 ppm, regardless of the backplane interface technology.

2.2.6 arbitration reset gap: The minimum period of idle bus that has to occur after a source using the fairness protocol has won an arbitration contest before it can once again compete for bus mastership. This is longer than a normal subaction gap.

2.2.7 arbitration sequence: For the backplane environment, a set of bits transmitted by nodes that wish to transmit packets that is used to determine which node will be able to transmit next.

2.2.8 arbitration signal: Bidirectional signal exchanged between nodes during arbitration. One of the PDUs for the physical layer (the other is the data bit).

2.2.9 asynchronous packet: A primary packet that contains the bus ID of the destination in the first quadlet. It is sent as the request subaction and/or response subaction of a transaction.

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

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

2.2.12 broadcast physical ID: A physical ID with a value of 111111_2.

2.2.13 bus ID: A 10-bit number uniquely specifying a particular bus within a system of multiple interconnected buses.

2.2.14 bus manager: The node that provides advanced power management, optimizes Serial Bus performance, describes the topology of the bus and cross-references 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.

2.2.15 byte: Eight bits of data.

2.2.16 cable physical layer: The version of the physical layer applicable to the Serial Bus cable environment.

2.2.17 cable PHY: Abbreviation for the cable physical layer.

2.2.18 concatenated transaction: A transaction where the request and response subactions are directly concatenated without a gap between the acknowledge of the request and the response packet.

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

2.2.20 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.

2.2.21 cycle master: The node that generates the periodic cycle start.

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

2.2.23 data bit: The smallest signaling element used by the physical layer for transmission of packet data on the medium. One of the PDU's for the physical layer (the other is the arbitration signal).

2.2.24 destination: A node that is addressed by a packet. If the destination is individually addressed by a source, then it has to return an acknowledge packet.

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

2.2.26 dribble bits: Extra bits added to the end of a packet that allow extra synchronization in implementations.

2.2.27 fairness interval: A group of back-to-back transfers during which each competing source using the fairness protocol gets a single transfer. The delimiters of the fairness interval are arbitration reset gaps.

2.2.28 gap: A period of idle bus.

2.2.29 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.

2.2.30 initial register space: A two kilobyte portion of initial node space with a base address of FFFF F000 000016. This address space is reserved for resources accessible immediately after a bus reset. Core registers defined by ISO/IEC 13213:1994 are located within initial register space as are Serial Bus-dependent registers defined by IEEE Std 1394-1995.

2.2.31 initial units space: A portion of initial node space with a base address of FFFF F000 040016. This places initial units space adjacent to and above initial register space. The CSR's and other facilities defined by unit architectures are expected to lie within this space.

2.2.32 isochronous: The essential characteristic of a time-scale or a signal such that the time intervals between consecutive significant instances either have the same duration or durations that are integral multiples of the shortest duration.

2.2.33 isochronous channel: A relationship between a node that is the talker and one or more nodes that are listeners, identified by a channel number. One isochronous packet, identified by the channel number, may be sent by the talker during each isochronous cycle. Channel numbers are allocated cooperatively through isochronous resource management facilities.

2.2.34 isochronous cycle: An operating mode of the bus that begins after a cycle start is sent and ends when a subaction gap is detected. During an isochronous cycle, only isochronous subactions may occur. An isochronous cycle begins every 125 µs, on average.

2.2.35 isochronous gap: The period of idle bus before the start of arbitration for an isochronous subaction.

2.2.36 isochronous resource manager: The node that contains the facilities needed to manage isochronous resources. In particular, the isochronous resource manager includes the BUS_MANAGER_ID, BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers. In addition, if there is no bus manager on the local bus, the isochronous resource manager may also perform limited power management and select a node to be the cycle master.

2.2.37 isochronous subaction: A complete link layer operation (arbitration and isochronous packet) that is sent only during an isochronous cycle.

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

2.2.39 link layer (LINK): The layer, in a stack of three protocol layers defined for Serial Bus, that provides the service to the transaction layer of one-way data transfer with confirmation of reception. The link layer also provides addressing, data checking and data framing. The link layer also provides an isochronous data transfer service directly to the application.

2.2.40 LINK: Abbreviation for the link layer.

2.2.41 listener: A node that receives an isochronous subaction for an isochronous channel. There may be zero, one or more than one listeners for any given isochronous channel.

2.2.42 local bus ID: A bus ID with a value of 1111111111_2.

2.2.43 lock-request packet: The packet transmitted during the request subaction portion of a lock transaction.

2.2.44 lock-response packet: The packet transmitted during the response subaction portion of a lock transaction.

2.2.45 lock transaction: A transaction that passes an address, subcommand and data parameter(s) from the requester to the responder and returns a data value from the responder to the requester. The subcommand specifies which indivisible update is performed at the responder; the returned data value is the previous value of the updated data.

2.2.46 module: The smallest component of physical management; i.e., a replaceable device.

2.2.47 natural priority: The order of packet transmission of a node given that all nodes start arbitration at the same instant using the same priority level. For the cable environment, the closer a node is to the root, the higher its natural priority. For the backplane environment, the priority level and node offset are concatenated to give its natural priority.

2.2.48 node: An addressable device attached to Serial Bus with at least the minimum set of control registers. Changing the control registers on one node does not affect the state of control registers on another node.

2.2.49 node controller: A component within a node that provides a coordination point for management functions exclusively local to a given node and involving the application, transaction, link and physical elements located at that node.

2.2.50 node ID: This is a unique 16-bit number, which distinguishes the node from other nodes in the system. 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 uniquely assigned as a consequence of bus initialization.

2.2.51 nonreturn to zero (NRZ): A signaling technique in which a polarity level high represents a logical “1” (one) and a polarity level low represents a logical level “0” (zero).

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

2.2.53 packet: A serial stream of clocked data bits. A packet is normally the PDU for the link layer, although the cable physical layer can also generate and receive special short packets for management purposes.

2.2.54 path: The concatenation of all the physical links between the link layers of two nodes.

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

2.2.56 PCB: Printed circuit board.

2.2.57 PDU: Abbreviation for protocol data unit.

2.2.58 peer: Service layer on a remote node at the same level. For instance a “peer link layer” is the link layer on a different node.

2.2.59 PHY: Abbreviation for the physical layer.

2.2.60 PHY packet: A packet either generated or received by the cable physical layer. These packets are always exactly 64 bits long where the last 32 bits are the bit complement of the first 32 bits.

2.2.61 physical connection: The full-duplex physical layer association between directly connected nodes. In the case of the cable physical layer, this is a pair of physical links running in opposite directions.

2.2.62 physical ID: The least-significant 6 bits of the node ID. This number is unique on a particular bus and is chosen by the physical layer during initialization.

2.2.63 physical layer (PHY): The layer, in a stack of three protocol layers defined for Serial Bus, that translates the logical symbols used by the link layer into electrical signals on the different Serial Bus media. The physical layer guarantees that only one node at a time is sending data and defines the mechanical interfaces for Serial Bus. There are different physical layers for the backplane and for the cable environment.

2.2.64 physical link: In the cable physical layer, the simplex path from the transmit function of the port of one node to the receive function of a port of a directly connected node.

2.2.65 port: A physical layer entity in a node that connects to either a cable or backplane and provides one end of a physical connection with another node.

2.2.66 primary packet: A packet made up of whole quadlets that contains a transaction code in the first quadlet. Any packet that is not an acknowledge or a PHY packet.

2.2.67 protocol data unit (PDU): Information delivered as a unit between peer entities that may contain control information, address information and data.

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

2.2.69 quadlet aligned address: An address with zeros in the two least significant bits.

2.2.70 register: A term used to describe quadlet aligned addresses that may be read or written by Serial Bus transactions. In the context of this standard, the use of the term register does not imply a specific hardware implementation. For example, in the case of split transactions that permit sufficient time between the request and response subactions, the behavior of the register may be emulated by a processor within the module.

2.2.71 request: A subaction with a transaction code and optional data sent by a node (the requester) to another node (the responder).

2.2.72 response: A subaction sent by a node (the responder) that sends a response code and optional data back to another node (the requester).

2.2.73 self-ID packet: A special packet (see 4.3.4.1) sent by a cable PHY during the self-ID phase following a reset. One to four self-ID packets are sent by a given node depending on the maximum number of ports it has.

2.2.74 Serial Bus management: The set of protocols, services and operating procedures that monitors and controls the various Serial Bus layers: physical, link and transaction.

2.2.75 services: A set of capabilities provided by one protocol layer entity for use by a higher layer or by management entities.

2.2.76 service primitive: A specific service provided by a particular protocol layer entity.

2.2.77 source: A node that initiates a bus transfer.

2.2.78 speed code: The code used to indicate various bit rates for Serial Bus: S25 indicates 24.576 Mbps/s for TTL backplanes; S50 indicates 49.152 Mbps/s for BTL and ECL backplanes; S100 indicates the 98.304 Mbps/s base rate for cable; S200 and S400 indicate 196.608 Mbps/s and 393.216 Mbps/s for the cable.

2.2.79 split transaction: A transaction where the responder releases control of the bus after sending the acknowledge and then some time later starts arbitrating for the bus so it can start the response subaction. Other subactions may take place on the bus between the request and response subactions for the transaction.

2.2.80 subaction gap: The period of idle bus between subactions. There is no gap between the request and response subaction of a concatenated split transaction.

2.2.81 subaction: A complete link layer operation: arbitration, packet transmission and acknowledgment. The arbitration may be missing when a node already controls the bus and the acknowledge is not present for subactions with broadcast addresses or for isochronous subactions.

2.2.82 talker: A node that sends an isochronous subaction for an isochronous channel. There shall be no more than one talker for any given isochronous channel.

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

2.2.84 transaction layer: The layer, in a stack of three protocol layers defined for Serial Bus, that defines a request-response protocol to perform bus operations of type read, write and lock.

2.2.85 transaction: A request and the corresponding response. The response may be null for transactions with broadcast destination addresses. This is the PDU for the transaction layer.

2.2.86 unified transaction: A transaction that is completed in a single subaction.

2.2.87 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 that is typically accessed by device driver software at an initiator. A node may have multiple units, which normally operate independently of each other.

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

3. Summary description

Xxxx....

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, (*e.g.*, clause 6., "Cable physical layer performance enhancement specifications,"), 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 (PCB) 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.

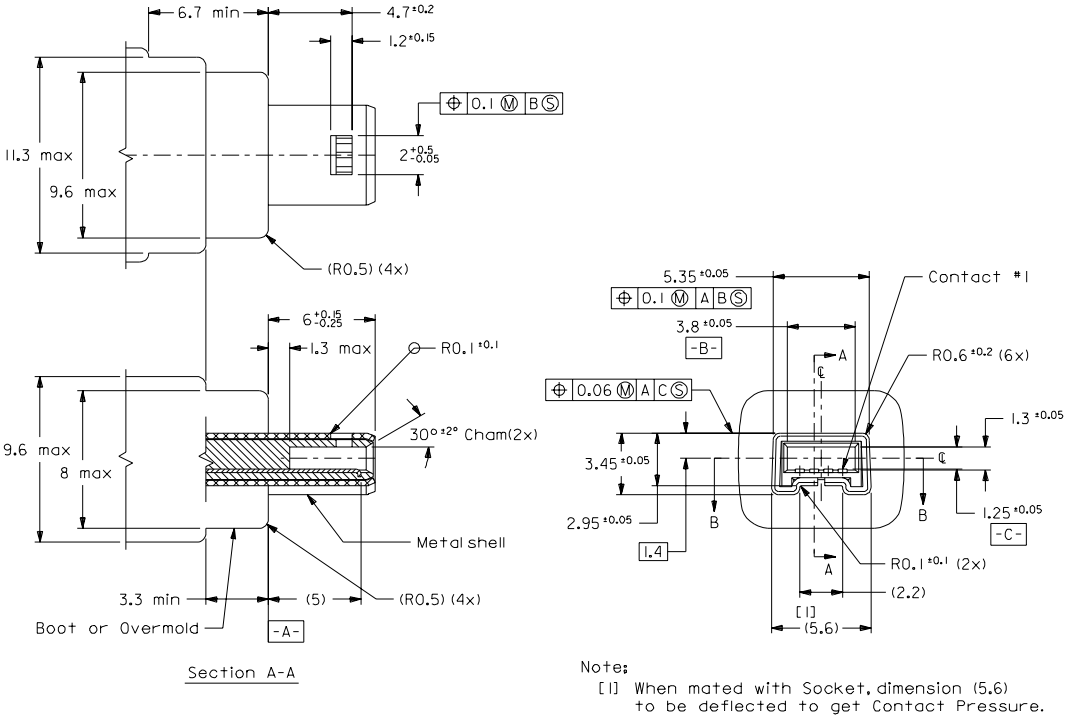


Figure 4-1 — Plug body

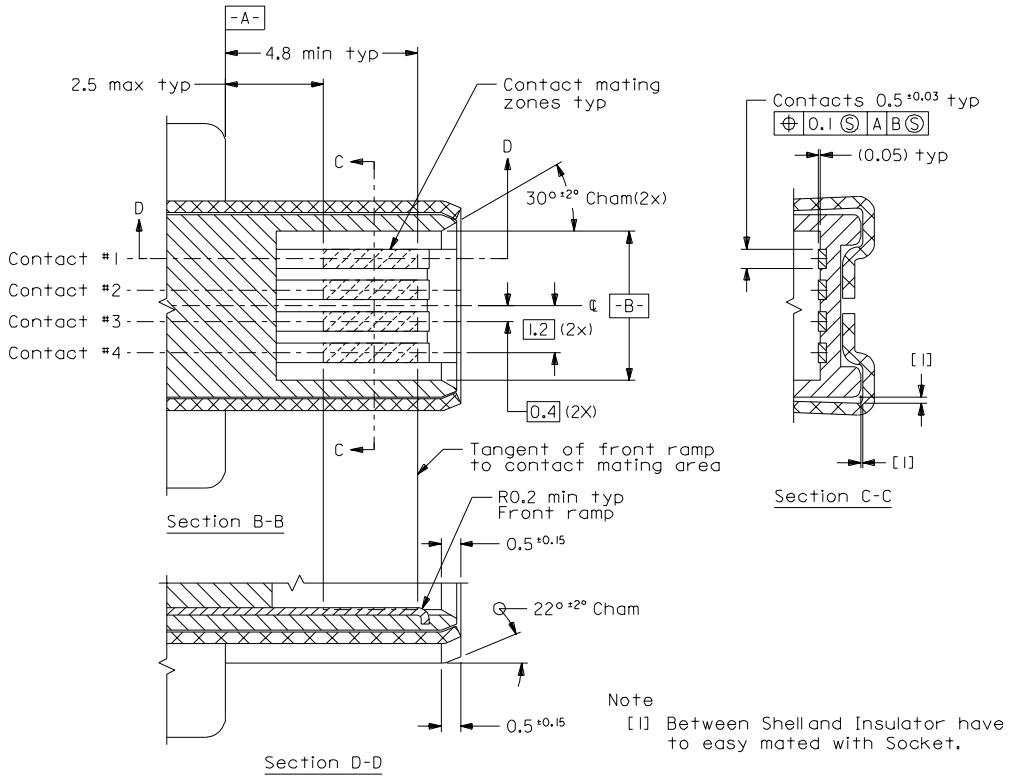


Figure 4-2 — Plug section details

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.

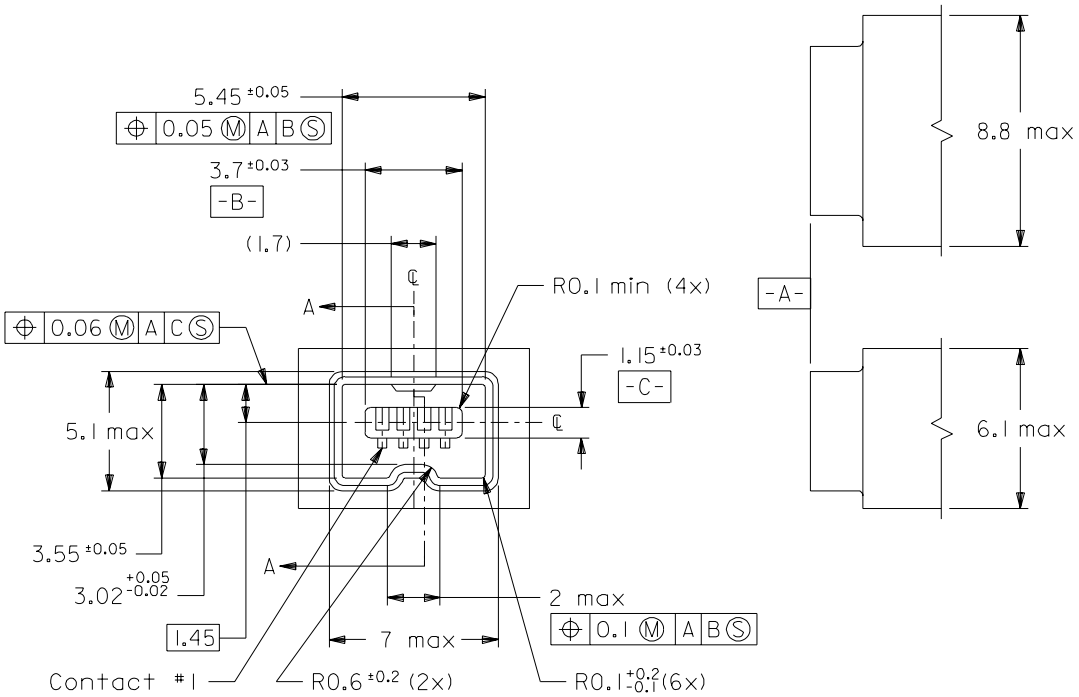


Figure 4-3 — Connector socket interface

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.

