

IEEE Standard for a High Performance Serial Bus

Sponsor

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

Approved 12 December 1995

IEEE Standards Board

Approved 22 July 1996

American National Standards Institute

Abstract: A high-speed serial bus that integrates well with most IEEE standard 32-bit and 64-bit parallel buses, as well as such nonbus interconnects as the IEEE Std 1596-1992, Scalable Coherent Interface, is specified. It is intended to provide a low-cost interconnect between cards on the same backplane, cards on other backplanes, and external peripherals. This standard follows the IEEE Std 1212-1991 Command and Status Register (CSR) architecture.

Keywords: backplane, bus, computers, high-speed serial bus, interconnect, parallel buses

The Institute of Electrical And Electronics Engineers, Inc.

345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1996 by the Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1996. Printed in the United States of America.

ISBN 1-55937-583-3

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

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

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

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

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

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

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

Introduction

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

This standard describes a serial bus that provides the same services as modern IEEE-standard parallel buses, but at a much lower cost. It has a 64-bit address space, control registers, and a read/write/lock operation set that conforms to the IEEE Std 1212-1991, Command and Status Register (CSR) standard. This simplifies bridging between the Serial Bus and the other interconnects using the IEEE 1212 architecture: IEEE Std 896-1991, FutureBus+[®] and IEEE Std 1596-1992, Scalable Coherent Interface (SCI).

There are two physical environments for the Serial Bus: backplane and cable. The backplane environment uses two single-ended signals on a broadcast multitapped bus using backplane transceiver logic (BTL) or emitter coupled logic (ECL) transceiver technology at 49.152 Mbit/s or enhanced transistor-transistor logic (TTL) transceiver technology at 24.576 Mbit/s. In all cases, bus arbitration on the backplane is done using a dominant-mode-logic bit serial approach. The cable environment uses two low-voltage differential signals to connect devices in a noncyclic topology at 98.304 Mbit/s, 196.608 Mbit/s, and 393.216 Mbit/s data rates. The cable arbitration system uses a self-configuring hierarchical request/grant protocol that supports hot plugging and widely varying physical topologies.

In addition to standard read/write/lock transactions, the Serial Bus provides extensive time-based services, including isochronous data transport (guaranteed latency and bandwidth) and an accurate submicrosecond global timebase for synchronizing events and data.

This standards effort started in 1986 at the request of the membership of the IEEE Microcomputer Standards Committee as an attempt to unify the different serial busses originally proposed as parts of the IEEE 1014 VME, IEEE 1296 Multibus II, and IEEE 896 FutureBus+[®] efforts. The original chair was Michael Teener, and much of the original architecture of this standard was invented by Mr. Teener, David James, and David Gustavson. In particular, the fairness and retry algorithms were originally proposed by David James during this early period.

As the proposed standard was developed, it attracted more interest from those that needed a much improved external I/O interconnect for multimedia information and for mass storage. This added the requirements for isochronous transport, much higher data rates, and a more rugged cable and connector system. To answer these needs

- a) David James and Michael Teener developed a isochronous access protocol that was compatible with the existing fairness scheme. This protocol was simplified and enhanced based on proposals by Ed Gardner.
- b) Roger Van Brunt and Florin Oprescu developed a much improved physical layer for the cable environment, which was capable of 400 Mbit/s peak rates using low-cost digital complimentary-symmetry metal-oxide semiconductor (CMOS) technology. Forrest Crowell further reduced the complexity of the cable physical layer proposing the data-strobe encoding from INMOS. Florin Oprescu's self-configuring hierarchical arbitration protocol was also adopted at this time, with simplifications developed by Willam Duckwall.
- c) Andy Carter proposed the connector and cable system for the cable environment, based on original work done by Hosiden Corporation. The connector task group under the leadership of David Hatch refined that proposal to meet the requirements of all concerned.
- d) Greg Floryance, Bill Ham, and Max Bassler led the specification for the internal connection system for the cable environment. Bill Ham also worked to ensure that system level grounding and power issues were handled correctly.
- e) Thom Potyraj coordinated the development of the backplane environment and contributed much of its technical content.
- f) Gerald Marazas defined the bus management protocols with extensive help from David James, Peter Johansson, Hisato Shima, Scott Smyers, and Michael Teener. Peter Johansson rewrote the bus management text during the ballot resolution process as the best way to respond to extensive ballot comments on this topic.

Gerald Marazas took over as chair of the working group in 1993, while Michael Teener continued as editor of the standard. Additional editing was done by Thom Potyraj for the backplane environment physical layer, Jeff Stai for the link and transaction layers, Gerald Marazas for the bus management layer, Bill Ham for the cable environment system

properties, Max Bassler and Greg Floryance for the internal device physical interface, Jim Baldwin for the PHY-Link interface definition, Florin Oprescu for the cable test procedures, and Steven Dunwoody for the shielding effectiveness test procedures.

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 P1394 working group at the time of publication.

Gerald Marazas, *Chair*
Michael D. Teener (*Editor*)
Ken Stewart, *Secretary*

Max Bassler
Bob Bellino
Charles Brill
Andy Carter
Greg Floryance
Giles Frazier
Bill Ham
Norm Harris
David Hatch

David V. James
James Kuo
Lawrence J. Lamers
Gene Milligan
Jay Neer
Florin Oprescu
Thomas J. Potyraj
Doug Riemer
Ron Roberts

Scott Smyers
Jeff Stai
Roger Van Brunt
Hans H. Wang
Harvey Watersdorf
Alan Wetzel
Bob Whiteman
Michael Wingard

The following is a list of other major participants in the IEEE P1394 working group (those that attended at least three working group meetings in the last four years).

Mark Andresen
John Atwood
James Baldwin
Peter Bartlett
Bryan Bell
Phil Bolton
Paul L. Borrill
David Brief
Ed Cady
Ramiro Calvo
Mike Chastain
Kim Clohessy
Forrest Crowell
Wayne Davis
Tom Debiec

Dhiru Desai
Frank Duffy
Sam Duncan
W. P. Evertz
Stephen Finch
Mike Foster
Richard Fryer
Bob Gannon
Edward A. Gardner
Charles Grant
Michael Griffin
David B. Gustavson
Steffen Hagene
Emil Hahn
Ken Hallam

Mark Hassel
Rick Heidick
Dan Hillman
Mark Jander
Greg Kite
Lawence Kopp
Ralph Lachenmaier
Michael Lazar
Wendell Lengefeld
Fred Leung
John Lohmeyer
John Lopata
Ivy Lui
Don May
Charles Monia

Claude Mosley
Ray Muggli
Gary Murdock
Michael Nguyen
Dan O'Connor
Erich Oetting
Thomas A. Patrick
Scott Petler

Kevin Pokorney
Jeffrey M. Rosa
Don Senzig
Patricia Smith
Robert N. Snively
Chris Stone
Paul Sweazey
Lars Thernsjö

Barry Thompson
Joel Urban
Mike Wenzel
Lee Whetsel
Dave Wickliff
Yoshihiko Yano

The following persons served on the ballot response committee:

Max Bassler
Larry Blackledge
Andy Carter
William Duckwall
Bill Gamble
Gary Hoffman

David V. James
Peter Johansson
Gerald Marazas
Richard Mourn
Dan O'Connor
Amrish Patel

Thomas J. Potyraj
Scott Smyers
Jeff Stai
Michael D. Teener
Roger Van Brunt
Alan Wetzel

The following persons were members of the balloting group:

M. Robert Aaron
Ray S. Alderman
Richard P. Ames
Harry A. Andreas
Keith D. Anthony
Harrison A. Beasley
Stephen E. Belvin
Alan Beverly
Christus Bezirtzoglou
John Black
Martin Blake
Lim Boon Lum
Paul L. Borrill
Michael L. Bradley
John C. Brightwell
Charles Brill
Hakon Ording Bugge
Andrew L. Carter
Stephen J. Cecil
Andy B. Cheese
C. H. Chen
Joseph Chen
Kim Clohessy
Steven R. Corbesero
Darrell Cox
Patrick Crane
Robert S. Crowder
Dante Del Corso
Roberto Divia
Ian Dobson
Samuel Duncan
Sourav K. Dutta
Roger D. Edwards
Wayne P. Fischer

Gordon Force
Giles Frazier
Martin Freeman
Richard E. Fryer
Paul J. Fulton
Joseph D. George
Stein Gjessing
Patrick Gonia
Julio Gonzalez Sanz
Bruce Grieshaber
William Groseclose
Peter B. Gutgarts
Mike Hasenfratz
David Hawley
Steven Hetzler
Thomas G. Hillyer
George Horansky
Edgar Jacques
David V. James
Norman C. Joehlin
David Kahn
Richard H. Karpinski
Stephen Kempainen
James R. Koser
Ernst H. Kristiansen
Thomas M. Kurihara
Ralph Lachenmaier
Tuvia Lamdan
Lawrence Lamers
Conrad A. Laurvick
Gerald E. Laws
Rollins Linser
Donald C. Loughry
Gerald A. Marazas

Roland Marbot
Joseph R. Marshall
William C. McDonald
S. Fenton McDonald
Thanos Mentzelopoulos
Dhenna Moongilan
Klaus-Dieter Mueller
J. Michael Munroe
J. D. Nicoud
Gregory C. Novak
Daniel C. O'Connor
Katsuyuki Okada
Peter Z. Onufryk
Florin Opreescu
Fred J. Orlando
Michael Orlovsky
Roman Orzol
Granville Ott
Elwood T. Parsons
Mira Pauker
Thomas Potyraj
Suzanne L. Price
Steve Quinton
Gordon Robinson
Fred U. Rosenberger
John Rynearson
Frederick E. Sauer
Donald Senzig
Gary K. Sloane
Larry C. Sollman
Robert K. Southard
Richard C. Spratt
Jeffrey Stai
Larry Stein

Nobuaki Sugiura
Michael D. Teener
Manu Thapar
Michael G. Thompson
Michael Timperman
Robert C. Tripi
Roger Van Brunt
Yoshiaki Wakimura
Paul Walker

Eike G. Waltz
Keith Weber
Thomas H. Wegmann
H. Michael Wenzel
Alan Wetzel
Colin Whitby-Stevens
Rob White
Dave Wickliff
Thomas Wicklund

Robert Widlicka
Mark Williams
Ronald T. Wolfe
David L. Wright
Forrest D. Wright
Yoshio Yamaguchi
Oren Yuen
Janusz Zalewski
Jonathan Zar

When the IEEE Standards Board approved this standard on December 12, 1995, it had 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 Rüsçh
Chee Kiow Tan
Leonard L. Tripp
Howard L. Wolfman

*Member Emeritus

Also included are 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

Futurebus+ is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

SCSI is a registered trademark of SCSI Solutions.

NuBus is a registered trademark of Texas Instruments, Inc.

CLAUSE	PAGE
1. Overview	1
1.1 Scope	1
1.2 References	1
1.3 Document organization	2
1.4 Serial Bus applications	2
1.5 Service model	3
1.6 Document notation	4
1.7 Compliance	11
2. Definitions and abbreviations	12
2.1 Conformance glossary	12
2.2 Technical glossary	12
3. Summary description	16
3.1 Node and module architectures	17
3.2 Topology	17
3.3 Addressing	19
3.4 Protocol architecture	19
3.5 Transaction layer	21
3.6 Link layer	24
3.7 Physical layer	31
3.8 Bus management	45
4. Cable PHY specification	46
4.1 Cable PHY services	46
4.2 Cable physical connection specification	50
4.3 Cable PHY facilities	81
4.4 Cable PHY operation	92
5. Backplane PHY specification	115
5.1 Backplane PHY services	115
5.2 Backplane physical connection specification	119
5.3 Backplane PHY facilities	128
5.4 Backplane PHY operation	131
5.5 Backplane initialization and reset	138
6. Link layer specification	139
6.1 Link layer services	139
6.2 Link layer facilities	145
6.3 Link layer operation	166
6.4 Link layer reference code	175

CLAUSE	PAGE
7. Transaction layer specification	177
7.1 Transaction layer services	177
7.2 Transaction facilities	180
7.3 Transaction operation	181
7.4 CSR Architecture transactions mapped to Serial Bus	203
8. Serial Bus management specification	203
8.1 Serial Bus management summary	203
8.2 Serial Bus management services	204
8.3 Serial Bus management facilities	207
8.4 Serial Bus management operations	234
8.5 Bus configuration state machines (cable environment)	244
Annex A (Normative) Cable environment system properties	250
Annex B (Normative) External connector positive retention	259
Annex C (Normative) Internal device physical interface	261
Annex D (Informative) Backplane PHY timing formulas	291
Annex E (Informative) Cable operation and implementation examples	301
Annex F (Informative) Backplane physical implementation example	320
Annex G (Informative) Backplane isochronous resource manager selection	326
Annex H (Informative) Serial Bus configuration in the cable environment	328
Annex I (Informative) Socket PCB terminal patterns and mounting	336
Annex J (Informative) PHY-link interface specification	342
Annex K (Informative) Serial Bus cable test procedures	360
Annex L (Informative) Shielding effectiveness and transfer impedance testing	379

IEEE Standard for a High Performance Serial Bus

1. Overview

1.1 Scope

This standard describes a high-speed, low-cost Serial Bus suitable for use as a peripheral bus or a backup to parallel backplane buses. Highlights of the Serial Bus include

- a) Automatic assignment of node addresses—no need for address switches.
- b) Variable speed data transmission based on ISDN-compatible¹ bit rates from 24.576 Mbit/s for TTL backplanes to 49.152 Mbit/s for BTL backplanes to 98.304 Mbit/s, 196.608 Mbit/s, and 393.216 Mbit/s for the cable medium.
- c) The cable medium allows up to sixteen physical connections (cable hops), each up to 4.5 m, giving a total cable distance of 72 m between any two devices. Bus management recognizes smaller configurations to optimize performance.
- d) Bus transactions that include both block and single quadlet reads and writes, as well as an “isochronous” mode that provides a low-overhead guaranteed bandwidth service.
- e) A physical layer supporting both cable media and backplane buses.
- f) A fair bus access mechanism that guarantees all nodes equal access. The backplane environment adds a priority mechanism, but one that ensures that nodes using the fair protocol are still guaranteed at least partial access.
- g) Consistent with ISO/IEC 13213 :1994 (IEEE Std 1212-1991).

1.2 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.²

¹The lowest data rate of 24.576 Mbit/s is exactly 18 times the 1.536 Mbit/s and 12 times the 2.048 Mbit/s ISDN primary rates. It is also an integer multiple of the ISDN basic rate and numerous other communication rates.

²EIA publications are available from Global Engineering, 1990 M Street NW, Suite 400, Washington, DC, 20036, USA.

IEEE Std 896.2-1991, IEEE Standard for Futurebus+[®]—Physical Layer and Profile Specification (ANSI).³

IEEE Std 896.5-1993, IEEE Standard for Futurebus+[®], Profile M (Military) (ANSI).

IEEE P896.6, IEEE Standard for Futurebus+[®] Telecommunications Systems, Profile T (Telecommunications).⁴

IEEE Std 1014-1987, IEEE Standard for a Versatile Backplane Bus: VMEbus (ANSI).

IEEE Std 1194.1-1991, IEEE Standard for Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits (ANSI).

ISO/IEC 9899: 1990, Programming languages—C.⁵

ISO/IEC 10857: 1994 (ANSI/IEEE Std 896.1 1994 Edition) Information technology—Microprocessor systems—Futurebus+[®]—Logical Protocol Specification.

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

1.3 Document organization

This standard contains this overview, a list of definitions, an informative summary description, chapters of technical specification, and application annexes. The new reader should read the document in order. The actual specification follows the summary and is organized from the bottom up; that is, the specification starts at the physical layer (cable and backplane), works up to the link layer, the transaction layer, and the bus management layer.

1.4 Serial Bus applications

Three primary applications have driven the design and architecture of the Serial Bus: an alternate for a parallel backplane bus, a low-cost peripheral bus, and a bus bridge between architecturally compatible 32-bit buses.

1.4.1 Alternate bus

There are five main reasons for providing a serial bus on a system that already has a parallel bus:

- a) The many modules that make up a system might operate on different backplane bus standards, yet they need to work together.
- b) Although located within the same enclosure, the system is too large or physically disperse to use a single backplane, yet modules in the different backplanes have to communicate.
- c) One or more modules of a system are located neither on the same backplane nor within the same enclosure.
- d) A redundant data path increases fault tolerance. The system can use the Serial Bus to isolate and diagnose errors without depending on the failed parallel bus.
- e) Many system modules are price-sensitive and do not need the full bandwidth of a parallel bus.

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

⁴This IEEE standards project was not approved by the IEEE Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE.

⁵ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

1.4.2 Low-cost peripheral bus

The Serial Bus can also be used as a powerful and low-cost peripheral interconnect. The compact Serial Bus cable and connector allow bandwidths comparable with existing I/O interconnect standards. The Serial Bus has the added advantage of architectural compatibility with parallel computer buses; this leads to lower communications overhead than limited function dedicated I/O interconnects.

1.4.3 Bus bridge

The Serial Bus architecture limits the number of nodes on any bus to 63, but supports multiple bus systems via bus bridges. The CSR Architecture defines the Serial Bus addressing structure, which allows almost 2^{16} nodes.

A bus bridge normally eavesdrops on the bus, ignoring all transactions between local addresses but listening carefully for remote transactions. When the bridge receives a packet for a remote address, it forwards the packet to an adjacent bus. After initialization, the bus bridges are transparent to the system.

Although the Serial Bus may be used in many bus configurations, when used to bridge CSR-Architecture-compliant buses it is expected to be used mostly in a hierarchical bus fashion, as illustrated in figure 1.1 (where bus #5 is a Serial Bus and bridges together other CSR-Architecture-compliant buses 1 through 4; #1 could be IEEE 896 FutureBus+,⁶ #2 could be IEEE 1596 Scalable Coherent Interface, and so on).

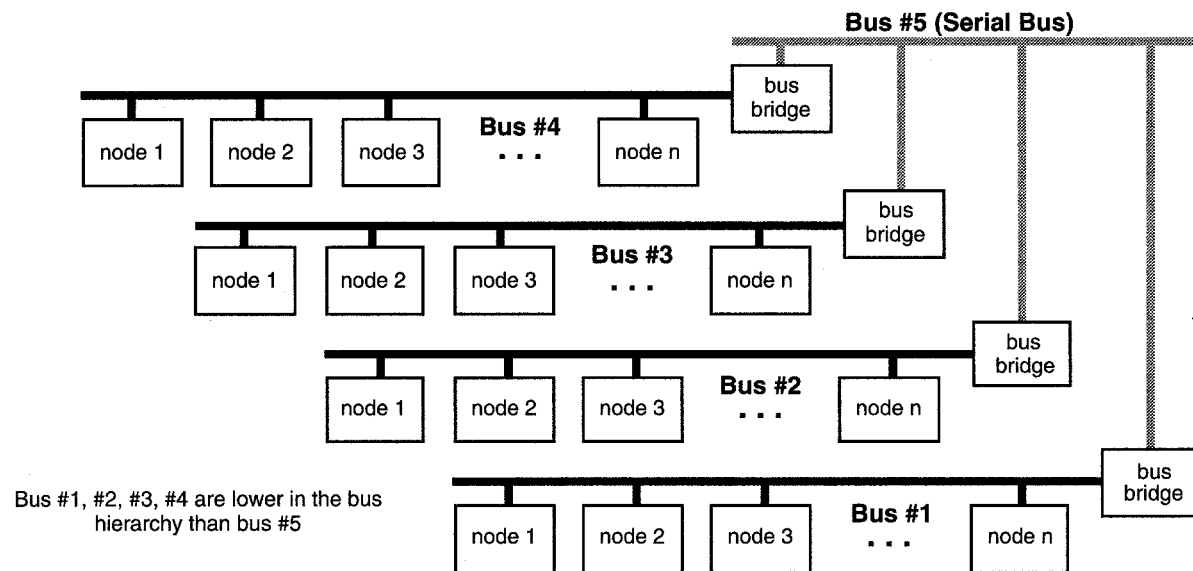


Figure 1.1—Example hierarchical bus topology

1.5 Service model

This standard uses 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:

⁶Information about references can be found in 1.2.

- 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.

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

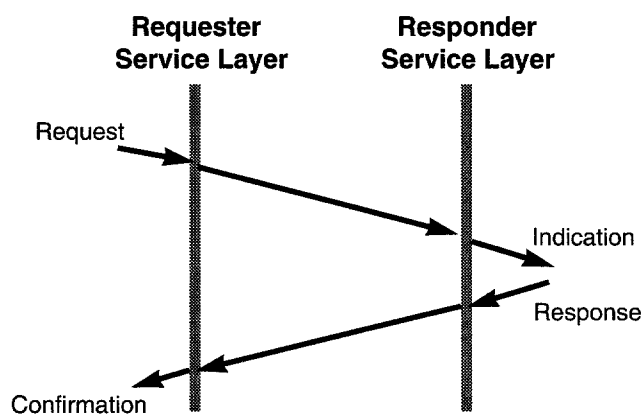


Figure 1.2—Service model

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 half-word, 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, and within a field the most significant bit is also transmitted first. This ordering convention is illustrated in figure 1.3.

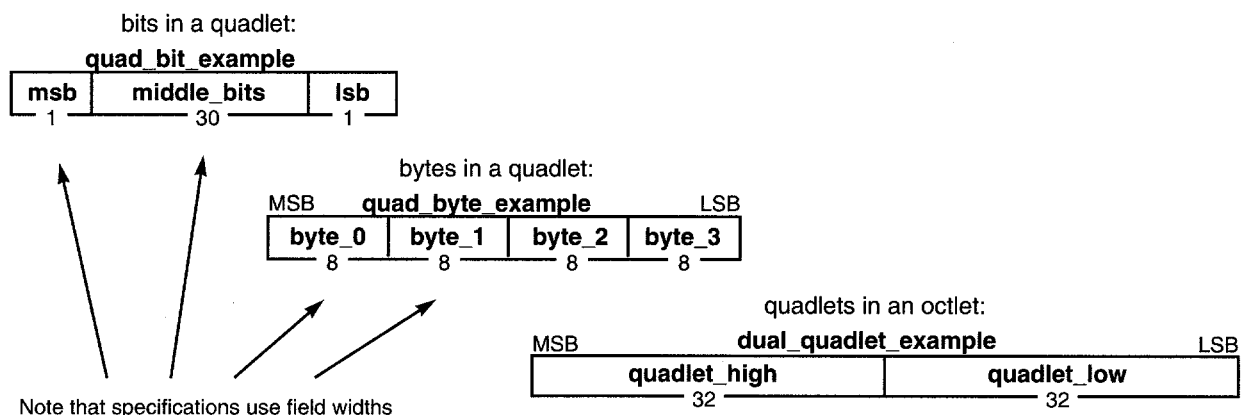


Figure 1.3—Bit and byte ordering

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 1.3.

When specific bit fields must be used, the CSR Architecture convention of consistent big-endian numbering is used. Hence, the most significant bit of a quadlet (“msb” in figure 1.2) 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 the control and status registers (CSRs) defined in clause 8..

1.6.4 Numerical values

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

Decimal numbers are represented in their standard 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0–9, A–F) digits followed by the subscript 16. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number “26” may also be represented as “1A₁₆” or “11010₂”. In C++ code examples, hexadecimal numbers have a “0x” prefix and binary numbers have a “0b” prefix, so the decimal number “26” would be represented by “0x1A” or “0b11010.”

1.6.5 Packet formats

Serial Bus packets consist of a sequence of quadlets. Packet formats are shown using the style given in figure 1.4.

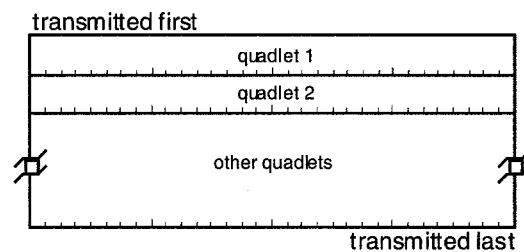


Figure 1.4—Example packet format

Fields appear in packet formats with their correct position and widths. 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
$\sim I$	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)
$I \wedge J$	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
$I \ll J$	Value of I, shifted left by J bits, zero fill
$I \gg J$	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 1.5.

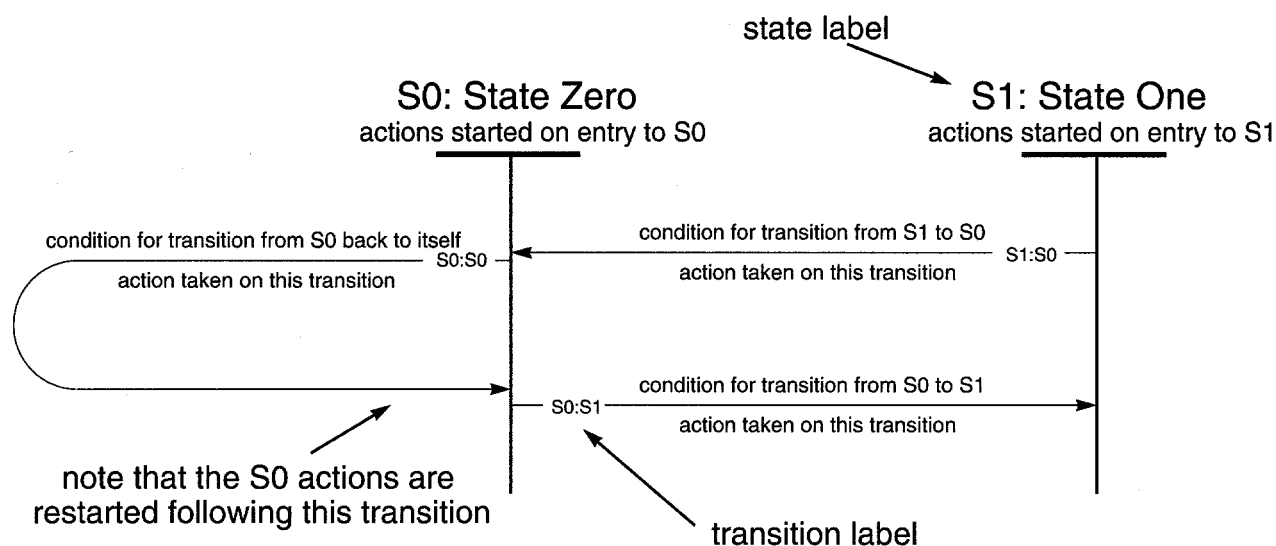


Figure 1.5—State machine example

These state machines make three assumptions:

- Time elapses only within discrete states.
- State transitions are logically instantaneous, so the only actions taken during a transition are setting flags and variables and sending signals.
- 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 lowercased (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 1.6.

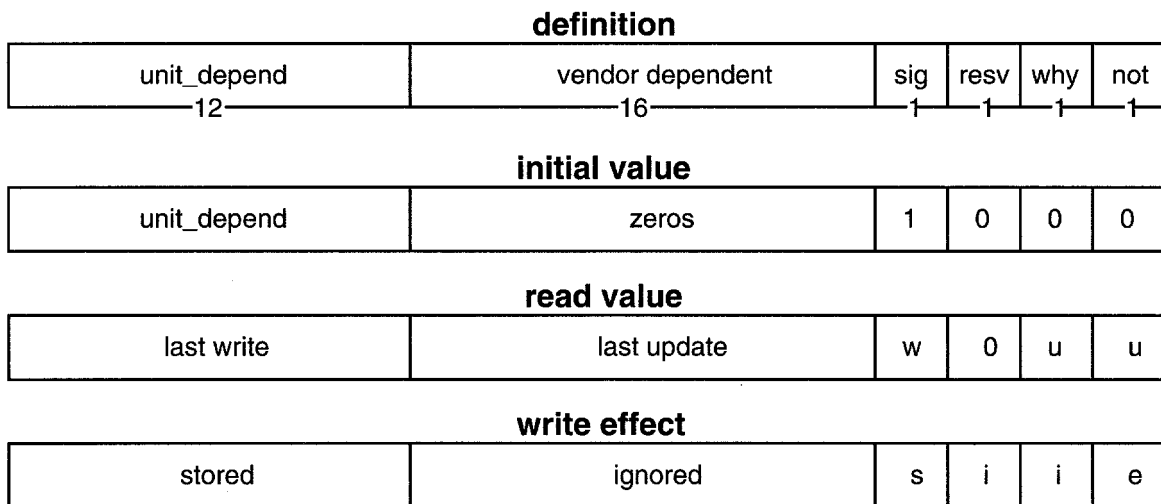


Figure 1.6—CSR format specification (example)

The register definition lists the names of register fields. These names are descriptive, but the fields are defined in the text; their function should not be inferred solely from their names. However, the register definition fields in 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.

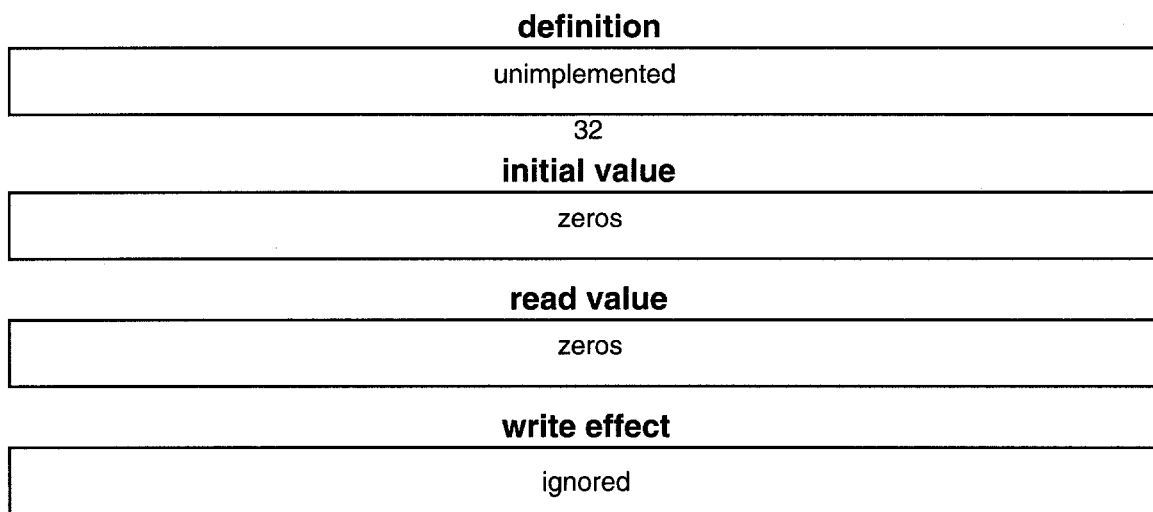
Table 1.7—Write value fields

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 1.7.

**Figure 1.7—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).

1.7 Compliance

1.7.1 CSR Architecture compliance

The Serial Bus follows ISO/IEC 13213 : 1994. Specifically:

- a) Addressing: the Serial Bus uses 64-bit fixed addressing. See 3.3.
- b) Transactions:
 - 1) The CSR-Architecture-defined transactions of read1-read64, write1-write64, lock4, and lock8 are equivalent to the Serial Bus transactions described in clause 7.3.5.2. The Serial Bus also specifies byte-aligned block read and write transactions from 1 byte to 2048 bytes, depending on the data rate as described in table 6.4.
 - 2) The clock strobe signal is implemented by the Serial Bus cycle_start packet described in clause 6.3.2.
- c) Error status codes: type_error, address_error, and conflict_error are all possible values of the status returned in a transaction response. The Serial Bus calls these “response codes,” and there are additional bus-specific values. See table 6.11 for more details.
- d) Resets: Power_reset and command_reset are described in clause 8.3.1.3.
- e) STATE_CLEAR: the *bus_depend* bits of the STATE_CLEAR register are all optional and are described in clause 8.3.1.5.
- f) NODE_IDS:
 - 1) *offset_id*: The Serial Bus calls this the “physical_ID.” For cable environments, the values are chosen during the self-ID process described in clause 4.4.2.3. In such environments, this physical_ID is a read-only value. For backplane environments, these values are dependent on the application environment (see clause 5.4.2.1).
 - 2) *bus_depend*: The bus-dependent field shall be all zeros and is a read-only value.
- g) SPLIT_TIMEOUT_HI and SPLIT_TIMEOUT_LO: Only the low-order 3 bits of SPLIT_TIMEOUT_HI (seconds) are to be implemented (8 s maximum timeout), and the higher order 13 bits of SPLIT_TIMEOUT_LO count at a granularity of 125 μs, not 1/8192 s.
- h) Bus-dependent registers: The Serial Bus has the following optional CSRs; see clause 8.3.2.2.7 for details:
 - 1) CYCLE_TIME
 - 2) BUS_TIME
 - 3) POWER_FAIL_IMMINENT
 - 4) POWER_SOURCE
 - 5) INCUMBENT_ANSWER

- 6) BANDWIDTH_ALLOCATE
- 7) CHANNELS_AVAILABLE_HI
- 8) CHANNELS_AVAILABLE_LO
- 9) MAINTENANCE_CONTROL
- 10) MAINTENANCE_UTILITY
- i) Bus_dependent ROM entries: The Serial Bus has the following extra ROM entries; see clause 8.3.2.3.10 for details:
 - 1) Node_Capabilities root entry
 - 2) NodeUniqueId leaf
 - 3) Bus_Dependent_Info directory
 - 4) Topological_Map location
 - 5) Speed_Map location

1.7.2 Serial Bus physical layers

The following minimum performance is required:

- a) Cable Physical Layer:
 - 1) 98.304 Mbit/s data bit transmit/receive
- b) Backplane Physical Layer:
 - 1) Data bit transmit/receive at 49.152 Mbit/s for BTL and ECL.
 - 2) Data bit transmit/receive at 24.576 Mbit/s for TTL.

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

2.1.3 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.

2.1.4 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 application environment: The physical environment of a backplane serial bus. This includes the bus itself, the modules, and the system that contains them. This environment may be a standardized host backplane (e.g., a

Futurebus+[®] profile) that describes signal requirements, transceivers, mechanical arrangement of the modules, and temperature range over which operation is guaranteed.

2.2.5 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.

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

2.2.7 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.