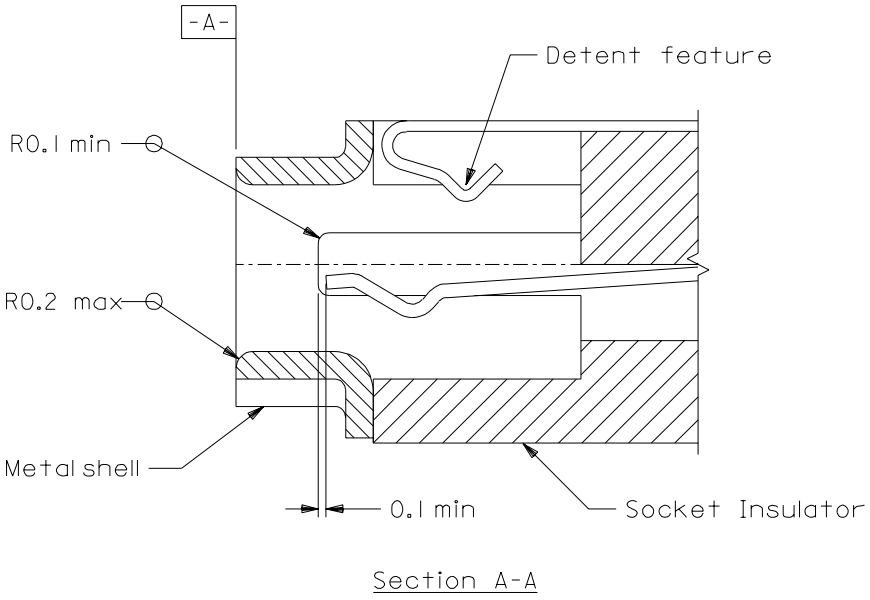


Figure 4-4 — Socket cross-section A–A

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

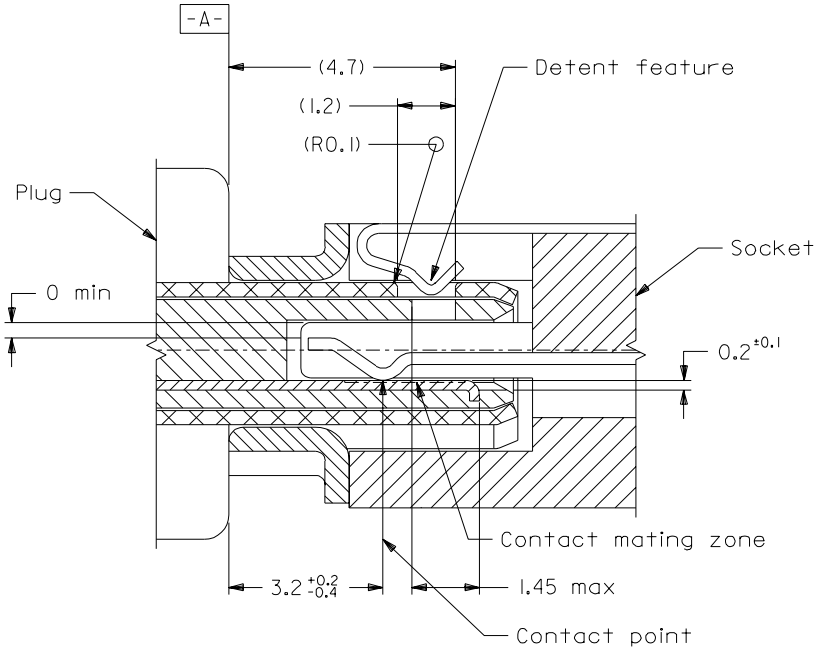


Figure 4-5 — Cross-section of plug and socket contacts

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

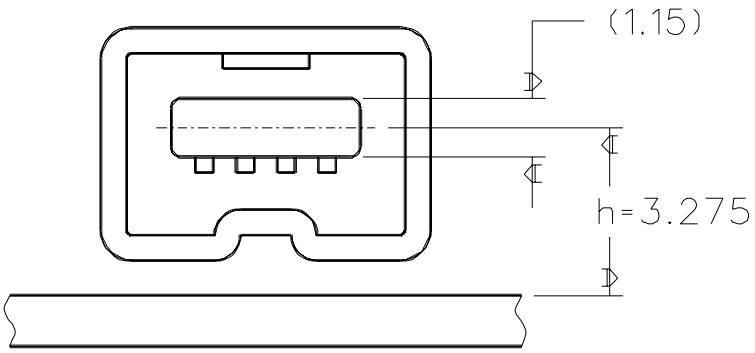


Figure 4-6 — Socket position when mounted on a PCB

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 insure 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 PCB footprints

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

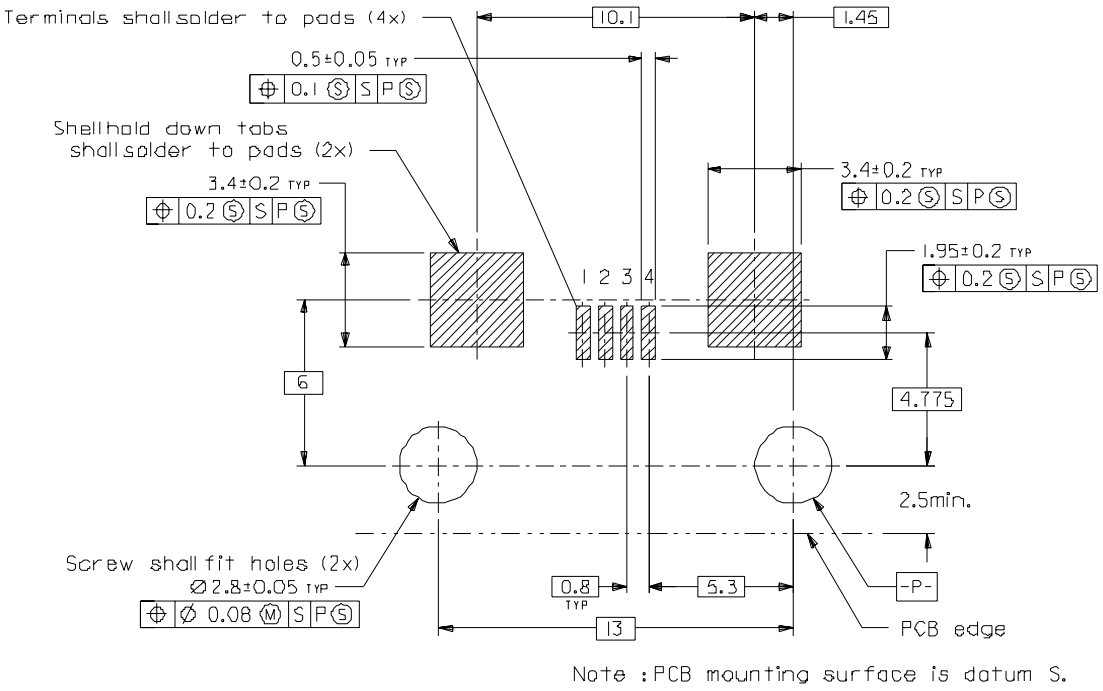


Figure 4-7 — Flat surface mount PCB connector footprint

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

Figure 4-8 — Flat through-hole mount PCB 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.

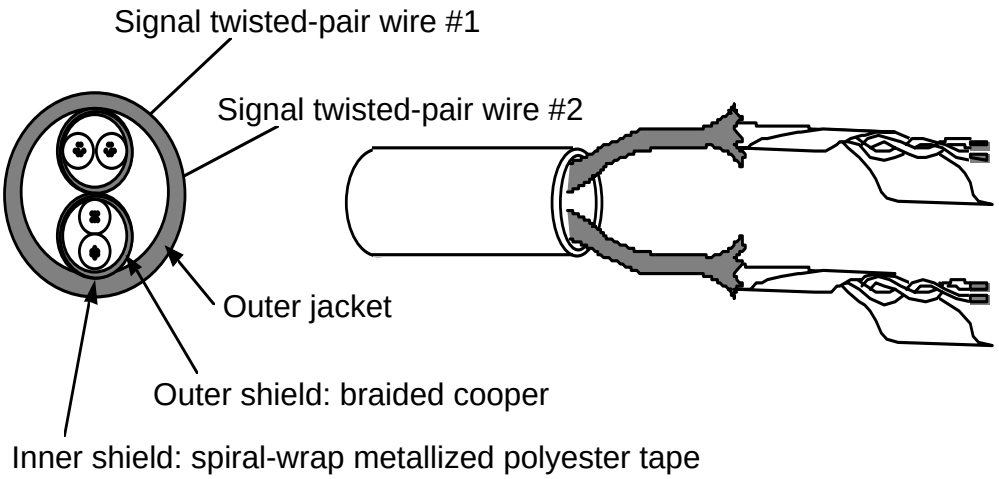


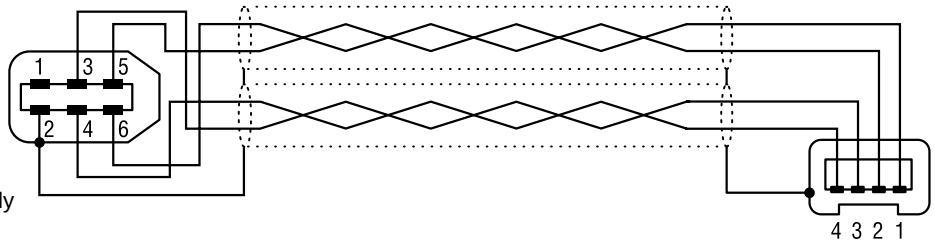
Figure 4-9 — Cable material construction example (for reference only)

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.



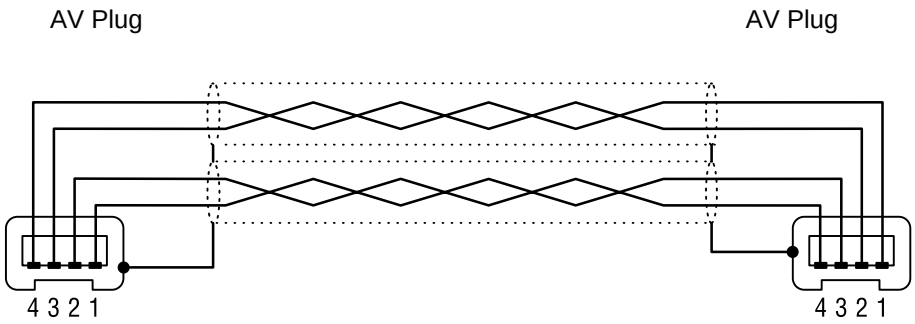
Signal Names
for reference only

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

Signal Names
for reference only

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

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



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 PCB used for Phase 1, A1 and A2 (one each).
[2] Sockets, assembled to PCB
[2] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one each).
[2] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-2 — Performance group A

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance Level
A1	Visual and dimensional inspection	ANSI/EIA 364-18A-84	Unmated connectors	Dimensional inspection	Per figures 4-1 through 4-5	No defects that would impair normal operations. No deviation from dimensional tolerances.
A2	Plating thickness measurement					Record thickness; see 4.1.4
A3	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated pair.
A4	Vibration	ANSI/EIA 364-28A-83	Condition I (See note)	Continuity	ANSI/EIA 364-46-84	No discontinuity at 1 µs or longer. (Each contact)
A5	None			Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.
A6	Mechanical shock (specified pulse)	ANSI/EIA 364-27A-83	Condition A (See note)	Continuity	ANSI/EIA 364-46	No discontinuity at 1 µs or longer. (Each contact)
A7	None			Low-level contact resistance	ANSI/EIA 364-23	20 mΩ maximum change from initial per mated contact.

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

a) An insulating member to prevent grounding of the shell to the panel
b) A PCB in accord with the pattern shown in figure? for the socket being tested. The PCB 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 PCB
[2] Sockets, assembled to PCB
[0] Plugs, unassembled to cable.
[2] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-3 — Performance group B

Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
B1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
B2	Thermal shock	IEC 68-2-14	10 cycles (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.
B3	Humidity	ANSI/EIA 364-31A-83	Condition A (96 h.) Method II (cycling) nonenergized Omit steps 7a and 7b. (mated)	Low-level contact resistance	ANSI/EIA 364-23A-85	20 mΩ maximum change from initial per mated contact.

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

Number of samples:

[2] Sockets, unassembled to PCB
[0] Sockets, assembled to PCB
[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:

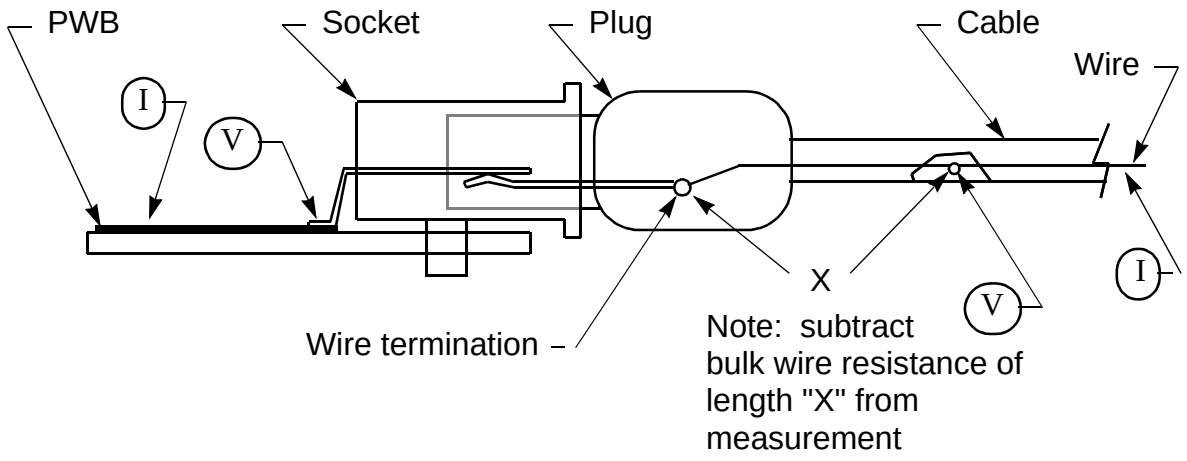
[0] Sockets, unassembled to PCB
[4] Sockets, assembled to PCB
[0] Plugs, unassembled to cable used for Phase 1, A1 and A2 (one).
[4] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4-5 — Performance group D

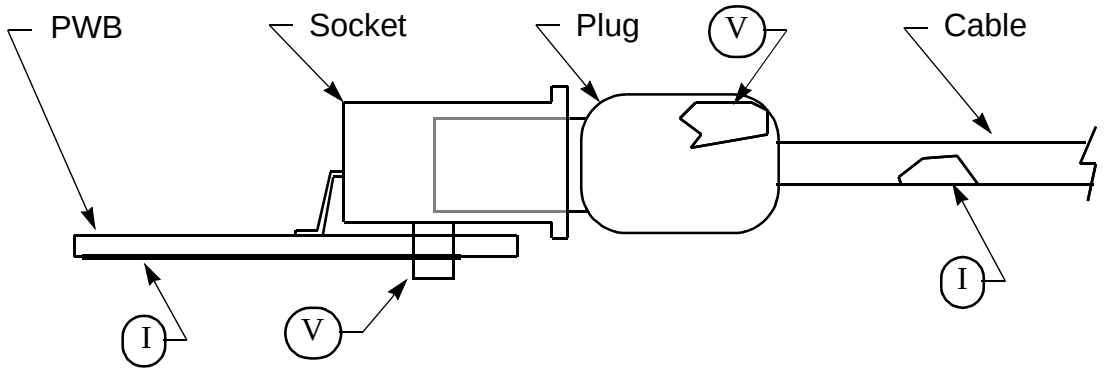
Phase	Test			Measurements to be performed		Requirements
	Title	ID No.	Severity or conditions	Title	ID No.	Performance level
D1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	50 mΩ maximum initial per mated contact.
D2	Continuity-housing (shell)		See figure 4-12 for measurement points	Contact 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 ~~at 100 mA, 5 Vdc open circuit max.~~
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 ~~at 100 mA, 5 Vdc open circuit max.~~



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 PCB
[2] Sockets, assembled to PCB
[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 ~~at 100 mA, 5 Vdc open circuit max.~~
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 ~~at 100 mA, 5 Vdc open circuit max.~~
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 PCB
[2] Sockets, assembled to PCB
[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 velocity of propagation

The differential velocity of propagation of the signal pairs 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 velocity of propagation of the signal pairs should be the same or less than the differential velocity of propagation. 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 covers the protocol and signal timing. It does not cover specific operation of the PHY except for behavior with respect to this interface.

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

The PHY has control over the bidirectional signals. The Link only drives these signals when control is transferred to it by the PHY. The Link performs all unsolicited activity through a dedicated request signal. The possible actions that may occur on the interface are categorized as transmit, receive, status and request. These actions are described in detail in the clauses that follow.

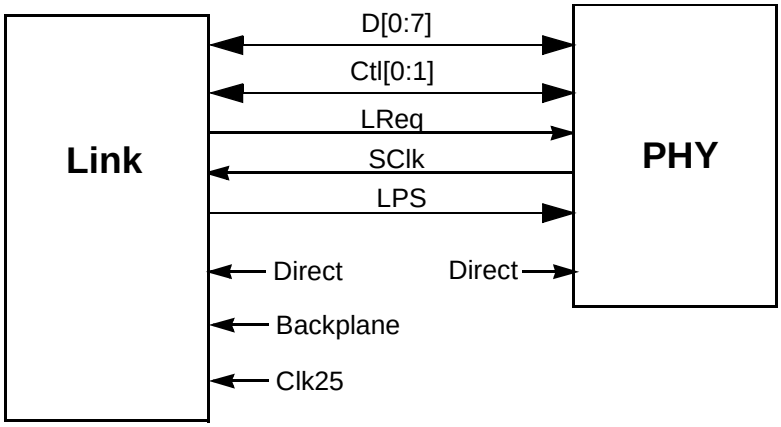


Figure 5-1 — PHY-Link interface

Table 5-1 — Signal description

Name	Driven by	Description
D[0:7]	Link and PHY (tri-state)	Data
Ctl[0:1]	Link and PHY (tri-state)	Control
LReq	Link	Link request port
SClk	PHY	49.152 MHz (sync to Serial Bus) clock
LPS	Link	Link Power Status
Direct	Neither	Indicates direct connection or isolation barrier
Backplane	Neither	Set high if backplane PHY
Clk25	Neither	Set high to notify the Link that SClk is 24.576 MHz

The interface described in this section supports the following data rates for the cable environment: S100, S200 and S400. This interface can also support the following data rates for the backplane environment: S25 and S50. The actual clock rate in a redundant encoding scheme is referred to as a “baud” rate and is independent of the clock rate of this interface. In the timing diagrams in this section, each bit cell represents one clock sample time. The specific clock-to-data timing relationships are described in 5.3.

Data is carried between the PHY and Link on the D bus. The width of the D bus depends on the maximum speed of the connected PHY device, 2 bits per 100 Mbps/s. In multiple-speed implementations, the portion of the D bus that carries packet data is left-justified in the D bus (starting with bit 0). Packet data for S100 transfers use D[0:1], S200 transfers use D[0:3] and S400 transfers use the full D[0:7]. The unused D[n] signals are to be driven low. The Ctl bus carries control information and is always 2 bits wide. The LReq signal is used by the Link to request access to Serial Bus and to read or write PHY registers. The Direct signal is used to disable the digital differentiator on the D, Ctl, SCLk, LPS and LReq signals, indicating that the two chips are directly connected, rather than through an isolation barrier.

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

Whenever control is transferred between the PHY and the Link, the side giving up control always drives the control and data buses to logic 0 levels for one clock before tri-stating its output buffers (an additional clock with 0 on the control and data signals is necessary for the Link when it is transferring control to the PHY without a Hold request). This is necessary to ensure that the differentiator circuit can operate properly. This procedure is part of the operational descriptions and timing diagrams that follow.

5.1 Operation

There are four basic operations that may occur in the interface: request, status, transmit and receive. All but request are initiated by the PHY. The Link uses the request operation to read or write an internal PHY register or to ask the PHY to initiate a transmit action. The PHY initiates a receive action whenever a packet is received from Serial Bus.

The Ctl bus is always 2 bits wide, independent of speed. The encoding of these signals is shown in tables 5-2 and 5-3.

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

Ctl[0:1]	Name	Meaning
00	Idle	No activity.
01	Status	The PHY is sending status information to the Link.
10	Receive	An incoming packet is being transferred from the PHY to the Link.
11	Transmit	The Link is granted the bus to send a packet.

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

Ctl[0:1]	Name	Meaning
00	Idle	Transmission complete, release bus.
01	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	Transmit	The Link is sending a packet to the PHY.
11	—	Unused.

The use of these signals is described in the following clauses.

5.1.1 Request

To request the bus or to access a PHY register, the Link sends a short serial stream to the PHY on the LReq signal. The information sent includes the type of request and parameter information. The nature and quantity of the parameters are dependent upon the type of request. Examples of parameter information are packet transmission speed, priority, PHY register address and data. The size of the request, inclusive of the stop bit, varies between 8 and 17 bits according to the type of request. A stop bit of 0 is required after each request transfer before the next transfer on LReq may begin.