2.2.8 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.

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

2.2.10 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.

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

2.2.12 backplane physical layer: The version of the physical layer applicable to the Serial Bus backplane environment.

2.2.13 backplane PHY: Abbreviation for the backplane physical layer.

2.2.14 base rate: The lowest data rate used by the 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 \pm 100 ppm.

2.2.15 broadcast_physical_ID: A physical_ID with a value of 111111₂.

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

2.2.17 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.

2.2.18 byte: Eight bits of data.

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

2.2.20 cable PHY: Abbreviation for the cable physical layer.

2.2.21 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.

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

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

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

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

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

2.2.27 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.

2.2.28 doublet: Two bytes of data.

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

2.2.30 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.

2.2.31 gap: A period of idle bus.

2.2.32 initial register space: The address space reserved for resources accessible immediately after a reset. This includes the registers defined by the CSR Architecture as well as those defined by this standard.

2.2.33 IRM: Abbreviation for the isochronous resource manager.

2.2.34 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.

2.2.35 isochronous channel: A relationship between a talker and one or more listeners, identified by a channel number. One packet for each channel is sent during each isochronous cycle. Channel numbers are assigned using the isochronous resource management facilities.

2.2.36 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 μ s, on average.

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

2.2.38 isochronous resource manager (IRM): 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.

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

2.2.40 link layer (LINK): The layer, in a stack of three protocol layers defined for the 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. See figure 3.4 for the relation of the link layer to the Serial Bus protocol stack.

2.2.41 LINK: Abbreviation for the link layer.

2.2.42 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.

2.2.43 local_bus_ID: A `bus_ID` with a value of 11111111₂.

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

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

2.2.46 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.

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

2.2.48 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.

2.2.49 node: An addressable device attached to the 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.

2.2.50 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.

2.2.51 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.

2.2.52 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).

2.2.53 octlet: Eight bytes of data.

2.2.54 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.

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

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

2.2.57 PCB: Printed circuit board.

2.2.58 PDU: Abbreviation for protocol data unit.

2.2.59 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.

2.2.60 PHY: Abbreviation for the physical layer.

2.2.61 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.

2.2.62 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.

2.2.63 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.

2.2.64 physical layer (PHY): The layer, in a stack of three protocol layers defined for the 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 the Serial Bus. There are different physical layers for the backplane and for the cable environment. See figure 3.4 for the relation of the physical layer to the Serial Bus protocol stack.

2.2.65 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.

2.2.66 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.

2.2.67 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.

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

2.2.69 quadlet: Four bytes of data.

2.2.70 quadlet aligned address: An address with zeros in the east significant two bits.

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 SBM: Abbreviation for Serial Bus management.

2.2.74 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.

2.2.75 Serial Bus management (SBM): The set of protocols, services, and operating procedures that monitors and controls the various Serial Bus layers: physical, link, and transaction. See figure 3.4 for the relation of Serial Bus management to the Serial Bus protocol stack.

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

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

2.2.78 source: A node that initiates a bus transfer.

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

2.2.80 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.

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

2.2.82 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.

2.2.83 talker: A node that sends an isochronous subaction for an isochronous channel.

2.2.84 transaction layer: The layer, in a stack of three protocol layers defined for the Serial Bus, that defines a request response protocol to perform bus operations of type read, write, and lock. See figure 3.4 for the relation of the transaction layer to the Serial Bus protocol stack.

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 architecture: The specification document describing the format and function of the software-visible resources of the unit.

3. Summary description

This clause describes the form and function of the Serial Bus in a general way and is not intended to be a normative part of this standard. Instead, it provides useful background information to aid the understanding and implementation of the specification in clauses 4. through 7.3.5.2.1 and annexes A, B, and C.

3.1 Node and module architectures

The Serial Bus architecture is defined in terms of nodes. A node is an addressable entity, which can be independently reset and identified. More than one node may reside on a single module, and more than one unit may reside in a single node, as illustrated in figure 3.1.

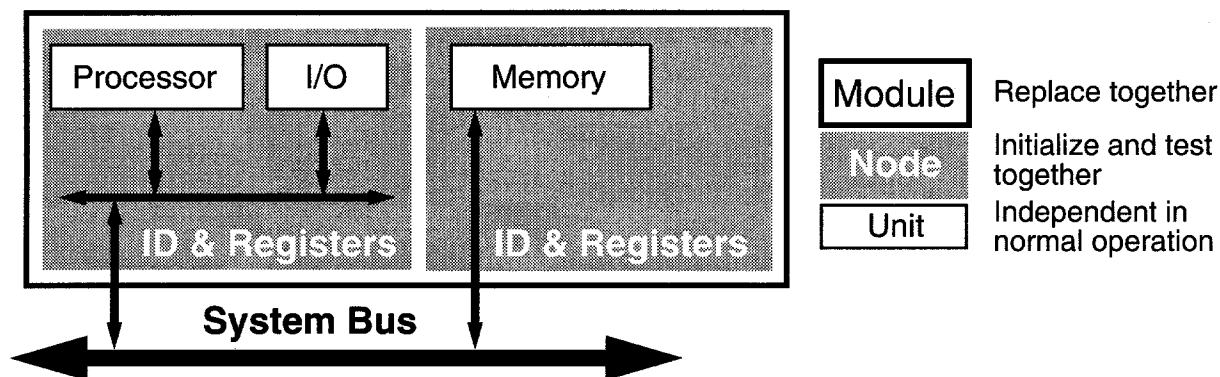


Figure 3.1—Module architecture

Each module consists of one or more nodes, which are independently initialized and configured. Note that modules are a physical packaging concept and nodes are a logical addressing concept. A module is a physical device, consisting of one or more nodes that share a physical interface. In normal operation, a module is not visible to software. Of course, this is not true when the module is replaced, when the shared bus interface fails, or when specialized module-specific diagnostic software is invoked.

A node is a logical entity with a unique address. It provides an identification ROM and a standardized set of control registers, and it can be reset independently.

The address space provided by a node can be directly mapped to one or more units. A unit is a logical entity, such as a disk controller, which corresponds to unique I/O driver software. On a multifunction node, for example, the processor and I/O interfaces could be different units on the same node. Unless the node is reset or reconfigured, the I/O driver software for each unit can operate independently. A unit may be defined by a unit architecture that describes the format and function of the software-visible registers of the unit. SCSI-3 Serial Bus Protocol (SBP) is a typical unit architecture.

Within a unit there may be multiple subunits, which can be accessed through independent control registers or uniquely addressed direct memory access (DMA)-command sequences. Although unit architectures should use the subunit concept to simplify I/O driver software, the definition of subunit architectures is beyond the scope of this standard.

3.2 Topology

The physical topology of the Serial Bus is divided into two “environments,” as shown in figure 3.2. One is called the “backplane environment” and is defined in this standard, although implementations may require additional physical-layer descriptions contained within other backplane bus standards. The other part is the “cable environment” and is completely specified in this standard. Interconnected nodes may reside in either environment without restriction.

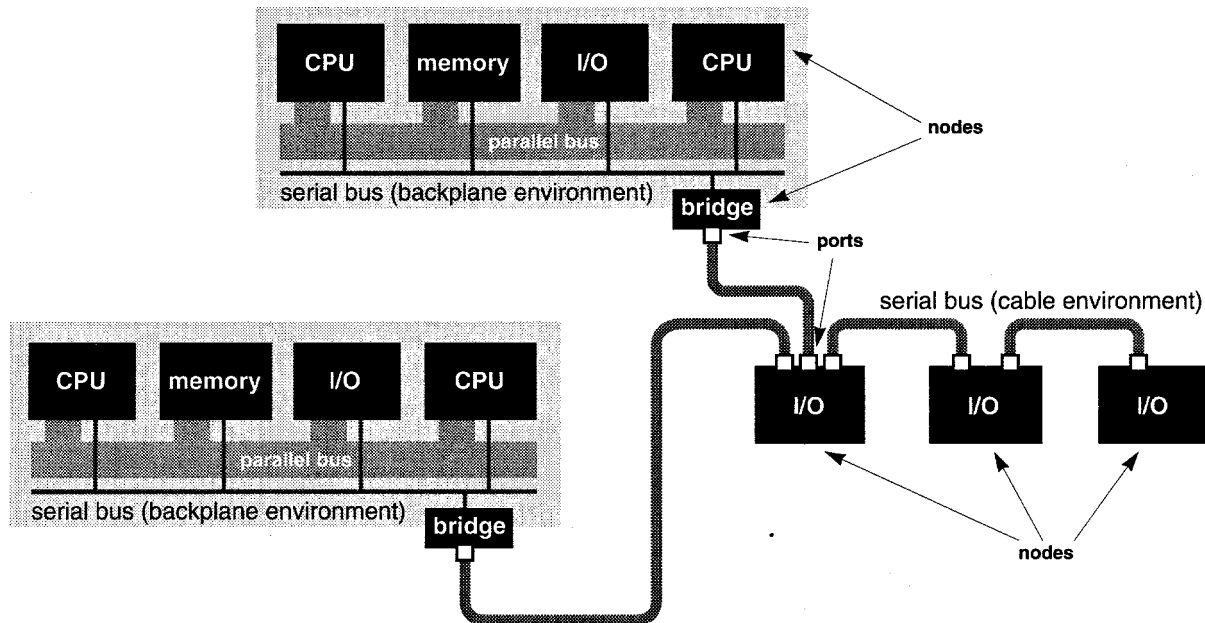


Figure 3.2—Serial Bus physical topology

Note that figure 3.2 shows several logically separate buses. A bus bridge is required to connect buses with different environments. The full implementation of multiple bus systems using bus bridges is not defined in this standard: only the addressing and transactions are specified.

3.2.1 Cable environment

The physical topology for the cable environment is a noncyclic network with finite branches and extent. “Noncyclic” means that closed loops are unsupported. The medium consists of two conductor pairs for signals and one pair for power and ground that connect together ports on different nodes. Each port consists of terminators, transceivers, and simple logic. The cable and ports act as bus repeaters between the nodes to simulate a single logical bus.

Since each node has to continuously repeat bus signals, the separate power and ground wires within the cable enable the physical layer of each node to remain operational even if node local power is off. The pair can even power an entire node if its requirements are modest. The Serial Bus cable provides 8–40 VDC at up to 1.5 A. The actual current available is system dependent.

3.2.2 Backplane environment

The physical topology of the backplane environment is a multidrop bus. The media consists of two single-ended conductors running the length of the backplane. Connectors distributed along the bus allow nodes to “plug into” the bus. Dominant mode (or “wired-OR”) logic allows all nodes to assert the bus.

When in the backplane environment, the Serial Bus typically accompanies a standard parallel bus within an equipment chassis. In such an environment, the Serial Bus is extended from the backplane into each physical device using two pins reserved for a serial bus by the various IEEE bus standards. Drivers and receivers for Serial Bus signals follow the conventions established by the appropriate parallel bus standard: e.g., Futurebus+ using BTL, VME using TTL, and Fastbus and SCI using ECL. Power and ground are distributed as specified for the parallel bus. Examples of backplane implementations are given in annex F

3.3 Addressing

The Serial Bus follows the CSR Architecture for 64-bit fixed addressing, where the upper 16 bits of each address represent the node_ID. This provides address space for up to 64 k nodes. The Serial Bus divides the node_ID into two smaller fields: the higher order 10 bits specify a bus_ID and lower order 6 bits specify a physical_ID. Each of the fields reserves the value of all “1”s for special purposes, so this addressing scheme provides for 1023 buses, each with 63 independently addressable nodes. This standardization is continued within the node, with 2^{48} bytes (256 terabytes) divided between **initial register space** (2048 bytes reserved for core CSR Architecture resources, registers specific to the Serial Bus, and the first 1024 bytes of a ROM ID area), **initial units space** (area reserved for node-specific resources), **private space** (area reserved for node-local uses), and **initial memory space**. Clause 4 of ISO/IEC 13213 : 1994 defines these terms in more detail. Figure 3.3 illustrates the address structure of the Serial Bus.

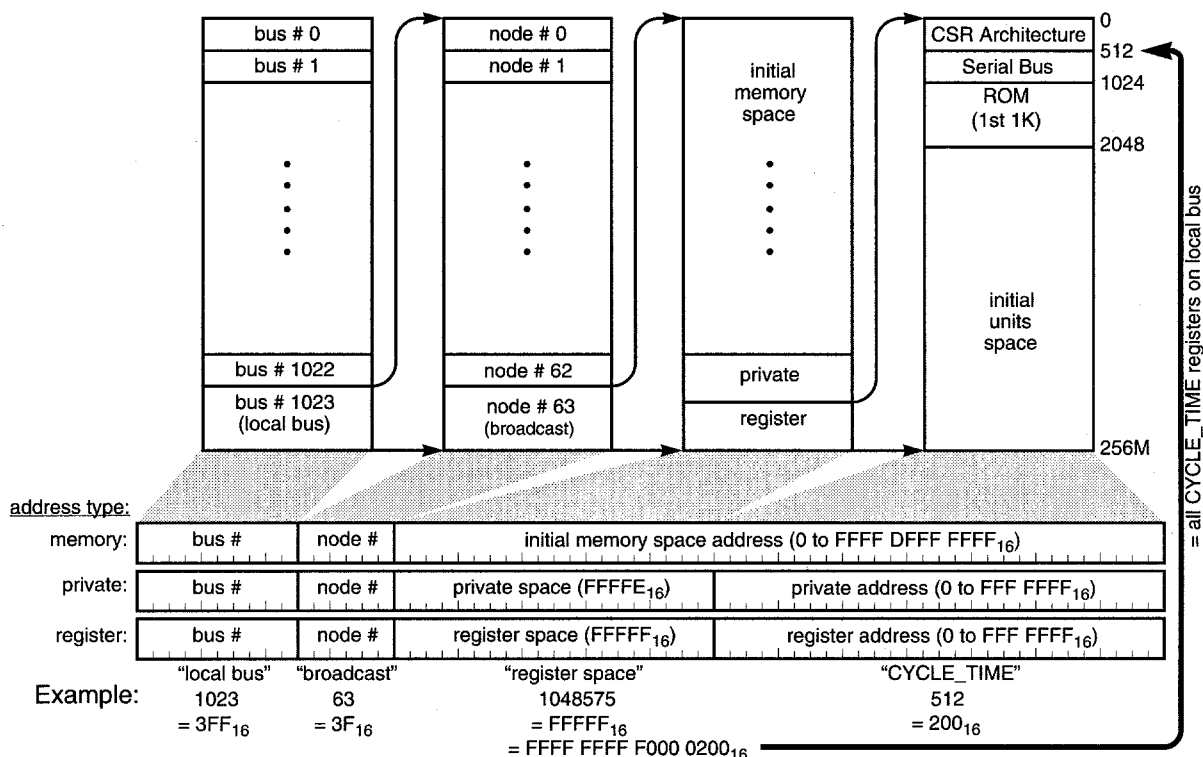
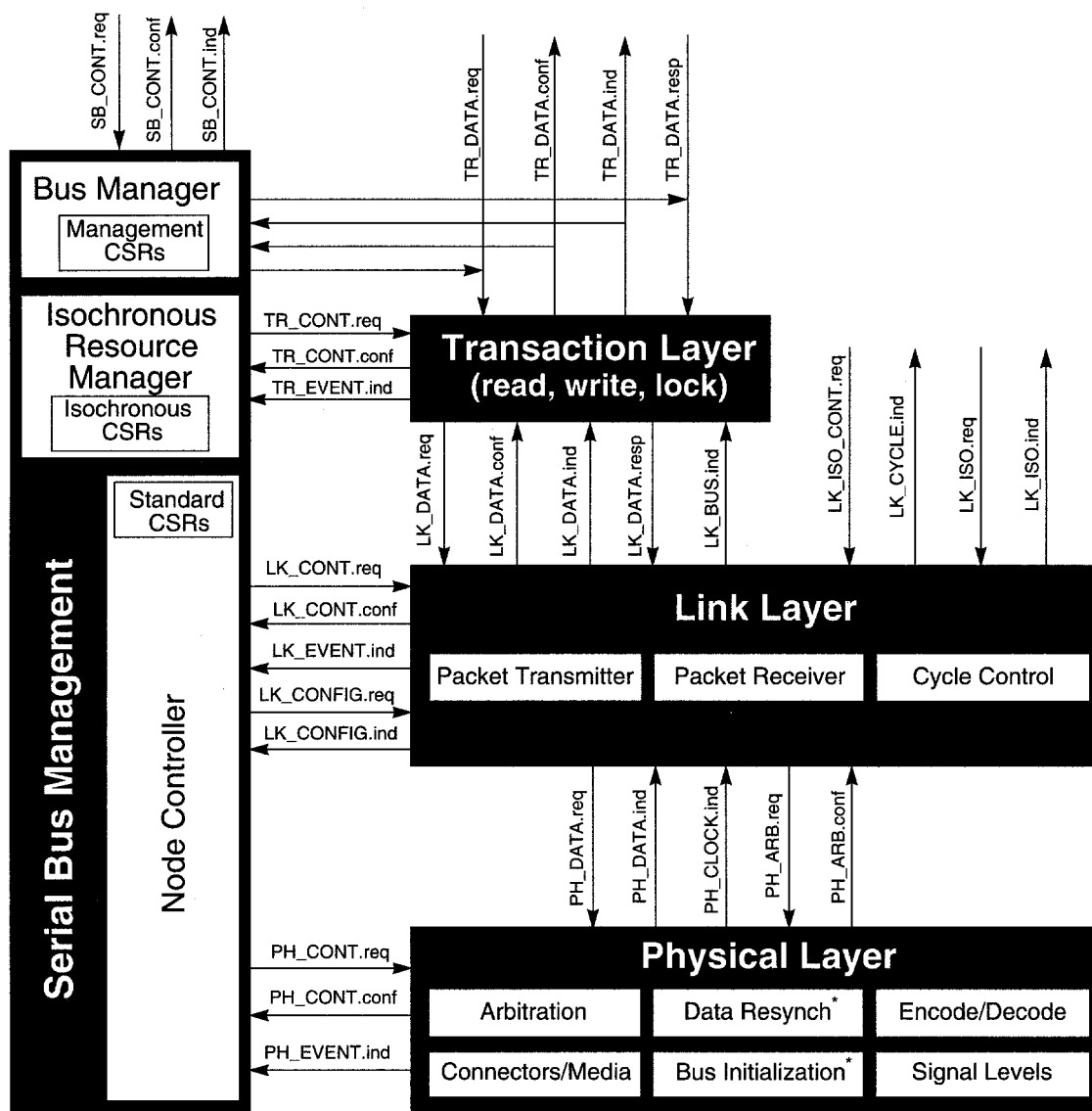


Figure 3.3—Serial Bus addressing

When practical, the node should use only the first 2048 bytes of the initial units space. This simplifies the design of heterogeneous-bus systems, since the 2048 bytes of standard CSRs and ROM and first 2048 bytes of the initial units space use up the 4096 bytes allocated to CSR space on buses using the CSR-Architecture-defined 32-bit extended addressing (Futurebus+ has several profiles that use 32-bit extended addressing). Note, however, that nodes intended solely for Serial Bus systems (or mixed with other 64-bit fixed interconnects, such as SCI) may take full advantage of the 256 megabytes of CSR address space provided by this addressing architecture.

3.4 Protocol architecture

The serial bus protocols are described as a set of three stacked layers, as shown in figure 3.4.



*Only for the cable environment

Figure 3.4—Serial Bus protocol stack

- The transaction layer defines a complete request-response protocol to perform the bus transactions required to support the CSR Architecture (the operations of **read**, **write**, and **lock**). Note that the transaction layer does not add any services for isochronous data, although it does provide a path for isochronous management data to get to the Serial Bus management via reads from and compare-swaps with the isochronous control CSRs.
- The link layer provides an acknowledged datagram (a one-way data transfer with confirmation of request) service to the transaction layer. It provides addressing, data checking, and data framing for packet transmission and reception. The link layer also provides an isochronous data transfer service directly to the application, including the generation of a “cycle” signal used for timing and synchronization. One link layer transfer is called a “subaction.”
- The physical layer has three major functions:
 - It translates the logical symbols used by the link layer into electrical signals on the different Serial Bus media.

- 2) It guarantees that only one node at a time is sending data by providing an arbitration service.
- 3) It defines the mechanical interfaces for the Serial Bus.

There is a different physical layer for each environment: cable and backplane. The cable physical layer also provides a data resynch and repeat service and automatic bus initialization.

The Serial Bus protocols also include Serial Bus management, which provides the basic control functions and standard CSRs needed to control nodes or to manage bus resources. The bus manager component is only active at a single node that exercises management responsibilities over the entire bus. At the nodes being managed (all those that are not the bus manager), the Serial Bus management consists solely of the node controller component. An additional component, the isochronous resource manager, centralizes the services needed to allocate bandwidth and other isochronous resources.

3.4.1 Data transfer services

This standard supports two basic data transfer services:

- a) Asynchronous data transfer
- b) Isochronous data transfer

The asynchronous (*asyn* = any, *chronous* = time) data transfer service provides a packet delivery protocol for variable-length packets to an explicit address and return of an acknowledgment. The isochronous (*iso* = same, *chronous* = time) data transfer service provides a broadcast packet delivery protocol for variable-length packets that are transferred at regular intervals. As shown in figure 3.4, the asynchronous data transfer service uses the transaction layer, whereas isochronous data transfer service is application driven.

3.5 Transaction layer

Data is transferred between nodes on the serial bus by three different transaction types:

- a) Read—data at a particular address within a responder is transferred back to a requester.
- b) Write—data is transferred from a requester to an address within one or more responders.
- c) Lock—data is transferred from a requester to a responder, processed with data at a particular address within the responder, and then transferred back to the requester.

Transactions are multithreaded, in that more than one transaction can be started by a requester before the corresponding response is returned. These are called a split-response transactions.

3.5.1 Transaction layer services

Transactions consist of four service primitives:

- a) Request—the primitive used by a requester to start the transaction.
- b) Indication—the primitive used to notify the responder of an incoming request.
- c) Response—the primitive used by the responder to return status and possibly data to the requester.
- d) Confirmation—the primitive used to notify the requestor of the arrival of the corresponding response.

These primitives and their relation to data flow is shown in figure 3.5.

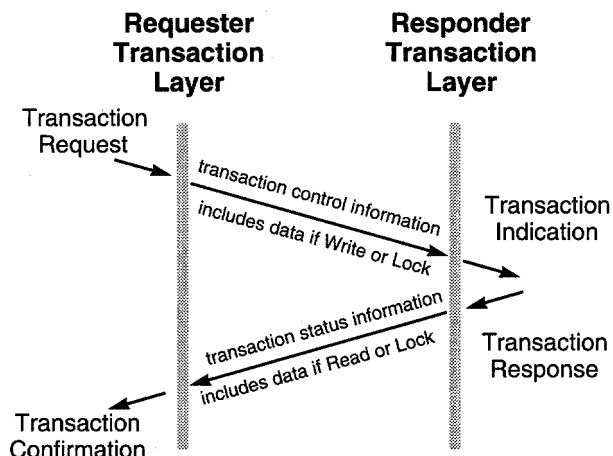


Figure 3.5—Transaction services

The Serial Bus architecture limits the maximum number of data bytes in a transaction to the largest power-of-two such that the whole asynchronous link-layer packet transmission takes less than 62 μ s (this is half the isochronous cycle time, and restricting asynchronous subaction to this value helps minimize buffer requirements). This means that the data payload of asynchronous packets is limited to 512 bytes at the cable base rate of 98.304 Mbit/s. Longer packets can be sent at higher data rates and shorter packets at lower data rates (see table 6.4). Implementations, however, can impose further restrictions. The only required transactions are quadlet write/read on quadlet aligned addresses (these are the transactions necessary to access the standard CSR resources). In addition, nodes that wish to contend for bus management resources shall also implement the quadlet “compare_swap” request and confirmation (see clause 4.) and the nodes that own those resources shall implement the quadlet “compare_swap” indication and response.

3.5.2 Lock subcommands

Since the Serial Bus supports split transactions, it cannot be easily locked while transaction sequences implement indivisible test-and-set operations. Therefore, special lock transactions are defined that communicate the intent from the requester to the responder, thus allowing the indivisible updates to be performed at the responder. There is one standard lock transaction format, but several different subcommands define conditional and unconditional update actions.

Lock subcommands follow the CSR Architecture and are based on the model necessary to implement the fetch and add and compare and swap primitives (the CSR Architecture calls them “fetch_add” and “compare_swap”). The other subcommands define other update actions that can be easily performed with minimal additions to the basic lock-operation hardware. In the lock implementation model, there is a data storage element at the destination (*memory_data*), two data values (*request_data* and *request_arg*) that are sent in the lock request, and one data value (*response_data*) that is returned in the lock response. A block diagram of this model is illustrated in figure 3.6.

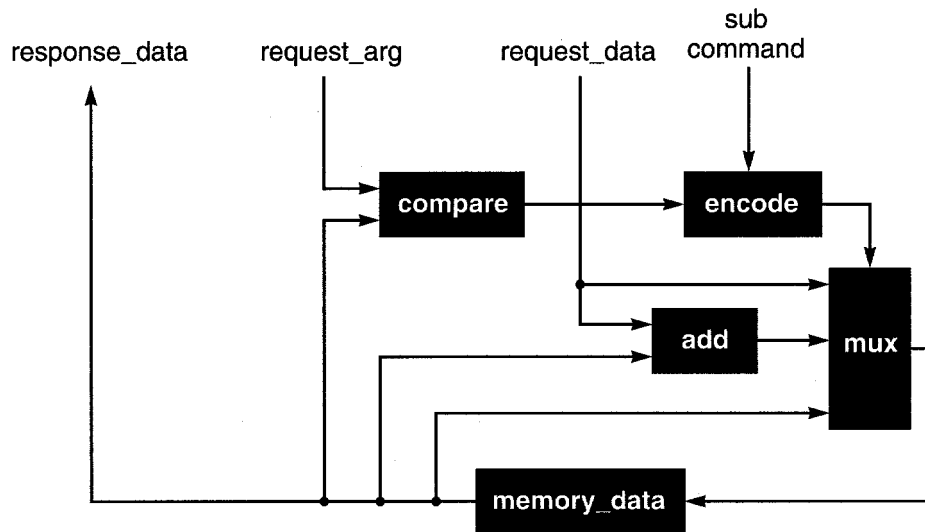


Figure 3.6—Simplified lock model

The compare_swap lock transaction compares the value of request_arg and memory_data. If these values are the same, then memory_data is loaded with the value of request_data. However, if these values are different, then memory_data remains unchanged. In either case, response_data returns the initial value of memory_data.

The mask_swap lock transaction performs a bitwise operation. For each request_arg mask bit that is set to 1, the corresponding memory_data bit is set to the corresponding request_data bit value. For each request_arg mask bit that is cleared, the corresponding memory_data bit is unchanged. In either case, response_data returns the initial value of memory_data.

The fetch_add lock transaction adds the value of request_data to memory_data. The request_arg value is ignored. The response_data returns the initial value of memory_data.

The four data values (memory_data, request_arg, request_data, and response_data) are all the same size, either 32 bits or 64 bits. All lock transactions are optional, but if any are implemented, then the “mask_swap” and “compare_swap” operations shall also be implemented. The C-language coding for these transactions is listed in table 7.1.

3.5.3 Subaction queue independence

For the split-response Serial Bus design model, separate and (effectively) independent queues exist for incoming and outgoing transaction requests and responses. For a simple responder that never explicitly generated read, write, or lock transactions, two queues are sufficient: one for incoming transaction requests and one for outgoing transaction responses, as illustrated in figure 3.7. These queues are independent, in the sense that the sending of response-queue subactions is not affected by the retry state of transaction request-queue packets (see 3.6.2.4 for a discussion of retries).

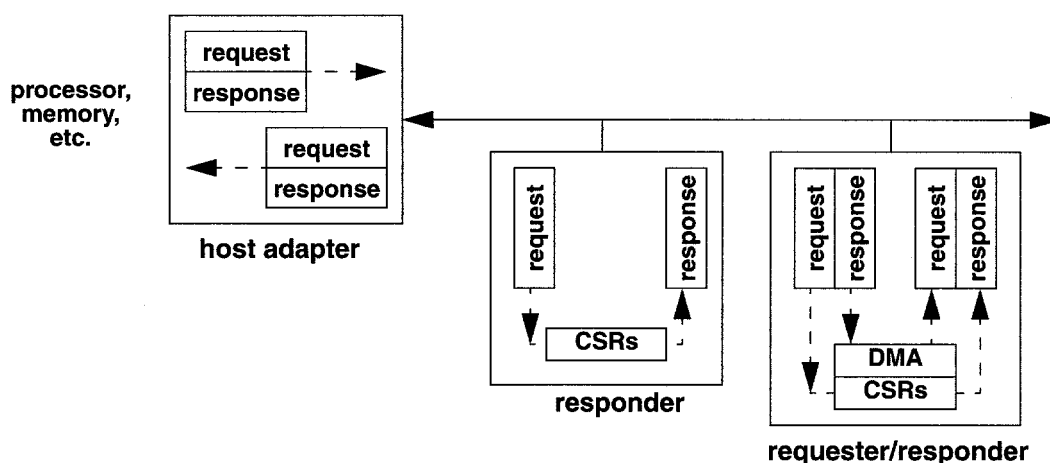


Figure 3.7—Transaction request and response subaction queues

A complex transaction requester/responder node (typically a node with asynchronous DMA capabilities) has two sets of queues. As a transaction requester, an outbound request queue holds the transaction requests waiting to be sent; the inbound response queue holds the transaction responses that are returned. As a responder, an inbound request queue holds the transaction requests addressed to this node; the outbound response queue holds the transaction responses being returned. As a central processing unit (CPU) or other initiator of higher layer protocols, these queues have to be processed independently: to avoid deadlock, sending of a transaction request never blocks the sending of a transaction response; to avoid starvation, sending of a transaction response never blocks sending of a transaction request. In a similar fashion, the DMA device queues also have to operate independently: to avoid deadlock, acceptance of previous transaction requests never blocks the acceptance of a transaction response; to avoid starvation, acceptance of a previous transaction response never blocks the acceptance of a following transaction request.

A bridge has separate queues for transaction requests and responses travelling in each direction, so that transactions initiated from Serial Bus nodes and transactions initiated from the other bus nodes can be processed independently.

The queue independence has an impact on the retry protocols of the node, in that the transaction request and response queues have to be serviced independently. When retry protocols described in 3.6.2.4 are used, requests and responses are retried in a sequential fashion (one per fairness interval as described in 3.7.2), and the retries from the transaction request and response queues have to be interleaved, so that each is retried at least once within four fairness intervals. The four-fairness-interval restriction provides the flexibility to be concurrently processing one retry_A/retry_B and one retry_X from each of the transaction request and response queues.

3.6 Link layer

The link layer provides a half-duplex data packet delivery service. The process of delivering a single packet is called a “subaction,” and there are two types used in the Serial Bus link layer:

- a) An asynchronous subaction—a variable amount of data and several bytes of transaction layer information are transferred to an explicit address and an acknowledge is returned.
- b) An isochronous subaction or “channel”—a variable amount of data is transferred on regular intervals with simplified addressing and no acknowledge.

The subaction has three possible parts:

- 1) Arbitration sequence—a node that wishes to transmit a packet requests the physical layer to gain control of the bus. The physical layer may respond immediately if the node already controls the bus (as it would be if the subaction is the transaction response corresponding to the immediately preceding acknowledge).
- 2) Data packet transmission—for asynchronous subactions, the source node sends a data prefix signal (including a speed code, if needed), addresses of the source and destination nodes, a transaction code, a transaction label, a retry code, data, one or two cyclic redundancy checks (CRCs) and a packet termination (either another data prefix or a data end signal). Isochronous subactions include a short channel identifier rather than source or destination addresses and do not have the transaction label or retry code.
- 3) Acknowledgment—uniquely addressed destination returns a code indicating the action taken by the packet receiver. Isochronous packets and asynchronous broadcast packets do not have acknowledgments. Acknowledgments are also preceded by a data prefix and terminated by another data prefix or a data end.

All asynchronous subactions are normally separated by periods of idle bus called “subaction gaps,” as shown in figure 3.8. Note that a gap opens up on the bus between the packet transmission and acknowledgment reception. This “ack gap” is of varying lengths depending on where the receiver is on the bus with respect to the senders of the link request and acknowledgment (ack). However, the maximum length of the ack gap is sufficiently shorter than a subaction gap to ensure that other nodes on the bus will not begin arbitration before the acknowledge has been received.

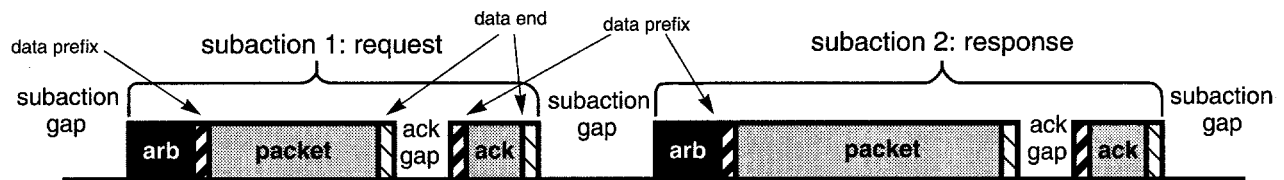


Figure 3.8—Example asynchronous subactions

Similarly, isochronous subactions are separated by periods of idle bus called “isoch gaps,” as shown in figure 3.9.

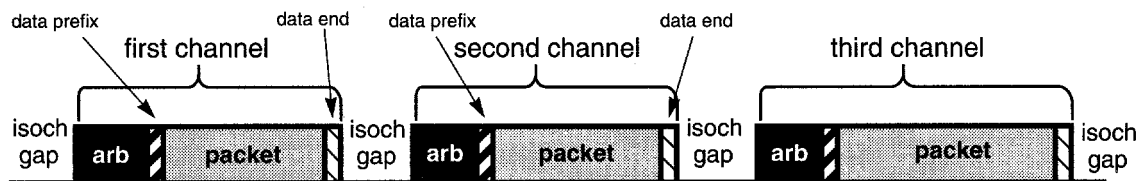


Figure 3.9—Example isochronous subactions

3.6.1 Link layer services

Link layer services also have the request, indication, response, and confirmation service primitives:

- a) Request—the primitive used by a link requester to transmit a packet to a link responder.
- b) Indication—the reception of a packet by a link responder.
- c) Response—the transmission of an acknowledgment by a link responder.
- d) Confirmation—the reception of the acknowledgment by the link requester.

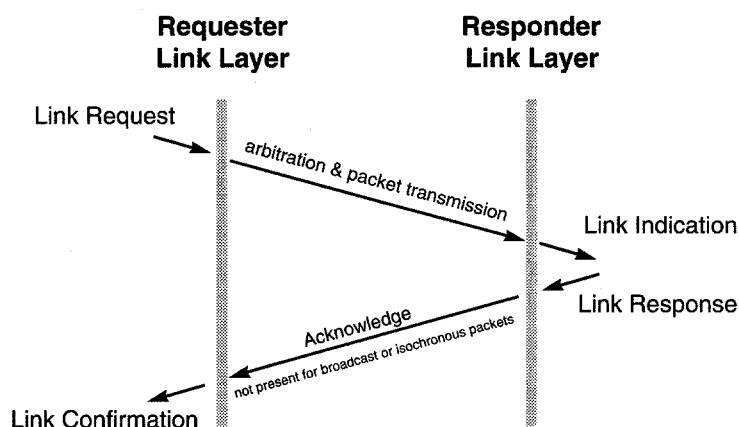


Figure 3.10—Link layer services

3.6.2 Link and transaction layer interactions

The transaction layer and link layer interact in a way that optimizes the use of the bus. In particular:

- a) Write transactions can be implemented in two different ways: unified or split
- b) Split transactions and isochronous subactions can be concatenated under certain circumstances
- c) A busy transaction layer can impose limited flow control on its peers

3.6.2.1 Unified transactions

If the responding transaction layer and link layer are fast enough, an entire write transaction takes a single link layer subaction: the transmission of a data packet and the corresponding acknowledgment as shown in figure 3.11.

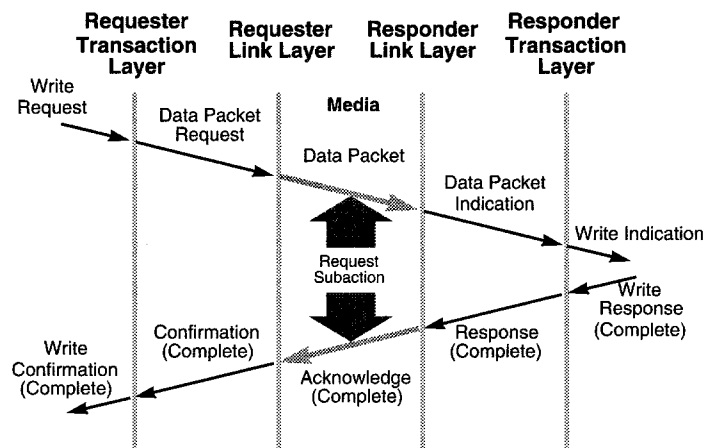


Figure 3.11—Unified transaction example

3.6.2.2 Split transactions

When the responding layers are slow, a split transaction is required with separate link layer subactions for request and response. Note that other link layer subactions may use the bus between the request and response of a single transaction.

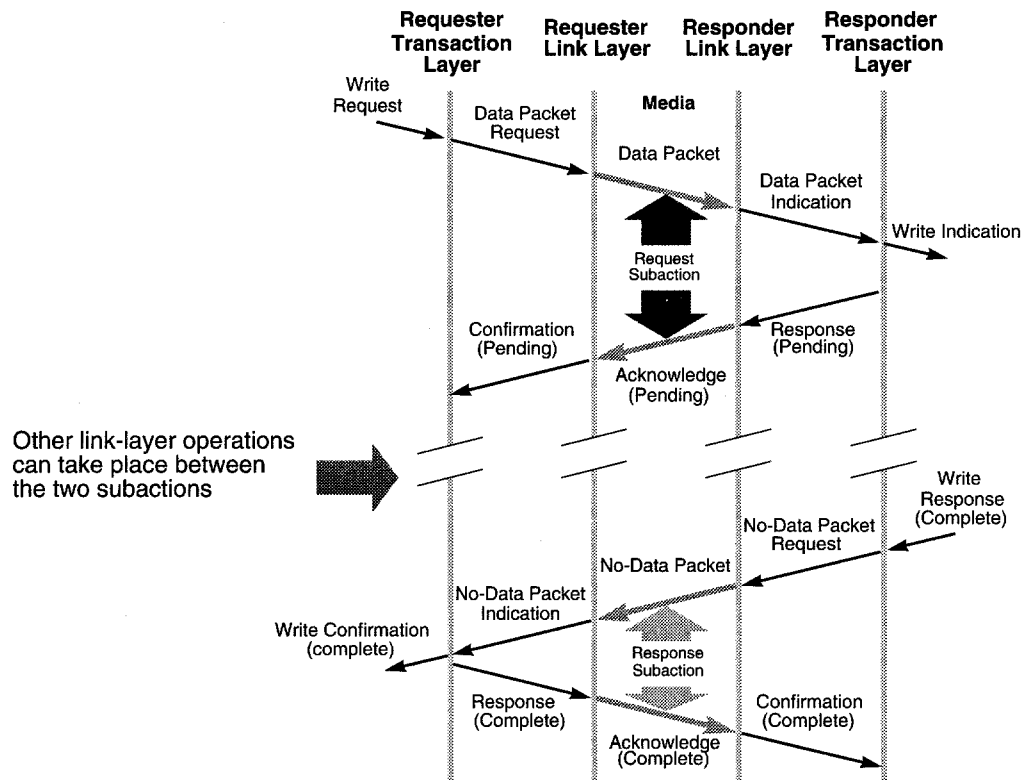


Figure 3.12—Split transaction example

3.6.2.3 Subaction concatenation

Split transactions are always used for read and lock transaction types since data must go both directions. If, however, the read or lock response is fast enough, the two subactions will be directly concatenated by skipping the data end, subaction gap, and arbitration for the response, as shown in figure 3.8 and figure 3.12, to produce the concatenated asynchronous subactions as shown in figure 3.13 and figure 3.14. Since the arbitration for the response is skipped, the fairness process described in 3.7.2 does not apply to the response.

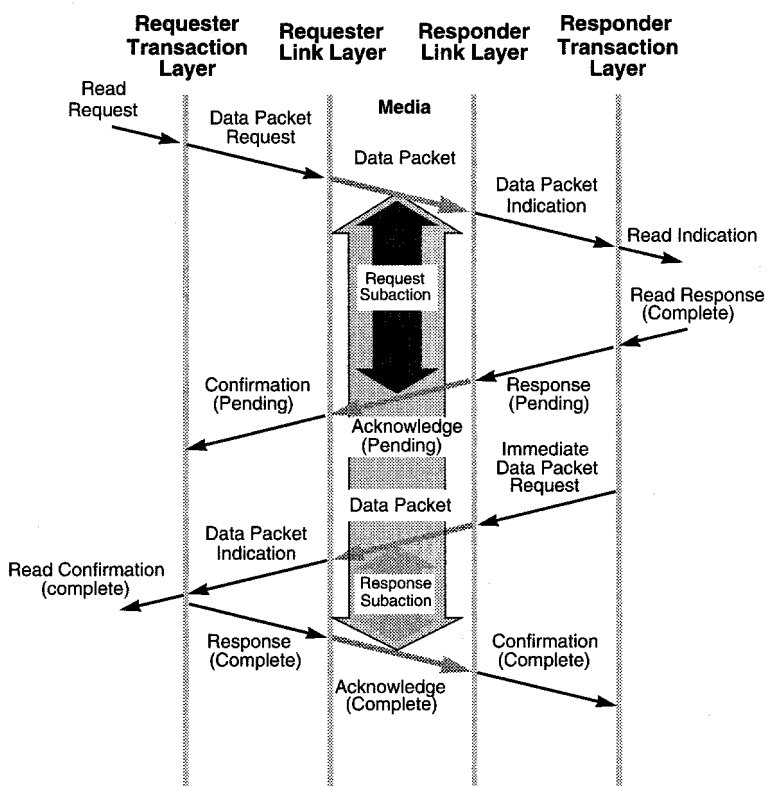


Figure 3.13—Split transaction using concatenated subactions

Note that the data prefix signal is used as a packet terminator for the request acknowledge to indicate the response packet is concatenated.

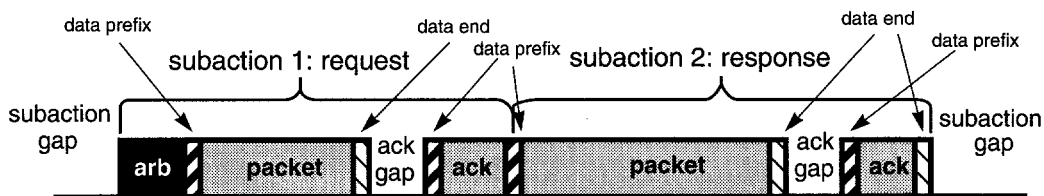


Figure 3.14—Example of concatenated asynchronous subactions

Similarly, isochronous subactions sent by the same node may be concatenated together, as shown in figure 3.15.

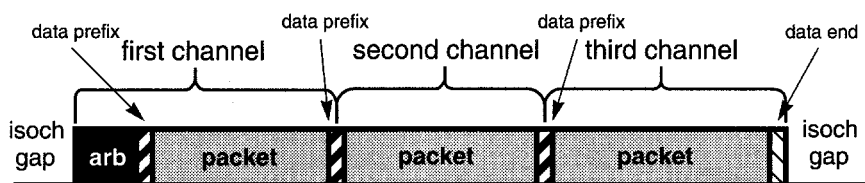


Figure 3.15—Example of concatenated isochronous subactions

3.6.2.4 Retries

On an idle bus, the resources of all devices are always available. On a congested multiple-bus system, however, device queues fill, and subactions have to be busied at the destination node. Two methods of reporting busy to an inbound primary packet and attempting to retry the packet are defined. Note that both transaction request packets and response packets may be retried.

The acknowledge code is used by the destination node to notify the sending node that it is busy and should try back later. The acknowledge code also indicates the phase of the retry, which is used by the destination node to manage the retry process. The retry code (in the primary packet header) is used by the sending node to indicate the retry phase of the packet to the destination node.

The first method is known as “single-phase” retry. This is a simple retry scheme. The transaction layer on a node that implements only single-phase retry will respond with an acknowledge code of `ack_busy_X` to any primary packet it receives (via the link data indication) when the node is busy. The sending node sends and retries the subaction using a retry code (`rt`; see 6.2.4.4) of `retry_X` until the subaction succeeds or its retry limit is exceeded. The transaction layer has the option of requeuing the pending retry and servicing other queued requests as long as the retry limit is not exceeded.

The second method is known as “dual-phase” retry, as specified by the state machine in 7.3.3.2.2. When the transaction layer becomes busy, it responds with an acknowledge code of `ack_busy_A` or `ack_busy_B` (A or B). When the subaction is retried, the `retry_A` is used if the `ack_busy_A` was previously returned or the `retry_B` is used if the `ack_busy_B` was previously returned. While the destination node is accepting `retry_A` retries, other subactions are marked `ack_busy_B`. Once all `retry_A` retries have been accepted, the destination node accepts `retry_B` retries, and other subactions are marked `retry_A`. This process continues in a “ping-pong” fashion, ensuring forward progress by the batch processing of A and B groups.

A timeout is used to determine when all `retry_A` retries have completed. Source nodes using the dual-phase retry shall retry the subaction during every four fairness intervals until their retry timeout expires. Destination nodes assume all `retry_A` subactions have been sent when four fairness intervals pass with no `retry_A` subactions having been busied; the same kind of timeout is also applied to `retry_B`. Higher performance destination nodes may count the number of `retry_l` packets busied with `retry_A` to avoid the timeout when the last of the previously-busied `ack_busy_A` subactions have been retried.

Note that separate `retry_A`/`retry_B` state machines are assumed for the transaction request and response queues of the destination node, so that the acceptance of transaction requests is not affected by the retry-state of transaction responses (and vice versa).

Due to the tight timing constraints on the pending retry (measured in fairness intervals, which can be quite short), these retries are expected to be performed by hardware.

If a source node that only implements single-phase retry receives any of the `ack_busy_X`, `ack_busy_A`, or `ack_busy_B` acknowledge codes, it shall retry using a retry code of `retry_X`. The dual-phase destination node will accept the `retry_X` when it has completed the current retry phase.

3.6.3 Asynchronous arbitration

Asynchronous arbitration is used whenever a link layer wants to send data packets “as soon as possible.” There are two types of asynchronous arbitration:

- a) “Fair” arbitration, which guarantees equal access to the bus for all participating nodes. This prevents nodes that have a higher natural priority from dominating traffic on the bus.
- b) “Urgent” arbitration (only available on the backplane environment) to support nodes that need to use the bus more frequently with less latency. This has to be used with care, since multiple nodes using urgent arbitration can interfere with each other.

3.6.4 Isochronous arbitration

Asynchronous arbitration is adequate for nodes that do not require a guaranteed high bandwidth or low latency or a precise timing reference (less than 1 μ s, for instance). However, data such as that related to digital sound or instrumentation are more efficiently handled with isochronous arbitration.

Isochronous services can be provided without upsetting the basic arbitration protocol by giving the highest priority access to a “cycle master” that maintains a common clock source.⁷ The cycle master tries to transmit a special timing request called a “cycle start” at specific intervals set by a “cycle synch” source (nominally 8 kHz \pm 100 ppm, or 125 μ s \pm 12.5 ns). If a transfer is in progress when the cycle synch occurs, then the cycle start will be delayed, causing significant jitter in the start time of the transfer. Since this jitter is frequently unacceptable, the amount of time that the cycle start was delayed is encoded within the packet as a transaction layer quadlet write request broadcast to the “cycle timer register” of each node.

Each node with isochronous service has a 32-bit cycle timer register. The low-order 12 bits of the register are a modulo 3072 count, which increments once each 24.576 MHz clock period. The next 13 higher order bits are a count of 8 kHz cycles, and the highest order 7 bits count seconds. The cycle master copies its cycle timer register to all isochronous nodes with the cycle start request, synchronizing all nodes within a constant phase difference.

Nodes sending isochronous data respond to cycle starts by immediately arbitrating for the bus without waiting for a subaction gap and sending an isochronous packet as soon as arbitration succeeds. This leads to a smaller minimum gap between isochronous packets than is needed for asynchronous arbitration to start. (The timing for gaps between isochronous packets is the same as for asynchronous acknowledges.) Only when *all* the nodes sending isochronous data have won arbitration and finished sending their data will the bus stay idle long enough for a subaction gap to appear. It is the subaction gap that will allow normal asynchronous arbitration to resume. Figure 3.16 illustrates the basic isochronous arbitration system.

⁷In the cable environment, the highest priority node is the root, so the cycle master must be the root. In the backplane environment, the cycle master uses the highest possible arbitration number (all ones).

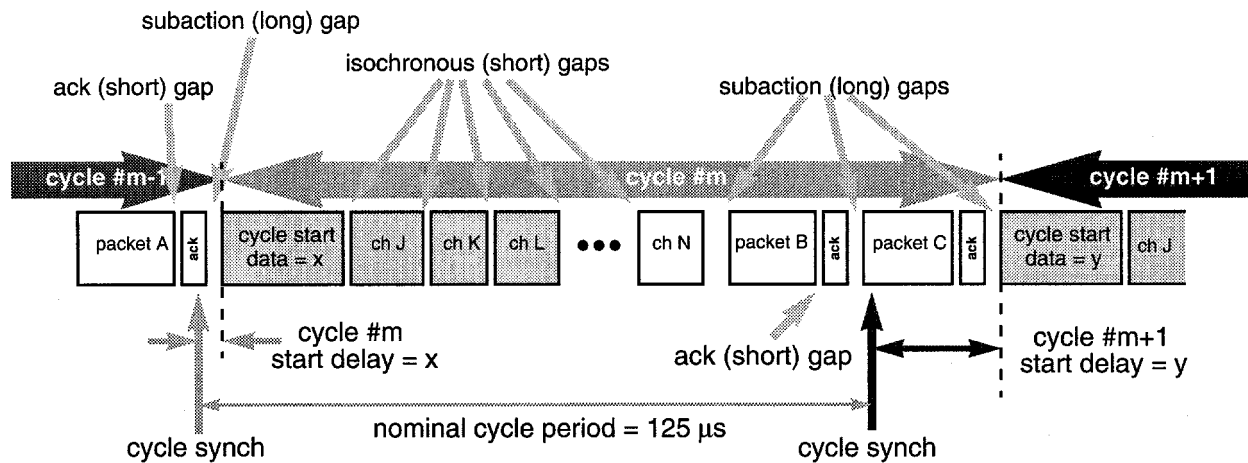


Figure 3.16—Cycle structure

The isochronous packets are labeled with 6-bit “channel” numbers, which have been previously assigned using the channel management protocol described in clause 7.3.5.2.1. Since nodes sending isochronous packets still arbitrate for the bus and the natural priority of a node is not related to channel number, the order of packet transmission is also not related with channel number.

When a node has been assigned a channel, it can send a variable amount of data up to a maximum specified during the bandwidth allocation protocol described in clause 7.3.5.2.1. This protocol will guarantee that the time consumed by all nodes sending isochronous data will not exceed the 125 μ s in a cycle.

3.7 Physical layer

The physical layer (the PHY) has three primary functions: transmission and reception of data bits, arbitration, and provision for the electrical and mechanical interface. The cable and backplane environments have different physical layers. Nonetheless, both physical layers share two fundamental concepts: “data-strobe” encoding for data bits, and a simple method for ensuring fair access to the bus. These similar concepts are described in 3.7.1 and 3.7.2 respectively. Concepts unique to the cable environment are described in 3.7.3, and those for the backplane environment are described in 3.7.4.

3.7.1 Data bit transmission and reception

During packet transmission, there is only a single node transmitting on the bus, so the entire media can operate in a half-duplex mode using two signals: Data and Strb. NRZ data is transmitted on Data and is accompanied by the Strb signal, which changes state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on either Data or Strb for each data bit. A clock that transitions each bit period can be derived from the exclusive-or of Data with Strb as shown below.

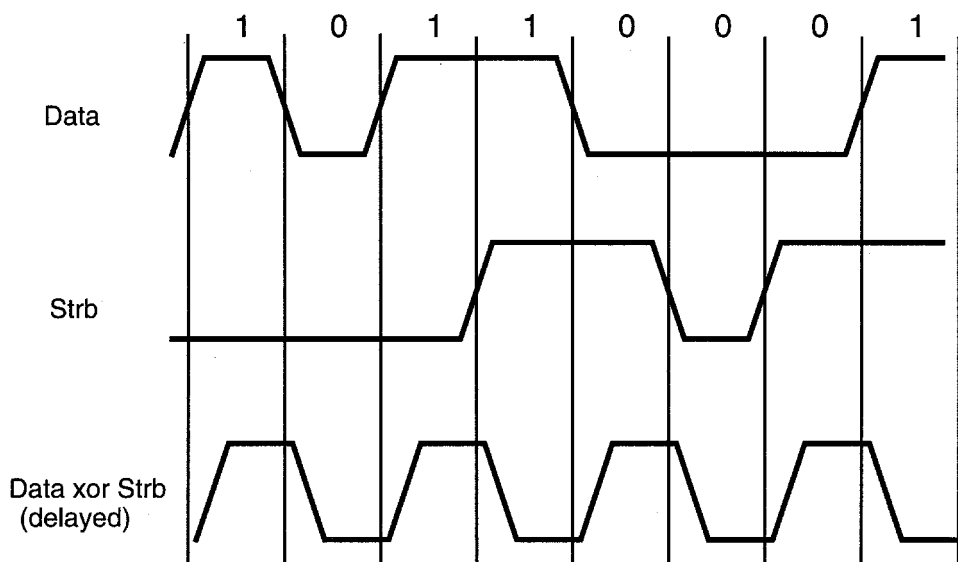


Figure 3.17—Data-strobe encoding

The primary rationale for use of this transmission code is to improve the skew tolerance of information to be transferred across the serial bus. In particular, the code ensures that transitions occurring on Strb and Data are approximately one bit period apart.

An example circuit for decoding data is shown in figure 3.18.

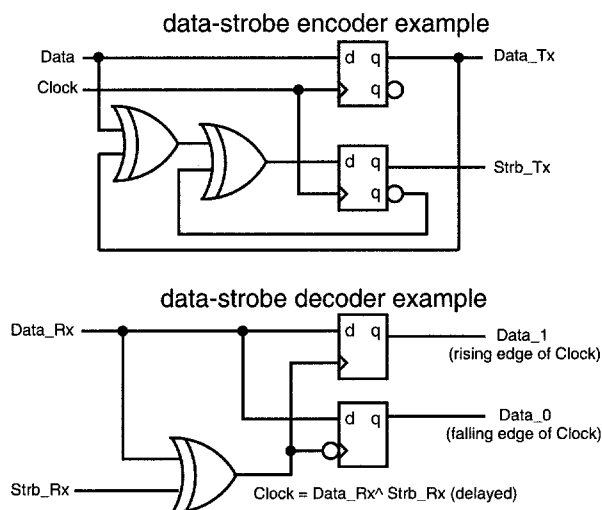


Figure 3.18—Data-strobe encoder and decoder example

3.7.2 Fair arbitration

The normal cable and backplane arbitration methods guarantee that only one node will be transmitting at the end of the arbitration period. As described below, these methods only provide a strict priority access; the node with the highest

natural priority (highest arbitration number on a backplane, closest to the root on a cable) will always win. The normal asynchronous arbitration for the Serial Bus adds a simple scheme that splits the access opportunities evenly among competing nodes.

The fairness protocol is based on the concept of a fairness interval. A fairness interval consists of one or more periods of bus activity separated by short idle periods called subaction gaps and is followed by a longer idle period known as an arbitration reset gap. At the end of each subaction gap, bus arbitration is used to determine the next node to transmit an asynchronous packet. This concept is shown in figure 3.19.

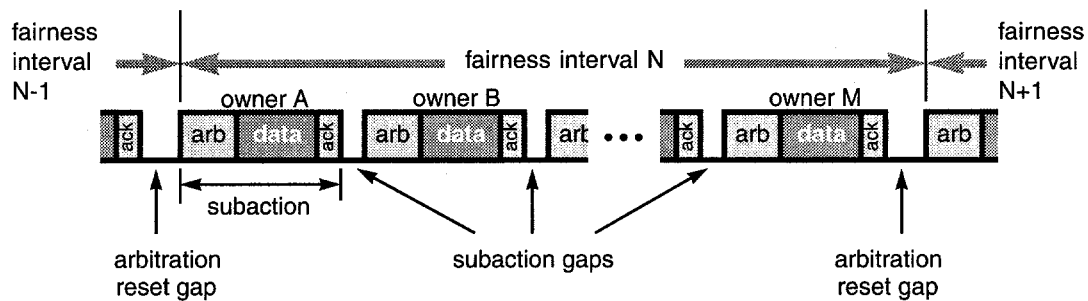


Figure 3.19—Fairness interval

The implementation of the fair arbitration protocol is defined in terms of these fairness intervals, as is discussed in the following clauses.

When using fair arbitration, an active node can initiate sending an asynchronous packet exactly once in each fairness interval. An active node can arbitrate only if its `arb_enable` signal is set. The `arb_enable` signal is set to one by an `arb_reset_gap` and is cleared when the node wins arbitration. This disables further arbitration requests for the remainder of the fairness interval. A fairness interval ends when arbitration by the final fair node is successful; this generates an `arb_reset_gap` since all nodes now have their `arb_enable` signals reset and cannot drive the bus. The `arb_reset_gap` reenables arbitration on all cards and starts the next fairness interval. This process is illustrated in figure 3.20.

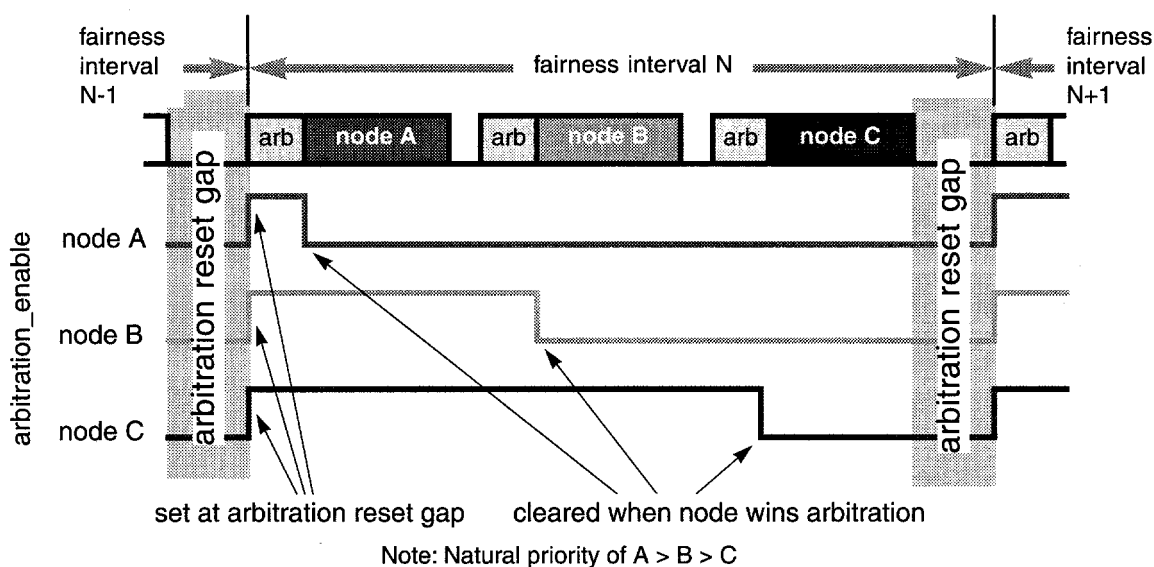


Figure 3.20—Fair arbitration

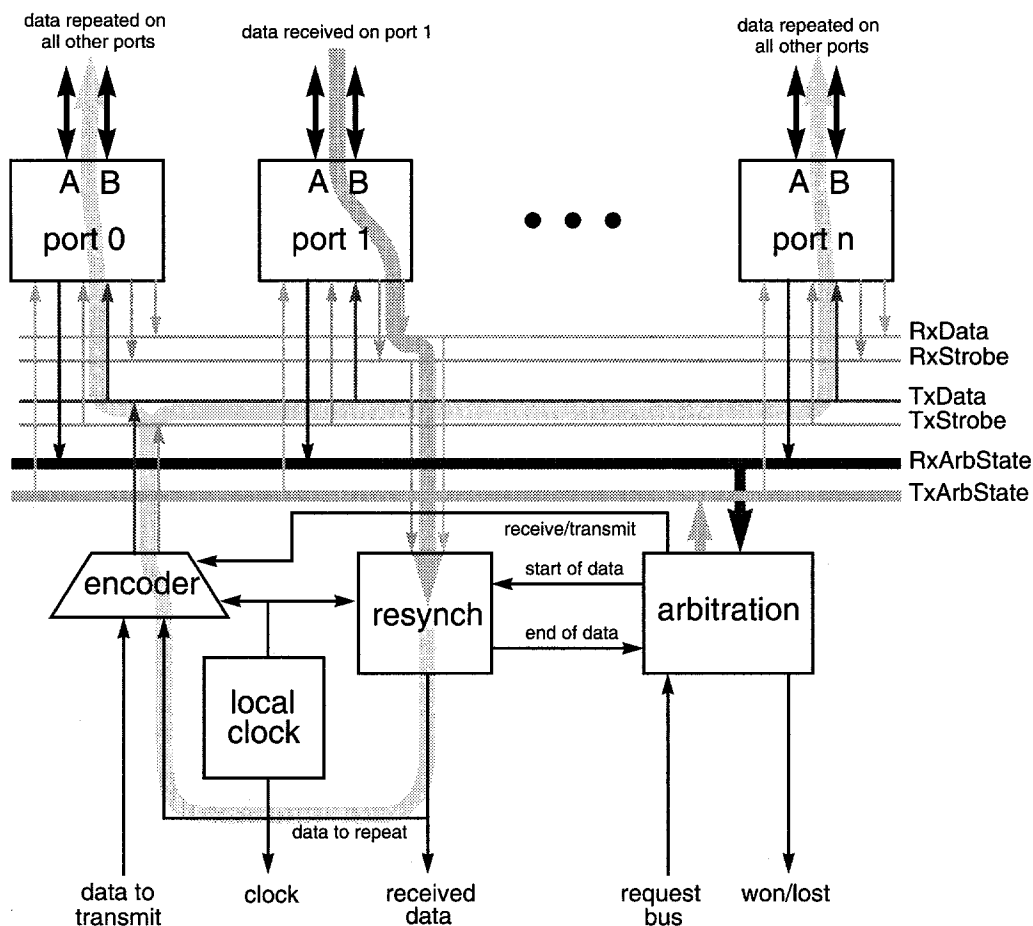
Note that a node sending a concatenated subaction (see 3.6.2.3) does not reset its arb_enable bit.

3.7.3 Cable physical layer

The cable environment is a network of nodes connected by point-to-point links called physical connections. The physical connection consists of a port on the PHY of each node and the cable between them. A PHY can have multiple ports, which allows a branching multihop interconnect as shown in figure 3.2. The primary restriction is that nodes have to be connected together as an acyclic graph (no loops). The cable PHY translates this physical point-to-point topology into the virtual broadcast bus expected by higher layers. The cable PHY does this by taking all data received on one port, resynchronizing it to a local clock, and repeating it out all of its other ports.

The cable PHY logically consists of four major components, as shown in figure 3.21:

- a) Ports, which provide the cable media interface.
- b) Arbitration logic, which provides access to the bus.
- c) The resynchronizer, which takes received data-strobe encoded data bits and generates data bits synchronized to a local clock.
- d) The encoder, which takes either the data being transmitted by the node (if the node has won arbitration) or the data received by the resynchronizer and encodes it in data-strobe format.



to Link Layer

Figure 3.21—Cable PHY

The cable arbitration takes advantage of the point-to-point (non-bused) nature of the cable environment by having each node handshake with its immediate neighbors to determine ownership of the media. There are four phases to this scheme, three to initialize the cable configuration and one for normal arbitration.

3.7.3.1 Cable configuration

Cable configuration is done in three phases: bus initialization, tree identification, and self-identification. During this process, a treelike topology is built; each node is assigned a physical node number and also sends node-specific information that is used by the management layer.⁸

3.7.3.1.1 Bus initialize

Whenever a node joins the bus, a bus reset signal forces all nodes into a special state that clears all topology information and starts the next phase. After bus initialize, the only information known to a node is whether it is a

⁸This process is illustrated in more detail in annex E.3.

branch (more than one directly connected neighbor), a leaf (only a single neighbor), or isolated (unconnected). A network of leaf and branch nodes is illustrated in figure 3.22.

NOTE — In figures 3.22 through 3.29, the two arrows representing the physical connection between the nodes are just an abstract representation of line state signaling. They do not imply that the signaling uses different wire pairs for the two directions; in fact, both directions use both pairs, and the received signal is the result of the dominance of “1” over “0” over “Z.”

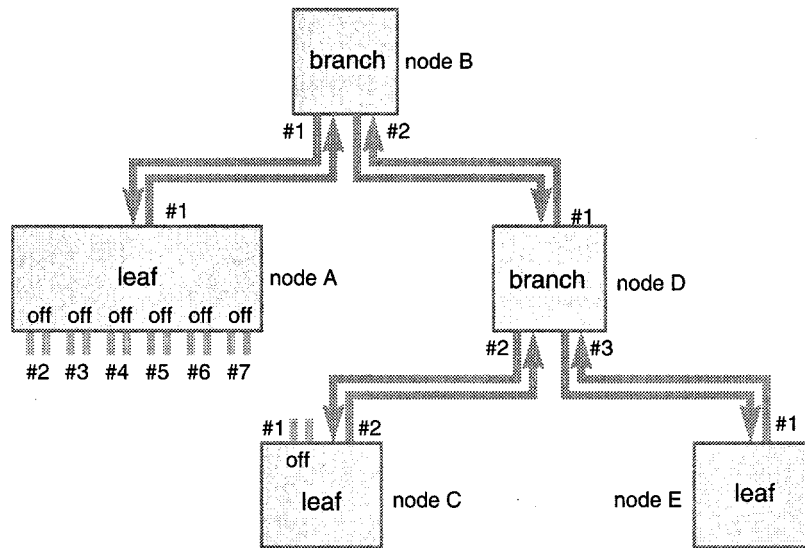


Figure 3.22—Example cable state after bus initialize

Note that each port of the node is individually numbered. There is no particular order to the numbering, it is just a way to give each port a unique label.

3.7.3.1.2 Tree identify

After a bus initialize, the tree-ID process translates the general network topology into a tree, where one node is designated a root and all of the physical connections have a direction associated with them pointing towards the root node. The direction is set by labeling each connected port as a “parent” (connected to a node closer to the root) or “child” port (connected to a node further from the root). Any unconnected ports are labeled “off” and do not participate in further arbitration processes. Any loop in the topology is detected by a configuration time-out in the tree-ID process. When the tree-identify process is complete, the example configuration will have each connected port labeled child (pointing to a child node) or parent (pointing to its parent node), as shown in figure 3.23.