The timing for this signal and the definition of the bits in the transfer are shown in figure 5-2.

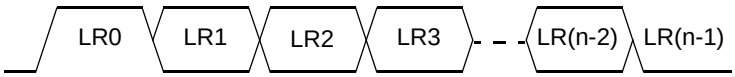


Figure 5-2 — LReq timing

If the LReq transfer is a bus request in the cable environment, it is 8 bits long and has the format given in table 5-4.

Table 5-4 — Bus request format for cable environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of transfer. Always 1.
1-3	Request Type	Indicates which type of bus request is being performed. See table 5-8 for the encoding of this field.
4-6	Request Speed	The speed at which the PHY will be sending the packet for this request. This field has the same encoding as the speed code from the first symbol of the receive packet. See table 5-9 for the encoding of this field.
7	Stop Bit	Indicates end of transfer. Always 0.

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

Table 5-5 — Bus request format for backplane environment

Bit(s)	Name	Description
0	Start Bit	Indicates start of transfer. Always 1.
1-3	Request Type	Indicates which type of bus request is being performed. See table 5-8 for the encoding of this field.
4-5	Request Speed	Ignored (set to 0) for the backplane environment.
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 read request, it is 9 bits long and has the format given in table 5-6.

Table 5-6 — Read request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of transfer. Always 1.
1-3	Request Type	Indicates that this is a register read. See table 5-8 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 write request, it is 17 bits long and has the format given in table 5-7.

Table 5-7 — Write request format

Bit(s)	Name	Description
0	Start Bit	Indicates start of transfer. Always 1.
1-3	Request Type	Indicates that this is a register write. See table 5-8 for the encoding of this field.

Table 5-7 — Write request format

Bit(s)	Name	Description
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.

The request type field is encoded as shown in table 5-8.

Table 5-8 — Request type field

LReq[1:3]	Name	Meaning
000	ImmReq	Take control of the bus immediately upon detecting idle; do not arbitrate. Used for acknowledge transfers.
001	IsoReq	Arbitrate for the bus, no gaps. Used for isochronous transfers.
010	PriReq	Arbitrate after a subaction ga;, ignore fair protocol. Used for cycle master request and responses.
011	FairReq	Arbitrate after a subaction gap, following fair protocol. Used for Fair transfers (with Request Priority field differentiating fair and urgent transfers for the backplane environment).
100	RdReg	Return specified register contents through status transfer.
101	WrReg	Write to specified register.
110	CycleSync	The link cycle timer has counted a 125 µs interval; arbitration acceleration after an observed acknowledge packet is not permitted until a subsequent subaction gap is observed.
111	—	Reserved for future standardization

The request speed field is encoded as shown in table 5-9.

Table 5-9 — Request speed field

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

NOTE—LR[4:6] is always set to 000 for transfers in the backplane environment, regardless of speed.

To request the bus for fair or priority access, the Link sends the request at least one clock after the interface becomes idle. The Link interprets the *receive* state on the Ctl signals as a lost request. If the Link sees the *receive* state anytime during or after it sends the request transfer, it assumes the request is lost and reissues the request on the next *idle*. This is necessary to insure that priority and isochronous request can be serviced properly. Meaning that the Link can prioritize cycle start packets over regular asynchronous packets and isochronous packets over cycle starts. The PHY will ignore a fair or priority request if it asserts the receive state anytime during the request transfer. Note that the minimum length of a packet is two clock cycles in the case of a 400 Mbps/s acknowledge packet. The minimum request packet is eight clock cycles.

The cycle master node uses a priority request (PriReq) to send the cycle start packet. To request the bus to send isochronous data, the Link issues an IsoReq request anytime after receiving the cycle start indication and before the next subaction gap. The PHY will clear an isochronous request only when the bus has been won.

To send an acknowledge, the Link shall issue a ImmReq request during the reception of the packet addressed to it. This is required to ensure that the ACK_RESPONSE_TIME is met and that other nodes do not detect a subaction gap. After the packet ends, the PHY immediately takes control of the bus and grants the bus to the Link. If the header CRC of the packet turns out to be bad, the Link releases the bus immediately. The Link cannot use this grant to send another type of packet. To ensure this, the Link shall wait 160 ns after the end of the received packet to allow the PHY to grant it the bus for the acknowledge, then release the bus and proceed with another request.

Though highly unlikely, it is conceivable that two different nodes can perceive (one correctly, one mistakenly) that an incoming packet is intended for them and both issue an acknowledge request before checking the CRC. The PHYs of both nodes would grab control of the bus immediately after the packet is complete. This condition will cause a temporary, localized collision of the data-on line states somewhere between two PHYs intending to acknowledge. All other PHYs on the bus would see the data-on state. This collision would appear as “ZZ” line state and would not be interpreted as a bus reset. The mistaken node would drop its request as soon as it has checked the CRC and the spurious “ZZ” line state would go away. The only side-effect of such a collision would be the loss of the intended acknowledge packet, which would be handled by the higher layer protocol.

For write requests, the PHY takes the value in the data field of the transfer and loads it into the addressed register as soon as the transfer is complete. For read requests, the PHY returns the contents of the addressed register at the next opportunity through a status transfer. The Link is allowed to perform a read or write operation at any time. If the status transfer is interrupted by an incoming packet, the PHY continues to attempt the transfer of the requested register until it is successful.

Once the Link issues a request for access to the bus (immediate, isochronous, fair, or priority), it shall not issue another bus request until the PHY indicates “lost” (incoming packet) or “won” (transmit). The PHY ignores new bus requests while a previous request is pending.

5.1.2 Status

When the PHY has status information to transfer to the Link, it will initiate a status transfer. The PHY will wait until the interface is idle to perform the transfer. The PHY initiates the transfer by asserting *status* (01b) on the Ctl signals, along with the first two bits of status information on D[0:1]. The PHY maintains Ctl == status for the duration of the status transfer. The PHY may prematurely end a status transfer by asserting something other than *status* on the Ctl signals. 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 normally sends just the first four bits of status to the Link. These bits are status flags that are needed by Link state machines. The PHY sends an entire status packet to the Link after a read request, or to indicate the nodes new phy_ID after a bus reset during the self-identify process. This is the only defined condition where the PHY automatically sends a register to the Link. The timing for the transfer is shown in figure 5-3.

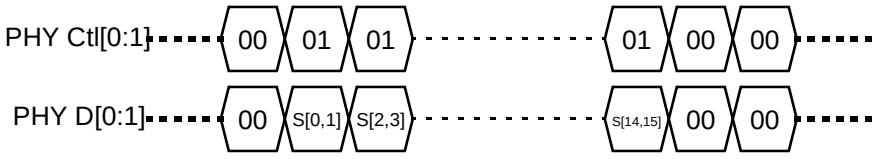


Figure 5-3 — Status timing

Table 5-10 — 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. This bit is used by the Link in the dual phase busy/retry state machine.
1	SUBACTION_GAP	The PHY has detected that Serial Bus has been idle for a Subaction gap time. This bit is used by the Link to detect the end of an isochronous cycle.
2	BUS_RESET_START	The PHY has entered bus reset state.
3	PHY_interrupt	This indicates an interrupt condition: - Loop detect interrupt - Cable power fail interrupt - Arbitration state machine time-out - Bias change detect (only on a disabled port) interrupt
4-7	Address	When transferring the contents of a register to the Link, such as when responding to a read through the LReq signal, this field holds the address of the register being read.
8-15	Data	This field holds the data corresponding to the register being transferred.

5.1.3 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 *transmit* on the Ctl signals for one SClk cycle, followed by *idle* for one cycle. After sampling the *transmit* state from the PHY, the Link asserts *idle* for one clock and takes control of the interface by asserting either *hold* or *transmit* on the Ctl bus. The Link asserts *hold* to keep ownership of the bus while preparing data. The PHY asserts the data prefix state on Serial Bus during this time. When it is ready to begin transmitting a packet, the Link asserts *transmit* on the Ctl signals along with the first bits of the packet. After sending the last bits of the packet, the Link asserts either *idle* or *hold* on the Ctl bus for one cycle and then it asserts *idle* for one additional cycle before tri-stating those signals.

An assertion of *hold* after the last bits of a packet indicates to the PHY that the Link needs to send another packet without releasing the bus. This function is useful in a number of cases: a) sending an acknowledge if the Link intends to send a unified response, b) the transmission of consecutive isochronous packets during a single cycle or c) transmission of a request packet within the scope of the arbitration enhancements specified in clause 6.3.1.3. The PHY responds to this *hold* state by waiting a MIN_PACKET_SEPARATION time and then asserting *transmit* as before.

When sending multiple packets during a single bus ownership, with one exception each may be transmitted at a different speed. The sole exception is that S100 packets shall not be concatenated after any packet with a higher speed. Link implementations compliant with this supplement shall signal the speed of each packet individually. This is done by signaling speed during the period *hold* is asserted.

NOTE—If the multi-speed capabilities of the PHY have not been enabled (see clause 5.2.1), the additional speed information transmitted by the Link for the second and subsequent concatenated packets shall be ignored by the PHY: all the 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 for the current bus ownership, it releases the bus by asserting *idle* on the Ctl signals for two SClk cycles. The PHY begins asserting *idle* on the Ctl signals one clock after sampling *idle* from the Link. Note that whenever the D and Ctl lines change “ownership” between the PHY and the Link, there is an extra clock period allowed so that both sides of the interface can operate on registered versions of the interface signals, rather than having to respond to a Ctl state on the next cycle.

Note that it is not required that the Link enter the *hold* state before sending the first packet if the implementation permits the Link to be ready to transmit as soon as bus ownership is granted. The timing for a single-packet transmit operation is shown in figure 5-4. 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 5-11) and ZZ represents high impedance state.

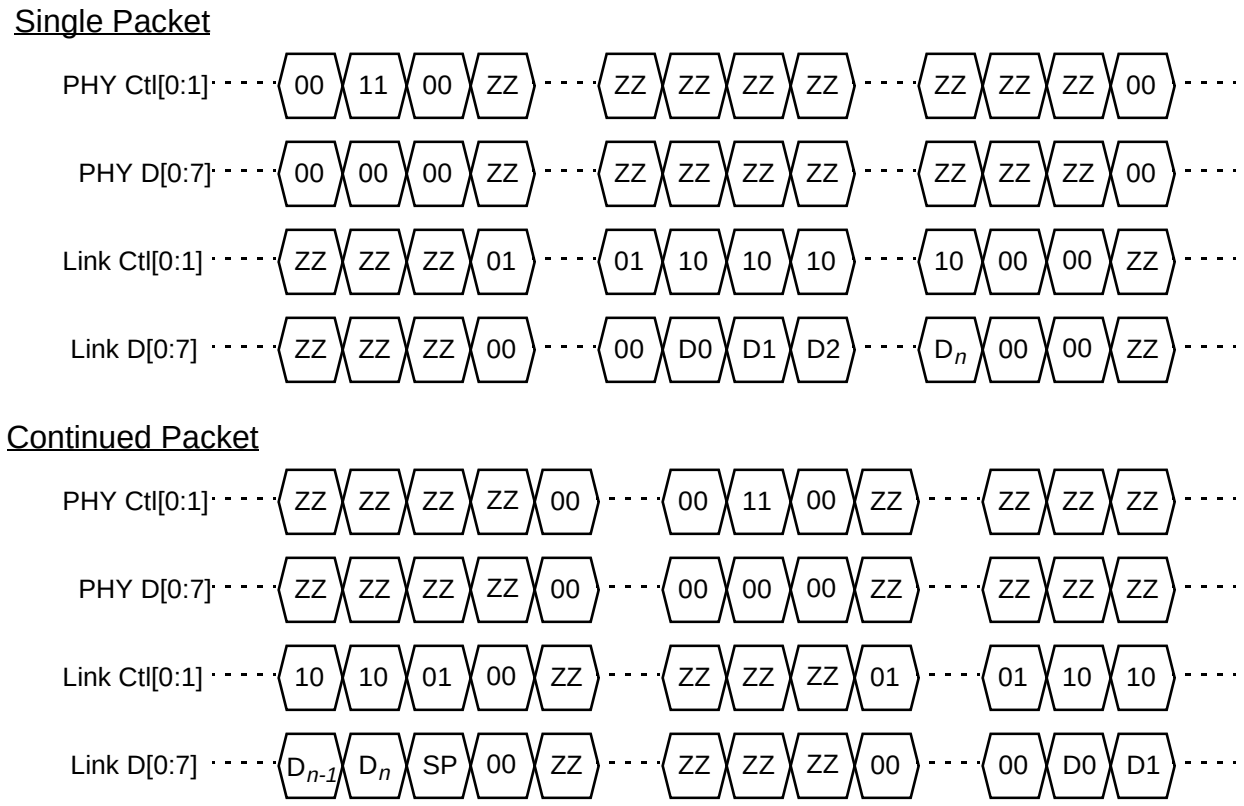


Figure 5-4 — Transmit timing

5.1.4 Receive

Whenever the PHY sees the “data-on” state on Serial Bus, it initiates a receive operation by asserting *receive* on the Ctl bus and 1’s on the D bus. The PHY indicates the start of a packet by placing the speed code (encoding shown in table 5-11) on the D bus, followed by the contents of the packet. The PHY holds the Ctl bus in *receive* until the last symbol of the packet has been transferred. The PHY indicates the end of the packet by asserting *idle* on the Ctl bus. Note that the speed code is a PHY-Link protocol and is not included in the calculation of the CRC or other data protection mechanisms.

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

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

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

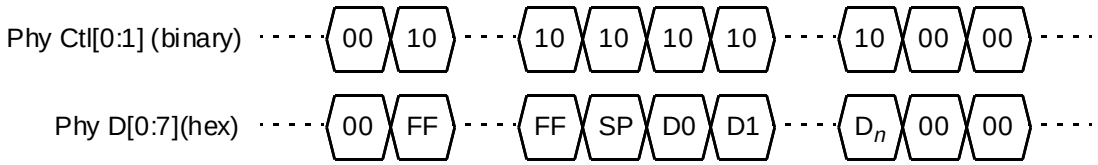


Figure 5-5 — Receive timing

The speed code for the receive operation is defined as shown in table 5-11. This is also the same speed encoding used by the link to signal speed to the PHY during concatenated packet transmission.

Table 5-11 — Speed code signalling

D[0:7]	Data rate
00xxxxxx[a]	S100
0100xxxx	S200
01010000	S400
01010001	S800
01010010	S1600
01010011	S3200
11111111	“data-on” indication

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

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

5.2 PHY register map

Although Annex J of IEEE Std 1394-1995, from which this clause is derived, originally described an interface to a discrete PHY, the material that follows is intended to be normative for both discrete and integrated PHY and link implementations. In addition, Link implementations shall provide a means for software or firmware to access to the PHY registers defined in the clauses that follow.

5.2.1 PHY register map (cable environment)

This supplement defines two PHY register maps for the cable environment, legacy and extended. The legacy PHY register map, illustrated by figure 5-6 below, is the same register map specified by informative Annex J of IEEE Std 1394-1995. The legacy register map is included to normalize existing PHY implementations at the time this supplement was developed. Newer PHY's that claim compliance with this supplement shall implement the extended PHY register map described later in this clause.

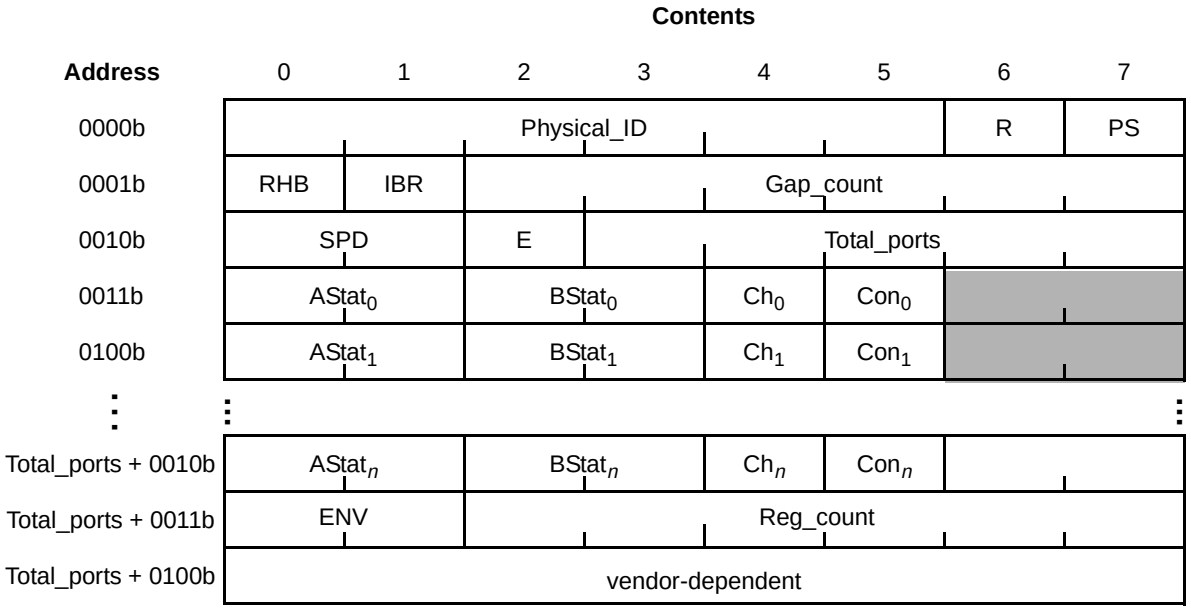


Figure 5-6 — Legacy PHY register map for the cable environment

Up to sixteen PHY registers may be defined in the format shown above. With the exception of the fields described in table 5-12 below, the PHY register fields in the cable environment have the same meaning for both the legacy and extended PHY register maps. Tables 5-13 and 5-14 summarize the PHY register fields for all register map formats in the cable environment.

Table 5-12 — Legacy PHY register fields for the cable environment

Field	Size	Type	Description
SPD	2	r	Indicates the maximum speed this PHY supports; the encoding is the same as for the speed code in the self-ID packet specified by IEEE Std 1394-1995.
E	1	r	If equal to zero, no enhanced registers are used. If equal to one, enhanced registers, *i.e.*, registers at address Total_Ports + 0011b and beyond are present.
Total_ports	5	r	The number of ports on this PHY. Also equal to the number of port status registers that follow. Total_ports shall be in the range one to 13 - E, inclusive.
ENV	2	r	Present if the E bit is one. ENV shall be equal to one in the cable environment; other values are reserved.
Reg_count	6	r	Present if the E bit is one. Indicates the number of enhanced registers that follow.

Note that the format of enhanced registers is vendor-dependent. Enhanced registers may be present regardless of the value of the E bit. If E is zero, all the enhanced registers are vendor-dependent. When E is one, a PHY register with the ENV and Reg_count fields is present; it specifies how many enhanced registers follow—but their format(s) are vendor-dependent.

In order to describe and manage PHY enhancements specified by this supplement (see 6., “Cable physical layer performance enhancement specifications,”), compliant PHY’s shall utilize the extended format illustrated by figure 5-7.