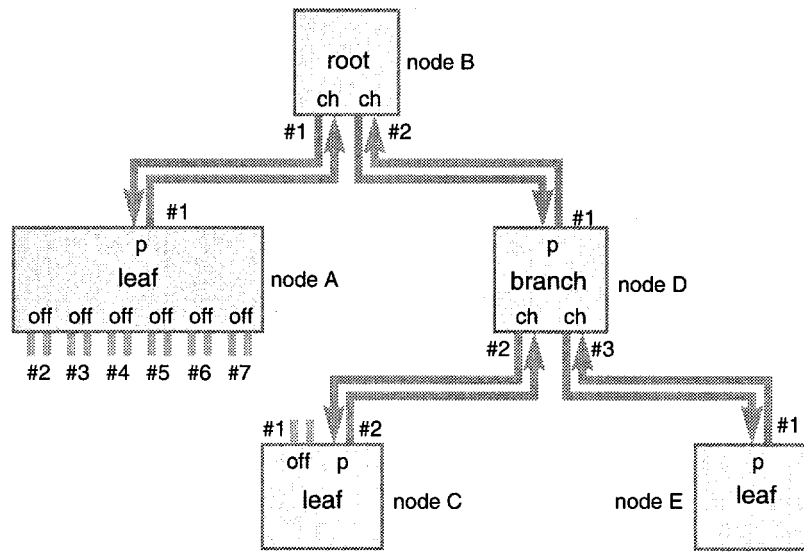


Figure 3.23—Tree identify complete

Note that the selection of the root node is not topology dependent. It is completely acceptable that the root node also be a leaf. The only requirement is that the cycle master (the isochronous cycle master described in 3.6.4) also has to be the root, since the root has the highest natural priority.

The node that has all of its connected ports identified as children becomes the root. A particular node can bias itself toward becoming the root by delaying participation in the tree identify process. Usually, the node that waits the longest time⁹ becomes the root.

A particular node can be forced to wait a longer time by using a special PHY configuration packet (described in 4.3.4.3).

3.7.3.1.3 Self-identify

The next step is to give each node an opportunity to select a unique physical_ID and identify itself to any management entity attached to the bus. This is needed to allow low-level power management and the building of a system topology map needed to determine the speed capabilities of each data path. Clause 7.3.5.2.1 discusses this process.

The self-ID process uses a deterministic selection process, where the root node passes control of the media to the node attached to its lowest numbered connected port and waits for that node to send an “ident_done” signal, indicating that it and all of its children have identified themselves. The root then passes control to its next highest port and waits for that node to finish. When the nodes attached to all the ports of the root are finished, the root itself does a self-identify. The child nodes use the same process in a recursive manner: the completion of the self-ID process is indicated by the bus going idle for a subaction gap period.

Sending the self-ID information is done by transmitting one to four very short packets at the base rate onto the cable that include the physical_ID and some management information. The physical_ID is simply the count of the number of times a node passes through the state of receiving self-ID information before having its own opportunity to send self-ID information, i.e., the first node sending self-ID packet(s) chooses zero as its physical_ID, the second chooses one, and so on. Note that a node is not required to decode the self-ID packet; it merely has to count the number of self-

⁹Longer than a worst-case unrestricted tree-identify process.

identify sequences since the bus reset (this is the number of times the Self-ID Receive state described in 4.4.2.3 is entered).

The management information included in the self-ID packet includes codes for the gap timer settings, the power needed to turn on the attached link layer, the state of the various ports (unconnected, connected to child, connected to parent), and data rate limitations.

Figure 3.24 illustrates the state of the bus after the self-ID process is complete. Note that each child port is labeled “ch-i”, meaning that the node attached to that port is identified.

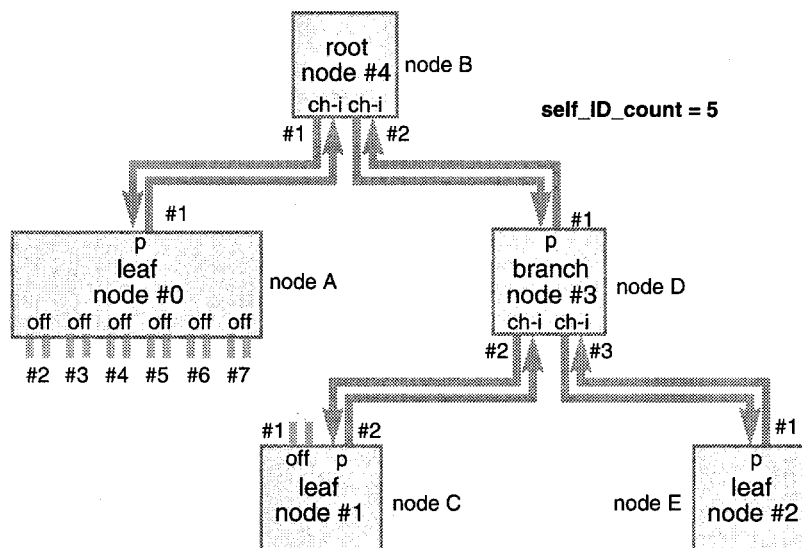


Figure 3.24—Example cable state after self-identify phase

3.7.3.2 Normal arbitration

Once the self-identify process is complete, nodes can use the normal arbitration method to send packets. This process is illustrated by the following example:

- a) Node A and node E begin arbitrating at the same time by sending a request to their parents.

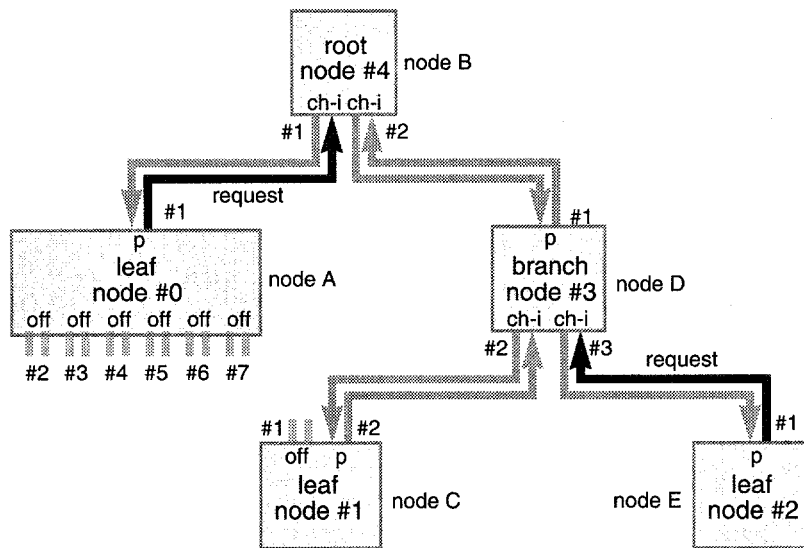


Figure 3.25—Arbitration request

- b) The parent of node E (node D) forwards the request to its parent (node B) and denies access to its other children (node C) by sending a data_prefix, while simultaneously the parent of node A (node B) denies access to its other children (node D). Since node B is root, it does not have to forward the request any further.

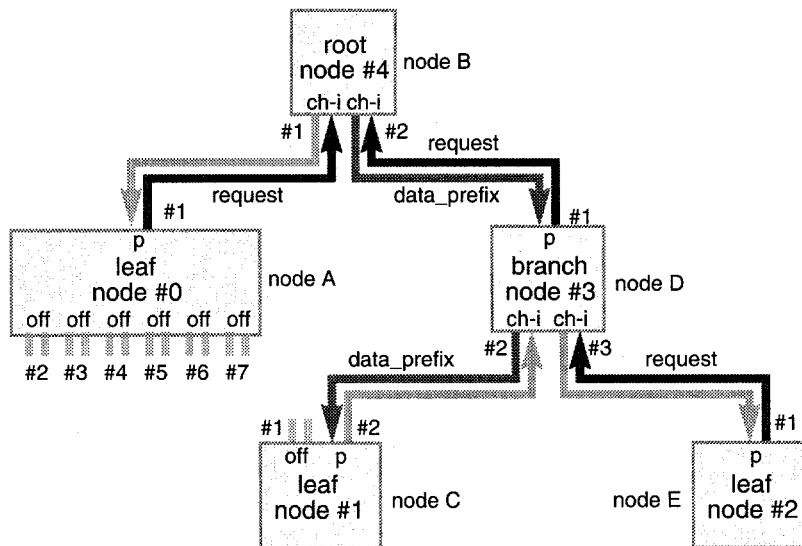


Figure 3.26—Arbitration request (continued)

- c) Instead, the root grants access to the first request (node A), while the other parent (node D) acknowledges the denial by withdrawing its request and passing on the denial.

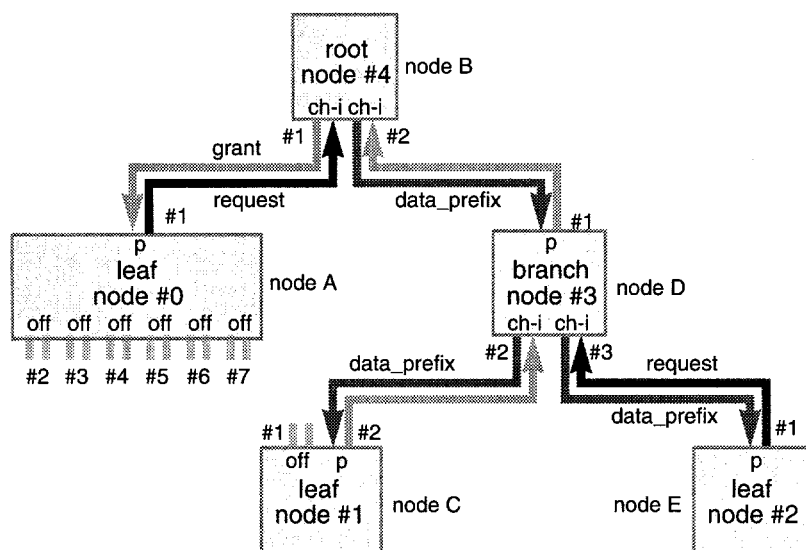


Figure 3.27—Arbitration grant

- d) This causes node E to withdraw its request. Simultaneously, node A receives the grant and, since it was the original requesting node,¹⁰ sends a data_prefix signal to warn all nodes that data is about to be sent.

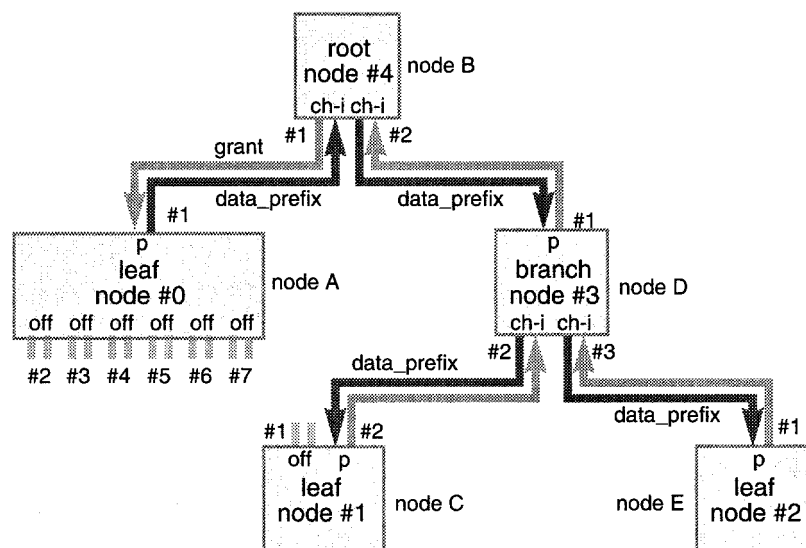


Figure 3.28—Data prefix

- e) The parent of node A (the root in this case) sees the data prefix and withdraws the grant. At this point, the physical connections between all the nodes are now in the same state and pointed away from the node that won the arbitration. This allows the unused second signal to be turned around and used as a strobe to time the transmission of data.

¹⁰If node A had children, they would have received a data_prefix when A started arbitration. If, however, one of the children of node A had requested the bus first, node A would have done an internal deny and passed on the request to the root, and later on the grant when it received it.

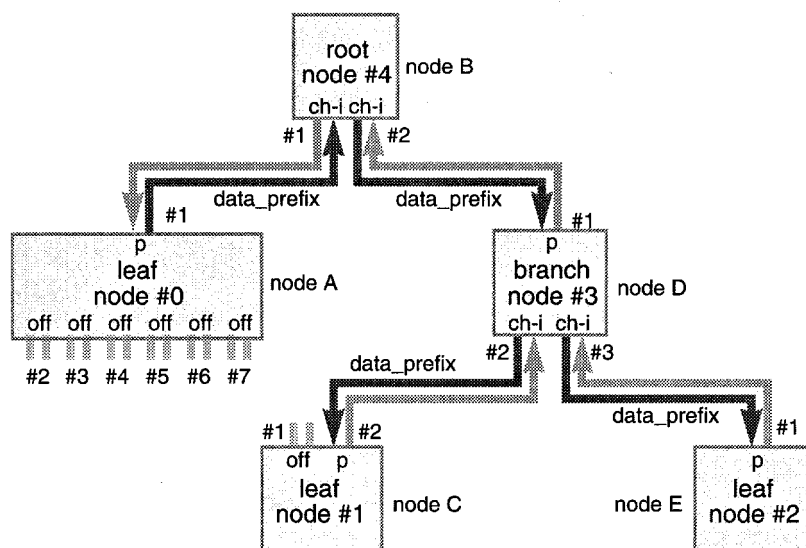


Figure 3.29—Start of data transmission

Clause 4.4.2 specifies the detailed algorithms of the cable environment arbitration.

3.7.3.3 Speed signaling

The cable environment supports multiple data rates of 98.304 Mbit/s, 196.608 Mbit/s, and 393.216 Mbit/s. (The lowest speed is known as the “base rate.”) If a higher rate is supported, then all lower rates are also required. Each node broadcasts its speed capabilities as part of the self-ID packet, and, in addition, each higher speed PHY (one capable of data rates greater than the base rate) exchanges speed information with its parent during the “ident_done” signaling at the end of the self-ID process of a node. This speed capability is recorded by both nodes for later use during normal arbitration. Since a node has already processed the “ident_done” of each of its children before passing on the “ident_done” to its parent, it then has a complete record of the speed capabilities of the nodes attached to each connected port.

In normal packet transmission, a speed code is sent during the data_prefix phase of arbitration. If a directly attached node is incapable of receiving high-speed data, then it is not sent any clocked data; instead, the data_prefix is continually sent on that port until the packet is complete. This will keep the slower attached node from arbitrating while the high-speed data is sent out through the other port(s). Since the slower node propagates the data_prefix to its other ports, all devices “downstream” from that node will also be kept from arbitrating.

Although this process guarantees that all nodes will arbitrate correctly, it is still possible for slower PHYs to act as blocking points for higher speed packets. To prevent this, the initiator of a transaction has to know the speed capabilities of the path between it and a responding node. On fully managed buses, this information is available from the bus manager based on data gathered from the self-ID packets.

3.7.3.4 Cable media interface

The cable media interface is the implementation of a physical connection. There are three major parts of the cable media interface:

- a) The electrical interface to the cable, which consists of
 - 1) TPA and TPB: two low-voltage, low-current, bidirectional differential signals to carry data bits or arbitration signals.

- 2) VP and VG: a power pair that provides the current needed by the physical layer to repeat signals. Nodes can either source or sink current, and there can be multiple power sources on a bus.
- b) The cable connectors, which are small and rugged and provide six electrical contacts plus a shield.
- c) The cable media itself, which has two well-shielded¹¹ signal pairs with relatively high impedance (meaning that little power is needed to drive an adequate signal), and one relatively low-impedance pair for the power.

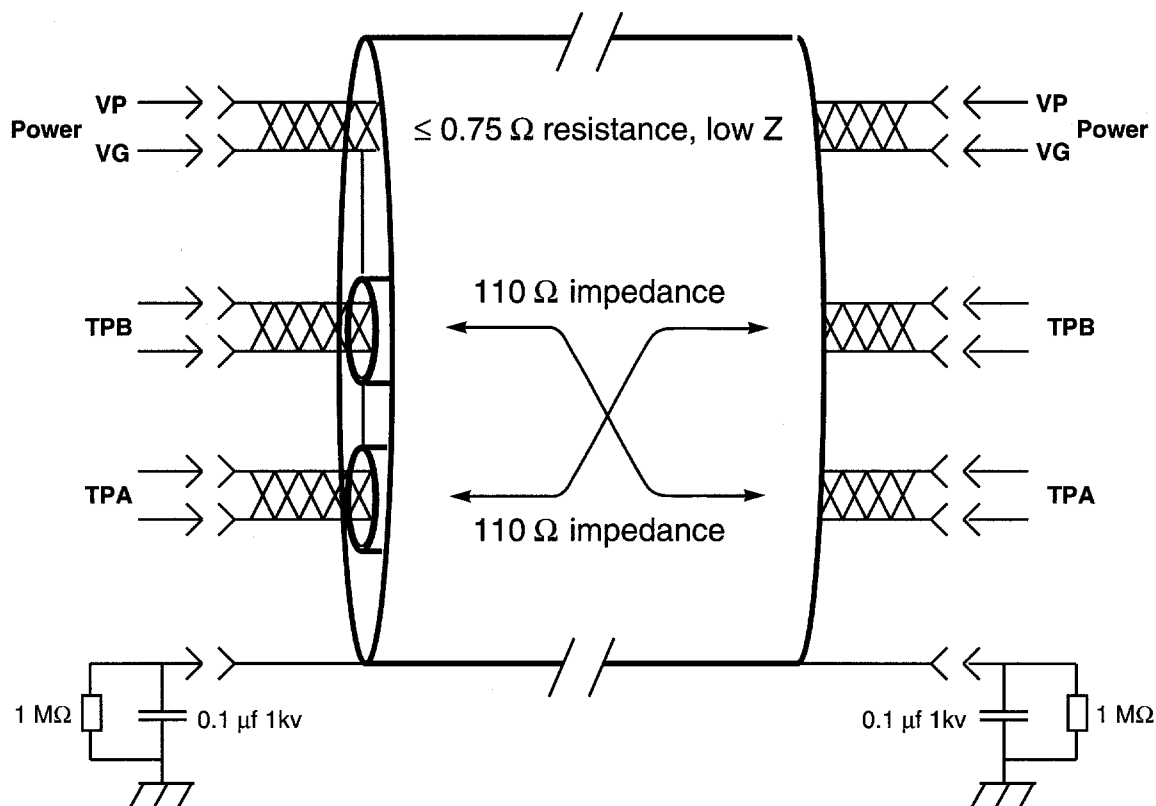


Figure 3.30—Cable interface

The two signals TPA and TPB have three values: “1”, “0” and “Z”, where the “Z” value represents an undriven or idle condition. Normally there is only one node driving a “1” or a “0” on a pair at a time, but the bus reset condition is an exception: a node forcing a bus reset drives both pairs with a “1” signal. Connected nodes can detect this condition because they will detect a different signal on the pair than they expect: i.e., if one node is driving a “0” on the pair and the other is driving a “1”, then the actual received signal will be a “Z” (the positive and negative currents cancel out). The node driving the “0” shall interpret the received “Z” as a “1”, while the node driving the “1” shall always assume the received signal is a “1”. This gives the cable PHY a “1’s dominant” signal mode.

3.7.4 Backplane physical layer

The backplane environment is that of a multidrop bus. There are two signals, Data and Strb, which are shared among all of the nodes on a broadcast bus. Typically, parameters of the bus have to be tightly controlled in order to ensure proper transmission characteristics.

¹¹Shielding is actually optional, although a cable has to be very carefully constructed to avoid common-mode signal crosstalk between the pairs.

3.7.4.1 Backplane arbitration

The backplane environment does not have the initialization requirements of the cable environment since the topology is fixed as a broadcast bus (no repeaters) and physical addresses may be set by the host backplane (e.g., with a built-in slot identifier mechanism).

The backplane arbitration scheme is bitwise comparison of an arbitration number on a dominant-mode medium. It depends on two assumptions:

- a) If any node asserts the bus, then all nodes perceive the bus as asserted after a certain length of time (“wired-or” and “open-collector” are other names for this).
- b) Each node has a guaranteed unique arbitration number.

The procedure followed by competing nodes is

```

START
wait for DATA and STRB to remain unasserted for a fixed period (a gap);
assert STRB;
for each bit in the arbitration sequence {
    if the arbitration bit is a one {
        assert DATA;
        wait one arbitration symbol period (sixteen arbitration clock times); }
    else {
        release DATA;
        wait one sample time (ten arbitration clock times);
        sample DATA;
        if DATA is asserted {
            lost arbitration,
            wait one hold time (six arbitration clock times);
            release STRB and go back to START }
        }
}
won arbitration, release DATA and continue to assert STRB;
begin packet transmission

```

STRB is asserted to ensure that long strings of “0” arbitration bits (when DATA is not asserted) are not interpreted as gaps between packets.

The minimum time required to sample the bus after the transmission of each bit is dependent upon the length and signal propagation speed of the bus. This amount varies between the different backplane environments, but it is assumed to be less than one “sample time.” The sample time is defined to be a constant value, regardless of the interface technology, and is measured in terms of arbitration clock times. One arbitration clock time is approximately 20.345 ns (1/49.152 MHz ± 100 ppm). Once an arbitration bit is put on the bus, the transmission of the next bit in the arbitration sequence cannot begin for a sample time plus an additional “hold time.” The hold time ensures that enough time has elapsed for the bit to be sampled along the length of the bus. It is also defined as a constant value measure in terms of arbitration clock times. Because the sample time and the hold time are independent of the interface technology, the arbitration bit period (equal to the sample time plus the hold time) is the same for all backplane buses.

More information on arbitration bit timing is included in annex D

3.7.4.2 Urgent arbitration

The backplane environment enhances the fair algorithm by splitting access opportunities among nodes based on two priority classes: “fair” and “urgent.” Nodes using an “urgent” priority may use up to three-fourths of the access

opportunities, with the remaining equally shared among nodes using the “fair” priority. All nodes are required to implement the fair priority class, while the urgent priority class is optional. Packets are labeled as “urgent” if that priority class was used.

The fair/urgent allocation uses the same fairness interval described in 3.7.2, but it accompanies the arbitration_enable flag with an “urgent_count.” The fair/urgent method works as follows:

- a) If the bus is idle for longer than an arbitration reset gap, a fairness interval begins and all nodes shall set their “arbitration_enable” flags, while nodes implementing urgent priority shall set their “urgent_count” to three.
- b) A node that is waiting to send a packet using the fair priority class shall begin arbitrating after detecting a subaction gap as long as its arbitration_enable flag is set. If its arbitration_enable flag is cleared, it shall wait for an arbitration reset gap before it begins arbitrating. When such a node wins an arbitration contest, it sends a packet without the “urgent” label and its arbitration_enable flag is cleared.
- c) A node that is waiting to send a packet with urgent priority shall begin arbitrating after detecting a subaction gap if its urgent_count is nonzero. If its urgent_count is zero, it shall wait for an arbitration reset gap before it begins arbitrating. Whenever such a node wins an arbitration contest, it sends a packet with the “urgent” label.
- d) A node implementing urgent priority shall set its urgent_count to three whenever an unlabeled (i.e., fair) packet is transmitted or received. This includes received packets that are addressed to other nodes.
- e) A node shall decrement its urgent_count whenever a packet with the “urgent” label is transmitted or received. This includes received packets that are addressed to other nodes. This ensures that there will be at most three “urgent” packets for every “fair” packet. (This does not ensure that every node using urgent priority will obtain the bus three times each fairness interval. The node arbitrating with the highest priority will always obtain the bus before other nodes arbitrating with an urgent, but lower, priority.)

In the presence of urgent nodes, a fairness interval ends after the final fair node and up to three remaining urgent nodes have successfully accessed the bus. Since all fair nodes now have their arbitration_enable signals reset and all urgent nodes have their urgent_count decremented to zero, none of the nodes can access the bus. The bus remains idle until an arbitration_reset_gap has occurred, re-enabling arbitration on all nodes and starting the next fairness interval. This process is illustrated in figure 3.31, which illustrates a situation where there are three nodes arbitrating for the bus with physical_IDs such that A has the highest, B is in the middle, and C has the lowest, with nodes A and C using fair priority and B using urgent.

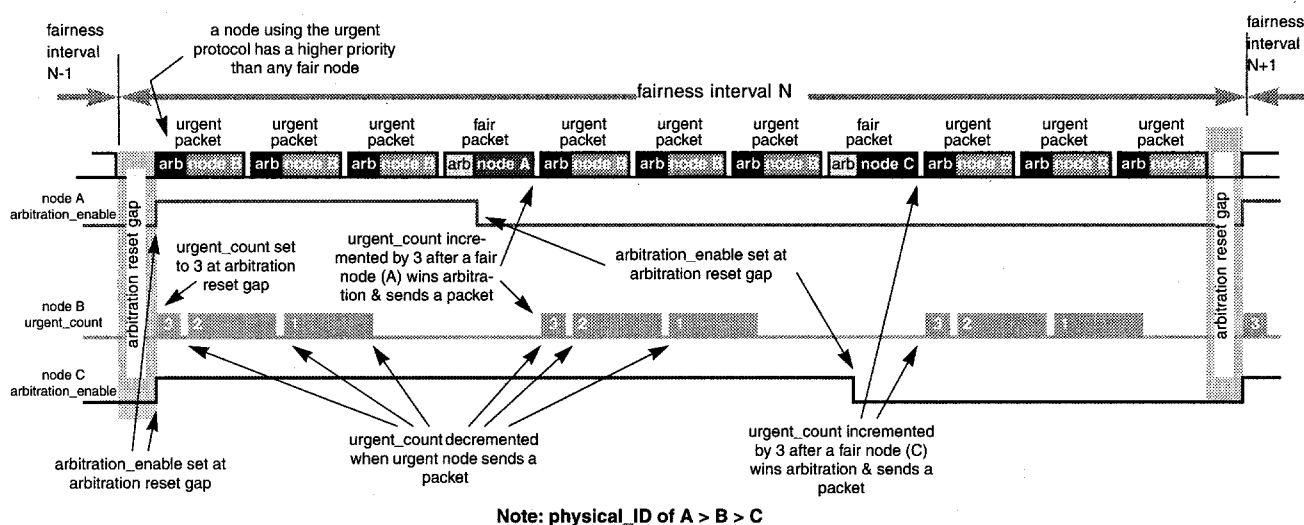


Figure 3.31—Fair/urgent arbitration example

In the backplane environment, the natural priority is the concatenation of the 4-bit urgent priority level with the `physical_ID`. This results in the following:

- a) A node using the urgent priority will always win an arbitration contest over all nodes using the fair priority.
- b) The node using the highest priority level will win the arbitration contest.
- c) If more than one node uses the highest priority level, then the one with the highest `physical_ID` will win.

3.7.4.3 Backplane media interface

The backplane media interface provides the mechanical and electrical interface that allows nodes to communicate. This includes the physical media, which consists of two single-ended conductors running the length of the backplane, as well as connectors distributed along the bus, which allow nodes to “plug into” the bus. The backplane media interface also includes transceivers for the transmission and reception of signals on the physical media. Dominant mode (or “wired-or”) drivers allow all nodes to assert the bus.

When in the backplane environment, the Serial Bus typically accompanies a standard parallel bus within an equipment chassis. In such an environment, the Serial Bus is extended from the backplane into each physical device using two pins reserved for a serial bus by the various IEEE bus standards. Drivers and receivers for Serial Bus signals follow the conventions established by the appropriate parallel bus standard: e.g., Futurebus+ using BTL, VME using TTL, and Fastbus and SCI using ECL. Power and ground are distributed as specified for the parallel bus. Example implementations of the backplane environment Serial Bus are included in annex F

3.8 Bus management

Serial Bus management describes the protocols, services, and operating procedures whereby one node exercising management level control is able to govern the operation of the remaining nodes on the bus. Within all environments the node controller component as well as CSR and configuration ROM facilities are used to configure and manage the activities at an individual node.

Within the cable environment, there are two management entities defined, the isochronous resources manager and the bus manager. These two management entities may reside at different nodes or on the same node. A valid combination for management purposes is the presence of the isochronous resource manager entity and the absence of the bus manager entity. In this circumstance, the isochronous resources manager exercises a subset of the management responsibilities normally assumed by the bus manager. Another valid combination is the absence of any bus manager entity, in which case no isochronous traffic is allowed.

The bus manager provides the services of

- a) Advanced bus power management
- b) Maintenance of the speed map
- c) Maintenance of the topological map
- d) Bus optimization based on information obtained from the topological map

The isochronous resources manager provides facilities for

- a) Allocation of isochronous bandwidth
- b) Allocation of channel numbers
- c) Selection of the cycle master

Within the backplane environment, significantly more elements are determined by the host backplane, so there is a much reduced need for any centralized level of control. The principal backplane area potentially needing management support occurs when the optional facility of isochronous data is supported.

4. Cable PHY specification

The cable environment physical layer provides the actual interface between nodes, including both the mechanical and electrical interface. At the lowest level, it includes the physical connection, which consists of the cable media itself, the connectors, and the electrical transceivers. The cable PHY also performs a number of higher-level functions including bus state determination, bus arbitration, all the encoding and decoding functions, synchronization of received data to a local clock, and a data repeat function.

This clause includes a formal services definition, a specification for the physical connection components, and the facilities and operations of the higher layer functions.

Annex A provides additional specifications appropriate for system designers. Note that some normative (required) specifications are included in this annex, including A.1.2, A.2, A.2.2, and A.4.

4.1 Cable PHY services

PHY layer services are provided at the interface between the PHY layer and higher layers; specifically, the link layer and the node controller, as illustrated in table 4.1. The method by which these services are communicated between the layers is not defined by this standard. PHY layer services may perform actions specified by the higher layer. PHY layer services may also communicate parameters that may or may not be associated with an action.

Table 4.1—Summary of cable PHY layer services

Service	Layer communicated with	Purpose of service
Cable PHY control request	From the node controller	Configure the cable PHY layer
Cable PHY control confirmation	To the node controller	Confirm cable PHY control request
Cable PHY event indication	To the node controller	Alert node controller to events detected in the cable PHY layer
Cable PHY arbitration request	From the link layer	Cause the cable PHY to request control of the bus
Cable PHY arbitration confirmation	To the link layer	Confirm cable PHY arbitration request
Cable PHY clock indication	To the link layer	Indicate when it is time for the link layer to make a cable PHY data request
Cable PHY data request	From the link layer	Cause the cable PHY layer to send one clocked data symbol or to start or end a data packet
Cable PHY data indication	To the link layer	Indicate the reception of one clocked data symbol or a bus status change

4.1.1 Cable PHY bus management services for the management layer

These services are used by the node controller component of the Serial Bus management layer to control the bus level actions of the PHY layer. The PHY layer uses these services to communicate changes of state within the PHY layer or on the bus.

4.1.1.1 PHY control request (PH_CONTROL.request)

The node controller uses this service to request the PHY layer to perform specific actions and to specify PHY layer parameters. It may also be used to request status about the PHY layer. The PHY layer shall service the request immediately upon receipt by the PHY layer. This service is confirmed.

The following actions shall be provided by this service:

- a) **Bus Reset.** The PHY layer shall reset the bus and initialize itself.
- b) **Disable Transmit.** The PHY layer shall set all bus outputs to the Z state. This bus output state shall be maintained until the node controller requests an Enable Transmit action. Link layer service actions that would require a change in bus output state shall not be performed.
- c) **Enable Transmit.** The PHY layer shall allow link layer service actions to change the state of the bus outputs.
- d) **Present Status.** The PHY layer shall return status to the node controller. The PHY layer shall return status via the PHY control confirmation service.
- e) **Set Link Active.** Set the link_active flag (sent during the self-ID process) so that remote nodes will be aware that this node has a functioning link.
- f) **Set Gap Count.** Set the duration of the subaction gap and the arbitration reset gap for the local node. (See 4.3.6.) This action causes the gap_count_reset_disable flag to be set (meaning that the next bus reset will not force the gap count back to the default value—this keeps the gap count constant across a *single* bus reset).
- g) **Set Force Root.** Set the local force_root parameter. This Boolean parameter is set TRUE to delay the start of the tree-ID process (see 4.4.2.2). If a single node has this flag set, then it will become the root node regardless of the topology of the bus. If more than one node has this bit set, any of those nodes or any node on the path between those nodes may become the root.
Important—This flag should not be set in single-node buses.
- h) **Clear Force Root.** Clear the local force_root parameter. See item g).
Important—This action shall be taken in single-node buses.

NOTE — The Gap Count and Set Force Root parameters are set in remote nodes by using the PHY configuration packets, as described in 4.3.4.3. These packets are sent using the services of the link layer, as described in 6.1.1.4. The bus or isochronous resource manager is responsible for sending these packets, as described in 8.4.4.4.

The following parameter is communicated via this service:

- **Gap Count.** (Optional, only present when the Set Gap Count action is requested.) The new value of gap_count for the local node.

4.1.1.2 PHY control confirmation (PH_CONTROL.confirmation)

The PHY layer uses this service to confirm the results of a PHY control request service. The PHY layer shall communicate this service to the node controller upon completion of a PHY control request. There are no actions provided by this service. When the corresponding control request is “Present Status,” the following parameters are communicated via this service:

- **Physical_ID.** As described in 4.3.8.
- **Gap_count.** As described in 4.3.8.
- **PHY_SPEED.** As described in 4.3.7.
- **NPORT.** As described in 4.3.7.
- **Child** [one for each port]. As described in 4.3.9.
- **Connected** [one for each port]. As described in 4.3.9.
- **Arb_A_rx** [one for each port]. As described in 4.2.2.
- **Arb_B_rx** [one for each port]. As described in 4.2.2.
- **Force_root.** As described in 4.3.8.
- **Root.** As described in 4.3.8.
- **Cable_power_active.** As described in 4.3.8.
- **Initiated_reset.** As described in 4.3.8.

4.1.1.3 PHY event indication (PH_EVENT.indication)

The PHY layer uses this service to indicate PHY-level events to the node controller. There are no actions provided by this service. No response is defined for this indication. The following parameters are communicated via this service:

- a) PHY Event. This parameter shall contain the event detected by the PHY layer. The following values are defined for this parameter:
 - 1) CABLE_POWER_FAIL. The PHY layer has detected a loss of cable power, as described in 4.2.2.7.
 - 2) BUS_RESET_START. The PHY layer has detected a bus reset.
NOTE — More than one BUS_RESET_START can be generated before the BUS_RESET_COMPLETE event is generated.
 - 3) BUS_RESET_COMPLETE. The PHY layer has detected a subaction gap after the bus reset process has started.
 - 4) CONFIG_TIMEOUT. The PHY layer has not completed the PHY layer initialization due to a timeout in the tree ID process.
NOTE — The CONFIG_TIMEOUT may indicate a loop in the topology.
 - 5) LINK_ON. The PHY layer has received a link_on packet addressed to this node, as described in 4.3.4.2.
 - 6) SELF_ID_COMPLETE. The PHY layer has transmitted its self-ID packet, as described in 4.4.2.3. The Physical_ID and Root parameters will also be valid for this indication.
NOTE — Other nodes will not have completed their reset processing until the PH_EVENT.indicate of BUS_RESET_COMPLETE has been received.
- b) Physical_ID. (Conditional, valid only when the PHY event is BUS_RESET_COMPLETE.) This parameter shall contain a unique 6 bit number, as determined by the self-ID process described in 4.4.2.3.
- c) Root. (Conditional, valid only when the PHY event is BUS_RESET_COMPLETE.) This Boolean parameter is TRUE only when this node is the root, as determined by the tree ID process described in 4.4.2.2.

4.1.2 PHY layer arbitration services for the link layer

These services are used to communicate arbitration requests between the PHY layer and the link layer. See 4.4.2.4.

4.1.2.1 PHY arbitration request (PH_ARB.request)

The link layer uses this service to request the PHY layer to start arbitration for the bus. The PHY layer shall arbitrate for the bus using the method specified by the service parameters. The PHY layer shall service the request immediately upon receipt from the link layer. This service shall be confirmed when the arbitration process completes. If a node loses arbitration (PH_ARB.confirmation with an arbitration request status of LOST), it shall reissue the arbitration request.

The following parameters are communicated via this service:

- a) Arbitration Class. This parameter shall contain the method of arbitration performed by the PHY arbitration request. The method of arbitration shall be one of the following:
 - 1) FAIR. The PHY layer shall start arbitration at the next subaction gap if its arb_enable flag is set; otherwise, it shall start arbitration at the next arbitration reset gap. This request can only be made when the PHY is in the arbitration idle (A0) state (see 4.4.2.4).
 - 2) CYCLE_MASTER. The PHY layer shall start arbitration at the next subaction gap. For the cable environment, this arbitration class may only be used by the root node. This request can only be made when the PHY is in the arbitration idle (A0) state (see 4.4.2.4).
 - 3) ISOCHRONOUS. The PHY layer shall start arbitration as soon as the bus is idle for at least an isochronous gap period. (The link layer uses this method to send an isochronous packet). This request can only be made when the PHY is in the arbitration idle (A0) or receive (A5) states (see 4.4.2.4).
 - 4) IMMEDIATE. A PHY arbitration confirmation with an Arbitration Request Status of WON shall be communicated to the link layer as soon as the bus is idle. (The link layer uses this method to send an acknowledge packet.) This request can only be made when the PHY is in the arbitration receive (A5) state (see 4.4.2.4).

- b) **Speed Code.** This parameter shall indicate which data rate is to be used to generate all subsequent PHY clock indication events until the next PHY data request with a DATA_END parameter. The speed code shall be one of those listed in table 4.2.

Table 4.2—Cable physical layer speed codes

Speed code	Comment
S100	Base rate; all nodes shall be capable of running at this rate.
S200	All nodes capable of the S200 data rate shall also be capable of operating at the S100 rate.
S400	All nodes capable of the S400 data rate shall also be capable of operating at the S100 and S200 data rates.

The data rate corresponding to each speed code is defined in table 4.21.

4.1.2.2 PHY arbitration confirmation (PH_ARB.confirmation)

The PHY layer uses this service to confirm the results of a PHY arbitration request service. The PHY layer shall communicate this service to the link layer upon completion of a PHY arbitration request. There are no actions provided by this service. The following parameter is communicated via this service:

- **Arbitration Request Status.** This parameter shall contain the result of a PHY arbitration request action. The following values are defined for this parameter:
 - **WON.** The PHY arbitration request action was completed successfully. The PHY layer shall begin communicating PHY clock indications. This confirmation is returned after the minimum length data_prefix state described in 4.4.1.1 has been transmitted.
 - **LOST.** The PHY arbitration request action was not successful.

4.1.3 PHY layer data services for the link layer

These services are used to communicate data symbols and control information between the PHY layer and the link layer.

4.1.3.1 PHY clock indication (PH_CLOCK.indication)

The PHY layer uses this service to indicate to the link layer that a data symbol transmission is about to occur. There are no actions provided by this service. The link layer shall respond to this indication with a PHY data request. No parameters are communicated via this service.

The PHY layer shall begin communicating this indication to the link layer after it has communicated a PHY arbitration confirmation with an Arbitration Request Status of WON. The PHY layer shall stop communicating this indication to the link layer after the link layer communicates a PHY data request with data of DATA_END.

This indication occurs once for each bit cell time (see 4.2.3) as specified by the corresponding PHY arbitration request.

4.1.3.2 PHY data request (PH_DATA.request)

The link layer uses this service to control the transmission of clocked data symbols by the PHY layer. The link layer shall communicate one PHY data request for each PHY clock indication. The PHY layer shall service the request immediately upon receipt by the PHY layer.

The following parameter is communicated via this service:

- Data. This parameter shall contain the symbol to be transmitted on the bus. The following values are defined for this parameter:
 - DATA_ONE. A symbol representing a data bit of one shall be transmitted on the bus.
 - DATA_ZERO. A symbol representing a data bit of zero shall be transmitted on the bus.
 - DATA_PREFIX. The PHY layer shall stop sending clocked data bits and leave the bus in the data_prefix state using the algorithm described in 4.4.1.1. This shall be used between the acknowledge and response packet of concatenated subactions.
 - DATA_END. The link layer shall stop communicating PHY data requests to the PHY layer. The PHY layer shall stop communicating PHY clock indications to the link layer. The PHY layer shall set all bus outputs to the idle state using the algorithm described in 4.4.1.1.

Once the link layer starts sending a data bit (DATA_ONE or DATA_ZERO), it shall transmit the entire packet, and the total number of continuous data bits shall be even.

Once the link layer starts sending a DATA_PREFIX, it shall continue sending it for at least MIN_PACKET_SEPARATION before sending new data bits.

The link layer shall stop transmitting (using the procedure described in 4.4.1.1) within MAX_BUS_OCCUPANCY of receiving the PHY Arbitration WON confirmation (receiving a “grant” arbitration signal).

4.1.3.3 PHY data indication (PH_DATA.indication)

The PHY layer uses this service to indicate to the link layer changes in the state of the PHY layer. These changes can include received data and other bus events. The PHY layer shall communicate this indication to the link layer for each data bit received. If the node is not receiving data bits, the PHY layer shall use this indication to communicate events needed by the link layer. No response is defined for this indication. The following parameters are communicated via this service:

- a) Data. This parameter shall contain the information decoded by the PHY that is needed by the link layer. The following values are defined for this parameter:
 - 1) DATA_START. The start of a packet has been detected on the bus.
 - 2) DATA_ONE. A data bit of one has been received on the bus.
 - 3) DATA_ZERO. A data bit of zero has been received on the bus.
 - 4) DATA_PREFIX. The end of a packet has been detected on the bus, but the transmitting node is not releasing control. This indicates a concatenated subaction.
 - 5) DATA_END. The end of a packet has been detected on the bus, and the transmitting node is releasing control.
 - 6) ARBITRATION_RESET_GAP. An Arbitration Reset Gap event has been detected on the bus. This event is needed for link layers that implement the retry A/B protocol.
 - 7) SUBACTION_GAP. A Subaction Gap event has been detected on the bus.
- b) Speed Code. This parameter shall indicate which data rate (see table 4.2) will be used during all subsequent PHY data indication services. This parameter shall be ignored if the data is not DATA_START.

4.2 Cable physical connection specification

For the cable medium, the Serial Bus uses point-to-point physical connections between nodes with a pair of differential signals TPA/TPA* and TPB/TPB* where the use of each pair is different depending on whether the node is arbitrating, transmitting data, or receiving data. Each physical connection consists of a port on each node and the cable between them. The actual cable assemblies use six conductor cables with small and rugged connectors. The limitations on this system are set by the requirement of the arbitration protocol for a fixed round-trip time for signals. The default timing is set after at most two bus resets, and it is adequate for 32 cable hops, each of 4.5 m for a total of 144 m.¹² In addition.

¹²Annex A describes other possible configurations.

the topology has to be “acyclic” (no closed loops). Any closed loops will prevent the tree-ID process described in 4.4.2.2 from completing.

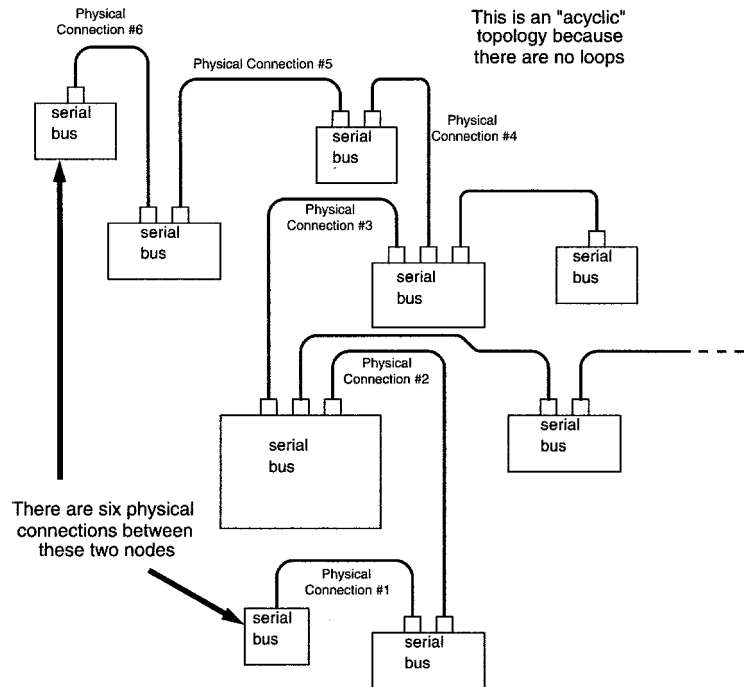


Figure 4.1—Cable topology

Serial Bus systems may be required to conform to (USA) FCC Part 15 rules for Class B operation. Because Serial Bus systems may be configured from equipment components and cables from more than one source, it is strongly recommended that the designs of individual cables and equipment follow the shielding and grounding practices required and/or suggested in this standard.

4.2.1 Media attachment

The Serial Bus permits a branching and daisy-chaining connection topology. Power can be provided or consumed by any node on the bus as described in 4.2.2.7. The topology puts certain constraints on the individual cable assembly lengths and wire gages. The remarks below apply to external (intercrate) cabling, where extra care has to be exercised for safety and EMC compliance. (Intracrate connections are not standardized in this clause.)

The connectors and cables have six contacts and six wires, plus shields. The plugs and sockets are mechanically identical at both ends of the cable, i.e., there is no polarization for directionality.

WARNING

The shield circuit of the Serial Bus external connection is an integral part of the electrical operation of the bus. The cable shield provides a primary electrical reference point for the conductors in the external cables and is the means by which the external cable shield is electrically connected to the enclosure wall. If this shield circuit does not perform as specified in this standard, three serious consequences may result:

- a) The external cable may provide a direct path between separate enclosures that could be a personal shock hazard or could result in serious physical damage to the cables themselves.
- b) The system may produce excessive electromagnetic radiation that may interfere with other electronic equipment.
- c) The data transmission through the external cable may suffer severe degradation in quality with resulting system performance problems.

In order to ensure that the shield circuit is properly designed, the transfer impedance testing procedure specified in annex L is recommended. This test should be applied to external shielded connectors and to complete enclosure connections that include the required shield circuit isolation circuit specified in 4.2.1.4.8.

4.2.1.1 Connectors

In typical applications, computer 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.

All dimensions, tolerances, and descriptions of features that affect the intermateability of the standardized shielded connector plugs and sockets are specified within this clause. Features of connector plugs and sockets that do not affect intermateability are not specified and may vary at the option of the manufacturer. Connector features that are not directly controlled within this clause shall be indirectly controlled by performance requirements in 4.2.1.3 and 4.2.1.4.

The holes and patterns (footprint) for the mounting of some of the possible versions of connectors to the PCB are recommended in annex I

4.2.1.1.1 Connector plug

The mating features of the connector plug are specified in figures 4.2 and 4.3. They will assure the intermateability of the plug with standardized sockets.

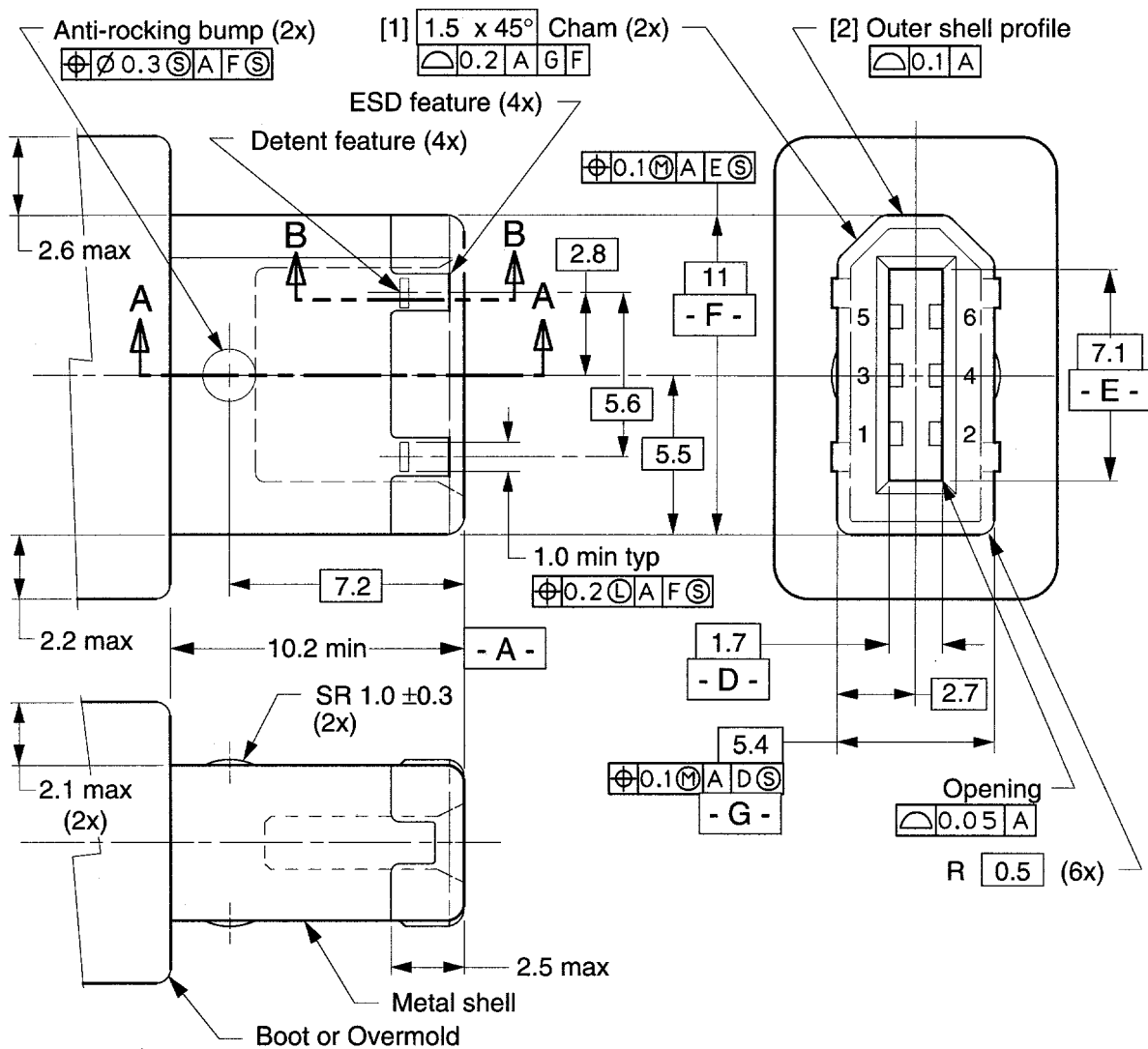
It is recommended that the plug contacts have a cylindrical section in the contact area that makes contact at a right angle to the cylindrical section of the socket contacts, thus creating a “crossed cylinders” configuration. The contacts should be designed to create a Hertzian stress (combination of cylindrical radius, normal force, and base and surface material hardnesses) of 1550 000–1 900 000 kPa in the mating area. This is to assure that the low-energy signals used in this physical layer are transmitted through the nonconductive films that are typically adsorbed on connector contacts.

NOTE — When a cable assembly plug is mated with a socket connector, there shall be a minimum of 1.0 mm clearance 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 4.2 and 4.3 describe a plug intended to be used when only detent retention with the socket is required.

4.2.1.1.2 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.



NOTES

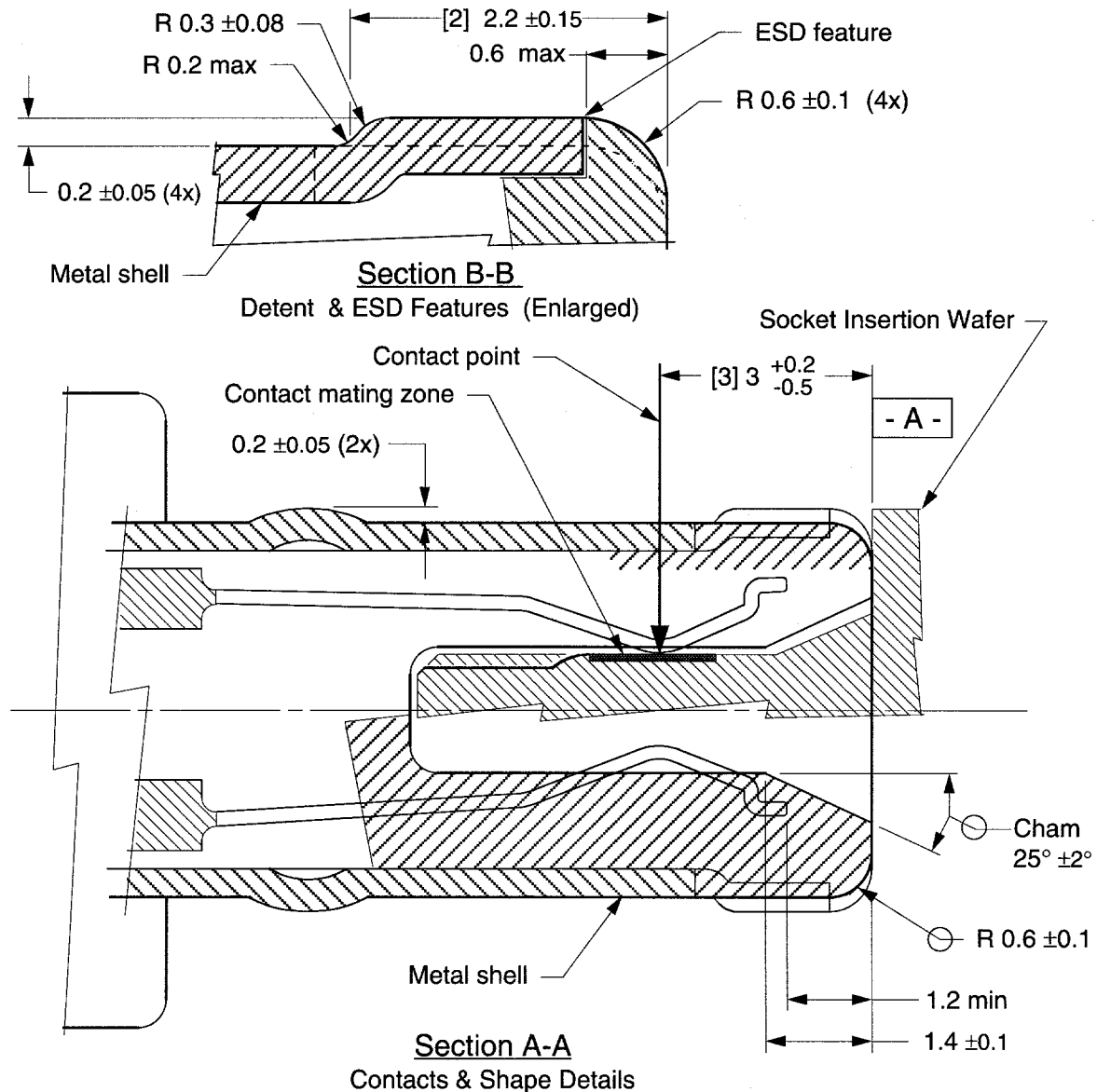
- 1—Theoretical sharp corner; chamfer profile tolerance excludes radii.
- 2—Chamfers and ESD features are excluded from profile tolerance.
- 3—Opening and contacts shall mate and function with the standardized socket insertion wafer.
- 4—Metal composition and thickness are at the option of the manufacturer.

Figure 4.2—Plug body

For reference, the following methods are listed: crimp, insulation displacement (IDC), insulation piercing, welding, and soldering.

4.2.1.1.3 Connector socket

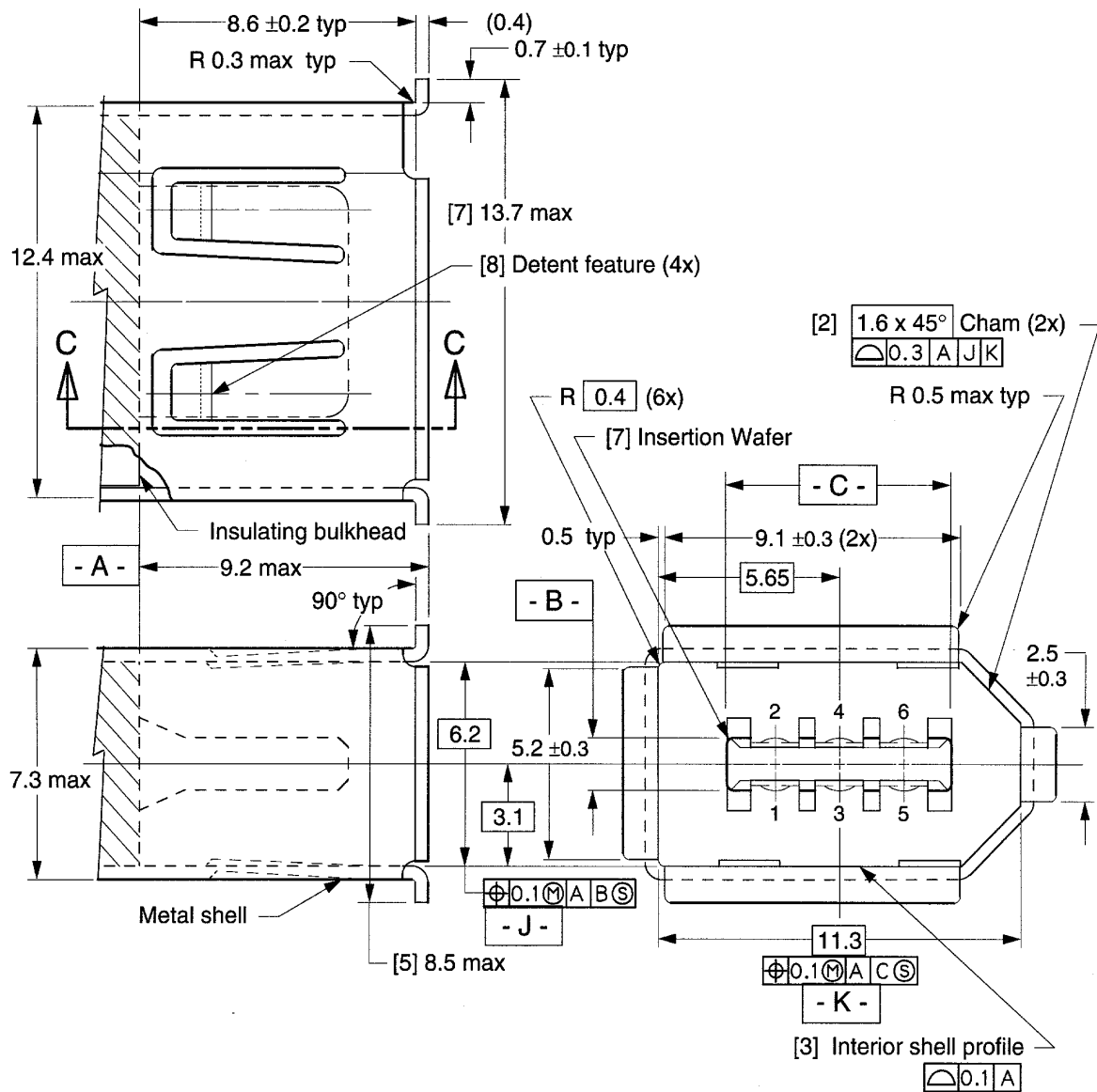
The mating features of the connector socket are described in figures 4.4 through 4.7. They will assure the intermateability of the socket with standardized plugs.



NOTES

- 1—Contact shape is at the option of the manufacturer.
- 2—Theoretical intersection of R 0.3 with main surface.
- 3—See figures 4-6 and 4-7.

Figure 4.3—Plug section details

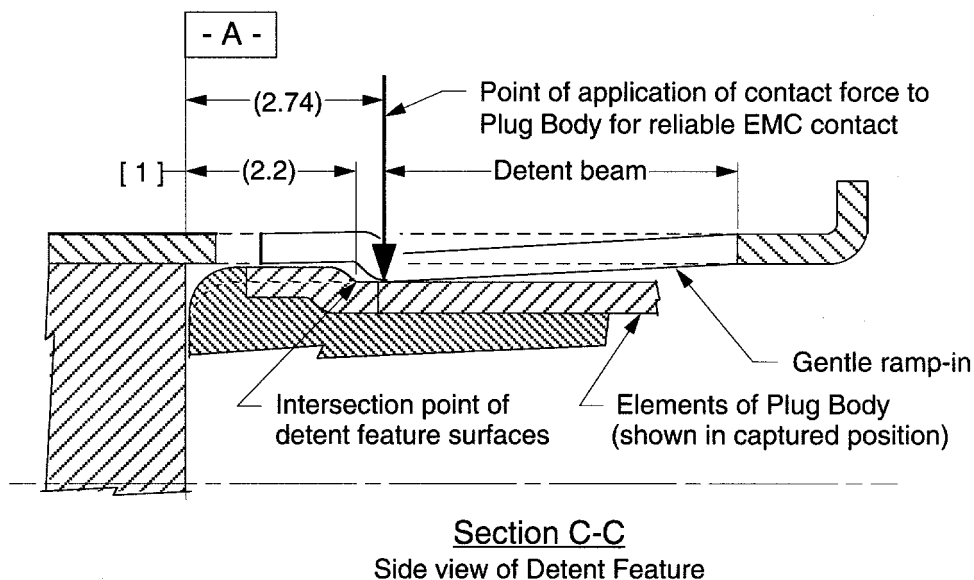


NOTES

- 1—Unless otherwise specified, tolerances are ± 0.15 , $\pm 2^\circ$.
- 2—Theoretical sharp corners; radii are excluded from chamfer tolerance zone.
- 3—Chamfers and ESD/Detent features are excluded from profile tolerance.
- 4—Detent features shall mate and function with the standardized plug shape.
- 5—Maximum size is less than the sum of the extreme tolerances.
- 6—Metal composition, thickness and detent details are at the option of the manufacturer.
- 7—See figures 4-6 and 4-7.
- 8—See figure 4-5.

Figure 4.4—Socket shell

Figures 4.4 and 4.5 describe the outer metal shell, which acts as a protective shield to prevent mechanical damage, to provide electrostatic protection, and to maintain shield continuity and integrity. They define features that provide “detent” retention of the mated plug.



NOTE—Socket detent features shall retain standard plug shape and contacts in reliable contact position under cable tension up to the minimum retention force.

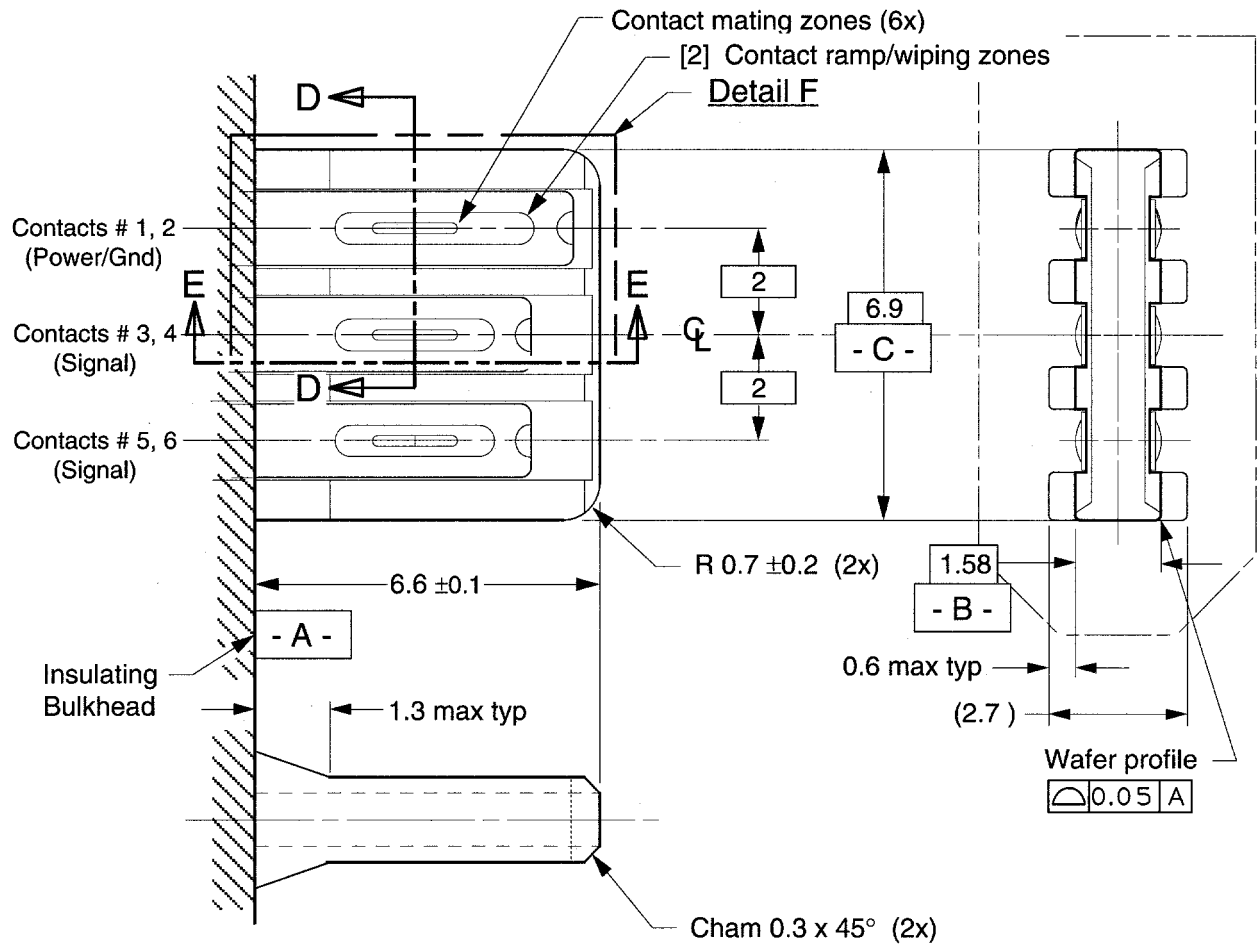
Figure 4.5—Socket shell detail

Figures 4.6 and 4.7 describe the “socket insertion wafer,” an insulating structure that positions and retains the contacts within the outer shell. They also define the “first-make, last-break” power contact (#1) and ground contact (#2). The leading edges of contacts #1 and #2 are intended to confine any erosion during “hot plugging” to a nonfunctional area of the contact surface. This will minimize contamination of the final resting area of the contact.

The contacts are attached to the signals using the guidance in table 4.3.