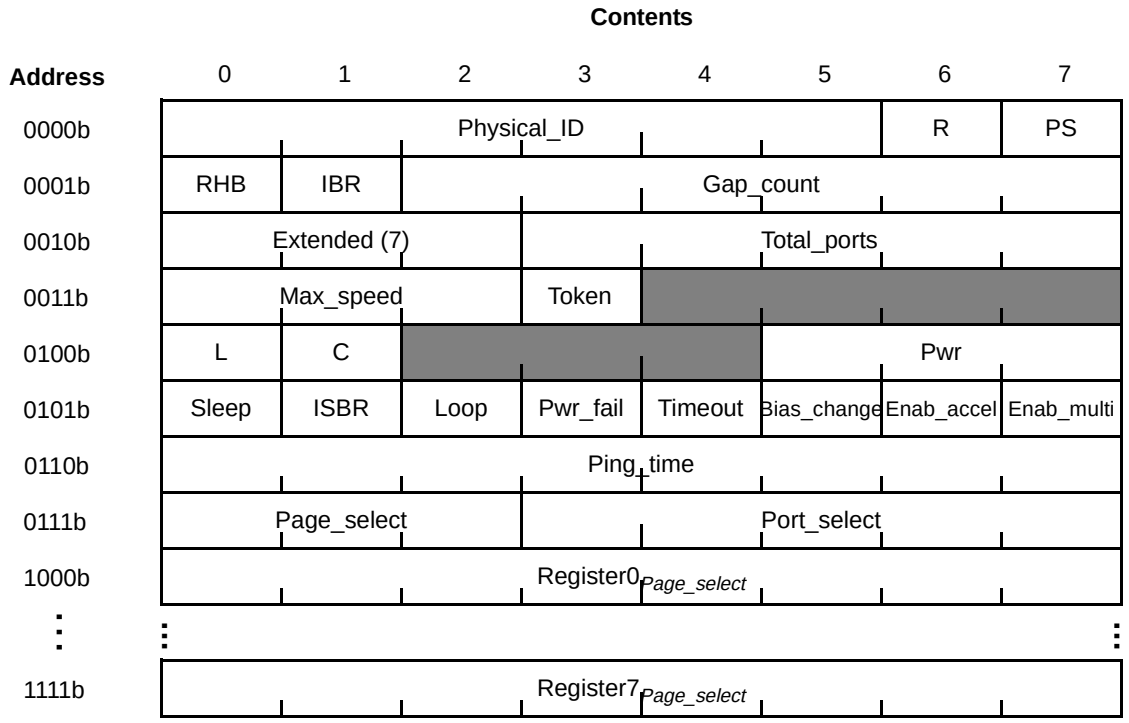


Figure 5-7 — Extended PHY register map for the cable environment

The meaning, encoding and usage of all the fields in the extended PHY register map is summarized by table 5-13 below. As previously described, many of these fields are also present in the legacy PHY register map; the descriptions below apply equally to both cases.

Table 5-13 — PHY register fields for the cable environment

Field	Size	Type	Description
Physical_ID	6	r	The address of this node determined during self-identification.
R	1	r	When set to one, indicates that this node is the root.
PS	1	r	Cable power status
RHB	1	rw	Root hold-off bit. When set to one, instructs the PHY to attempt to become the root during the next tree identify process.
IBR	1	rw	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 state for 166 us and is self-clearing.
Gap_count	6	rw	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.

Table 5-13 — PHY register fields for the cable environment (Continued)

Field	Size	Type	Description
Extended	3	r	This field shall have a constant value of seven, which indicates the extended PHY register map.
Total_ports	5	r	The number of ports implemented by this PHY.
Max_speed	3	r	Indicates the maximum speed this PHY supports; the encoding is the same as for the *xspd* bit in self-ID packet 7 (see clause 6.1.1).
L	1	rw	Link enabled. Default value of one subsequent to a power reset. Otherwise cleared or set by software to control the value of the L bit transmitted in the self-ID packet. The transmitted L bit shall be the logical AND of this bit and the LPS signal.
C	1	rw	Contender. Cleared or set by software to control the value of the C bit transmitted in the self-ID packet. The value of this bit subsequent to a power reset is implementation-dependent.
Pwr	3	rw	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	Initiate short (arbitrated) bus reset. A write of one to this bit requests the PHY to arbitrate and issue a short bus reset. This bit is self-clearing.
Loop	1	rw	Loop detect. A write of one to this bit clears it to zero.
Pwr_fail	1	rw	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	Arbitration state machine timeout. A write of one to this bit clears it to zero.
Bias_change	1	rw	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	Enable arbitration acceleration. When set to one, the PHY shall use the enhancements specified in X.
Enab_multi	1	rw	Enable multi-speed packet concatenation. When set to one, the Link shall signal the speed of all packets to the PHY.
Ping_time	8	r	Round-trip time of the last ping packet originated by this node.
Page_select	3	rw	Selects which of eight possible PHY register pages are accessible through the window at PHY register addresses 1000b through 1111b, inclusive.
Port_select	5	rw	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 1000b through 1111b, inclusive.

The *PHY_interrupt* state reported as bit 3 of PHY status transferred to the link is the logical OR of the *Loop*, *Pwr_fail*, *Timeout* and *Bias_change* bits in PHY register five.

The upper half of the PHY register space, addresses 1000b through 1111b, inclusive, provides a windows through which additional pages of PHY registers may be accessed. This supplement defines page zero, the port status page, which is used to access configuration and status information for each of the PHY's port. The port is selected by writing both Page_select and Port_select to the PHY register at address 0111b. The format of the port status page is illustrated by figure 5-8 below.

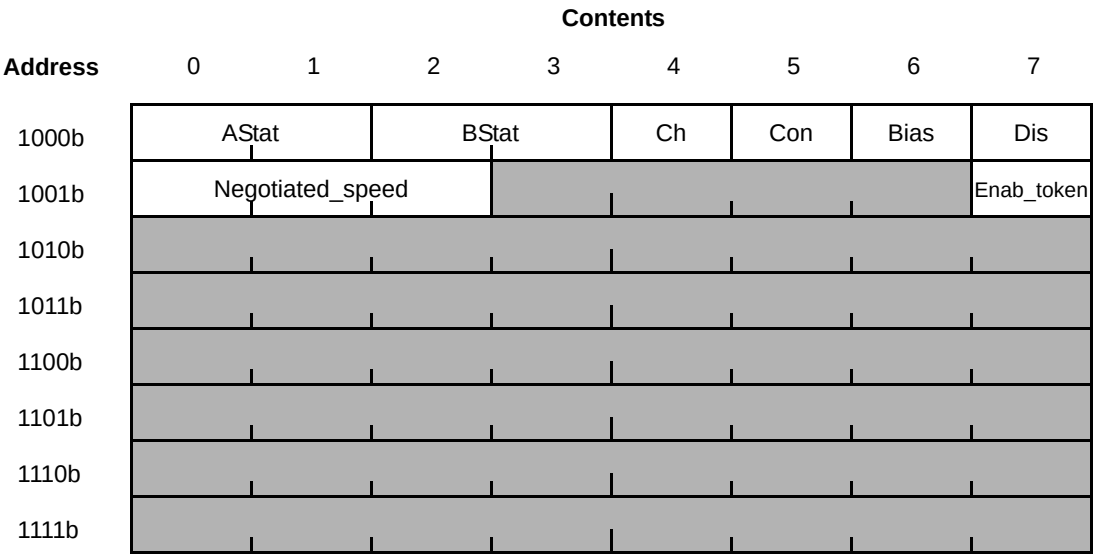


Figure 5-8 — PHY register page 0: Port Status page

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

Table 5-14 — PHY register Port Status page fields

Field	Size	Type	Description
AStat	2	r	TPA line state for the port: 00 = invalid 01 = 1 10 = 0 11 = 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 value reported by this bit is filtered by hysteresis logic to reduce multiple status changes caused by contact scrape when a connector is inserted or removed.
Bias	1	r	If equal to one, bias voltage is detected (possible connection). The value reported by this bit is unfiltered and represents the instantaneous state of detected bias voltage.
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 strappable 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 *xspd* bit in self-ID packet 7 (see clause 6.1.1).
Enab_token	1	rw	Enable token-style arbitration. When set to one, the enhancements specified in X shall be enabled for this port.

The Bias and Dis fields are not present in the legacy PHY port status registers; the meanings of all the other fields are identical for both the legacy and the extended PHY registers.

5.2.2 PHY register map for the 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.

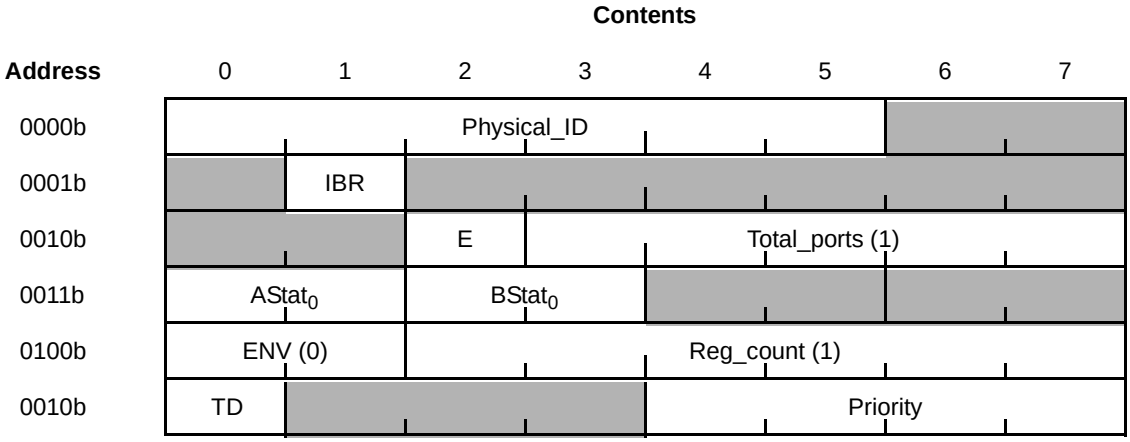


Figure 5-9 — PHY register map for the backplane environment

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 5-13).

Table 5-15 — 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.
E	1	r	If equal to zero, no enhanced registers are used. If equal to one, enhanced registers at address 0100b and 0101b 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 0110b 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.

5.2.3 Integrated link and PHY

The register map described in the preceding clauses was specified to guarantee 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 one of the register maps 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.

5.3 AC timing

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 and chip outputs can come directly from registers without combinational delay or additional loading. This configuration provides generous margins on setup and hold time. The timing defined assumes there some delay from the SClk input to the input registers due to a clock tree. As a result, the timing in table 5-16 provides for little or no setup time and generous hold time. The SClk output from the PHY should originate directly from the root of the internal clock tree of the PHY to provide similar insertion delay on inputs to the PHY.

The AC timing parameters for this interface also assumes a maximum delay through an isolation barrier, such as the example given in J.6, “Isolation barrier,” of IEEE Std 1394-1995.

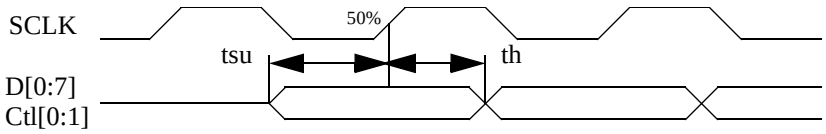


Figure 5-10 — SCLK duty cycle

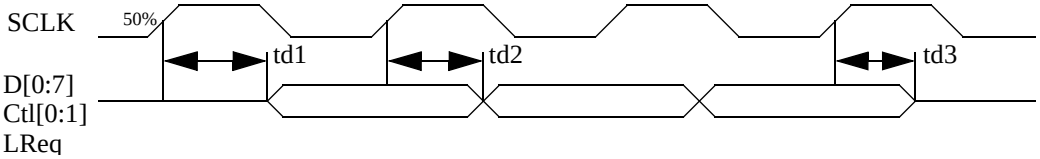


Figure 5-11 — Link to PHY transfer waveform

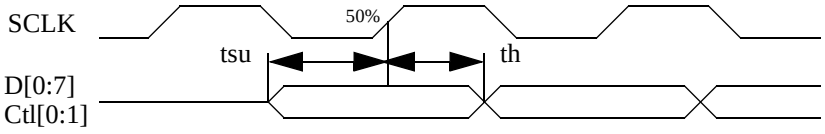


Figure 5-12 — PHY to link transfer waveform

Table 5-16 — AC timing parameters

Name	Description	Unit	Minimum	Nominal	Maximum
tc	SCLK cycle time	ns	20.24		20.45
tw(H)	Pulse duration, SCLK high	ns	9		11
tw(L)	Pulse duration, SCLK low	ns	9		11
td1	Delay time, SCLK high to D[0:7}, Ctl[0:1] and LReq valid	ns	3		14
td2	Delay time, SCLK high to D[0:7], Ctl[0:1] and LReq valid	ns	3		14
td3	Delay time, SCLK high to D[0:7], Ctl[0:1] and LReq invalid	ns	3		14
tsu	Setup time, D[0:7], Ctl[0:1] and LReq before SCLK	ns	6		
th	Hold time, D[0:7], Ctl[0:1] and LReq after SCLK	ns	0		
	Delay through isolation barrier	ns			3
	Hysteresis input rising threshold	V	Vcc/2 + 0.2		Vcc/2 + 1.3
	Hysteresis input falling threshold	V	Vcc/2 - 1.3		Vcc/2 - 0.2

6. 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 completely 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 PHY's 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. This contradiction has already resulted in observed interoperability problems with PHY's from different vendors, as different vendors attempted to sensibly interpret a nonsensical specification. New requirements for PHY's in this supplement correct the defect and promote interoperability between existing PHY's 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 furthest from the root is permitted to arbitrate before the others in a particular asynchronous or isochronous arbitration sequence. After this furthest node arbitrates, the nodes successively closer to the root employ ack-accelerated arbitration to concatenate their transmissions without the necessity for subaction gaps

— Transmission delay calculation (PHY pinging). The ability for a PHY to transmit a "ping" packet to another PHY and time its return permits the inclusion of cables longer than 4.5 meters or PHY's 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.

6.1 Cable PHY packets

This clause extends the definition of PHY packets and completely replaces IEEE Std 1394-1995 clause 4.3.4, "Cable PHY packets." For convenience of reference and to correct typographical errors, all of the existing PHY packet definitions are reproduced followed by the new definitions. With the exception of the expansion of the speed code to four bits in the self-ID packet, there are no substantive changes to the previously defined packet formats.

The cable physical layer sends and receives a small number of short packets which are used for bus management. These PHY packets all consist of 64 bits, with the second 32 bits being the logical inverse of the first 32 bits; they are all sent at the S100 speed. If the first 32 bits of a received PHY packet do not match the complement of the second 32 bits, the PHY packet shall be ignored.

The cable physical layer packet types are

a) The self-ID packet
b) The link on packet
c) The PHY configuration packet
d) The extended PHY configuration packets (which family includes, at present, only the ping packet)

NOTE—The PHY packets can be distinguished from the null-data isochronous packet (the only link packet with exactly two quadlets) since the latter uses a 32-bit CRC as the second quadlet, while the PHY packets use the bit inverse of the first quadlet as the second.

6.1.1 Self-ID packet

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

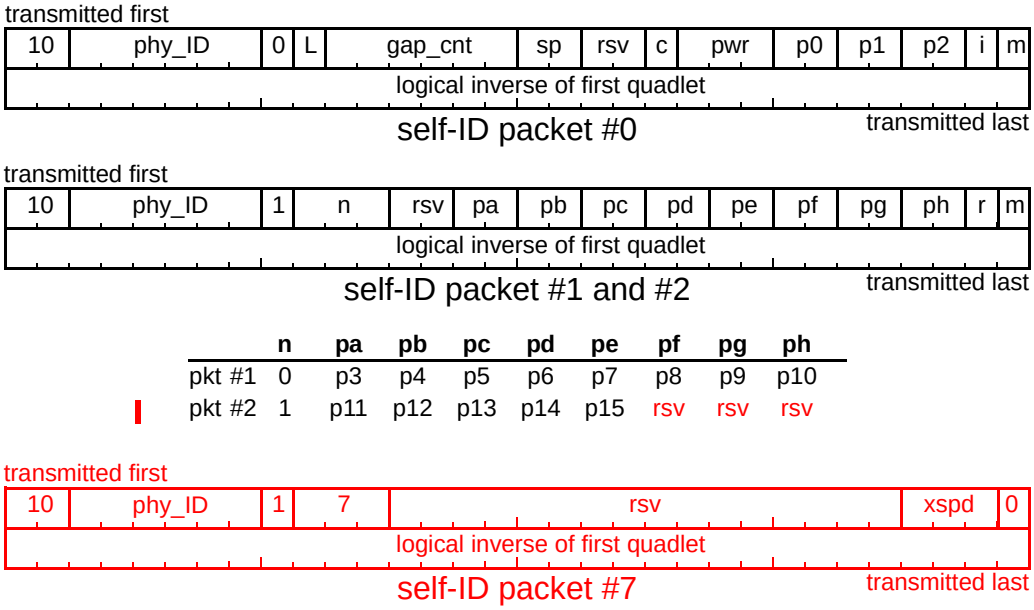


Figure 6-1 — Self-ID packet format

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

Table 6-1 — Self-ID packet fields

Field	Derived from	Comment
10		Self-ID packet identifier
phy_ID	`physical_ID`	Physical node identifier of the sender of this packet
L	`LPS` `PHY register L bit`	If set, this node has active Link and Transaction layers
gap_cnt	`gap_count`	Current value for this nodes' PHY_CONFIGURATION.*gap_count* field.
sp	`PHY_SPEED`	Speed capabilities: 00 98.304 Mbit/s 01 98.304 and 196.608 Mbit/s 10 98.304, 196.608 and 393.216 Mbit/s 11 Extended speed capabilities reported in self-ID packet 7
c	`CONTENDER`	If set and the link_active flag is set, this node is a contender for the bus or isochronous resource manager as described in clause 8.4.2 of IEEE Std 1394-1995.

Table 6-1 — Self-ID packet fields (Continued)

Field	Derived from	Comment
pwr	`POWER_CLASS`	Power consumption and source characteristics: 000 Node does not need power and does not repeat power. 001 Node is self-powered and provides a minimum of 15 W to the bus. 010 Node is self-powered and provides a minimum of 30 W to the bus. 011 Node is self-powered and provides a minimum of 45 W to the bus. 100 Node may be powered from the bus and is using up to 1 W. 101 Node is powered from the bus and is using up to 1 W. An additional 2 W is needed to enable the Link and higher layers[a]. 110 Node is powered from the bus and is using up to 1 W. An additional 5 W is needed to enable the Link and higher layers[a]. 111 Node is powered from the bus and is using up to 1 W. An additional 9 W is needed to enable the Link and higher layers[a].
p0 … p15	`NPORT`, `child[NPORT]`, `connected[NPORT]`	Port status: 11 Connected to child node 10 Connected to parent node 01 Not connected to any other PHY 00 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 returned as a 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
xspd	`PHY_SPEED`	Extended speed capabilities: 000 98.304 Mbit/s 001 98.304 and 196.608 Mbit/s 010 98.304, 196.608 and 393.216 Mbit/s 011 98.304, 196.608, 393.216 and 786.43 Mbit/s 100 98.304, 196.608, 393.216, 786.432 and 1,572.864 Mbit/s 101 98.304, 196.608, 393.216, 786.432, 1,572.864 and 3,145.728 Mbit/s All other values are reserved for future definition
r, rsv		reserved for future definition, set to zeros

a. The link and higher layers are enabled by the link-on PHY packet described in clause 6.1.2 of IEEE Std 1394-1995.

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

6.1.2 Link-on packet

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

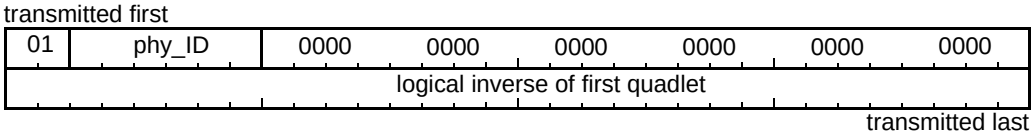


Figure 6-2 — Link-on packet format

Table 6-2 — Link-on packet fields

Field	Derived from	Comment
01		Link-on packet identifier
phy_ID	`physical_ID`	Physical node identifier of the destination of this packet

6.1.3 PHY configuration packet

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

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

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

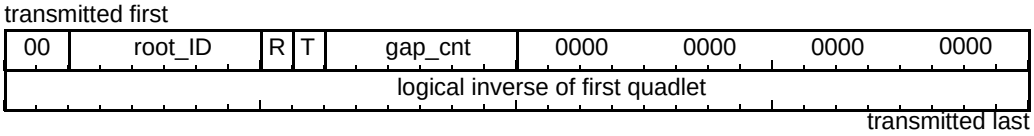


Figure 6-3 — PHY configuration packet format

Table 6-3 — PHY configuration packet fields

Field	Derived from	Comment
00		PHY configuration packet identifier
root_ID	`physical_ID`	Physical_ID of node to have its force_root bit set (only meaningful if R bit set)
R	`R`	If set, then the node with its physical_ID equal to this packet's root_ID shall have its force_root bit set, all other nodes shall clear their force_root bit. If cleared, the root_ID field shall be ignored.
T	`T`	If set, all nodes shall set their PHY_CONFIGURATION.gap_count field to the value in this packet's gap_count field
gap_cnt	`gap_count`	New value for all nodes' PHY_CONFIGURATION.gap_count field. This value goes into effect immediately on receipt and stays valid after the next bus reset (gap_count is set to 63 after a second bus reset without an intervening PHY configuration packet as described in `reset_start_actions()` in clause 6.3.1.1.2 and `receive_actions()` in clause 6.3.1.3.3)

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

6.1.4 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 (in combination with the R and T bits, 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.

6.1.5 Ping packet

The reception of the cable PHY packet shown in figure 6-4 shall cause the node identified by phy_ID to transmit a self-ID packet that reflects the current configuration and status of the PHY. Because of other actions, such as the receipt of a PHY configuration packet, the self-ID packet transmitted may differ from that of the most recent self-identify process.

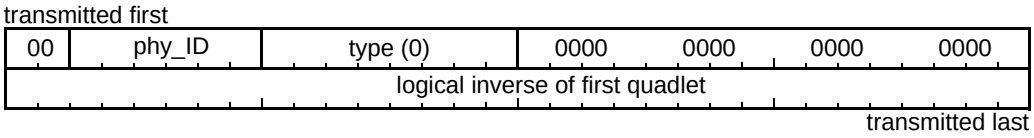


Figure 6-4 — Ping packet format

Table 6-4 — Ping packet fields

Field	Derived from	Comment
00		PHY configuration packet identifier
phy_ID	`physical_ID`	Physical node identifier of the destination of this packet
type		Extended PHY configuration packet type

6.2 Cable PHY timing constants

This clause defines new values 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 6-5 — Cable PHY timing constants

Timing constant	Minimum	Maximum	Comment
CONNECT_TIMEOUT	336.0 µs	341.3 µs	Connection debounce time
RESET_DETECT	80.0 µs	85.3 µs	Time for a connected node to confirm a reset signal
`SHORT_RESET_TIME`	1.3 µs	1.4 µs	Short reset hold time (~128/BASE_RATE)
`RESET_WAIT`	0.16 µs	0.16 µs	Reset wait delta time (~16/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.

6.3 Cable physical layer operation

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

— 4.4.1, “Data transmission and reception”; and
— 4.4.2.2, “Tree identify”

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

— The duration of the bus reset signal may be reduced to approximately 1.3 µs if the port that initiates the reset first arbitrates for the bus; and

— The probability of a multitude of bus resets during and subsequent to a physical cable insertion or removal is reduced.

The operation of the cable physical layer can best be understood with reference to the architectural diagram shown in figure 6-5:

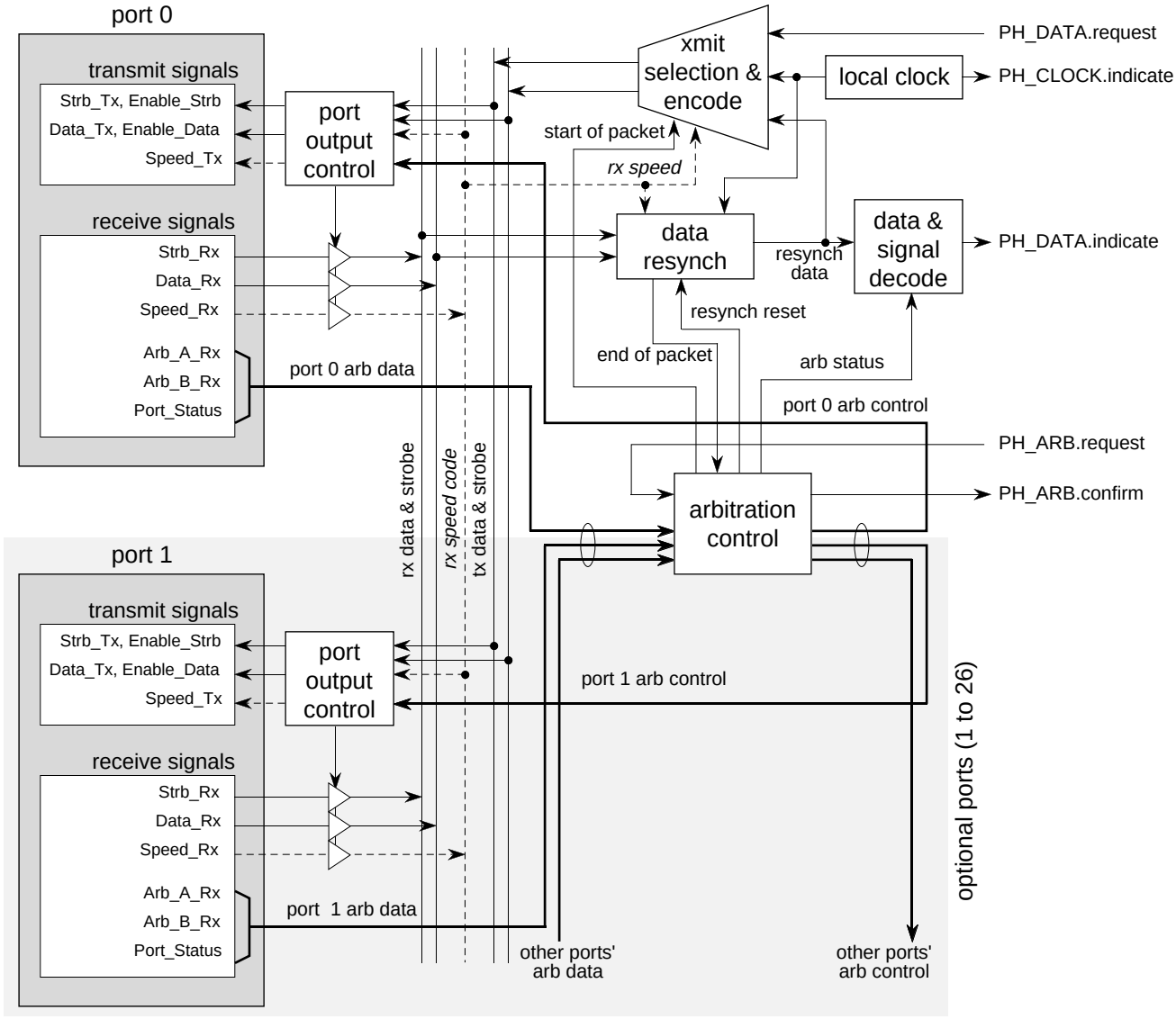


Figure 6-5 — Cable physical layer architecture

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 the attached 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 6.3.1

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 attached 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 following definitions:

Table 6-6 — Cable PHY packet definitions (Sheet 1 of 2)

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

Table 6-6 — Cable PHY packet definitions (Sheet 2 of 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;
```

Table 6-7 — Cable PHY code definitions

```
enum phyData {L, H, Z};             // Types of data to transmit
enum speedCode {S100, S200, S400};  // Speed codes
struct portData {phyData TpA; phyData TpB;};    // Port data structure
boolean random_bool();              // Returns a random TRUE or FALSE value
void portT(int port_number, portData txData);   // Transmit txData on indicated port
portData portR(int port_number);    // Return current rxData signal from indicated port
void portTspeed (int port_number, speedCode speed); // Set transmit speed on indicated port
speedCode portRspeed(int port_number);  // Return current speed from indicated port
int fifo_rd_ptr, fifo_wr_ptr;       // Data resynch buffer pointers
const int FIFO_DEPTH = 8;           // IMPLEMENTATION-DEPENDENT!
dataBit fifo[FIFO_DEPTH];           // Data resynch buffer
boolean waiting_for_data_start;     // First data bit not yet received
speedCode rx_speed;                 // Speed of received packet
timer arb_timer();                  // Timer for arbitration state machines
boolean root_test;                  // Flag that is randomly set during root contention
baserate_time contend_time;         // Amount of time to wait during root contention
int child_count;                    // Number of child ports
int lowest_unidentified_child;      // Lowest numbered active child that has not sent its self-ID
boolean all_child_ports_identified; // Set if all child ports have been identified
boolean self_ID_complete;           // Set if the self_ID transmission is complete
int requesting_child;               // Lowest numbered requesting child
boolean force_root;                 // Set to delay start of tree-ID process for this node
boolean end_of_reception;           // Set when reception of packet is complete
boolean link_req_active;            // Set when a request has started for the local link layer
boolean arb_enable;              // Set when a node only needs to wait for a subaction gap to arbitrate
boolean old_connect[NPORT];         // Connection history for each port
boolean isolated_node;              // Set if no ports connected
boolean bus_initialize_active;      // Set while the PHY is initializing the bus
boolean gap_count_reset_disable;    // If set, a bus reset will not force the gap_count to the maximum
```

6.3.1 Cable environment arbitration

The cable environment supports the immediate, priority, isochronous and fair arbitration classes. Immediate arbitration is used to transmit an acknowledge immediately after packet reception; the bus is expected to be available. Priority arbitration is used by the root for cycle start requests or may be used by any node to override fair arbitration. Isochronous arbitration is permitted between the time a cycle start is observed and the subaction gap that concludes an isochronous cycle; 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.

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

6.3.1.1 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.

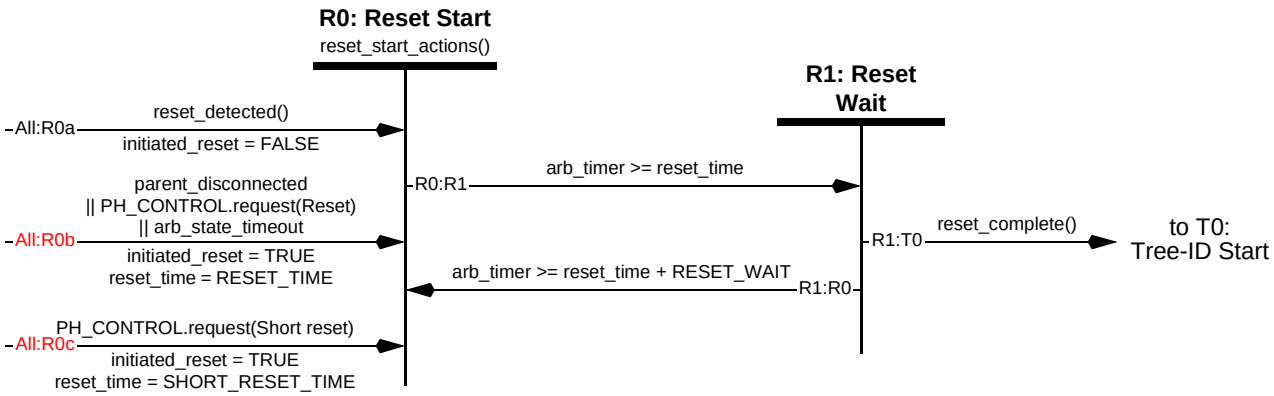


Figure 6-6 — Bus reset state machine

6.3.1.1.1 Bus reset state machine notes

Transition All:R0a. 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:R0b. 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 other than Idle, Transmit or Receive for longer than MAX_ARB_STATE_TIME.

Transition All:R0c. This is the entry point to the bus reset process if this node is initiating the process as the result of a Serial Bus management request for an arbitrated reset.

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 directly attached 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 attached peer PHY's 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 attached peer PHY's are in reset wait or sta.rting the tree ID process).

6.3.1.1.2 Bus reset actions and conditions

Table 6-8 — Bus reset actions and conditions (Sheet 1 of 2)

```
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 &= (port_status[i] != DISCONNECTED);
      if (port_status[i] == IN_PROGRESS) {
         if (!connected[i])
            port_status[i] = DISCONNECTED;  // Lost attempted connection
         else if (connect_timer >= CONNECT_TIMEOUT)
            port_status[i] = CONNECTED; // Confirmed connection
      } else if (port_status[i] == DISCONNECTED) {
         if (connected[i]) {            // Possible new connection?
            connect_timer = 0;          // Start connect timer
            port_status[i] = IN_PROGRESS;
         }
      } else if (!connected[i]) {       // Disconnect?
         portStatus[i] = DISCONNECTED;  // Effective immediately!
         if (child[i]) {
            child_disconnected = TRUE;  // Attempt arbitrated (short) reset
            arb_req = PRIORITY_REQ;
            reset_time = SHORT_RESET_TIME;
         } else
            parent_disconnected = TRUE; // Long reset is inevitable
      }
   }
}

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

   for (i = 0; i < NPORT; i++)
      if (portR(i) == BUS_RESET)    // More than 20 ns (transient DS == 11)
         switch (portStatus[i]) {
            case DISCONNECTED:
               break;               // Ignore reset if connection not already present

            case IN_PROGRESS:
               reset_time = 0;
               if (isolated_node)
                  reset_time = SHORT_RESET_TIME;
               else if (connect_timer >= 85 ms)
                  reset_time = RESET_TIME;
               return(reset_time != 0);
               break;

            case CONNECTED:
               reset_time = (PHY_state == RX) ? SHORT_RESET_TIME : RESET_TIME;
               return(TRUE);
               break;
         }
   return(FALSE);
}

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

   physical_ID = 0;
   child_count = 0;
   bus_initialize_active = TRUE:
```

Table 6-8 — Bus reset actions and conditions (Sheet 2 of 2)

```
   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 for reset_time + RESET_WAIT
   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) && connected[i])
         return(FALSE);
   return(TRUE);
}
```


6.3.1.2 Self identify

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

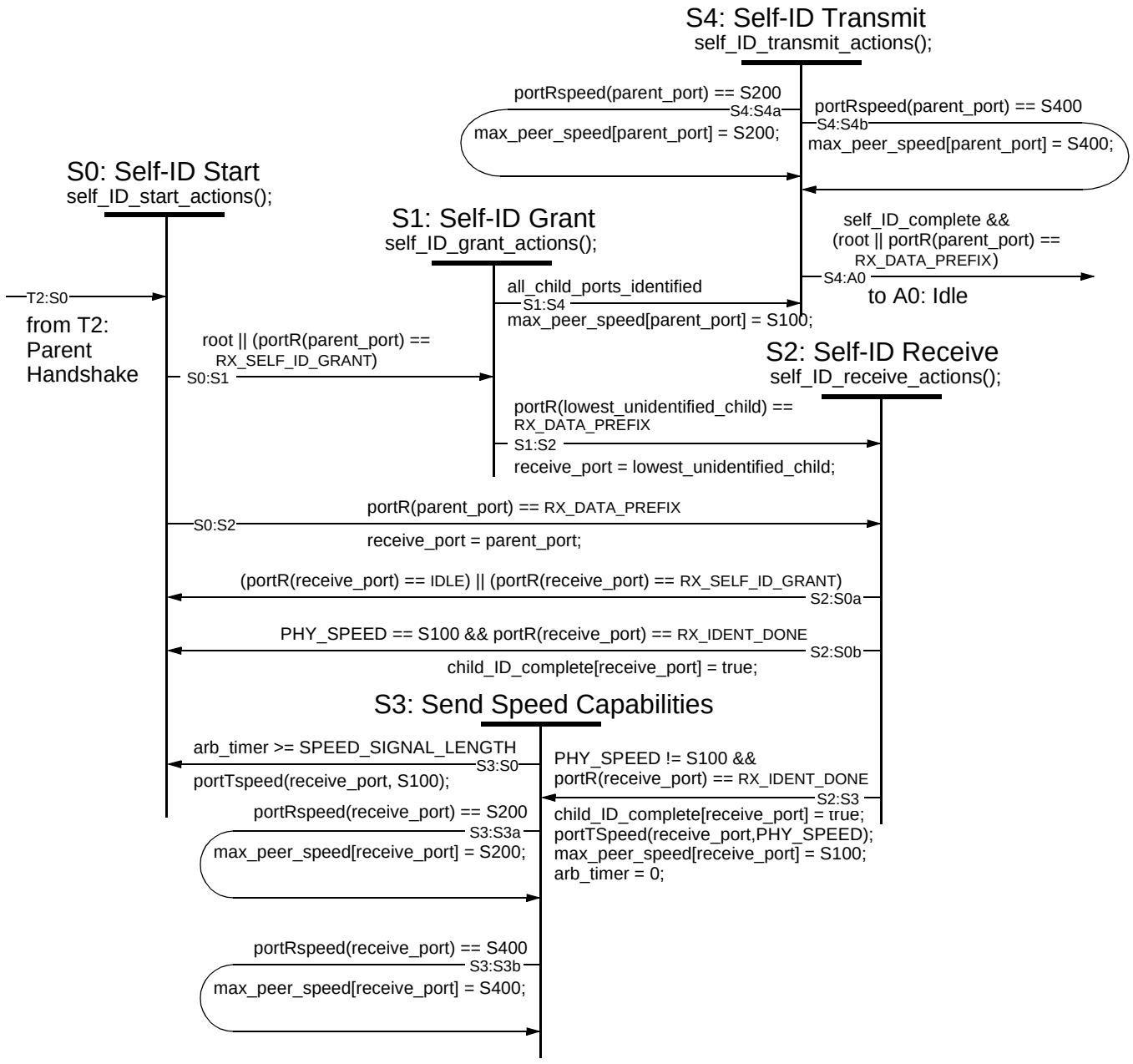


Figure 6-7 — Self-ID state machine

NOTE—This state machine does not include the reset conditions, since that would result in an overly-complex figure. See the All:R0a and All:R0b transition statements on page 61.

6.3.1.2.1 Self-ID state machine notes

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

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

Transition S0:S2. If a node receives a RX_DATA_PREFIX signal (10) from its parent, it knows that a self-ID packet is coming from a node in another branch in the tree.

State S1: Self-ID Grant. This state is entered when a node is given permission to send a self-ID packet. If it has any unidentified children, it sends a TX_GRANT signal (Z0) to the lowest numbered of those. All other connected ports are sent a TX_DATA_PREFIX signal (01) to warn them of the start of a self-ID packet.

Transition S1:S2. When the PHY receives a RX_DATA_PREFIX signal (10) from its lowest numbered unidentified child, it enters the Self-ID Receive state.

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

State S2: Self-ID Receive. As data bits are received from the bus they are passed on to the link layer as PHY data indications. This process is described in clause 4.4.1.2 of IEEE Std 1394-1995. Note that multiple self-ID packets may be received in this state.

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

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

Transition S2:S3. If the PHY gets an RX_IDENT_DONE (Z1) signal from the receiving port and the node is capable of running at the S200 or S400 data rates, it flags that port as identified and starts sending the speed capabilities signal. It also starts the speed signaling timer and sets the port speed to the S100 rate.

State S3: Send Speed Capabilities. If a node is capable of sending data at a higher rate 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_TIME.

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

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

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

State S4: Self-ID Transmit. At this point, all child ports have been flagged as identified, so the PHY can now send its own self-ID packet (see clause 6.1) using the process described in clause 4.4.1.1 of IEEE Std 1394-1995. When a non-root node is finished, it sends a TX_IDENT_DONE signal (1Z) and a speed capability signal as defined in clause 4.2.2.3 of IEEE Std 1394-1995 to its parent and IDLE (ZZ) to its children. The speed capability signal is transmitted for a fixed time duration (SPEED_SIGNAL_LENGTH). Simultaneously it monitors the bus for a speed capability transmission from the parent. The root node just sends IDLE (ZZ) to its children. Note that the children will then enter the Idle state described in the next clause, but they will never start arbitration since an adequate arbitration gap will never open up until the Self-ID process is completed for all nodes.

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

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

Transition S4:A0. The PHY then enters the Idle state described in the next clause when the self-ID packet has been transmitted **and** if either of the following conditions are met:

1) The node is the root. When the root enters the Idle state, all nodes are now sending IDLE signals (ZZ) and the gap timers will eventually get large enough to allow normal arbitration to start.
2) The node starts to receive a new self-ID packet (RX_DATA_PREFIX – 10). This will be the self-ID packet for the parent node or another child of the parent. This event shall cause the PHY to transition immediately out of A0:Idle into A5:Receive.

6.3.1.2.2 Self-ID actions and conditions

Table 6-9 — Self ID actions and conditions (Sheet 1 of 2)