Table 4.3—Connector socket signal assignment

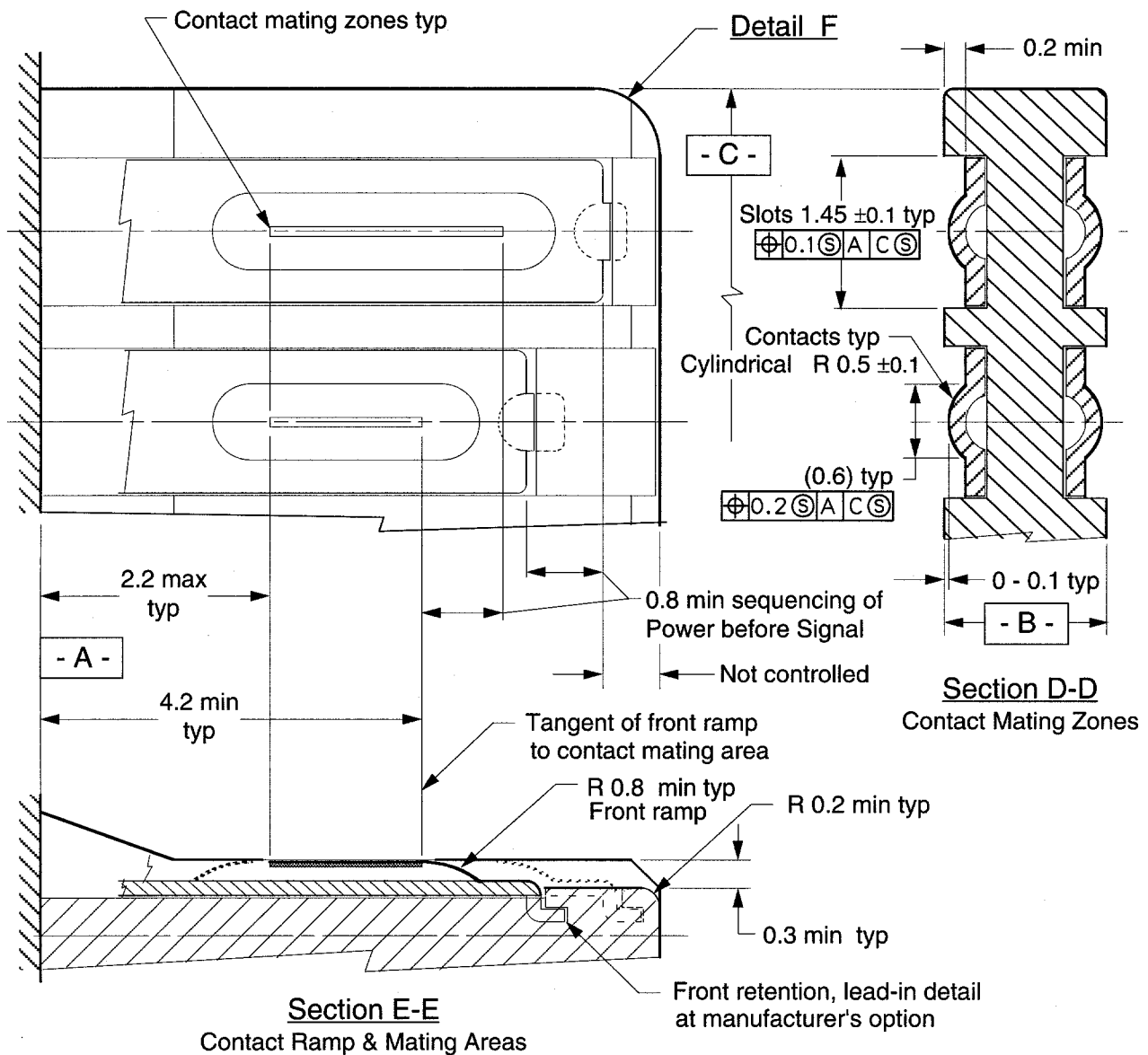
Contact number	Signal name	Comment
1	VP	Cable power
2	VG	Cable ground
3	TPB*	Strobe on receive, data on transmit (differential pair)
4	TPB	
5	TPA*	Data on receive, strobe on transmit (differential pair)
6	TPA	



NOTES

- 1—Note differences of Power/Ground (#1, 2) compared to Signal (#3, 4, 5, 6) contacts.
- 2—Refer to enlarged detail, figure 4-7.

Figure 4.6—Socket insertion wafer



NOTE—Refer to figure 4-6 for datum and section definitions.

Figure 4.7—Socket insertion wafer details

It is likely that there will be several variations of sockets to meet differing mounting orientation requirements, panel/bulk-head mounting, and/or assembly techniques. 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 annex I

4.2.1.1.4 Positive retention

The plug and socket are inherently retained together by means of a detent, which may be overcome by a prevailing force to effect a release upon unmating. Additional parts may be added to the plug and socket to provide a much stronger positive retention feature. This is specified in annex B

A socket that is equipped with the positive retention feature shall permit a plug without this feature to be inserted and retained by the detent [only], and to function normally otherwise.

A socket that is not equipped with the positive retention feature shall permit a plug that is equipped with the positive retention feature to be inserted and retained by the detent [only], and to function normally otherwise. It is also possible that the panel mounting configuration (which is not controlled within this standard) may prevent the insertion of a plug equipped with positive retention features if the equipment was not designed for this purpose. The minimum mounting interval for sockets with and without the positive retention feature is shown in annex I

4.2.1.1.5 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 should be used on contacts.

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

NOTES:

- 1 — Selective plating on contacts is acceptable. In that case, one of the above electroplatings shall cover the complete area of contact, including the contact wipe area.
- 2 — A copper strike is acceptable under the nickel electroplate.
- 3 — A palladium strike is acceptable over the nickel electroplate.

4.2.1.1.6 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 a minimum thickness of 1.27 μm (50 μin) nickel. A copper strike is acceptable under the nickel.

4.2.1.1.7 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, or a galvanically compatible alloy, over a suitable underplate.

4.2.1.1.8 Connector durability

The requirements of different end-use applications call for connectors that can be mated and unmated a different number of times without degrading performance beyond acceptable limits. Accordingly, this standard specifies minimum performance criteria of 1500 mating cycles.

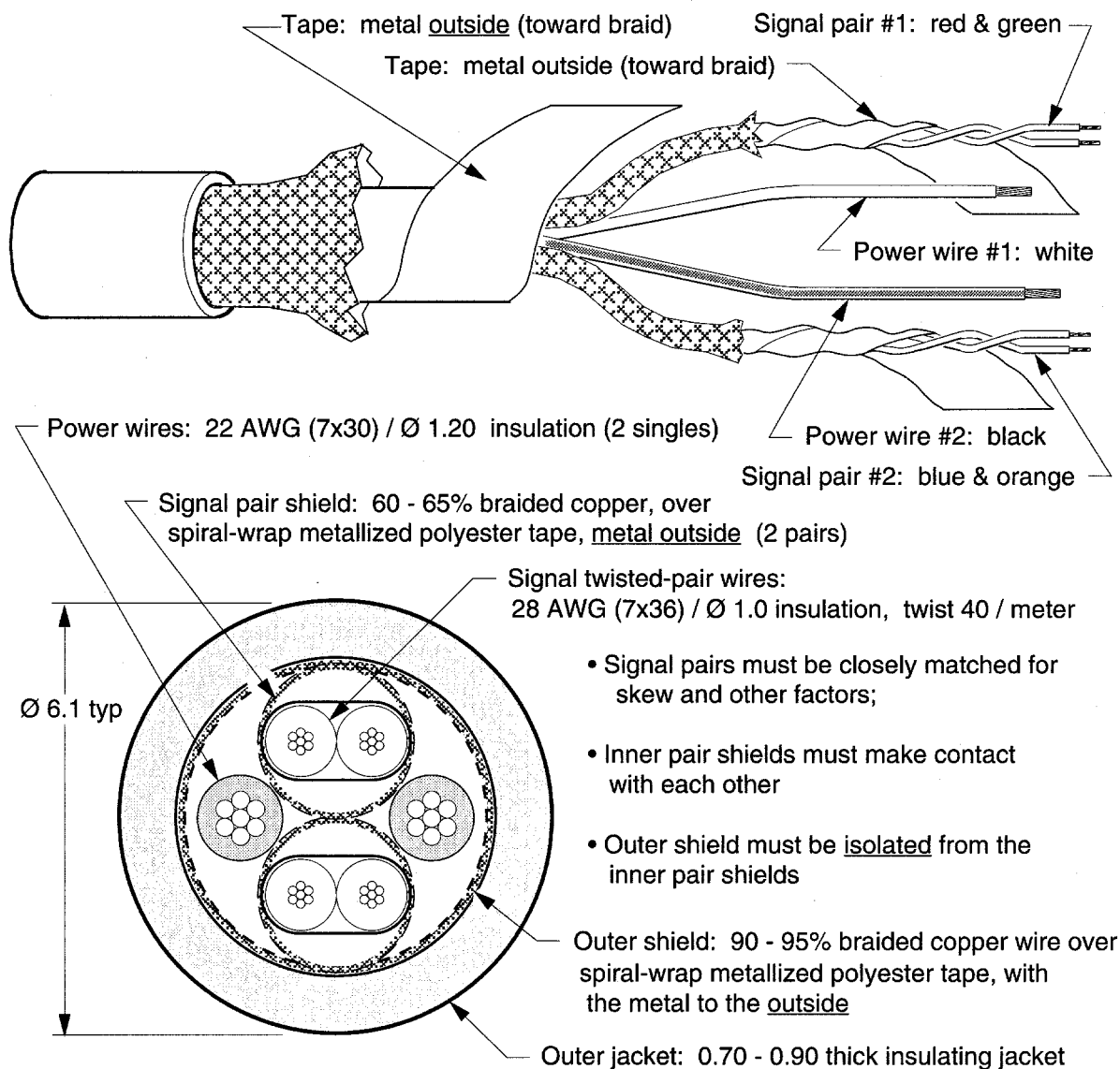
4.2.1.2 Cables

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

4.2.1.2.1 Cable material

Linear cable material typically consists of two twisted pair and two power conductors. The two twisted pairs carry the balanced differential data signals. The third pair of wires carries the bus power (nominally 8–40 Vdc). Figure 4.8

illustrates a reference design adequate for a 4.5 m cable. Insulation shall be sufficient for Safety Extra-Low Voltage (SELV) or 42 V rms. 4.2.1.4 describes the performance requirements for the cable assembly.



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

Figure 4.8—Cable material construction example (for reference only)

4.2.1.2.2 Cable assemblies

Cable assemblies consist of two identical plug connectors 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 consideration is given to the actual Serial Bus system topology to be used, as discussed in greater detail in annex A

The connector pins are terminated as shown in figure 4.9. The two signal pairs “cross” in the cable to effect a transmit-to-receive interconnection.

The individual shields on the signal pairs shall be connected to the VG pins of the plug. The outer shield shall have a secure low-impedance connection to the plug shield shell.

4.2.1.3 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 the 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 of ANSI/EIA 364-B-90 are: Temperature; + 15 °C to + 85 °C, Humidity; 95% maximum. Class 1.3 is further described as operating in a “harsh environmental” state, but with no marine atmosphere.

Accordingly, the performance groupings, the sequences within each group, and the test procedures shall follow the recommendations of ANSI/EIA 364-B-90, except where the unique requirements of the Serial Bus connector and cable assembly may call for tests that are not covered in ANSI/EIA 364-B-90, or where the requirements deviate substantially from those in that standard. In those cases, test procedures of other recognized authorities or specific procedures described in the annexes will be cited.

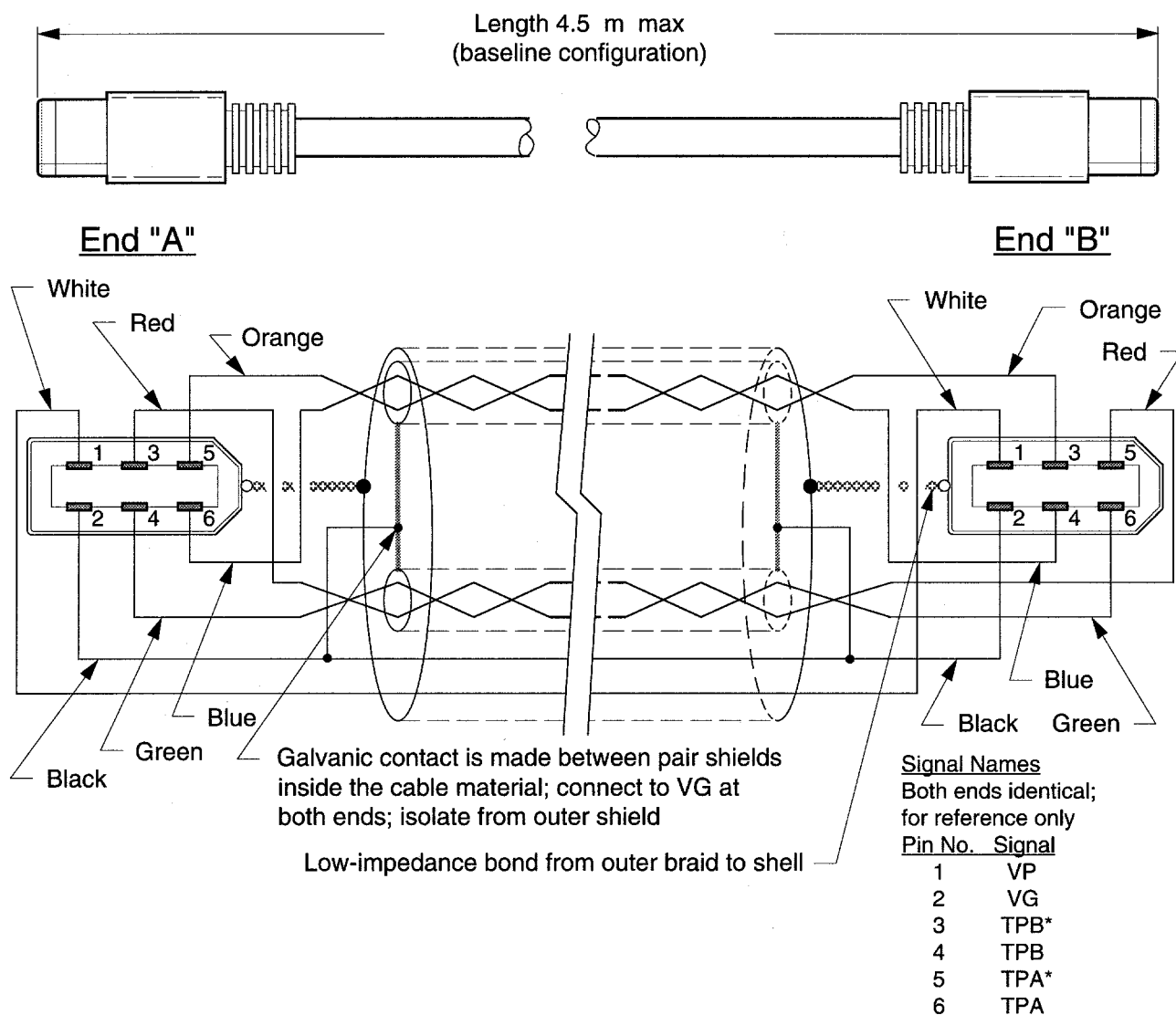
Sockets, plugs, and cable assemblies shall perform to the requirements and pass all the following tests in the groups and sequences shown.

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 that has successfully passed performance testing according to this standard.
- d) Plug only or socket only. In this case, the other half of the interface shall be procured from a source that has successfully passed performance testing according to this standard. For those performance groups that require it, the plugs may be assembled to the cable, to provide a cable assembly, by the connector manufacturer, or by a cable assembly supplier.

NOTES:

- 1 — All performance testing is to be done with cable material that conforms to this standard. In order to test to these performance groups, ANSI/EIA 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 PCB 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-B-90.
- 3 — The numbers 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.



NOTES

- 1—Connectors are viewed as looking at the front plug face.
- 2—Wire colors are indicated for reference only for understanding of schematic connections; note that pins 5 and 6 on one end map to pins 3 and 4 on the opposite end.

Figure 4.9—Example cable assembly and schematic

4.2.1.3.1 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 4.4—Performance group A

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
A1	Visual and dimensional inspection	ANSI/EIA 364-18A-84	Unmated connectors	Dimensional inspection	Per figures 4-2 through 4-7	No defect that would impair normal operations. No deviation from dimensional tolerances.
A2	Plating thickness measurement					Record thickness; see 4.2.1.1.5.
A3	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated pair.
A4	Vibration	ANSI/EIA 364-28A-83	Condition III (See note and figure 4.10).	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	30 mΩ maximum change from initial per mated contact.
A6	Mechanical shock (specified pulse)	ANSI/EIA 364-27A-83	Condition G (See note and figure 4.10).	Continuity	ANSI/EIA 364-46-84	No discontinuity at 1 μs or longer. (Each contact)
A7	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
<p>NOTE — Connectors are to be mounted on a fixture that simulates typical usage. The socket shall be mounted to a panel that is permanently affixed to the fixture. The mounting means shall include typical accessories such as</p> <ul style="list-style-type: none"> a) An insulating member to prevent grounding of the shell to the panel. b) A PCB in accordance with the pattern shown in Annex E for the socket being tested. The PCB shall also be permanently affixed to the fixture. <p>The plug shall be mated with the socket, and the other end of the cable shall be permanently clamped to the fixture.</p>						

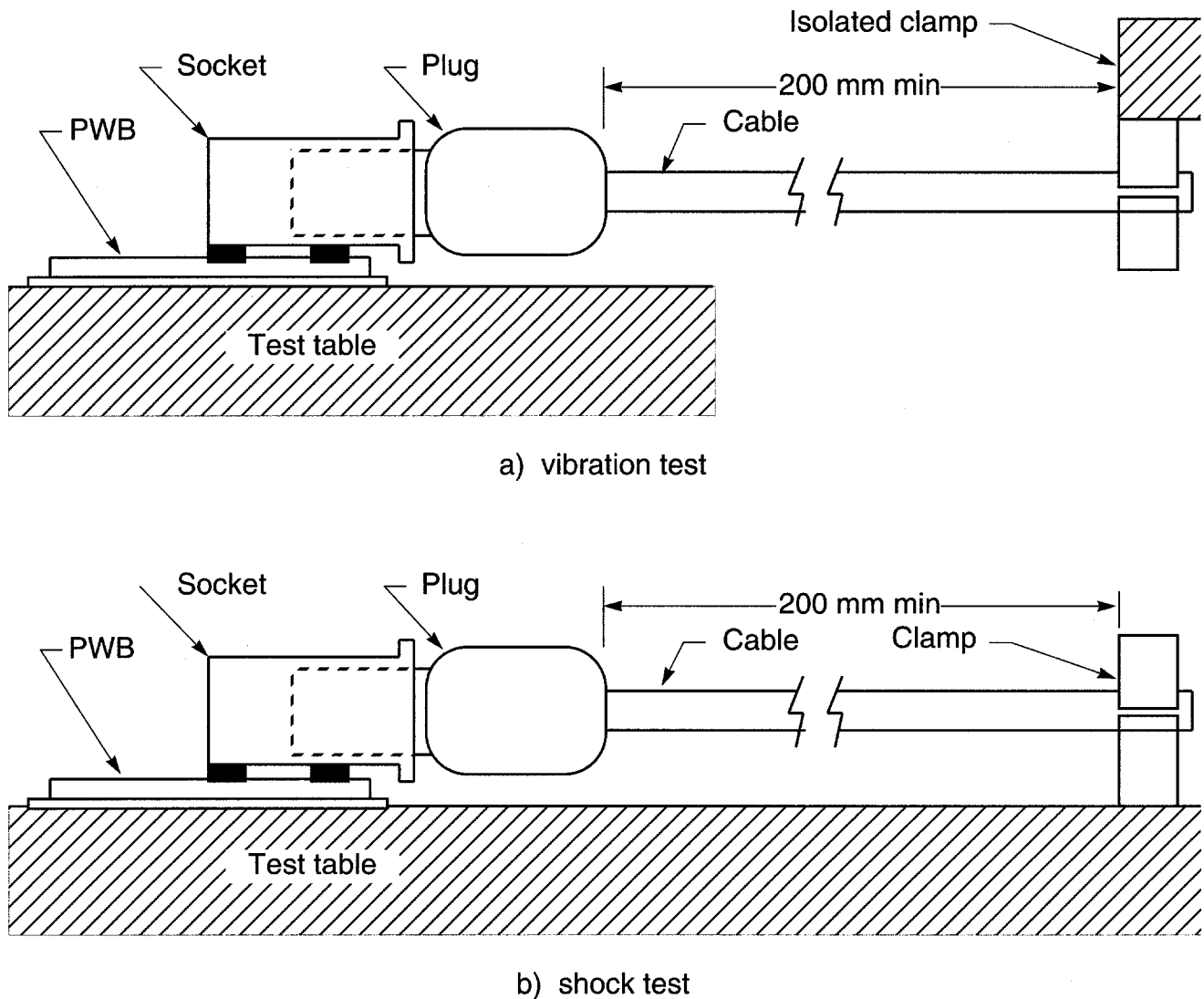


Figure 4.10—Shock and vibration fixturing diagram

4.2.1.3.2 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 4.5—Performance group B

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
B1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 m Ω , maximum, initial per mated contact.
B2	Thermal shock	ANSI/EIA 364-32B-92	Condition I 10 cycles (mated).	Low-level contact resistance	ANSI/EIA 364-23A-85	30 m Ω maximum change from initial per mated contact.
B3	Humidity	ANSI/EIA 364-31A-83	Condition C (504h). Method III (cycling) nonenergized. Omit steps 7a and 7b (mated).	Low-level contact resistance	ANS/EIA 364-23A-85	30 m Ω maximum change from initial per mated contact.

4.2.1.3.3 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, A 1, and A2 (one).
- [0] Cable assemblies with a plug assembled to one end, 2 m long.

Table 4.6—Performance group C

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
C1	Withstanding voltage	ANSI/EIA 364-20A-83	Test voltage 500 Vdc \pm 50 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	ANSI/EIA 364-32B-92	Condition I 10 cycles (unmated).	Withstanding voltage (same conditions as C1)	ANSI/EIA 364-20A-83	No flashover. No sparkover. No excess leakage. No breakdown.
C3	Insulation resistance	ANSI/EIA 364-21A-83	Test voltage 500 Vdc \pm 50 Vdc (unmated and unmounted).	Insulation resistance	ANSI/EIA 364-21A-83	100 M Ω 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	100 M Ω minimum.

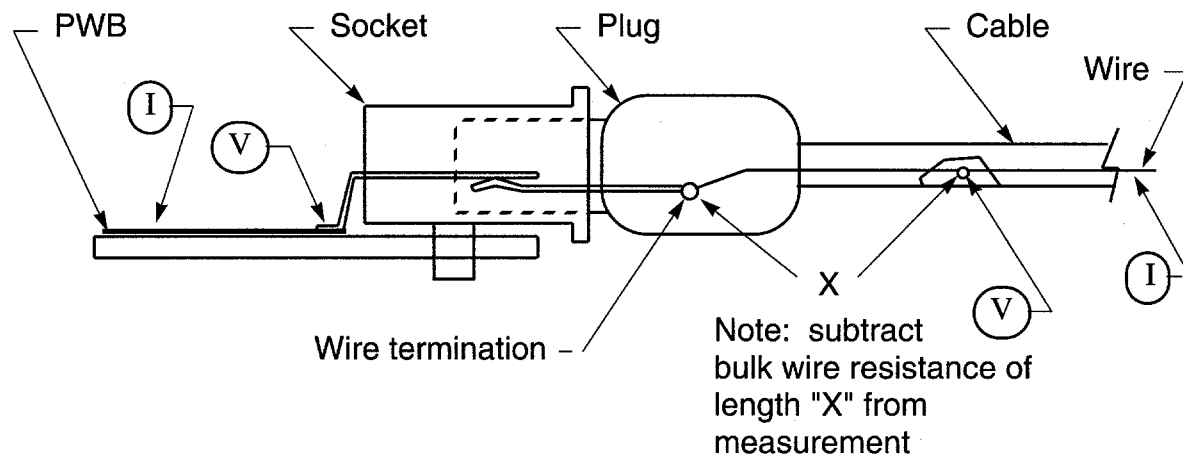
4.2.1.3.4 Performance group D: Contact life and durability when subjected to mechanical cycling and corrosive gas exposure

Number of samples:

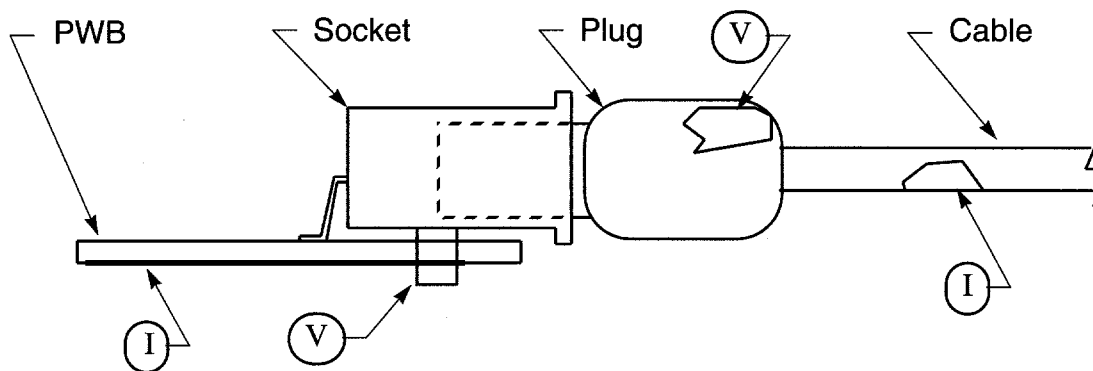
- [0] Sockets, unassembled to PCB.
- [4] Sockets, assembled to PCB.
- [0] Plugs, unassembled to cable used for Phase 1, A1, and A2 (one).
- [4] Cable assemblies with a plug assembled to one end, 25.4 cm long.

Table 4.7—Performance group D

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
D1	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated contact.
D2	Continuity-housing (shell)		See figure 4.11 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 maximum.
D3	Durability	ANSI/EIA 364-09B-91	(a) 2 mated pairs, 5 cycles. (b) 2 mated pairs, automatic cycling to 750 cycles, rate 500 cycles/h ± 50 cycles.			
D4	None			Low-level contact resistant	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D5	Continuity-housing (shell)		See figure 4.11 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 maximum.
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 for 10 days.	Low level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D7	Durability	ANSI/EIA 364-09B-91	Class II Exposures: (a) 2 mated pairs, 5 cycles. (b) 2 mated pairs, automatic cycling to 750 cycles, rate 500 cycles/h ± 50 cycles.			
D8	Mixed flowing gas	ANSI/EIA 364-65-92	Class II Exposures: Expose mated for 10 days.	Low-level contact resistance at end of exposure	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D9	Continuity-housing (shell)		See figure 4.11 for measurement points.	Contact resistance	ANSI/EIA 364-06A-83	50 mΩ, maximum, initial from braid to socket shell at 100 mA, 5 Vdc open circuit maximum.



a) contact resistance



b) shield resistance

Figure 4.11—Shield and contact resistance measuring points

4.2.1.3.5 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 4.8—Performance group E

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	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: 9.8 N minimum 39.2 N maximum
E2	None			Low-level contact resistance	ANSI/EIA 364-23A-85	30 m Ω , maximum, initial per mated contact.
E3	Continuity-housing (shell)		See figure 4-11.	Contact resistance	ANSI/EIA 364-06A-83	50 m Ω , maximum, initial from braid to socket shell at 100 mA, 5 Vdc open circuit maximum.
E4	Temperature life	ANSI/EIA 364-17A-87	Condition 4 (105° C) 250 h. Method A (mated).	Low-level contact resistance	ANSI/EIA 364-23A-85	30 m Ω maximum change from initial per mated contact.
E5	Continuity-housing (shell)			Contact resistance	ANSI/EIA 364-06A-83	50 m Ω maximum change from initial from braid to socket shell at 100 mA, 5 Vdc open circuit maximum.
E6	Mating and unmating forces	ANSI/EIA 364-13A-83	Mount socket rigidly. Insert plug by hand.	Mating only		
			Auto Rate: 25 mm/min	Unmating only	ANSI/EIA 364-13A-83	Unmating force: 9.8 N minimum 39.2 N maximum

4.2.1.3.6 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 4.9—Performance group F

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	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: 9.8 N minimum 39.2 N maximum
F3	Durability	ANSI/EIA 364-09B-91	Automatic cycling to 1500 cycles. 500 cycles/h \pm 50 cycles	Unmating only	ANSI/EIA 364-13A-83	Unmating force at end of durability cycles: 9.8 N minimum 39.2 N maximum

4.2.1.3.7 Performance group G: General tests

Suggested procedures to test miscellaneous but important aspects of the interconnect are given in table 4.10.

Since the tests listed below may be destructive, separate samples shall be used for each test. The number of samples to be used is listed under the test title.

Table 4.10—Performance group G

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
G1	Electrostatic discharge [1 plug] [1 socket]	IEC 801-2	1–8 kV in 1. kV steps. Use 8 mm ball probe. Test unmated.	Evidence of discharge		No evidence of discharge to any of the six contacts; discharge to shield is acceptable.
G2	Cable axial pull test [2 plugs]		Fix plug housing and apply a 98 N load for 1 min on cable axis.	Continuity, visual	ANSI/EIA 364-46-84	No discontinuity on contacts or shield greater than 1 μ s under load. No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.
G3	Cable flexing [2 plugs]	ANSI/EIA 364-41B-89	Condition I, dimension $X = 3.7 \times$ cable diameter; 100 cycles in each of two planes.	(a) Withstanding voltage	Per C1	Per C1
				(b) Insulation resistance	Per C3	Per C3
				(c) Continuity	ANSI/EIA 364-46-84	No discontinuity on contacts or shield greater than 1 μ s during flexing.
				(d) Visual	-	No jacket tears or visual exposure of shield. No jacket movement greater than 1.5 mm at point of exit.

4.2.1.4 Signal propagation performance

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

4.2.1.4.1 Signal impedance

The differential mode characteristic impedance of the signal pairs shall be measured by time domain reflectometry at less than 100 ps rise time using the procedure described in K.3:

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

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

The common mode characteristic impedance of the signal pairs shall be measured by time domain reflectometry at less than 100 ps rise time using the procedure described in K.3:

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

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

4.2.1.4.2 Signal pairs attenuation

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

Table 4.11—Cable attenuation

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

4.2.1.4.3 Signal pairs velocity of propagation

The differential velocity of propagation (V) of the signal pairs shall be measured in the frequency domain using the procedure described in K.5:

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

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

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 K.5 since this will be the case for all but the most exotic cable constructions:

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

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

4.2.1.4.4 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 K.5.2:

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

4.2.1.4.5 Power pair characteristic impedance

The differential mode characteristic impedance of the power pair shall be measured by TDR at less than 100 ps rise time using the procedure described in K.7:

$$Z_{VG} \leq 65 \text{ (differential)}$$

4.2.1.4.6 Power pair dc resistance

The dc resistance of the power wires is measured with a milliohmeter capable of “four-wire” resistance measurements using the procedure described in K.6.3:

$$R_V \leq 0.333 \text{ } \Omega$$

$$R_G \leq 0.333 \text{ } \Omega$$

4.2.1.4.7 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 K.8:

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

4.2.1.4.8 Shield ac coupling

Current in the cable shield has to be minimized. To do this, all nodes that use cable power or have more than one port shall have a shield isolation circuit equivalent to a resistor and a capacitor in parallel between the equipment shield and the socket/cable shield, such that

- Resistor = $(1 \pm 0.1) \text{ M}\Omega$
- Capacitor with an impedance Z, where
 - $Z \geq 25 \text{ k}\Omega$ at 60 Hz
 - $Z \leq 15 \text{ }\Omega$ from 2 MHz to 500 MHz

4.2.2 Media signal interface

The cable media signal interface is called a port. It consists of two twisted pair interfaces (TPA/TPA* and TPB/TPB* and a power distribution pair (VP/VG). A node may have several such ports. Each port has associated circuitry that provides separate signals for packet data reception/transmission and for arbitration, as shown in figure 4.12.

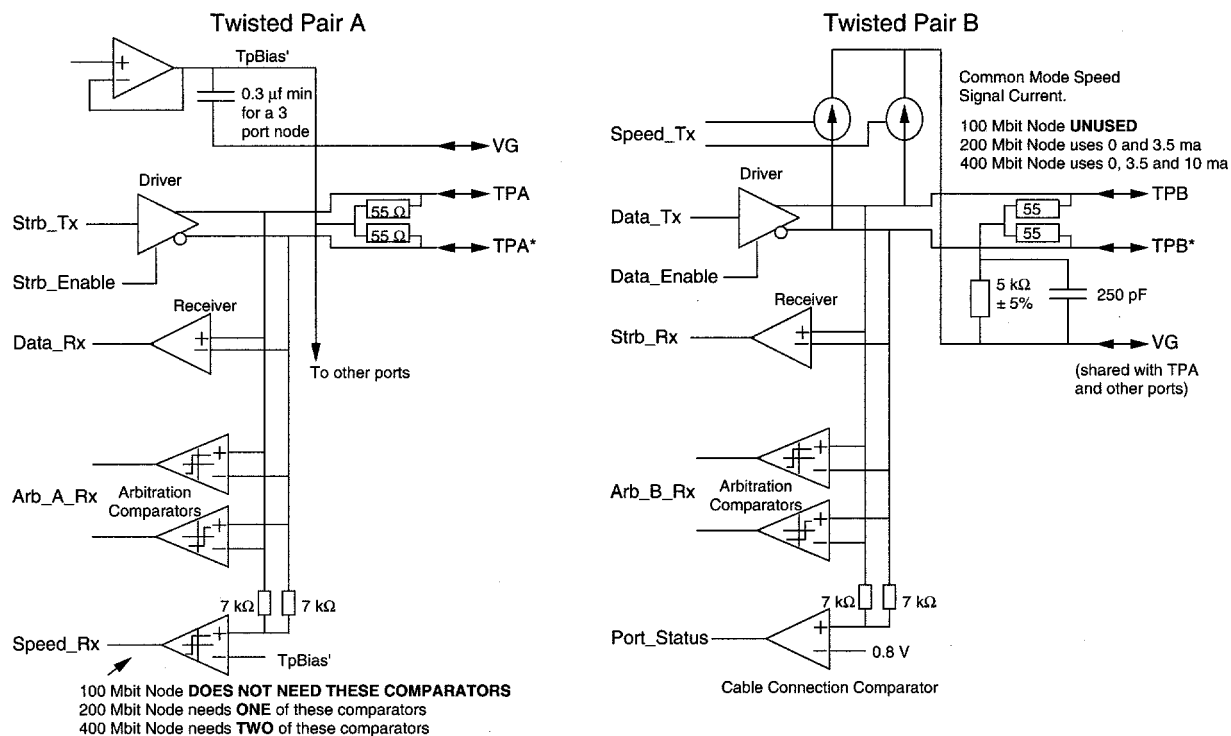


Figure 4.12—Cable media signal interface configuration

The TPA/TPA* pair transmit the Strb_Tx signal and receive the Data_Rx, Arb_A_Rx, and Speed_Rx signals, while the TPB/TPB* pair transmits the Data_Tx and Speed_Tx signals and receives the Strb_Rx, Arb_B_Rx, and

Port_Status signals. The Strb_Tx, Data_Tx, Strb_Enable, and Data_Enable signals are used together to generate the arbitration signals. The Arb_A_Rx and Arb_B_Rx signals are each generated by two comparators since they have three states: 1, 0, and Z.

In addition, the TPA/TPA* pair transmits TpBias while the TPB/TPB* pair receives the TpBias signal, which is used by the Port_Status comparator to determine that a cable connection exists.