```
void self_ID_start_actions() {
   int i;

   all_child_ports_identified = TRUE;   // Will be reset if any active children are unidentified
   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_port))
         portT(i, TX_GRANT);// Send grant to lowest unidentified port (if any)
      else
         portT(i, TX_DATA_PREFIX);// Otherwise, tell others to prepare for packet
}

void self_ID_receive_actions() {
   int i;

   portT(receive_port, IDLE);       // Turn off grant, get ready to receive
   receive_actions();               // Receive (and repeat) packet
   physical_ID = physical_ID + 1;   // Increment 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;

   self_ID_complete = FALSE;
   receive_port = NPORT;            // Indicate that we are transmitting (no port has this number)
   start_tx_packet(S100);           // Send data prefix and 98.304 Mbit/sec speed code
   for (SID_pkt_number = 0; SID_pkt_number <= last_SID_pkt; SID_pkt_number++) {
      selfID.dataQuadlet = 0;       // Clear all zero fields in self ID packet
      selfID.type = 0b10;
      selfID.phy_ID = physical_ID;
      if (SID_packet_number == 0) { // First self ID packet?
         selfID.L = link_pwr;       // Link active or not
         selfID.gap_cnt = gap_count;
         selfID.sp = PHY_SPEED;
         selfID.del = PHY_DELAY;
         selfID.c = CONTENDER;
         selfID.pwr = POWER_CLASS;
         selfID.i = initiated_reset;
      } else {
         selfID.seq = 1;            // Indicates second and subsequent packets
         selfID.n = SID_pkt_number - 1; // Sequence number
      }
```


Table 6-9 — Self ID actions and conditions (Sheet 2 of 2)

```
      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 */
      } 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
      }
   }
   if (!ping_response)          // Skip if self-ID packet was in response to a ping
      for (port_number = 0; port_number < NPORT; i++) {
         if (port_number == parent_port) {
            portT(i, TX_IDENT_DONE);            // Notify parent that self-ID is complete
            portTspeed(port_number, PHY_SPEED); // Send speed signal (if any)
            wait_time (SPEED_SIGNAL_LENGTH);
            portTspeed(port_number, S100);      // Stop sending speed signal
            PH_EVENT.ind(SELF_ID_COMPLETE, physical_ID, root);
         } else
            portT(port_number, IDLE);           // Turn off transmitters to others
      }
   self_ID_complete = TRUE;                     // Signal completion
}

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

   for (i = 0; i < 32; i++) {
      tx_bit(quad_data & 0x80000000);           // Send high order bit
      quad_data <<= 1;                          // Shift to position next bit
   }
}
```

6.3.1.3 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).

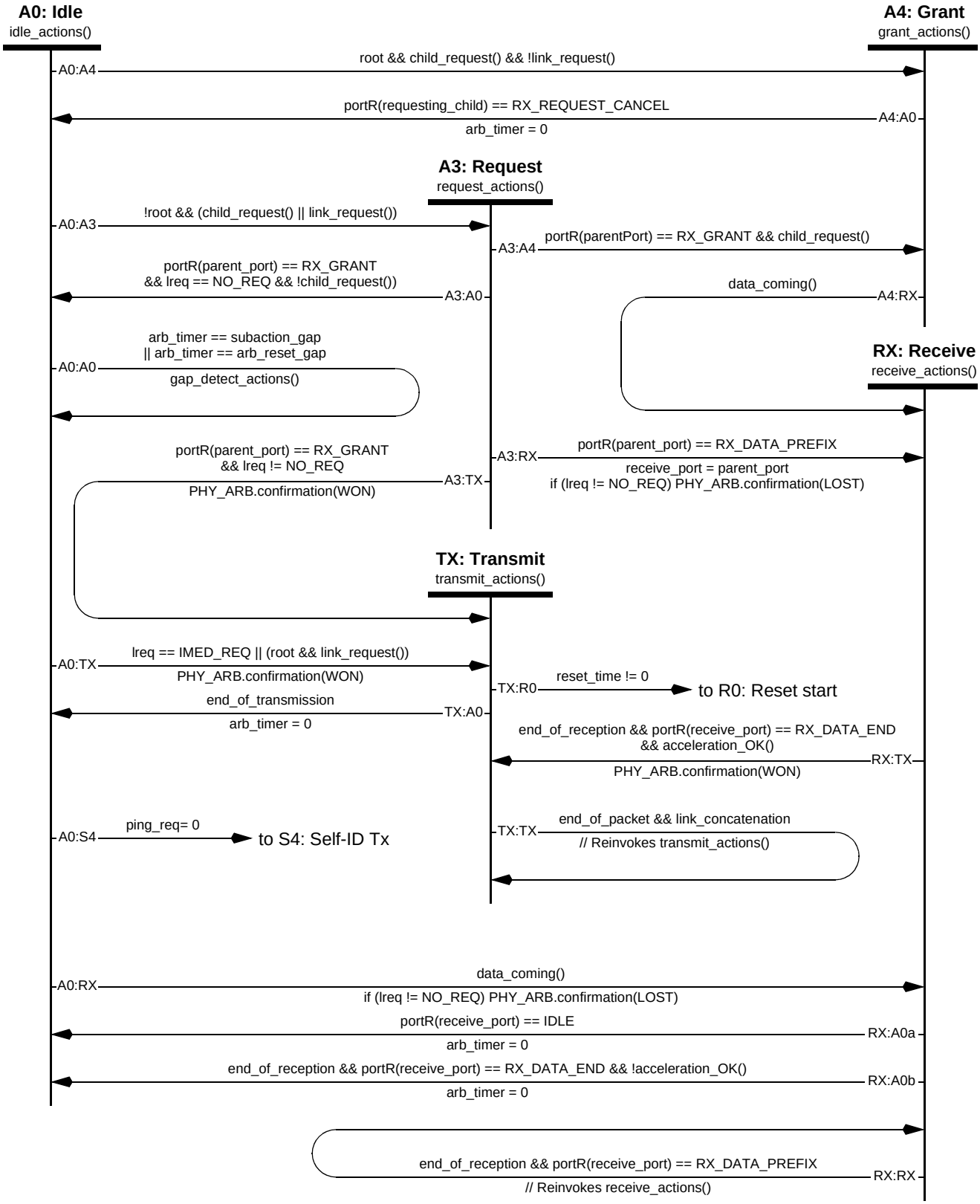


Figure 6-8 — Cable arbitration state machine

6.3.1.3.1 Normal arbitration state machine notes

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

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

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

Transition A0:A3. If the PHY has a queued isochronous request or receives a RX_REQUEST signal (0Z) from one of its children and the node is not the root, it passes the request on to its parent.

Transition A0:A4. If, on the other hand, the PHY receives a RX_REQUEST signal (0Z) from one of its children and the node **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 attached link layer needed to send an acknowledge), the PHY notifies the link layer that it is ready to transmit and enters the Transmit state.

State A3: 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 A3: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 A3:A4. 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 A3: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 attached link layer was making the request, it is notified.

Transition A3:TX. If the PHY receives a RX_GRANT signal (00) from its parent and the attached 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 A4: 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 A4: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 A4: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 attached link layer to send new requests if there were any queued) notifies the attached link layer that the bus is busy and starts the packet receive process described below. Note that the packet received could be a PHY packet (self-ID, link-on or PHY configuration), acknowledge, or normal data packet. PHY configuration and link-on packets are interpreted by the PHY, as well as being passed on to the link layer.

Transition RX:A0a. If a packet ends and the received signal is RX_DATA_END (01), then the receive process is complete and the Idle state is entered.

Transition RX:A0b. If transmitting node stops sending any signals (received signal is ZZ), then it is releasing the bus so the PHY can return to Idle state.

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

State TX: Transmit. Unless an arbitrated (short) bus reset has been request, 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.

6.3.1.3.2 Normal arbitration actions and conditions

Table 6-10 — Normal arbitration actions and conditions (Sheet 1 of 2)

```
void link_req() {       // Parse serial LReq input into parameters
   switch (req_type) {          // 3-bit Request Type at the start of LReq
      case IMMED_REQ:
      case ISOCH_REQ:
      case PRIORITY_REQ:
      case FAIR_REQ:
         if (lreq == NO_REQ) {
            lreq = req_type;    // Request Type from LReq transfer
            speed = req_speed;  // Request Speed from LReq transfer
         }
         break;

      case CYCLE_SYNCH:
         cycle_sync = TRUE;     // Will clear when subaction gap detected
         break;

      case RD_REG:              // How are these handled if the PHY is not idle?
         break;

      case WR_REG:
         break;
   }
}

boolean acceleration_OK() {         // TRUE if either ACK or fly-by acceleration OK

   if (receive_port == parent_port)
      return(FALSE);
   else if (cycle_synch)
      return(lreq == ISOCH_REQ);
   else if (ack)
      return(lreq == PRIORITY_REQ || arb_enable);
   else
      return(FALSE);
}

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

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

boolean 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
```

Table 6-10 — Normal arbitration actions and conditions (Sheet 2 of 2)

```
     PH_DATA.indication(RESET_GAP);     // Alert link
  } else if (arb_timer >= subaction_gap_time) {
     PH_DATA.indication(SUBACTION_GAP); // Notify link
     if (bus_initialize_active) {       // End of self-identify process?
        PH_EVENT.indication(BUS_RESET_COMPLETE);
        bus_initialize_active = FALSE;
     }
  }
}

void idle_actions() {
   int i;

   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] && (lreq != NO_REQ || 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 link_request() {              // TRUE if OK to request the bus
   if (lreq == NO_REQ || lreq == IMMED_REQ)
      return(FALSE);
   else if (lreq == ISOCH_REQ)
      return(TRUE);
   else if (arb_timer >= arb_reset_gap_time + arb_delay)
      return(TRUE);
   else if (lreq == FAIR_REQ && !arb_enable)
      return(FALSE);
   else if (arb_timer == subaction_gap_time + arb_delay)
      return(TRUE);
   else if (ack && !cycle_synch && arb_timer < subaction_gap_time)
      return(TRUE);
   else
      return(FALSE);
}

void grant_actions() {
   int i;

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


6.3.1.3.3 Receive actions and conditions

Table 6-11 — Receive actions and conditions (Sheet 1 of 2)

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

   cycle_master_req = fair_req = FALSE; // Cancel requests
   if (!root)
      pri_req = FALSE;                  // Clear priority request if not root
   PH_DATA.ind(DATA_PREFIX);            // Send notification of bus activity
   rx_speed = start_rx_packet();        // Start up receiver and repeater
   PH_DATA.ind(DATA_START(rx_speed));   // Send speed indication
   ping_timer_enable = FALSE;           // Halt ping timer
   do {
      rx_bit(&rx_data, &end_of_data);
      if (!end_of_data) {               // Normal data, send to link layer
         PH_DATA.ind(rx_data);
         if (bit_count == 8) {          // After 8 bits, more are coming...
            accFair_req = FALSE;        // Clear request variables
            accPri_req = FALSE;
            pri_req = FALSE;
            ack = FALSE;
         }
         if (bit_count < 64)            // Accumulate first 64 bits
            rx_phy_pkt.bits[bit_count++] = rx_data;
         else
            bit_count++;
      }
   } while (!end_of_data);
   switch(portR(receive_port)) {        // Send appropriate end of packet indicator
      case RX_DATA_PREFIX:
         PH_DATA.ind(DATA_PREFIX);      // Concatenated packet coming
         break;

      case RX_DATA_END:
         if (iso_req || (ack && (   accpri_req
                                 || (pri_req && root)
                                 || (accfair_req && arb_enable))))
            ;                           // Transition to transmit state next
         else
            PH_DATA.ind(DATA_END);      // Normal end of packet
         break;
   }
   stop_rx_packet();
   end_of_reception = TRUE;
   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.check_bits[i + 32])
            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.ext_type == 1)
               ping_command = (rx_phy_pkt.proxy_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;
               }
```

Table 6-11 — Receive actions and conditions (Sheet 2 of 2)

```
            }
            break;

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


6.3.1.3.4 Transmit actions and conditions

Table 6-12 — Transmit actions and conditions

```
void transmit_actions() {

   boolean imm_req = FALSE, ping_timer_enable = FALSE, test_end = FALSE;
   int bit_count = 0, i;
   PHY_packet rx_phy_pkt, tx_phy_pkt;
   phyData data_to_transmit;

   if (fair_req)
      arb_enable = fair_req = FALSE;
   isoch_req = FALSE;
   receive_port = NPORT;        // Impossible port number ==> PHY transmitting
   if (ping_command) {
      start_tx_packet(S100);
      tx_phy_pkt.dataQuadlet = 0;
      tx_phy_pkt.ext_type = byte2;   // Physical ID of node to be pinged
      tx_quadlet(tx_phy_pkt);
      tx_quadlet(~tx_phy_pkt);
      stop_tx_packet(tx_data_end, S100);
   } else {
      start_tx_packet(req_speed);   // Send data prefix & speed signal
      while (~test_end) {
         PH_CLOCK.ind;              // Tell link to send data
         while (~PH_DATA.req & (data_to_transmit)); // Wait for data from 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;
               break;

            case DATA_PREFIX:
               stop_tx_packet(DATA_PREFIX);
               wait_time(min_packet_separation);    // Hold bus for concatenated packet
             start_tx_packet(req_speed);         // Send data prefix and speed signal for next packet
               break;

            case DATA_END:
               stop_tx_packet(data_end);
               test_end = TRUE;         // End of packet indicator
               if (bit_count == 64) {   // We have transmitted a PHY packet
                  // FIX THIS STUFF! //
                  for (i = 0; i < 32; i++)
                     if (rx_phy_pkt.dataBits[i] == rx_phy_pkt.checkBits[i])
                        break;
                  if (rx_phy_pkt.type == 0b00)
                     if (rx_phy_pkt.ext_type == 0) {
                        ping_timer_enable = TRUE;      // Clear and enable timer
                        ping_response = (rx_phy_pkt.phy_ID == physical_ID);
                     } else {
                        if (rx_phy_pkt.R)
                           force_root = (rx_phy_pkt.root_ID == physical_ID);
                        if (rx_phy_pkt.T) {
                           gap_count = rx_phy_pkt.gap_cnt;
                           gap_count_reset_disable = TRUE;
                     }
                  }
               break;
         }
   end_of_transmission = TRUE;
}
```

6.4 Port disable

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

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

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

Figure 6-9 below illustrates a possible implementation of the port disable logic.

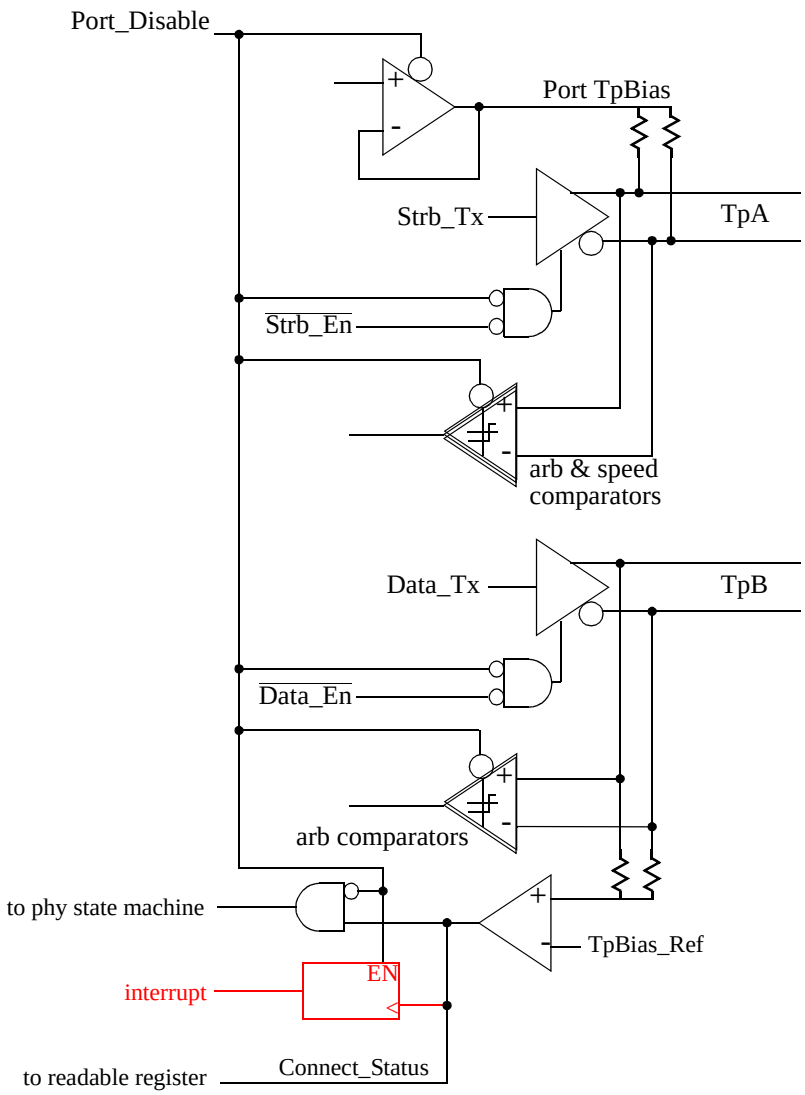


Figure 6-9 — Port disable logic

7. Isochronous connection management specifications

Need to add a model or overview description here....