4.2.2.1 Signal amplitude

For the test loads shown in figure 4.13, the drivers for TPA and TPB shall provide the differential output signal amplitude given in table 4.12 (an additional 10% is allowed for signal overshoot).

Table 4.12—Differential output signal amplitude

Maximum	Minimum	Units
265	172	mV

The TPA differential output amplitude is measured between the TPA and TPA* pins; the TPB differential output amplitude is measured between the TPB and TPB* pins. The magnitude of the TPA differential output voltage shall meet the limits of table 4.12 for all values of the common mode current (I_{cm}) as specified in table 4.17. The magnitude of the TPB differential output voltage shall meet the limits of table 4.12 for all values of the common mode voltage (V_{cm}) as specified in table 4.15.

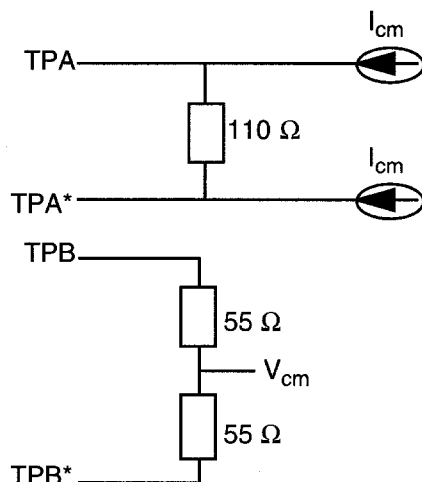


Figure 4.13—Differential output test loads

At the receiving end of the cable, the signal amplitude detection limits shall be as given in table 4.13.

Table 4.13—Differential receive signal amplitude

Signal	S100 (mV)		S200 (mV)		S400 (mV)	
	Max	Min	Max	Min	Max	Min
During arbitration (Arb_A, Arb_B valid)	260	173	262	171	265	168
During clocked data reception (Data_Rx, Strb_Rx valid)	260	142	260	132	260	118

At the receiving end of the cable, the receivers and arbitration comparators shall operate with signals having amplitudes as specified in table 4.13. The TPA receivers and arbitration comparators shall meet the input requirements for common mode voltages as specified in table 4.14. The TPB receivers and arbitration comparators shall meet the input requirements for common mode voltages as specified in table 4.15.

4.2.2.2 Common mode voltage

TPA is the source of the common mode bias voltage (relative to the power ground pin, VG), and it shall meet the requirements of table 4.14 when common mode signaling is on (speed signaling active) and off (no speed signaling).

Table 4.14—TPA common mode output voltage

Condition		Limit (V)
Maximum		2.015
Minimum	Speed signaling off	1.665
	S100 speed signaling	1.665
	S200 speed signaling	1.438
	S400 speed signaling	1.030

The test load is shown in figure 4.13.

The TPA common mode output voltage is measured as the average of the voltages on the TPA and TPA* pins. The value of this voltage shall meet the limits of table 4.14 for all values of I_{cm} as specified in table 4.17.

The common mode input voltage received at the TPB pair shall remain within the limits shown in table 4.15 relative to the power ground pin VG.

Table 4.15—TPB common mode input voltage

Condition		Limit (V)
Maximum		2.515*
Minimum	Speed signaling off	1.165
	S100 speed signaling	1.165
	S200 speed signaling	0.935
	S400 speed signaling	0.523

*For an end-of-wire power-consuming device, the maximum will be 2.015 V since there is no need to budget for the 0.5 V power supply drop possible, given the power pair resistance specified in 4.2.1.4.6 and the maximum power pair current specified in 4.2.2.7.

The common mode input voltage is also used to determine the connection status of the port as shown in table 4.16.

Table 4.16—Port_Status signal condition

Port_Status	Condition
DISCONNECTED	TPB common mode input ≤ 0.6 V
Undefined	0.6 V < TPB common mode input < 1.0 V
CONNECTED	TPB common mode input ≥ 1.0 V

This signal shall *not* be sampled when common mode signaling is active (speed signaling), so there is no danger of accidentally assuming that the port is disconnected.

4.2.2.3 Speed signaling

When common mode signaling is on (speed signaling), the TPB common mode signal driver shall source the appropriate current to indicate the maximum data rate that can be received by this port. Similarly, the TPA common mode signal receiver shall use the levels in table 4.17 to determine the maximum data rate that can be received by the attached port.

NOTE — Since the link layer has a minimum packet size and the PHY imposes packet starting and ending times, the maximum pulse repetition frequency for speed signaling is 1.89 MHz.

The TPB common mode speed signaling output current is measured as one half of the algebraic total current flowing out the TPB and TPB* pins. The value of this current shall meet the limits of table 4.17 for all values of V_{cm} as specified in table 4.15.

Table 4.17—TPB common mode output current and TPA common mode input current

Signal	Data rate (mA)					
	Speed_Tx = S100		Speed_Tx = S200		Speed_Tx = S400	
	Max	Min	Max	Min	Max	Min
Common mode signaling off (Speed_Tx = S100)	0.44	-0.81	0.44	-0.81	0.44	-0.81
Common mode signaling on	0.44	-0.81	-2.53	-4.84	-8.10	-12.40

This current is measured for the test loads shown in figure 4.14.

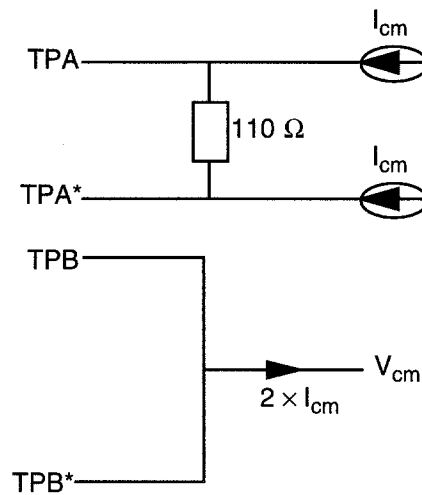


Figure 4.14—Common mode current test loads

4.2.2.4 Arbitration signal voltages

During cable arbitration (including tree ID and self-ID as well as normal operation), the arbitration receive signals, Arb_A_Rx and Arb_B_Rx, will take three different values when TPA/TPB meet the requirements of table 4.18.

Table 4.18—Arbitration signaling levels

Arb_n_Rx*	Max (mV)	Min (mV)
1		168
Z	89	-89
0	-168	

*"n" is "A" or "B."

NOTE — The minimum signal level for the 1 Arb_n_Rx value and maximum signal level for the 0 value correspond to the minimum differential receive amplitudes for all speeds in table 4.13.

4.2.2.5 Input impedance

Both TPA and TPB shall meet the input impedance requirements of table 4.19.

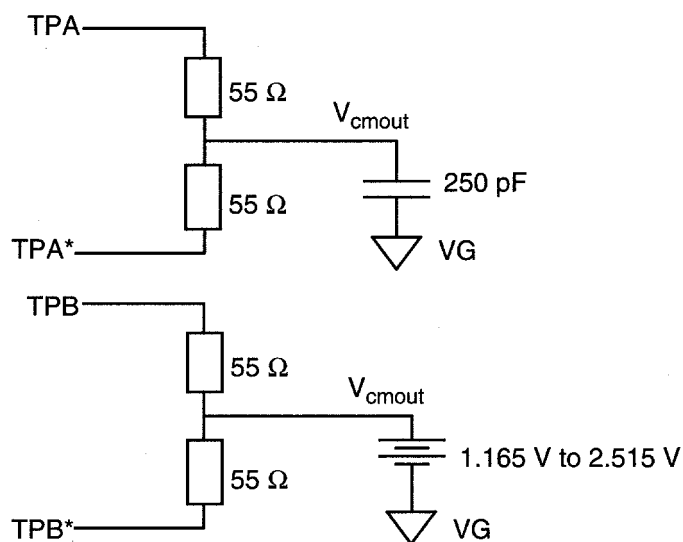
Table 4.19—Differential input impedance

Parameter	Test Condition	Max	Min	Units
R _{in}	Receive mode	111	109	Ω
R _{intx}	Transmit mode	111	105	Ω
C _{in}	S100-only port	9		pF
	S200-capable port	7		pF
	S400-capable port	4		pF

If a node cannot be characterized as a point impedance, then the alternate requirement is that the amplitude of a reflection using a differential TDR with a 0.5 V and 1 ns rise time shall not exceed 240 mV for S100-only ports, 187 mV for a S200-capable port, and 107 mV for a S400-capable port.

4.2.2.6 Noise

The maximum output common mode noise, V_{cmout} , shall be 200 mV peak-to-peak (p-p) during arbitration and 30 mV during data transmission, as measured in a 400 MHz bandwidth using the test loads shown in figure 4.15.

**Figure 4.15—Common mode output noise test loads**

The maximum input common mode noise shall be 225 mV p-p during arbitration and 55 mV p-p during packet reception. This shall be measured in a bandwidth of 20 MHz to 400 MHz.

The maximum input differential noise shall be 8mV p-p.

4.2.2.7 Power and ground

A node may be a power source, a power sink, or neither. Since there may be more than one power source, the power sources shall be diode-connected to the power wire such that current from a node providing a higher voltage does not flow into sources of lower output voltage. This is illustrated in figure 4.16.

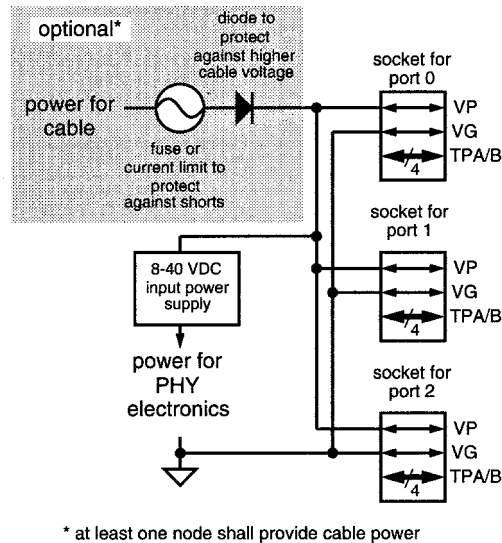


Figure 4.16—Power distribution interface example

When implemented, a cable power source shall meet the requirements of table 4.20

Table 4.20—Cable power source requirements

Condition	Limit	Units
Maximum output current per port	1.5	Amp
Minimum output voltage	8	Vdc
Maximum output voltage	40	Vdc
Maximum output ripple (10 kHz to 400 MHz)	100	mV p-p
Maximum output ripple (below 10 kHz)	1	V p-p

In addition, the source shall provide overvoltage and short-circuit protection.

Input voltage range is from 7.5 V to 40 V. If the input voltage is within this range, the node variable `cable_power_active` shall be true. If a node detects a voltage lower than 8 V, it shall generate the `PH_EVENT`. indicate of `CABLE_POWER_FALL`. If a node uses cable power, it shall meet the following requirements:

- a) It cannot use more than 1 W of power after a power reset or after being initially connected from the bus (transition from all ports unconnected to any port connected). Any additional power usage shall depend on receiving a link-on PHY packet.
- b) The inrush energy shall not exceed 18 mJ in 3 ms.
- c) Maximum peak-to-peak ripple and slew rate of the power supply current consumption of a node as a function of the maximum current requirement of the node.
 - 1) I_{load} = Maximum current requirement of the node in A(s)
 - 2) Maximum peak-to-peak ripple = 100 mA * ($I_{load}/1.5$ A)
 - 3) Maximum change in load current shall be less than I_{load} in 100 μ s.

The sum of the dc currents on VG and VP shall be less than 50 μ A. Note that this requires special consideration in system grounding. See annex A

All nodes with two or more cable ports shall repeat bus signals, even if the local power for that node is off. Cable power shall be used for the operation of the physical layer functions in such cases. Single port nodes are not required to repeat signals.

4.2.2.8 Driver and receiver fault protection

The drivers and receivers shall tolerate -0.5 V to $+2.83$ V without damage.

NOTE — All transceivers will have the additional protection of the termination network.

4.2.3 Media signal timing

4.2.3.1 Data rate

The cable media supports three different data rates. The lowest rate (98.304 Mbit/s) is required and is called the “base rate.” All ports that support a higher data rate shall also support all lower rates.

Table 4.21—Cable media data rates

	Data rate			Units
	S100	S200	S400	
Data rate	98.304 ppm \pm 100 ppm	196.608 ppm \pm 100 ppm	393.216 ppm \pm 100 ppm	Mbit/s
Nominal bit cell time*	10.17	5.09	2.54	ns

*Bit cell times are shown for reference. The standard is based on the data rate.

4.2.3.2 Data signal rise and fall times

The output rise and fall times for data signals are measured from 10% to 90% and are dependent on the data rate as specified in table 4.22.

Table 4.22—Output rise and fall times

	Maximum rise or fall time (ns)
S100	3.2
S200	2.2
S400	1.2

The input slope is measured at a differential offset of $+25$ mV for rising signals and -25 mV for falling signals and shall have a minimum value as shown in table 4.23.

Table 4.23—Input slope

	Minimum slope for data signals (mV/ns)
S100	60
S200	110
S400	175

4.2.3.3 Jitter and skew

The following tables give the maximum jitter and skew limits for the three cable data rates. The complete jitter budget is given in E.2.

Table 4.24—Cable environment jitter and skew (in ns)

	S100	S200	S400
Maximum transmitter skew	0.40	0.15	0.10
Maximum transmitter jitter	0.80	0.25	0.15
Maximum skew at receive pins	0.80	0.55	0.50
Maximum jitter at receive pins	1.08	0.50	0.315

4.3 Cable PHY facilities

4.3.1 Coding

Attached peer physical layer entities on the bus communicate via NRZ data. The NRZ data is transmitted and received as Data_Tx and Data_Rx and is accompanied by a strobe signal, Strb_Tx and Strb_Rx. This strobe signal changes state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on either Data or Strb. A clock that transitions every bit period can be extracted by performing an exclusive OR of Data_Rx and Strb_Rx, as shown in figure 4.17.

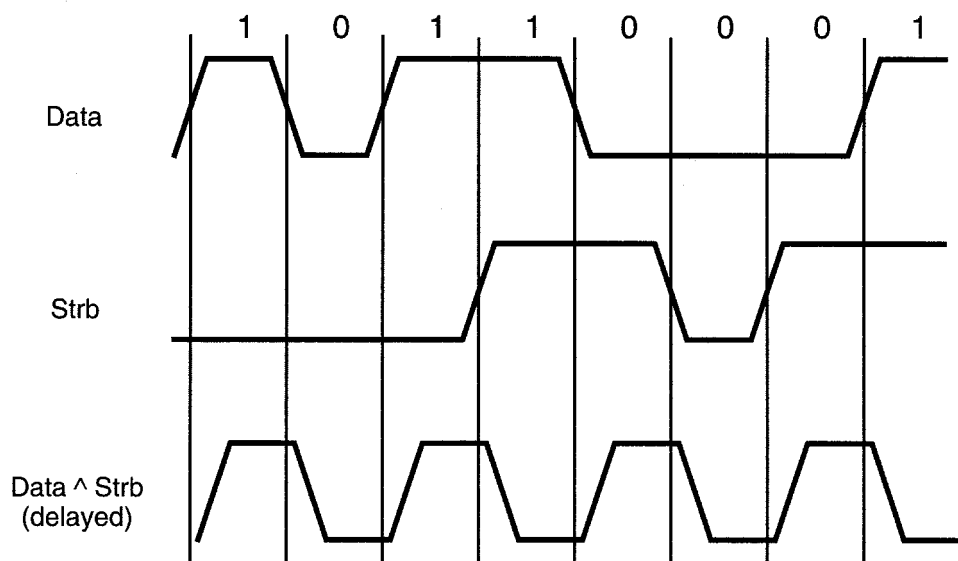


Figure 4.17—Data-strobe coding

The primary rationale for use of this transmission code is to improve the transmission characteristics of information to be transferred across the serial bus. In particular, the code ensures that transitions occurring on Strb and Data are approximately one bit-period apart. This results in an increase in skew tolerance that could not be obtained with a clocked NRZ format.

The procedure for encoding data during transmission is described in 4.4.1.1 and decoding during reception in 4.4.1.2.

4.3.2 Cable PHY signals

The cable PHYs use bidirectional dominant mode signaling during arbitration. The logical values are “1”, “0”, and “Z” and are generated by setting the drivers in figure 4.12 according to the rules in table 4.25 and decoded using the rules in table 4.26.

Table 4.25—Arbitration signal generation rules

Transmit arbitration signal A (Arb_A_Tx)	Drivers		Comment
	Strb_Tx	Strb_Enable	
Z	-	0	TPA driver is disabled
0	0	1	TPA driver is enabled, strobe is low
1	1	1	TPA driver is enabled, strobe is high

Transmit arbitration signal B (Arb_B_Tx)	Drivers		Comment
	Data_Tx	Data_Enable	
Z	-	0	TPB driver is disabled
0	0	1	TPB driver is enabled, data is low
1	1	1	TPB driver is enabled, data is high

Table 4.26—Arbitration signal decoding rules

Received arbitration comparator value (Arb_n*_Rx)	Transmitted arbitration signal for this port (Arb_n*_Tx)	Interpreted arbitration signal (Arb_n*)	Comment
Z	Z	Z	If this port is transmitting a Z, then the received signal will be the same as transmitted by the port on the other end of the cable.
0	Z	0	
1	Z	1	
Z	0	1	If the comparator is receiving a Z while this port is sending a 0, then the other port must be sending a 1. This is the first half of the dominance rule for 1.
0	0	0	The other port is sending a 0 or a Z.
Z	1	1	The other port must be sending a 0. This is the other half of the dominance rule for 1.
1	1	1	The other port is sending a 1 or a Z.

*"n" is "A" or "B." This table applies to both signal pairs.

4.3.3 Cable PHY line states

The cable PHY encodes arbitration line states on the two transmit arbitration signals, Arb_A_Tx and Arb_B_Tx, using the rules in table 4.27. Note that a particular encoding may have different meanings depending on whether it is sent to a parent or a child.

Table 4.27—Cable PHY transmitted arbitration line states

Arbitration transmit		Line state name	Comment
Arb_A_Tx	Arb_B_Tx		
Z	Z	IDLE	Sent to indicate a gap
Z	0	TX_REQUEST	Sent to parent to request the bus
		TX_GRANT	Sent to child when bus is granted
0	Z	TX_PARENT_NOTIFY	Sent to parent candidate during tree-ID
0	1	TX_DATA_PREFIX	Sent before any packet data and between blocks of packet data in the case of concatenated subactions
1	Z	TX_CHILD_NOTIFY	Sent to child to acknowledge the parent_notify
		TX_IDENT_DONE	Sent to parent to indicate that self-ID is complete
1	0	TX_DATA_END	Sent at the end of packet transmission
1	1	BUS_RESET	Sent to force a bus reconfiguration

The cable PHYs decode the interpreted arbitration signals (Arb_A and Arb_B) into line states using the rules in table 4.28.

Table 4.28—Cable PHY received arbitration line states

Interpreted arbitration signals		Line state name	Comment
Arb_A	Arb_B		
Z	Z	IDLE	The attached peer PHY is inactive
Z	0	RX_PARENT_NOTIFY	The attached peer PHY wants to be a child
		RX_REQUEST_CANCEL	The attached peer PHY has abandoned a request (this PHY is sending a grant)
Z	1	RX_IDENT_DONE	The child PHY has completed its self-ID
0	Z	RX_SELF_ID_GRANT	The parent PHY is granting the bus for a self-ID
		RX_REQUEST	A child PHY is requesting the bus
0	0	RX_ROOT_CONTENTION	The attached peer PHY and this PHY both want to be a child
		RX_GRANT	The parent PHY is granting control of the bus
0	1	RX_PARENT_HANDSHAKE	The attached peer PHY acknowledges parent_notify
		RX_DATA_END	The attached peer PHY has finished sending a block of data and is about to release the bus
1	Z	RX_CHILD_HANDSHAKE	The attached peer PHY acknowledges TX_CHILD_NOTIFY (the peer PHY is a child of this PHY)
1	0	RX_DATA_PREFIX	The attached peer PHY is about to send packet data or has finished sending a block of packet data and is about to send more
1	1	BUS_RESET	Sent to force a bus reconfiguration
NOTE — Some of the possible received values are unused by the arbitration protocols, and some of the above codes are “don't care” values for some of the states in the state machines of 4.4.2.			

4.3.4 Cable PHY packets

The cable PHY sends and receives a small number of short packets that 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, and 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

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.

4.3.4.1 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 format shown in figure 4.18.

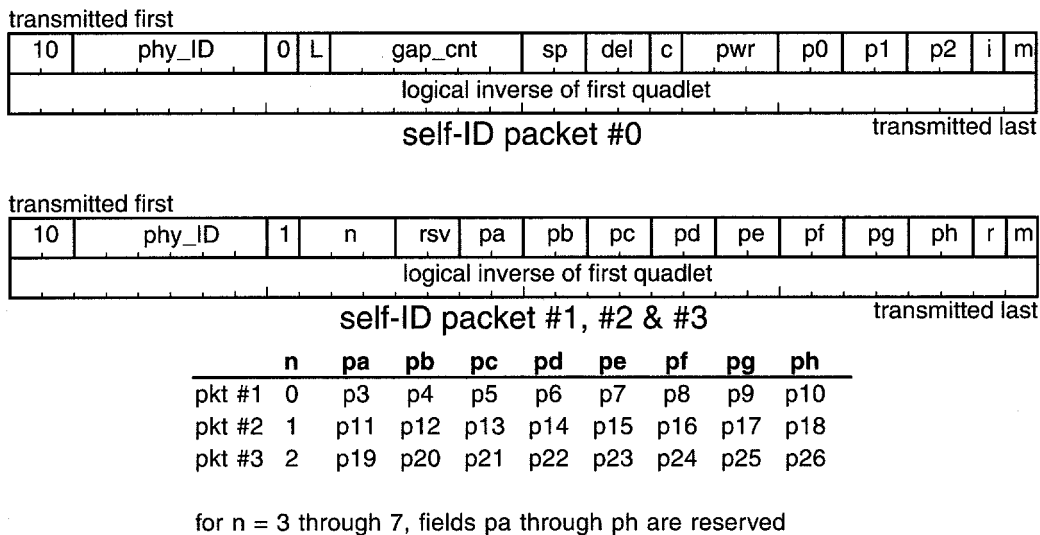


Figure 4.18—Self-ID packet format

Table 4.29—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	link_active	If set, this node has an active Link and Transaction Layer.
gap_cnt	gap_count	Current value for the PHY_CONFIGURATION.gap_count field of this node.
sp	PHY_SPEED	Speed capabilities: 00 98.304 Mbit/s 01 98.304 Mbit/s and 196.608 Mbit/s 10 98.304 Mbit/s, 196.608 Mbit/s, and 393.216 Mbit/s 11 Reserved for future definition
del	PHY_DELAY	Worst-case repeater data delay: 00 ≤ 144 ns (~14/BASE_RATE) 01 Reserved 10 Reserved 11 Reserved
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 8.4.1.
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.* 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. 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.
p0 ... p26	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). [†] (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 (0 through 2, corresponding to self-ID packets #1 through #3). If n has the value of 3 through 7, then the rsv, pa-ph, r, and m fields in figure 4.18 are reserved.
r, rsv		Reserved for future definition, set to zeros.

*The link and higher layers are enabled by the Link-On PHY packet described in 4.3.4.2.

[†]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.

4.3.4.2 Link-on packet

For those nodes that do not automatically power-on their link layer circuitry, reception of the cable PHY packet shown

in figure 4.19 shall cause a PH_EVENT.indicate of LINK_ON. (See 8.4.3 for further information.)

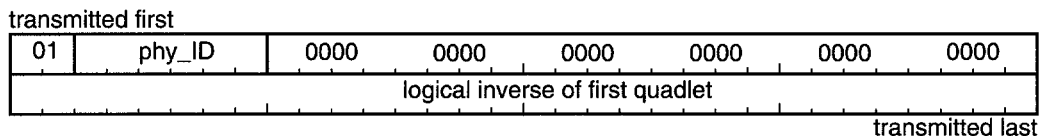


Figure 4.19—Link-on packet format

Table 4.30—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

4.3.4.3 PHY configuration packet

It is possible to optimize Serial Bus performance for particular configurations in 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 done for remote nodes using the PHY configuration packet shown in figure 4.20. (For the local node, the PH_CONT.request service is used) The procedures for using this PHY packet are described in clause 7.3.5.2.1.

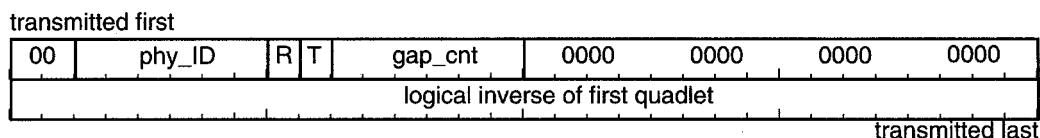


Figure 4.20—PHY configuration packet format

Table 4.31—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 the root_ID of this packet 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 the gap_count field of this packet.
gap_cnt	gap_cnt	New value for the PHY_CONFIGURATION.gap_count field of all nodes. 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 4.4.2.1.2 and receive_actions in 4.4.2.4.2)
NOTE — A PHY configuration packet with R = 0 and T = 0 is reserved and shall be ignored when received.		

4.3.5 Cable PHY timing constants

The configuration and timing of the physical layer in the cable environment is derived from the constants given in table 4.32.

Table 4.32—Cable PHY timing constants

Timing constant	Minimum	Maximum	Comment
ACK_RESPONSE_TIME	0.05 μ s	0.17 μ s	Time between reporting the end of a packet (DATA_END) and requiring that an acknowledging link layer respond with an arbitration request
ARB_SPEED_SIGNAL_START	-0.02 μ s		Time delay between a transmitting port generating the data_prefix signal and the same transmitting port generating the transmission speed signal
BASE_RATE	98.294 Mbit/s	98.314 Mbit/s	Base bit rate (98.304 \pm 100 ppm) Mbit/s
CONFIG_TIMEOUT	166.6 μ s	166.9 μ s	Loop detect time (\sim 16384/BASE_RATE)
ROOT_CONTENTEND_FAST	0.24 μ s	0.26 μ s	Time to wait in state T3 during root contention if random bit off (\sim 24/BASE_RATE) as described in 4.4.2.2
ROOT_CONTENTEND_SLOW	0.57 μ s	0.60 μ s	Time to wait in state T3 during root contention if random bit on (\sim 56/BASE_RATE) as described in 4.4.2.2
DATA_END_TIME	0.24 μ s	0.26 μ s	End of packet signal time (\sim 24/BASE_RATE)
DATA_PREFIX_TIME	0.04 μ s	0.16 μ s	Remaining time after speed sampling before clocked data starts; also the time between the acknowledge and response data in a concatenated response (between \sim 4/BASE_RATE and \sim 16/BASE_RATE)
FORCE_ROOT_TIMEOUT	83.3 μ s	CONFIG_TIMEOUT	Time to wait in state TO (Tree-ID Start; see 4.4.2.2) before acknowledging parent_notify signals (between \sim 8192/BASE_RATE and \sim 16384/BASE_RATE)

Timing constant	Minimum	Maximum	Comment
			Important: This time shall be less than or equal to the CONFIG_TIMEOUT value for the node.
NOMINAL_CYCLE_TIME	124.988 μ s	125.013 μ s	Average time between the start of one isochronous cycle and the next (125 μ s \pm 100 ppm)
MAX_ARB_STATE_TIME		RESET_WAIT (see below)	Maximum time in any arbitration state other than Idle, Reset, or Data Reception
MAX_BUS_HOLD		1.63 μ s	Maximum time a node may transmit a TX_DATA_PREFIX signal between the request acknowledge and data packet of concatenated asynchronous subactions or between data packets of concatenated isochronous subactions
MAX_BUS_OCCUPANCY		100 μ s	Maximum time a node may transmit
MAX_DATA_PREFIX_DELAY		PHY_DELAY (see table 4.29)	Maximum delay between a RX_DATA_PREFIX signal arriving at a receive port and a TX_DATA_PREFIX being sent at a transmit port (this means that the data prefix signal has to be delayed less than the clocked data)
MIN_PACKET_SEPARATION	0.34 μ s		Minimum time between packets (\sim 32/BASE_RATE)
RESET_TIME	166.6 μ s	166.7 μ s	Reset hold time (\sim 16384/BASE_RATE)
RESET_WAIT	166.8 μ s	166.9 μ s	Reset wait time (\sim 16400/BASE_RATE)
SID_SPEED_SIGNAL_START	-0.02 μ s	0.02 μ s	Time delay between a child port generating the TX_IDENT_DONE signal and the same child port generating the speed capability signal
SPEED_SIGNAL_LENGTH	0.10 μ s	0.12 μ s	Time while speed signaling is active (\sim 10/BASE_RATE)

4.3.6 Gap timing

The interpacket gap is a period of time during which the bus is unasserted (Arb_A_Rx and Arb_B_Rx in state Z). There are four types of gaps:

- a) Acknowledge gap—Appears between the end of an asynchronous primary packet and the acknowledge packet.
- b) Isochronous gap—Appears before nonconcatenated channels (isochronous packets).
- c) Subaction gap—Appears before nonconcatenated asynchronous subaction within a fairness interval.
- d) Arbitration reset gap—Appears before nonconcatenated asynchronous subactions when the fairness interval starts.

The timing for these gaps is set by the node variable gap_count (see 4.3.8) using the formulas given in table 4.33.

Table 4.33—Arbitration gap times

Gap type	Detection time		Comment
	Minimum	Maximum	
Acknowledge gap Isochronous gap	0.04 μ s	0.05 μ s	$\sim 4/\text{BASE_RATE}$
Subaction gap	$(27 + \text{gap_count} * 16) / \text{BASE_RATE}$	$(29 + \text{gap_count} * 16) / \text{BASE_RATE}$	After two resets, gap_count is 63, so subaction gaps are $\sim 10 \mu$ s
Arb reset gap	$(51 + \text{gap_count} * 32) / \text{BASE_RATE}$	$(53 + \text{gap_count} * 32) / \text{BASE_RATE}$	After two resets, $\sim 20 \mu$ s

In order to guarantee that all nodes see a gap, nodes shall wait an additional arb_delay (see table 4.34) before starting arbitration as documented in state A2 in figure 4.25.

Table 4.34—Arbitration timing variables

Timing variable	Timing	Comment
arb_delay	$\text{gap_count} * 4 / \text{BASE_RATE}$	After two resets $\sim 2.6 \mu$ s

4.3.7 Cable PHY node constants

There are also a set of parameters that can have different values in each node:

- a) NPORT. The number of ports in this PHY, from 1 to 27.
NOTE — The port numbering follows C-language conventions, with the first port as port #0, the second as port #1, etc.
- b) PHY_DELAY_CATEGORY. Indicates the value of PHY_DELAY according to table 4.29.
- c) PHY_SPEED. The maximum speed packet that a PHY can repeat or source. A value of 0 indicates a maximum speed of S100, 1 a maximum of S200, and 2 a maximum of S400. Other values are reserved.
- d) POWER_CLASS. Nodes may source cable power in 15 W increments or consume power in 1. W increments (for values, see table 4.29).
- e) CONTENDER. A Boolean constant with the value of TRUE if the node wishes to be a contender in the bus management process whenever its link is active. The bus management contention process is described in 8.4.1.

4.3.8 Node variables

Each node has a set of variables that are set by the Bus Manager via PHY packets, the arbitration process, and/or the bus reset process.

Table 4.35—Node variables

Variable name	Comment
arb_enable	Set when a node only needs to wait for a subaction gap to arbitrate
cable_power_active	Set true if the cable power is within normal operating range (see 4.2.2.7)
force_root	Tree-ID delay flag; will force the node to be root if only one node has this flag set; controlled locally by PH_CONT.req, remotely by the PHY configuration packet (see 4.3.4.3)
gap_count	Gap control value; set locally by PH_CONT.req, remotely by the PHY configuration packet (see 4.3.4.3)
initiated_reset	Set true if this node started the bus reset process (is not repeating it)
link_active	Set true if this node has an active link
more_packets	Flag to indicate that more self-ID packets are to be sent
parent_port	Port number that is connected to parent node; set during tree-ID
physical_ID	Physical ID (0..62); set during self-ID
receive_port	Port number that is receiving packet data; set during arbitration
root	Set if the node is the root; set during tree-ID

4.3.9 Port variables

Each port has a set of variables that control the operation of the arbitration state machines.

Table 4.36—Port variables

Variable name	Comment
child	Set if this port is connected to a child node
connected	Set if there is a peer PHY attached to this port
child_ID_complete	The child connected to this port has finished its self-ID
max_peer_speed	Maximum speed accepted by peer PHY attached to this port
speed_OK	The connected port can accept a packet at the requested speed

4.4 Cable PHY operation

The operation of the cable PHY can best be understood with reference to the architectural diagram shown in figure 4.21.