7.1 Plug control registers

The CSR Architecture reserves a portion of the initial register space for bus-dependent uses. The addresses of these bus-dependent registers are specified in terms of byte offsets within the initial register space, where the base of the initial register space (from the beginning of the initial node space) is FFFF F000 0000_{16}. Table 7-1 summarizes the optional bus-dependent registers specified by this supplement.

Table 7-1 — Plug control registers

Offset	Name	Notes
900_{16}	OUTPUT_MASTER_PLUG	Common output plug controls for the node
904_{16}–$97C_{16}$	OUTPUT_PLUG	Output plug control registers for individual isochronous channels, OUTPUT_PLUG[0] through OUTPUT_PLUG[30]
980_{16}	INPUT_MASTER_PLUG	Common input plug controls for the node
984_{16}–$9FC_{16}$	INPUT_PLUG	Input plug control registers for individual isochronous channels, INPUT_PLUG[0] through INPUT_PLUG[30]

A node that implements plug control registers shall support quadlet read requests for all implemented registers. The node shall also support lock requests for all implemented registers so long as the *destination_offset* is quadlet aligned, the *extended_tcode* is equal to two (compare and swap) and the *data_length* is equal to four. Neither quadlet write nor block write requests shall be supported for any plug control registers, whether implemented or not. If an otherwise valid request is received for an unimplemented plug control register, the node may either reject the request with a response of *resp_address_error* or behave in accordance with clause 1.6.11 of IEEE Std 1394-1995.

A node may optionally support block read requests addressed to the address space of the plug control registers. If the combination of *destination_offset* and *data_length* for the block read request includes unimplemented plug control register(s), the node may reject the request with a response of *resp_address_error*. However, if the node successfully completes the transaction the response data returned for the unimplemented register(s) shall be as specified by clause 1.6.11 of IEEE Std 1394-1995.

The following clauses provide detailed definitions of the registers required to support isochronous connection management.

7.1.1 OUTPUT_MASTER_PLUG register

The OUTPUT_MASTER_PLUG register provides information about and permits control of common aspects of a node's OUTPUT_PLUG registers. If a node can function as an isochronous talker, it shall implement the OUTPUT_MASTER_PLUG register. The register definition is given by figure 7-1 below.

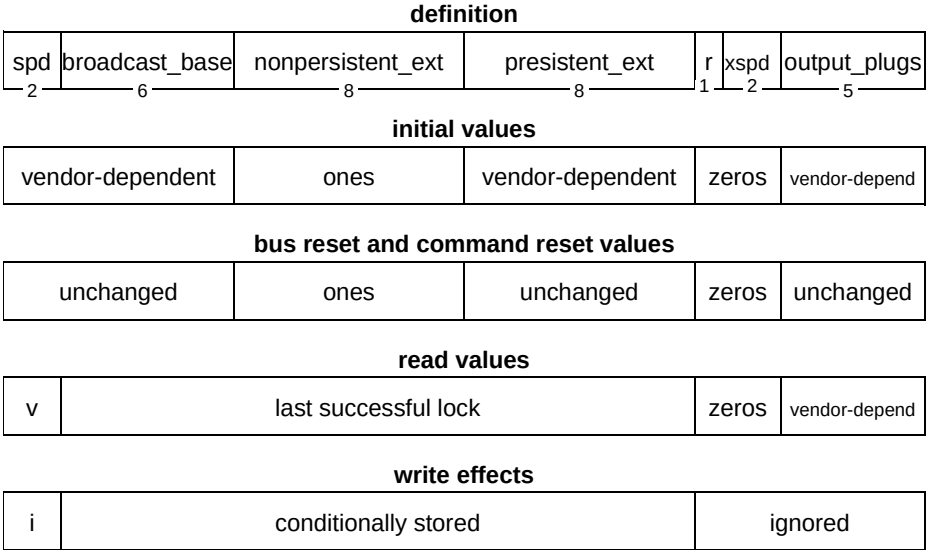


Figure 7-1 — OUTPUT_MASTER_PLUG format

The ***spd*** field shall specify the maximum speed for isochronous data transmission that any of the OUTPUT_PLUG registers may use, as encoded by table 7-2.

Table 7-2 — Speed encoding

Value of *spd*	Data rate
0	S100
1	S200
2	S400
3	Maximum data rate specified by *xspd*

The ***broadcast_base*** field shall specify the base isochronous channel number used to determine the channel number transmitted for broadcast out connections. When broadcast out connections are established for plug(s) for which a point-to-point connection does not simultaneously exist, the *channel* field of the OUTPUT_PLUG register(s) shall be set to 63 if *broadcast_base* equals 63 and otherwise shall be set to (*broadcast_base* + *n*) modulo 63, where *n* is the ordinal of OUTPUT_PLUG[*n*]. See clause 7.2 for more information on broadcast out and point-to-point connections.

The ***nonpersistent_ext*** and ***persistent_ext*** fields are reserved for future standardization.

When the *spd* field has a value of three, the ***xspd*** field shall specify the maximum speed for isochronous data transmission that any of the OUTPUT_PLUG registers may use, as encoded by table 7-3.

Table 7-3 — Extended speed encoding

Value of *xspd*	Data rate
0	S800
1	S1600
2	S3200
3	Reserved for future standardization

The ***output_plugs*** field shall specify the total count of OUTPUT_PLUG registers implemented by a node. Between zero and 31 OUTPUT_PLUG registers may be implemented. If one or more OUTPUT_PLUG registers are implemented, they shall lie within the contiguous address range FFFF F000 0904_{16} to FFFF F000 0900_{16} + 4 * *output_plugs*, inclusive.

7.1.2 OUTPUT_PLUG registers

Each OUTPUT_PLUG register permits the description and control of both broadcast and point-to-point connections that originate with the associated plug. OUTPUT_PLUG registers are optional but one or more are required if the node can function as an isochronous talker. OUTPUT_PLUG registers shall be implemented within a contiguous address space and are referenced by an ordinal *n*, where *n* starts at zero; OUTPUT_PLUG[*n*] refers to the register addressable at FFFF F000 0904_{16} + 4 * *n*. The register definition is given by figure 7-2 below.

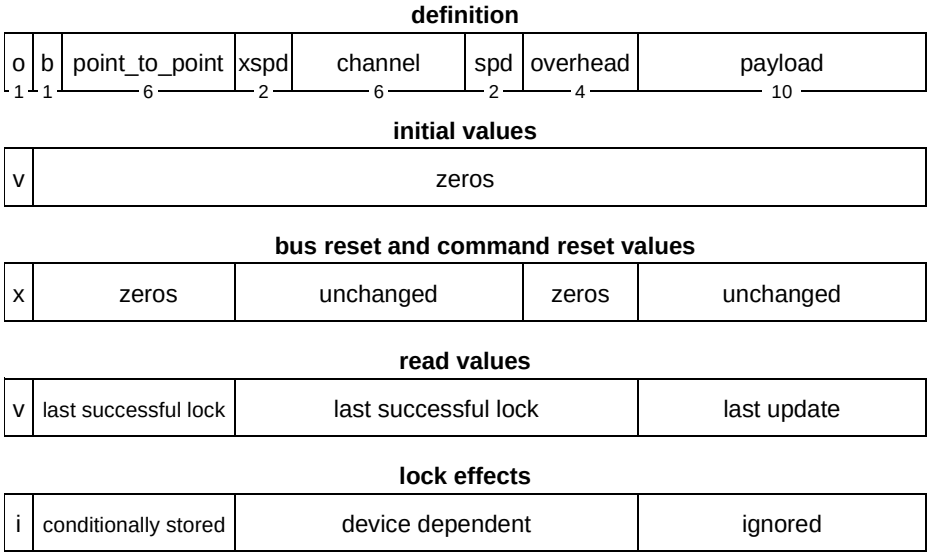


Figure 7-2 — OUTPUT_PLUG format

The ***online*** bit (abbreviated as *o* in the figure above) shall specify the on-line status of the plug resources controlled by the OUTPUT_PLUG register. An *online* bit value of zero shall indicate that the plug is off-line and not capable of transmitting isochronous data. An *online* value of one shall indicate that the plug may be configured and used for isochronous data transmission.

NOTE—Plug status may change dynamically from off-line to on-line as device resources become unavailable or available. The causes of a change in plug status reported by the *online* bit are vendor-dependent.

The ***broadcast*** bit (abbreviated as *b* in the figure above) shall specify whether or not a broadcast connection exists for the output plug, where a value of zero indicates that no such connection exists.

The ***point_to_point*** field shall specify the number of point-to-point connections that exist for the output plug.

When the *spd* field has a value of three, the ***xspd*** field shall specify the speed to be used for isochronous data transmissions for the plug, as encoded by table 7-3. If x*spd* is set to a value greater than the value of the x*spd* field in the OUTPUT_MASTER_PLUG register, isochronous data transmissions shall be disabled for the plug.

The ***channel*** field shall specify the isochronous channel number used in isochronous data transmissions for the plug.

The ***spd*** field shall specify the speed to be used for isochronous data transmissions for the plug, as encoded by table 7-2. If *spd* is set to a value greater than the value of the *spd* field in the OUTPUT_MASTER_PLUG register, isochronous data transmissions shall be disabled for the plug.

The ***overhead*** field shall encode a value used in the calculation of the isochronous bandwidth necessary to allocate for isochronous data transmissions associated with the plug. Isochronous bandwidth is expressed in terms of bandwidth allocation units, as defined by IEEE Std 1394-1995. One bandwidth allocation unit represents the time required to transmit one quadlet of data at a future (larger than the present maximum definition) S1600 data rate, roughly 20 nanoseconds. If *overhead* is nonzero, the total bandwidth allocation necessary is expressed as $overhead * 32 + (payload + 3) * 2^{4 - spd}$. Otherwise, the total bandwidth allocation may be obtained from $512 + (payload + 3) * 2^{4 - spd}$. In the preceding formulae, *overhead*, *payload* and *spd* represent the values of these fields in the OUTPUT_PLUG register.

The ***payload*** field shall specify the maximum number of quadlets that may be transmitted in a single isochronous packet for this plug. A *payload* value of zero indicates a maximum of 1024 quadlets; all other values for *payload* represent a maximum of the value itself.

NOTE—The value of *payload* does not include the isochronous header, header CRC or data CRC required as part of an isochronous packet; it counts only those quadlets that are part of the isochronous data payload.

7.1.3 INPUT_MASTER_PLUG register

The INPUT_MASTER_PLUG register provides information about and permits control of common aspects of a node's INPUT_PLUG registers. If a node can function as an isochronous listener, it shall implement the INPUT_MASTER_PLUG register. The register definition is given by figure 7-3 below.

definition

spd	reserved	nonpersistent_ext	presistent_ext	r	input_plugs
2	6	8	8	3	5

initial values

v	zeros	ones	vendor-dependent	zeros	vendor-depend

bus reset and command reset values

x	zeros	ones	unchanged	zeros	unchanged

read values

v	zeros	last successful lock	zeros	vendor-depend

write effects

ignored	conditionally stored	ignored

Figure 7-3 — INPUT_MASTER_PLUG format

The ***spd*** field shall specify the maximum speed at which any of the node's input plugs may receive isochronous data, as encoded by table 7-2.

The ***nonpersistent_ext*** and ***persistent_ext*** fields are reserved for future standardization.

When the *spd* field has a value of three, the ***xspd*** field shall specify the maximum speed at which any of the node's input plugs may receive isochronous data, as encoded by table 7-3.

The ***input_plugs*** field shall specify the total count of INPUT_PLUG registers implemented by a node. Between zero and 31 INPUT_PLUG registers may be implemented. If one or more INPUT_PLUG registers are implemented, they shall lie within the contiguous address range FFFF F000 0984_{16} to FFFF F000 0980_{16} + 4 * *input_plugs*, inclusive.

7.1.4 INPUT_PLUG registers

Each INPUT_PLUG register permits the description and control of point-to-point connections that terminate at the associated plug. INPUT_PLUG registers are optional and are required only if the node can function as an isochronous listener. INPUT_PLUG registers shall be implemented within a contiguous address space and are referenced by an ordinal *n*, where *n* starts at zero; INPUT_PLUG[*n*] refers to the register addressable at FFFF F000 0984_{16} + 4 * *n*. The register definition is given by figure 7-4 below.

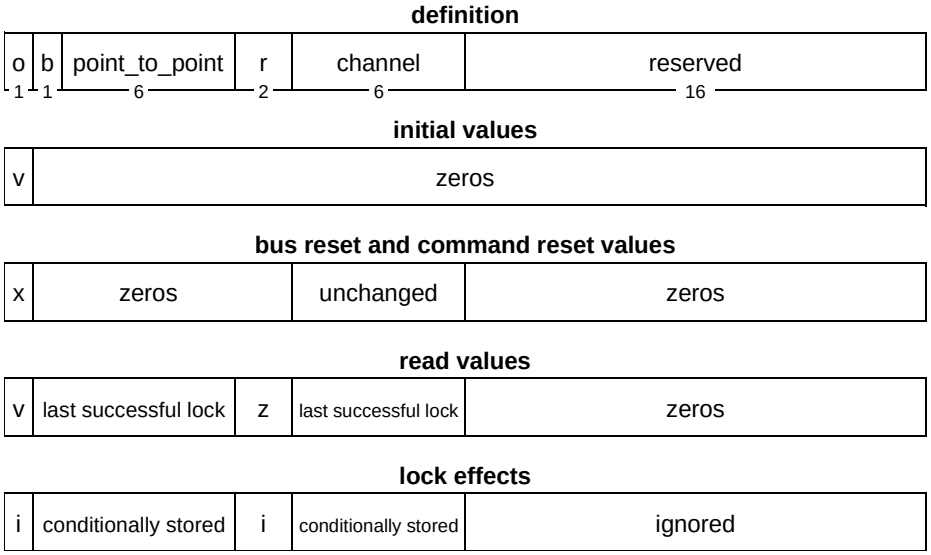


Figure 7-4 — INPUT_PLUG format

The ***online*** bit (abbreviated as *o* in the figure above) bit shall specify the on-line status of the plug resources controlled by the INPUT_PLUG register. An *online* bit value of zero shall indicate that the plug is off-line and not capable of receiving isochronous data. An *online* value of one shall indicate that the plug may be configured and used for isochronous data transmission.

NOTE—Plug status may change dynamically from off-line to on-line as device resources become unavailable or available. The causes of a change in plug status reported by the *online* bit are vendor-dependent.

The ***broadcast*** bit (abbreviated as *b* in the figure above) shall specify whether or not a broadcast connection exists for the input plug, where a value of zero indicates that no such connection exists.

The ***point_to_point*** field shall specify the number of point-to-point connections that exist for the input plug.

The ***channel*** field shall specify the isochronous channel number used in isochronous data reception for the plug.

7.2 Isochronous connection management

All applications that control isochronous devices on Serial Bus shall conform to procedures defined in this clause for the management of isochronous connections between the devices. The plug control registers, previously defined in clause 7.1, provide the facilities upon which connection management operates. All isochronous devices shall implement plug control registers.

Connection management procedures are defined as a set of rules for the manipulation of plug control registers by direct access, through read and lock transactions, to these registers. In addition, devices may autonomously modify their own plug control registers or they may be modified by indirect methods beyond the scope of this supplement. The same procedures shall apply whether a plug control register is modified autonomously, directly or indirectly.

There are three fundamental connection types:

— point-to-point;
— broadcast out; and
— broadcast in.

A point-to-point connection exists when both endpoints of an isochronous data flow between a talker and a listener have been made visible in plug control registers at each device. That is, an OUTPUT_PLUG register at a talker and an INPUT_PLUG register at a listener both have the same value in their *channel* fields and both have nonzero values in their *point_to_point* fields.

A broadcast out connection exists when a talker transmits isochronous data but has no indication that there are listeners. In this case the talker's OUTPUT_PLUG register identifies an isochronous channel by the value in the *channel* field and indicates a broadcast out connection by a value of one for the *broadcast* bit.

A broadcast in connection exists when a listener is receiving (or is prepared to receive) isochronous data without indication that a talker exists. Analogously to a broadcast out connection, the listener's INPUT_PLUG register identifies an isochronous channel by the value in the *channel* field and indicates a broadcast in connection by a value of one for the *broadcast* bit.

Multiple point-to-point connections may exist simultaneously at a single output or input plug but only zero or one broadcast connection, out or in according to the nature of the plug, may exist at any time at a single plug. Broadcast and point-to-point connections may coexist simultaneously at a single plug.

The same isochronous data may be transported between two devices that share a point-to-point connection as may be transported if one has a broadcast out connection and the other a broadcast in connection. The essential difference is that point-to-point connections are considered protected: through conformance to these connection management procedures, only entities that establish point-to-point connections are permitted to break them. By contrast, broadcast connections may be terminated by any entity.

7.2.1 Plug states

Plugs are one of the concepts used in the description of connection management procedures. Each OUTPUT_PLUG or INPUT_PLUG register implemented by a target represents a plug. Plugs do not physically exist; they represent state information associated with the transmission or reception of an isochronous channel. The value of a plug control register is the visible manifestation of the state of that plug.

A plug may be in one of four states: IDLE, READY, SUSPENDED or ACTIVE. The state is evidenced by the values of the *online* and *broadcast* bits and the *point_to_point* field in the plug control register. State transitions are caused by modifications to the plug control register, either directly as the result of a Serial Bus lock transaction or indirectly as the result

of a internal device state changes[1]. Figure 7-5 below illustrates the state transitions. For the sake of clarity, the *broadcast* bit and the *point_to_point* field of the plug control register are described in the state diagram as if they comprised a single field named connections. If either *broadcast* or *point_to_point* is nonzero, connections is considered nonzero; this is also referred to as a connected condition for the plug. Otherwise, if both *broadcast* and *point_to_point* are zero, connections is deemed zero; the plug is called unconnected. Also, the symbol PLUG in the diagram indicates either an OUTPUT_PLUG or INPUT_PLUG register as appropriate.

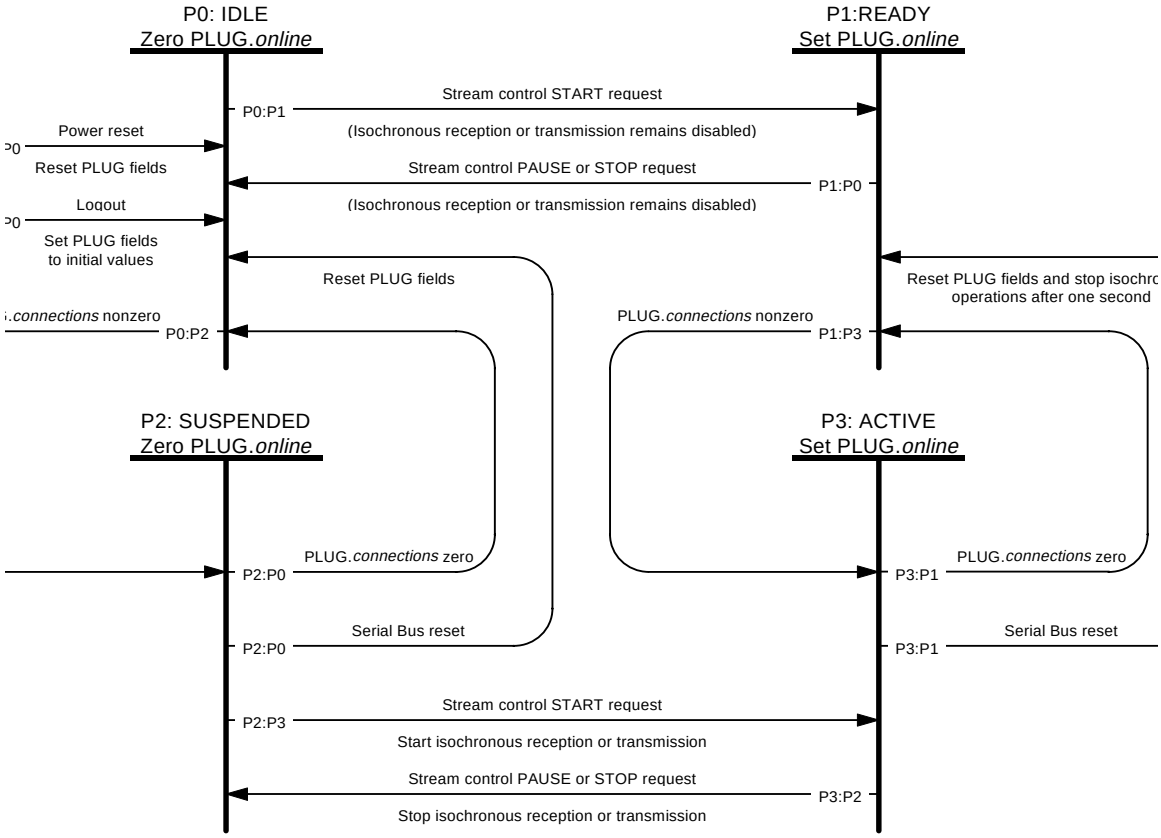


Figure 7-5 — Plug state transitions

Transition Any:P0. Any of a command reset or power reset shall cause the plug control register to be reset to its initial values, as specified in clause 7.1, and shall place the plug in the idle state.

State P0: Idle. Upon entry to this state, the *online* bit in the plug control register shall be zeroed. The plug is available for use.

Transition P0:P1. The successful completion of an autonomous device operation that makes a source of, e.g., activation of PLAY on a video cassette recorder, shall cause the *online* bit to be one. The plug shall transition to the READY state. The plug is still unable to transmit or receive isochronous data because no connections have been established.

Transition P0:P2. When either the *broadcast* bit or the *point_to_point* field becomes nonzero, the plug becomes connected and shall transition to the SUSPENDED state.

[1] A device changes its state in response to a command or as the result of manual operation of a control.

State P1: Ready. Upon entry to this state, the *online* bit in the plug control register shall be set to one. Since no connections exist, the plug is not yet configured to transmit or receive isochronous data.

Transition P1:P0. An internal device state change causes the *online* bit to be zeroed. The plug shall transition to the IDLE state.

Transition P1:P3. When either the *broadcast* bit or the *point_to_point* field becomes nonzero, the plug becomes connected and shall transition to the ACTIVE state.

State P2: Suspended. Upon entry to this state, the *online* bit in the plug control register shall be zeroed. The plug is connected and may transmit or receive isochronous data in response to a change in internal device state, *e.g.*, the operation of a front panel control or the execution of a command.

Transition P2:P0. When both the *broadcast* bit and the *point_to_point* field are zero, the plug becomes unconnected and shall transition to the IDLE state.

Transition P2:P0. A Serial Bus reset shall cause the fields in the plug control register to be set to their bus reset values, as specified in clause 7.1. This causes both the *broadcast* bit and the *point_to_point* field to be zeroed; the plug becomes unconnected and shall transition to the IDLE state.

Transition P2:P3. An internal device state change causes the *online* bit to be set to one. The plug shall transition to the ACTIVE state and may transmit or receive isochronous data.

State P3: Active. Upon entry to this state, the *online* bit in the plug control register shall be set to one. The plug is both connected and able to transmit or receive isochronous data.

Transition P3:P1. When both the *broadcast* bit and the *point_to_point* field are zero, the plug becomes unconnected and shall transition to the READY state.

Transition P3:P1. A Serial Bus reset shall cause the fields in the plug control register to be set to their bus reset values. This causes both the *broadcast* bit and the *point_to_point* field to be zeroed; the plug becomes unconnected and shall transition to the READY state. However, the target shall, for one second, continue to behave as if the plug control register retained its value prior to the bus reset.

NOTE—This one second isochronous delay permits isochronous operations to continue for a short period of time without disruption. During this time, the controlling application(s) are expected to reallocate the necessary isochronous channel(s) and bandwidth and then to reconfirm these by a write to the plug control register(s). In the event that the necessary resources cannot be reallocated, the expiration of the one second isochronous delay forces the plug to transition to the READY state and stops isochronous transmission or reception.

Transition P3:P2. An internal device state change causes the *online* bit to be zeroed. The plug shall transition to the SUSPENDED state.

7.2.2 Establishing and breaking connections

A connection is established at a plug whenever the *broadcast* bit is modified from zero to one or whenever the *point_to_point* field is incremented. As described above, a plug makes a transition from an unconnected to a connected state upon the establishment of the first connection. A connection is broken at a plug whenever the *broadcast* bit is modified from one to zero or whenever the *point_to_point* field is decremented. In like fashion to the transition to connected, a plug makes a transition from a connected to an unconnected state upon the breaking of the last connection.

The following rules shall apply when an OUTPUT_PLUG or INPUT_PLUG register is modified, whether directly or indirectly:

— The channel number and necessary isochronous bandwidth shall be allocated in the isochronous resource manager's CHANNELS_AVAILABLE and BANDWIDTH_AVAILABLE registers before the corresponding output plug is connected. The channel and bandwidth shall not be deallocated, except as a result of a command or power reset at the isochronous resource manager or as a result of a Serial Bus reset, while the output plug remains connected;

— The *channel* and *spd* fields of an OUTPUT_PLUG register shall not be modified while the plug is connected;

— The *channel* field of an INPUT_PLUG register shall not be modified while the *point_to_point* field in the same register is nonzero;

— The *broadcast* bit shall not be set to one by a Serial Bus lock transaction; a lock transaction may zero the bit. Autonomous device actions or other requests addressed to the device (beyond the scope of this standard) may zero the *broadcast* bit or set it to one;

— When an output plug transitions to a connected state, the *broadcast* bit and the *point_to_point*, *spd* and *overhead* fields shall be modified in a single operation, either one Serial Bus lock transaction, one autonomous device action or a single other request addressed to the device; and

— If the *broadcast* bit of an OUTPUT_PLUG register is modified from zero to one at the same time that the *point_to_point* field remains zero, the *channel* field shall be modified in the same operation, either a Serial Bus lock transaction, an autonomous device action or some other request addressed to the device. The value for *channel* shall be determined from the OUTPUT_MASTER_PLUG register's *broadcast_base* field, as defined in clause 7.1.1.

When an entity attempts to establish a connection through an OUTPUT_PLUG or INPUT_PLUG register (or both), it shall retain sufficient prior state information to reverse the plug control register modifications in the event that the connection cannot be made.

8. 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.

8.1 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 an nonbroadcast primary packet. This supplement defines two new acknowledge codes, *ack_address_error* and *ack_tardy*. Table 8-1 below replaces existing table 6-13 in IEEE Std 1394-1995.

Table 8-1—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	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
8 — B_{16}	reserved	
C_{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.
D_{16}	ack_data_error	The node could not accept the block packet because the data field failed the CRC check, or because the length of the data block payload did not match the length contained in the data_length field. This code shall not be returned for any packet that does not have a data block payload.
E_{16}	ack_type_error	A field in the request packet header was set to an unsupported or incorrect value, or an invalid transaction was attempted (e.g., a write to a read-only address).
F_{16}	reserved	

Although an address error in a request packet is normally detected by the transaction or higher application layer, there may be circumstances in which the link is capable of detecting and address error within the time permitted for a unified response to a request subaction. The new acknowledge code ***ack_address_error*** is defined to provide the same utility as the existing *resp_address_error.*

The ***ack_tardy*** code has been defined to enable low power consumption states for Serial Bus devices. Such a device may be able to place its link layer in a partially functional state and suspend the transaction and all higher application layers. The link layer shall be able to recognize nonbroadcast request packets whose *destination_ID* addresses the suspended node. Upon recognition of such a packet, the link shall send an immediate response of *ack_tardy* and shall initiate the resumption of full link and transaction layer functionality. The recipient of an *ack_tardy* may retransmit the request packet in a subsequent fairness interval. The time required for the link and transaction layers to become fully operational is implementation-dependent but is expected to be on the order of *min* to *max* milliseconds.

8.2 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 8-2—Response code encoding

Code	Name	Comment
0	resp_complete	The node has successfully completed the command.
1 to 3	Reserved	
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	

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 specifies response code usage so as to assure equivalent behavior, and hence interoperability, of link layer, transaction layer and application implementations from different vendors.

8.2.1 resp_complete

Nodes shall respond with resp_complete in the circumstances described below (this is not an exhaustive list, just some examples of circumstances for which there might be confusion with other response codes):

A write transaction 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.

8.2.2 resp_conflict_error

Nodes shall respond with resp_conflict_error in the circumstances described below:

An otherwise valid request packet is received but the resources required to act upon the request are not available. The requester may reasonably expect the same packet to succeed at some point in the future when the resources are available. Note that the distinction between resp_conflict_error and ack_busy_X, ack_busy_A or ack_busy_B hinges upon the possibility of deadlock. The busy acknowledgments are appropriate for transient conditions of expected short duration that cannot cause a deadlock. On the other hand, resp_conflict_error shall be returned when an end-to-end retry is necessary to avoid the possibility of deadlock. Potential deadlocks typically arise when a request cannot be queued and blocks a node's transaction resources.

8.2.3 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 ECC error reported by the underlying medium 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.

8.2.4 resp_type_error

Nodes shall respond with resp_type_error in the circumstances described below:

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

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

A request packet is received with valid *tcode* and *extended_tcode* values, but the referenced address does not implement the indicated request. An example of this is a write request to an address that is entirely read-only (note that this is distinct from a write request that references an otherwise writable location that contains read-only bits or fields). 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.

8.2.5 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.

8.3 Transaction layer services

Section 6 of IEEE Std 1394-1995, “Link layer specification”, in the descriptions of the different types of primary Serial Bus packets, requires that the transaction codes (*tcode*) used in response to data requests correspond to the original *tcode* of the request. That is, a read response for data quadlet shall be sent only in response to a read request for data quadlet, a read response for data block shall be sent only in response to a read request for data block and a lock response shall be sent only in response to a lock request. This is a pleasing symmetry but a close examination of clause 7, “Transaction layer specification”, reveals that insufficient information is communicated to the transaction layer in order for it to meet this requirement.

The portion of section 7 that describes which *tcode* the transaction layer shall select for a READ or WRITE request mandates, at present, that a quadlet *tcode* shall be used if the data length of the transaction is four, independent of whether or not the destination offset is quadlet aligned. This 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.

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

a) Limit the use of quadlet READ and WRITE transactions to the case where the data length is four and the destination offset is quadlet aligned;
b) Optionally permit the use of block READ and WRITE transactions in the case where the data length is four and the destination offset is quadlet aligned;
c) Add a new parameter to a transaction layer data service so that quadlet responses may be properly generated for quadlet requests and block responses for block requests; and
d) Emphasize that support for quadlet transactions is mandatory in all Serial Bus implementations but that support for block transactions is optional.

8.3.1 Transaction data request (TRAN_DATA.request)

In clause 7.1.2.1, a new parameter is added to the list of parameters communicated to the transaction layer *via* this service:

— 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.

8.3.2 Transaction data response (TRAN_DATA.response)

In clause 7.1.2.4, a new parameter is added to the list of parameters communicated to the transaction layer *via* this service:

— 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.

8.3.3 Sending a transaction request

Clause 7.3.3.1.2, under the heading “State TX1: Send Transaction Request”, currently has language that 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.

8.3.4 Sending a transaction response

Clause 7.3.3.1.3, under the heading “State TX2: Send Transaction Response”, currently has language that describes how the transaction code parameter (communicated to the link layer *via* LK_DATA.response) 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.

8.3.5 CSR Architecture transactions mapped to Serial Bus

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

At a minimum, all Serial Bus nodes shall implement support for transaction data requests with a transaction type of READ or WRITE, a data length of four and a destination address that is quadlet aligned. These correspond to the read4 and write4 requests of the CSR Architecture.

All other transaction support, *i.e.*, transaction data requests with a data length other than four, a destination address that is not quadlet aligned or lock requests, is optional.

NOTE—Transaction support for block reads or writes for some arbitrary data length *n* does not necessarily imply transaction support for any other length block read or write.

8.4 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 8-3 below replaces existing table 8-4 in IEEE Std 1394-1995.

Table 8-3—Serial Bus-dependent registers in initial units space

Offset	Name	Notes
800_{16} — $8FC_{16}$	TOPOLOGY_MAP	Present at the bus manager, only.
900_{16}	OUTPUT_MASTER_PLUG	Common output plug controls for the node
904_{16} — $97C_{16}$	OUTPUT_PLUG	Output plug control registers for individual isochronous channels, OUTPUT_PLUG[0] through OUTPUT_PLUG[30]
980_{16}	INPUT_MASTER_PLUG	Common input plug controls for the node
984_{16} — $9FC_{16}$	INPUT_PLUG	Input plug control registers for individual isochronous channels, INPUT_PLUG[0] through INPUT_PLUG[30]
$A00_{16}$ — AFC_{16}		Reserved for Serial Bus
$B00_{16}$ — CFC_{16}	FCP command frame	Specified by draft standard ISO/IEC-1883
$D00_{16}$ — EFC_{16}	FCP response frame	Specified by draft standard ISO/IEC-1883
$F00_{16}$ — FFC_{16}		Reserved for Serial Bus
1000_{16} — $1FFC_{16}$	SPEED_MAP	Present at the bus manager, only
2000_{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 CSR's that fall within the above address space.

8.4.1 SPEED_MAP_REGISTERS (cable environment)

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

The three least-significant bits of the ***speed_code*** bytes specify one of the data transfer speeds S100, S200 or S400, S800, S1600 or S3200. The *speed_code* bytes use the same encoding as the *xspd* field specified in table 6-1. The remaining most-significant six bits are reserved for future standardization and may not be relied upon to be read as zeros.

8.4.2 TOPOLOGY_MAP registers (cable environment)

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

8.5 Configuration ROM Bus_Info_Block

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 new format of the Bus_Info_Block is shown in figure 8-1.

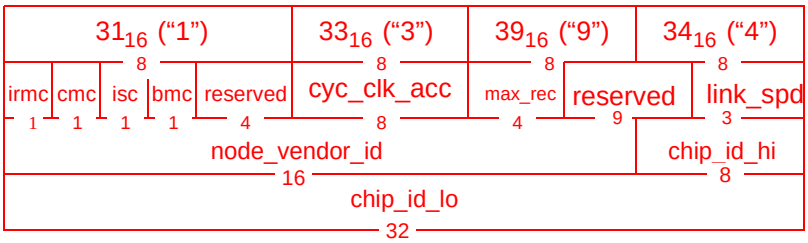


Figure 8-1—Bus_Info_Block format

The definitions of all fields previously specified by IEEE Std 1394-1995 are unchanged.

The ***link_spd*** field shall report the maximum speed capability of the node's link layer; the encoding used is the same as for the self-ID packet *xspd* field specified in table 6-1.

8.6 Automatic activation of the cycle master

As presently 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 presently specified, 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 8.3.2.2.1of 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 this 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 or if this 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).

8.7 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.

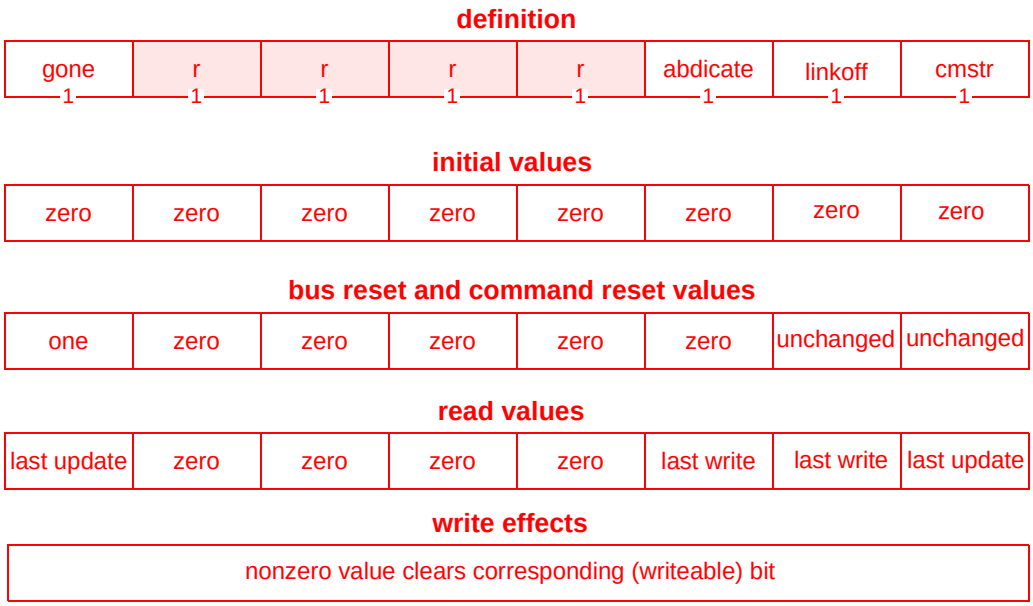


Figure 8-2—STATE_CLEAR.*bus_depend* field

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 8.6 below.

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 CSR's for the presence or absence of advanced features or the two nodes may engage in some negotiation to determine which is more capable.

8.8 Security extensions

IEEE Std 1394-1995 makes no provisions for facilities or device-implementation constraints that enable a secure architecture for transactions. 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 necessity to provide building blocks upon which more secure systems may be constructed. In particular it is important for Serial Bus devices to possess unforgeable identities and to not be able to snoop asynchronous request or response packets addressed to other nodes.

With respect to these security considerations, a device compliant with this supplement shall be subject to the following restrictions:

— 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 at the time of manufacture;
— A node shall not originate any asynchronous request or response packets with a *source_ID* field that is not equal to either a) the most significant 16 bits of the node's NODE_IDS register or b) the concatenation of $3FF_{16}$ (the local *bus_ID*) and the physical ID assigned to the node's PHY during the self-identify process; and
— A node's link shall not receive nor make available to the transaction layer or any other application layer any asynchronous request or response packet unless the *destination_ID* field is equal to either a) the most significant 16 bits of the node's NODE_IDS register or b) the concatenation of $3FF_{16}$ (the local *bus_ID*) and the physical ID assigned to the node's PHY during the self-identify process.

All exemptions to these restrictions, 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 exemptions are for Serial Bus to Serial Bus bridges, which work is in progress in the IEEE P1394.1 working group. Only for the purposes of forwarding asynchronous request and response packets may a bridge recognize packets not addressed to it and transmit packets as a proxy for other nodes.