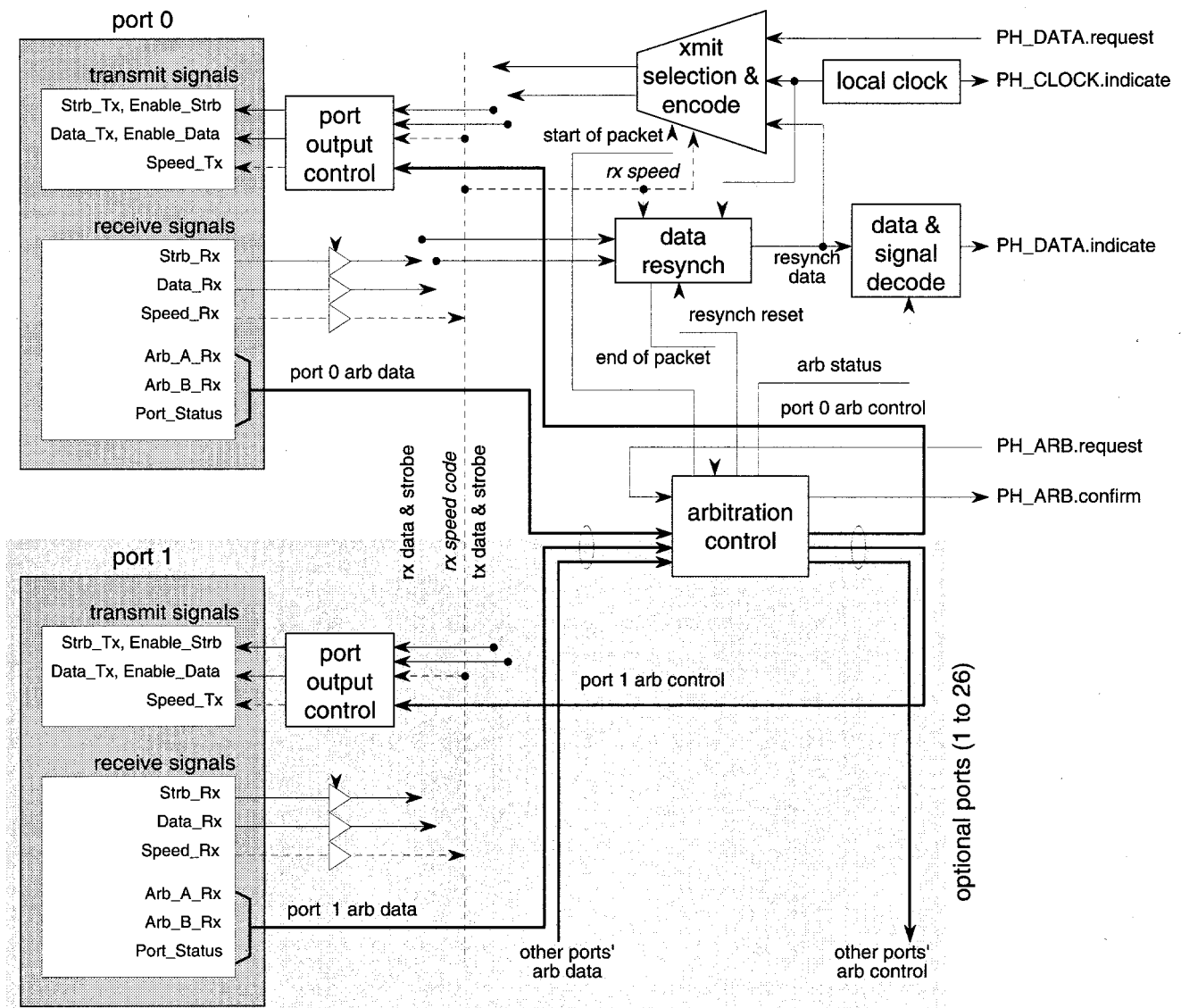


Figure 4.21—Cable PHY architecture

The main controller of the cable PHY is the block labeled “arbitration control,” which responds to arbitration requests from the link layer (PHY_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 4.4.2.

The “data resynch” block decodes the data-strobe signal and retimes the received data to a local fixed-frequency clock provided by the “local clock” block. Since the clocks of receiving and transmitting nodes can be up to 100 ppm different from the nominal, the data resynch function has to be able to compensate for a difference of 200 ppm over the maximum packet length of 84.31 μ s (1024 byte isochronous packet at 98.304 Mbit/s). The operation of this block is described in 4.4.1.2.

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

The “xmit selection & encode” block is the selector between repeated data and data sent by the attached link layer. It also generates the strobe signal for the transmitted data. Its operation is described in 4.4.1.1.

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

All of the procedures in this clause use the syntax specified in 1.6.7 and the definitions in table 4.37.

Table 4.37—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 (); // function that returns a random true or false value
void portT(int port_number, portData txData);
// function that transmits the signal txData on port port_number
portData portR(int port_number);
// function that returns the current signal at port port_number
void portTspeed (int port_number, speedCode speed);
// function that sets the speed code to be transmitted by a port
speedCode portRspeed(int port_number);
// function that returns the current speed signal at a 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 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

```

4.4.1 Data transmission and reception

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

4.4.1.1 Cable environment data transmission

Data transmission is a straightforward function: the data bits are sent to the attached peer PHY along with the appropriately encoded strobe signal using the timing provided by the PHY transmit clock. If the connected port cannot accept data at the requested speed (indicated by the speed_OK[i] flag being false), then no data is sent (leaving the drivers in the “01” data prefix condition).

Table 4.38—Data transmit code

```

static dataBit tx_data, tx_strobe; // memory of last signal sent
void tx_bit (dataBit bit) { // transmit a bit
int i;
wait_event (PHY_CLOCK_indication); // wait for clock
if (bit == tx_data) // if no change in data
    tx_strobe = ~tx_strobe; // invert strobe
tx_data = bit;
for (i = 0; i < NPORT; i++) {
    if ((i != receive_port) && speed_OK[i]) { // do not send data out receive port or to
low speed peers
        portData pd = {phyData(tx_strobe), phyData(tx_data)};
        portT(i,pd); }
    else
        portT(i,TX_DATA_PREFIX);
}
}

```

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

Starting data transmission requires sending a special data prefix signal and a speed code. The speed_OK[i] flag for each port is set if the attached peer PHY has the capabilities to receive the data.

Table 4.39—Start data transmit code

```

void start_tx_packet (speedCode tx_speed) { // send data prefix and speed code
int i;
for (i = 0; i < NPORT; i++) {
    portT(i, TX_DATA_PREFIX); // send data prefix
    speed_OK[i] = (tx_speed <= max_peer_speed[i]);
    if (speed_OK[i])
        portTspeed(i, tx_speed); // receiver can accept, send speed intentions
}
wait_time(SPEED_SIGNAL_LENGTH);
for (i = 0; i < NPORT; i++)
    portTspeed(i, S100); // go back to normal signal levels
wait_time(DATA_PREFIX_TIME); // finish data prefix
}

```

Ending a data transmission requires sending extra bits (known as “dribble bits”). Flushing the last bit through the receiving circuit requires one (for the S100 data rate), three (for the S200 data rate), or seven (for the S400 data rate) extra bits. An extra bit is required to put the two signals TPA and TPB into the correct state depending whether the bus is being held (PH_DATA.request(DATA_PREFIX) or not [PH_DATA.request(DATA_END)]).

Table 4.40—Stop data transmit code

```

void stop_tx_packet (phyData ending_status, speedCode tx_speed) {
    // stop transmitting packet and send new port state
    int i;
    switch (tx_speed) { // add one bit if necessary to align stop
        case(S100)
            break;
        case(S200)
            tx_bit (1); // pad with two extra bits
            tx_bit (1);
            break;
        case(S400)
            for (i = 1; i <= 6; i++)
                tx_bit(1); // pad with six extra bits
            break;
    }
    switch(ending_status) { // add ending-status-dependent penultimate bit
        case(RX_DATA_PREFIX)
            tx_bit(1); // TPB needs to be a 1
            break;
        case(RX_DATA_END)
            tx_bit(0); // TPB needs to be a 0
            break;
    }
    wait_event(PH_CLOCK_indication); // wait for clock
    for (i = 0; i < NPORT; i++)
        switch(ending_status) { // add ending-status-dependent last bit
            case(RX_DATA_PREFIX)
                portT(i, TX_DATA_PREFIX); // send data prefix signal
                break;
            case(RX_DATA_END)
                portT(i, TX_DATA_END); // send data end signal
                break;
        }
    wait_time(DATA_END_TIME);
}

```

NOTE — This algorithm works to force the ending port state to TX_DATA_PREFIX or TX_DATA_END since two things are known: there have to be an even number of bits between a start_tx_packet and a stop_tx_packet, and the start_tx_packet starts with tx_strobe at 0 and tx_data at 1. This means that when stop_tx_packet is called, the port state can only be either 01 or 10. If the port state has to end as 01 (TX_DATA_PREFIX) and the current port state is 01, then this algorithm will set port state to 11 for one bit time, then back to 01. If the ending state has to be 10 (TX_DATA_END), then the port state sequence will be 00, then 10. The process is similar if the current port state is 10.

4.4.1.2 Cable environment data reception and repeat

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

Table 4.41—Data reception and repeat code

```

static phyData old_data, old_strobe;           // memory of last signal sent
//
// decode data-strobe stream and loaf FIFO - this routine is always running
// (speed code recording is also done here)
//
void decode_bit (void) {
repeat {
    if (rx_speed == S100) {                   // store speed signals, if any
        if (portRspeed(receive_port) != S100) {
            rx_speed = portRspeed(receive_port);
            signal (SPEED_SIGNAL_RECEIVED);    // notify start_rx_packet
        }
    }
    new_signal = portR(receive_port);         // get signal
    new_data = new_signal.TPA;                // received data is on TPA
    new_strobe = new_signal.TPB;             // received strobe is on TPB
    if ((new_signal.TPA != old_strobe)       // (new_data != old_data) {
                                                // either data or strobe changed
        FIFO[fifo_wr_ptr] = new_data;        // put data in fifo
        fifo_wr_ptr == (fifo_wr_ptr == FIFO_DEPTH - 1 ? 0 : fifo_wr_ptr++);
        signal (DATA_STARTED);              // signal rx_bit to start
    }
    old_strobe = new_strobe;
    old_data = new_data;
}
}
//
// unload FIFO and repeat data
//
void rx_bit(dataBit *rx_data, boolean *end_of_data) {
int i;
wait_event(PHY_CLOCK_indication);          // wait for clock
if (fifo_rd_ptr == fifo_wr_ptr)            // is FIFO empty?
    end_of_data = TRUE;                    // if so, set flag
else {
    end_of_data = FALSE;                   // if not, clear flag
    rx_data = FIFO[fifo_rd_ptr];           // ... get data bit
    fifo_rd_ptr = (fifo_rd_ptr == FIFO_DEPTH - 1 ? 0 : fifo_rd_ptr++);
    tx_bit (rx_data);                      // ... and repeat it
}
}

```

Starting data reception requires initializing the data resynchronizer and doing the speed signaling with the sender of the data. At the same time, the node has to start up the transmitting ports by sending a special data prefix signal and repeating the received speed code. As in the start_tx_packet function, the node has to do the speed signaling exchange for each transmitting port.

Table 4.42—Start data reception and repeat code

```

// send data prefix and do speed signaling
//
speedCode start_rx_packet ( ) {
int i;
fifo_rd_ptr = 0; // reset data resynch buffer
fifo_wr_ptr = 0;
waiting_for_data_start = TRUE; // first data bit not yet received
portT(receive_port, IDLE); // turn off grant, get ready to receive
wait_event (SPEED_SIGNAL_RECEIVED | DATA_STARTED); // wait for speed sig or first bit
if (rx_speed != S100) { // if speed signalling, then must repeat it
tx_speed = rx_speed; // get speed of packet to repeat
for (i = 0; i < NPORT; i++)
if (i != receive_port) {
portT(i, TX_DATA_PREFIX); // send data prefix out repeat ports
speed_OK[i] = (tx_speed <= max_peer_speed[i]);
if (speed_OK[i])
portTspeed(i, tx_speed); // receiver can accept, send speed intentions
}
wait_time(SPEED_SIGNAL_LENGTH);
for (i = 0; i < NPORT; i++)
if (i != receive_port)
portTspeed(i, S100); // go back to normal signal levels
wait_time(DATA_PREFIX_TIME); // finish data prefix
wait_event (DATA_STARTED); // wait for decoder to start
}
for (i = 0; i < FIFO_DEPTH/2 - 1; i++)
wait_event (PHY_CLOCK_indication); // make sure FIFO is centered
}

```

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

Table 4.43—Stop data reception and repeat

```

void stop_rx_packet ( ) { // receiving done, clean up ending
int i;
portData ending_data;
ending_data = portR(receive_port); // get ending state on receive port
stop_tx_packet(ending_data, rx_speed); // finish repeat
}

```

4.4.2 Cable environment arbitration

The cable environment supports the immediate, fair, isochronous, and cycle_master arbitration classes, where the cycle_master class is only available at the root node.

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 this standard. The state machine notes are informative.

4.4.2.1 Bus reset

The bus reset process starts when a bus reset signal is received or generated locally. Its purpose is to guarantee that all nodes propagate the reset signal.

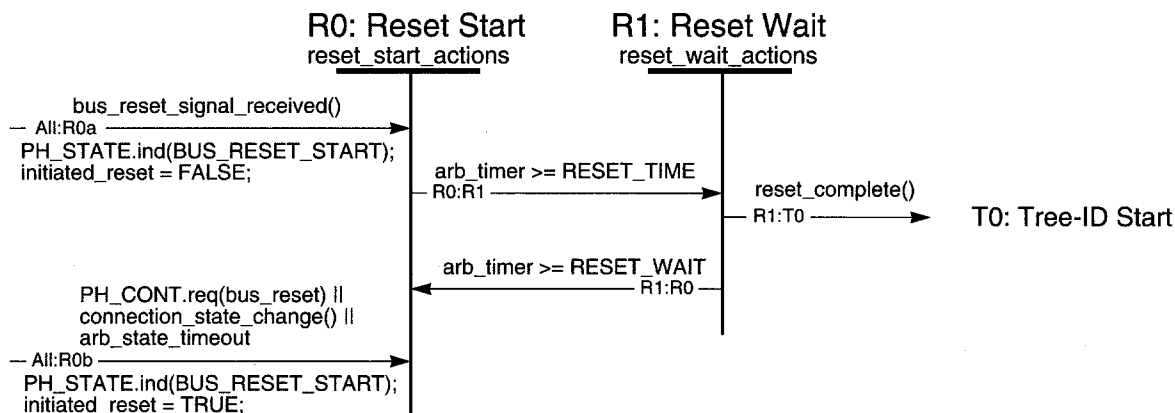


Figure 4.22—Bus reset state machine

4.4.2.1.1 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 the arbitration signal lines of any port (see table 4.28).

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 PHY_CONTROL.request.
- 2) The PHY detects a change in the connection state of any port.
- 3) The PHY stays in any state other than Idle, Reset_Wait, Transmit, or Receive for longer than MAX_ARB_STATE_TIME.

State R0:Reset Start. The node sends a BUS_RESET signal long enough that all other bus activity settles down (RESET_TIME is longer than the worst-case packet transmission plus the worst-case bus turnaround time).

Transition R0:R1. The node sends 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 go idle (indicating that the attached peer PHYs have left their R0 state).

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

Transition R1:T0. All the connected ports are idle, so the directly connected nodes are in state R1 or already in the start of tree ID (state T0).

4.4.2.1.2 Bus reset actions and conditions

Table 4.44—Bus reset actions and conditions

```

boolean bus_reset_signal_received() {
// true if a bus reset signal is received on any connected port
int i;
for (i = 0; i < NPORT; i++) {
    if (connected[i] && portR(i) == BUS_RESET; // found a reset signal
        return (TRUE);
    }
return (FALSE);
}
boolean connection_state_change() {
// true if any port goes from connected to disconnected or vice-versa
int i;
state_change = FALSE;
for (i = 0; i < NPORT; i++) {
    if (connected[i] !=
        old_connected[i]; { // did connection change?
        state_change = TRUE;
        old_connect[i] = ~old_connect[i];
    }
}
return (state_change);
}
void reset_start_actions() {
// transmit the bus_reset signal for a RESET_TIME period on all ports
int i;
root = false;
physical_ID = 0;
child_count = 0;
bus_initialize_active = TRUE;
if (gap_count_reset_disable) // first reset since setting gap_count?
    gap_count_reset_disable = FALSE // if so, leave it as is and arm it for next
else
    gap_count = 0x3F; // otherwise, set it to the maximum
for (i = 0; i < NPORT; i++) {
    if (connected[i])
        portT(i, BUS_RESET); // propagate reset signal
    else
        portT(i, IDLE); // but only on connected ports
    child[i] = false;
    child_ID_complete[i] = false;
}
arb_timer = 0; // start timer
}
void reset_wait_actions() {
// transmit the idle signal for a RESET_WAIT period on all ports
int i;
for (i = 0; i < NPORT; i++) portT(i, IDLE); // reset signal is finished
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;
}

```


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

State T2: Parent Handshake. At this point, a node is waiting to receive the RX_PARENT_HANDSHAKE signal (01) [the result of the node sending TX_PARENT_NOTIFY (0Z) and its parent sending TX_CHILD_NOTIFY (1Z—reversed to Z1 by the tx/rx signal swap)]. This step is bypassed if the node is root (receiving RX_PARENT_NOTIFY on all connected ports). Another way this state can be exited is if it receives the RX_ROOT_CONTENTION signal from its parent.

Transition T2:S0. When the node receives the RX_PARENT_HANDSHAKE signal, it starts the self-ID process by sending the IDLE signal (ZZ) (see 4.4.2.3.1). It shall also take this transition if it is root, since it does not have a parent.

Transition T2:T3. If a node receives a RX_PARENT_NOTIFY signal on the same port upon which it is sending a TX_PARENT_NOTIFY signal, the merged signal is called root_contention (00). This will only happen for a single pair of nodes as they both bid to make the other node its parent.

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

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

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

4.4.2.2.2 Tree-ID actions and conditions

Table 4.45—Tree-ID actions and conditions

```

void tree_ID_start_actions() {
int i, temp_count;
arb_timer = 0; // start timer
while(true) { // loop forever
temp_count = 0; // temporary child counter
for (i = 0; i < NPORT; i++)
if (~connected[i] || portR(i) == RX_PARENT_NOTIFY) {
parent" // when unconnected or receiving "you are my
parent"
child[i] = true; // set child flag
temp_count++; // and increment counter
child_count = temp_count; // set current child count
} // end of forever loop
}
void child_handshake_actions() {
int i;
root = true; // root will stay true if all ports are child ports
for (i = 0; i < NPORT; i++) {
if (connected[i] && child[i])
portT(i, TX_CHILD_NOTIFY); // you are my child
else if (connected[i]) {
portT(i, TX_PARENT_NOTIFY); // you are my parent
parent_port = i; // there is at most one port with child==false
root = false; // cannot be root since there is a parent
}
}
boolean
child_handshake_complete() { // true if all active children in "Start Self_ID"
int i;
for (i = 0; i < NPORT; i++)
if (child[i] && connected[i] && (portR(i) != RX_CHILD_HANDSHAKE))
return false; // active child not giving "you are my parent"
return true; // will also be true if there are no active
children
}
void root_contend_actions() {
int i;
contend_time = (random_bool() ? CONTEND_SLOW : CONTEND_FAST);
for (i = 0; i < NPORT; i++) {
if (child[i])
portT(i, TX_CHILD_NOTIFY); // you are my child
else
portT(i, IDLE); // abandon "you are my parent" request;
}
arb_timer = 0; // start arbitration timer
}
}

```

4.4.2.3 Self-identify

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

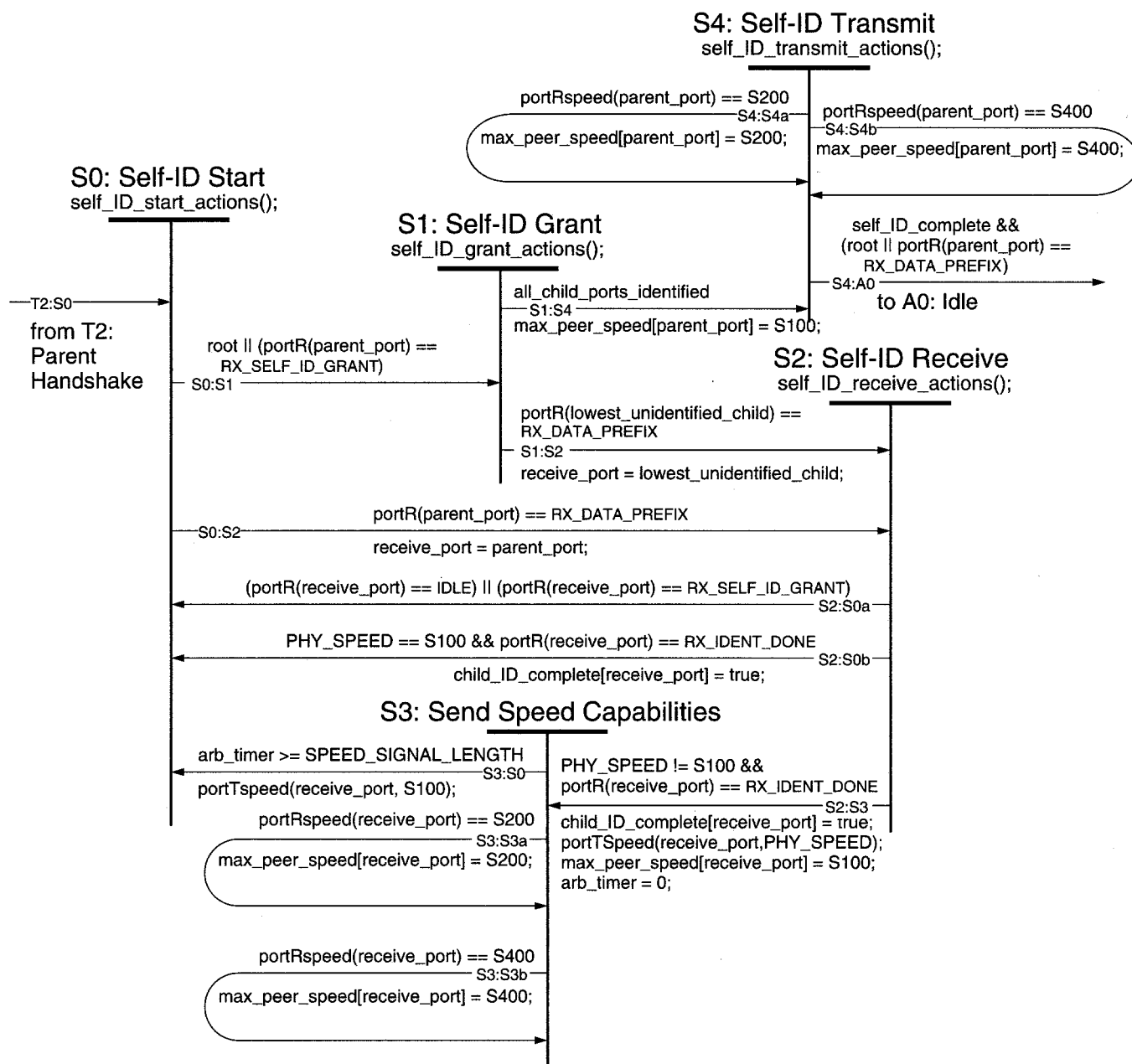


Figure 4.24—Self-ID state machine

NOTE — This state machine does not include the reset conditions, since that would result in an overly complex figure. See 4.4.2.1.1.

4.4.2.3.1 Self-ID state machine notes

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

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 4.4.1.2. 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 than S100, it transmits on the receiving child port its speed capability signals as defined in 4.2.2.3 for a fixed duration SPEED_SIGNAL_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 4.3.4) using the process described in 4.4.1.1. When a nonroot node is finished, it sends a TX_IDENT_DONE signal (1Z) and a speed capability signal as defined in 4.2.2.3 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 Transition S4:A0, 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 4.4.2.4 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 become 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.

4.4.2.3.2 Self-ID actions and conditions

Table 4.46—Self-ID actions and conditions (Sheet 1 of 3)

```

void self_ID_start_actions() {
int i;
all_child_ports_identified = true; // will be reset if any active children are
unidentified
for (i = 0; i < NPORT; i++) {
    if (child_ID_complete[i])
        portT(i, TX_DATA_PREFIX); // tell identified children to prepare to receive data
    else {
        portT(i, IDLE); // allow parent to finish
        if (child[i] && connected[i]) { // if connected child
            if (all_child_ports_identified)
                lowest_unidentified_child = i;
            all_child_ports_identified = false;
        }
    }
}
}

void self_ID_grant_actions() {
int i;
for (i = 0; i < NPORT; i++)
    if (~all_child_ports_identified && (i == lowest_unidentified_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, see 4.4.2.4.2
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 node is transmitting (no port has this
number)

```

Table 4.46—Self-ID actions and conditions (Sheet 2 of 3)

```

start_tx_packet(S100);          // send data prefix and 98.304 Mbit/s speed code
for (SID_pkt_number = 0; SID_pkt_number <= last_SID_pkt; SID_pkt_number++) {
    if (SID_packet_number == 0) // high 24 bits of first self-ID packet
        self_ID_pkt = 0x00800000 | // constant 10 for self-ID header
            (physical_ID << 16) | // phy ID number (offset-ID for Link)
            link_pwr << 14 | // link power, set if on
            (gap_count << 8) | // gap_count for gap timers
            (PHY_SPEED << 6) | // speed capability (S100/S200/S400)
            (PHY_DELAY << 4) | // phy delay for gap count calculations
            (CONTENDER << 3) | // set if node wants to be bus or limited manager
            POWER_CLASS; // power supply/consumption class
    else // high 14 bits of second, third, and fourth self-ID packet
        self_ID_pkt = 0x00002020 | // constant 10 for self-ID plus extension flag
            (physical_ID << 6) | // phy ID number (offset-ID for Link)
            ((SID_pkt_number - 1) << 2); // packet sequence number and reserved 00 field
    while (port_number < ((SID_pkt_number + 1)*8 - 5)) {
        // concatenate port status
        if (port_number >= NPORT)
            ps = 0b00; // unimplemented
        else if (~connected[port_number])
            ps = 0b01; // unconnected
        else if (child[port_number])
            ps = 0b11; // active child
        else
            ps = 0b10; // parent
        self_ID_pkt = (self_ID_pkt << 2) | ps;
        port_number++;
    }
    self_ID_pkt = (self_ID_pkt << 2)
    if (initiated_reset && SID_pkt_number == 0)
        self_ID_pkt = self_ID_pkt | 0x00000002; // set initiated reset flag if first packet
    if (SID_pkt_number == last_SID_pkt) { // last packet has data_end
        tx_quadlet(self_ID_pkt); // send packet
        tx_quadlet(~self_ID_pkt); // send check bits
        stop_tx_packet(TX_DATA_END, S100);
    } else { // other packets are concatenated and need "more" bit
        self_ID_pkt = self_ID_pkt | 0x00000001;
        tx_quadlet(self_ID_pkt); // send packet
        tx_quadlet(~self_ID_pkt); // send check bits
        stop_tx_packet(TX_DATA_PREFIX, S100);
    }
}
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);
    }
}

```

Table 4.46—Self-ID actions and conditions (Sheet 3 of 3)

```

// 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 = quad_data << 1; // shift to position next bit
}
}

```

4.4.2.4 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).

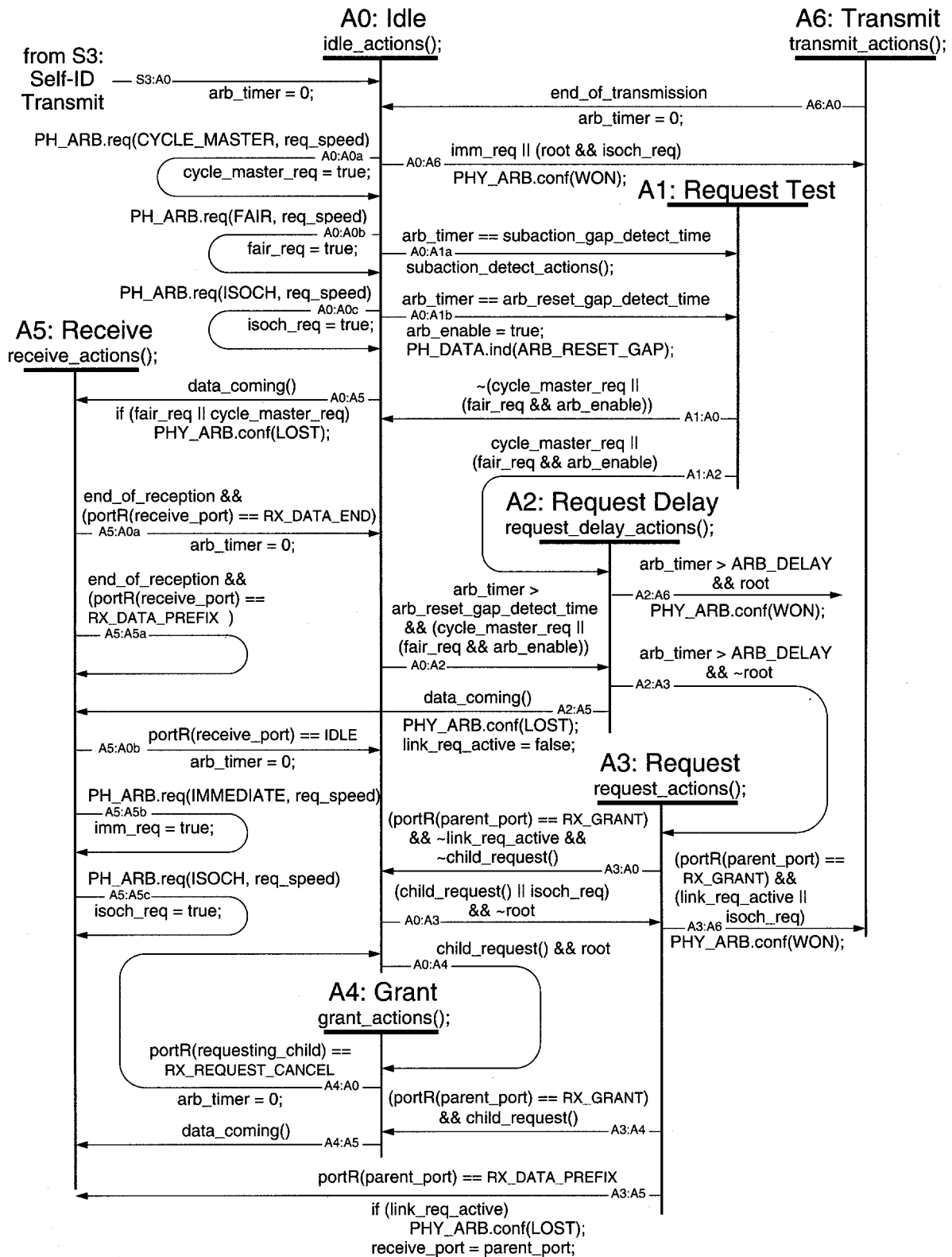


Figure 4.25—Cable arbitration state machine

NOTE — This state machine does not include the reset conditions, since that would result in an overly complex figure. See 4.4.2.1.1.

4.4.2.4.1 Normal arbitration state machine notes

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

Transition A0:A0a. If the node is the cycle master and the cycle timer expires, the link layer shall make a cycle master arbitration request. The node queues the request since an appropriate gap may not have occurred.

Transition A0:A0b. If the link layer makes a fair arbitration request, the node queues it.

Transition A0:A0c. If the link layer makes an isochronous arbitration request, the node queues it.

Transition A0:A1a. If a subaction gap occurs, the PHY notifies the link layer (and the node controller, if this is the first subaction gap after a bus reset) and tests whether there is an active request. This test is only performed for a single value of the idle timer (notice the equality test). This is necessary since starting a request in the interval between the detection of a subaction gap and an arbitration reset gap may cause some nodes to see an arbitration reset gap while others do not.

Transition A0:A1b. If an arbitration reset gap occurs, a fairness interval is over, so the PHY sets the arbitration enable flag and notifies the link layer.

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* if the node is not the root, it passes the request on to its parent.

Transition A0:A4. If the PHY receives a RX_REQUEST signal (0Z) from one of its children, and if the node *is* the root, it starts the bus grant process.

Transition A0:A5. 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:A6. 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 A1: Request Test. This state consumes zero time; it only operates as a convenient point for testing whether a link request is active or not after a timing event.¹³

Transition A1:A0. If a link request has not been queued, the PHY continues to wait for one.

Transition A1:A2. If a link request has been queued, the PHY can start the request process.

State A2: Request Delay. This state marks the start of the request handshake on the bus. No actual action is taken—instead, this state is just a delay to guarantee that all nodes will detect the appropriate gap. If this delay was not present, a very busy node with a slightly faster clock could prevent other nodes from ever seeing an arbitration reset gap. This state also records that the request is coming from the local link layer, not from a child node.

¹³Implementation note: this state is just a documentation aid to simplify the state machine. No time should be consumed in this state; otherwise, all the other transitions of state A0 would also have to be implemented.

Transition A2:A3. When the delay is over and the node is not the root, then it can make a request to its parent.

Transition A2:A5. If the PHY receives the RX_DATA_PREFIX signal on any of its ports while waiting to make its request, it shifts into the Receive state and notifies the link layer that the pending arbitration request has been lost.

Transition A2:A6. When the delay is over and the node is the root, it can start transmitting a packet immediately.

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 prepare to receive a packet.

Transition A3:A0. If the PHY receives a RX_GRANT signal (00) from its parent, and if 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 if the requesting child is still making a request, the PHY grants the bus to that child.

Transition A3:A5. 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:A6. If the PHY receives a RX_GRANT signal (00) from its parent, and if 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 signal (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:A5. 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 A5: 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 by the receive actions in table 4.47. Note that the packet received could be a PHY packet (self-ID, link-on, or PHY configuration), acknowledge packet, or normal data packet. The PHY configuration and link-on packets are interpreted by the PHY, as well as being passed on to the link layer.

NOTE — To simplify the writing of this standard, the self-ID packets are also handled by the PHY layer, although most real systems will reserve this function for the same hardware that implements the link layer.

Transition A5: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 A5: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 A5:A5a. If a packet ends and the received signal is RX_DATA_PREFIX (10), then there may be another packet coming, so the receive process is restarted.

Transition A5:A5b. During reception of a packet, the attached link layer will determine if it needs to send an acknowledge. If it does, it shall make the PHY arbitration request with the IMMEDIATE arbitration class. This request is queued so that the A0:A6 or A0:A3 transitions can be taken immediately after PHY returns to the Idle state.

Transition A5:A5c. During reception of a packet, the attached link layer will determine if the packet is a cycle start and if any isochronous data needs to be transmitted. If this is true, it shall make the PHY arbitration request with the ISOCHRONOUS arbitration class. This request is queued so that the A0:A6 or A0:A3 transitions can be taken immediately after PHY returns to the Idle state.

State A6: Transmit. The transmission of a packet starts by the node sending a TX_DATA_PREFIX and speed signal for 100 ns as described in 4.2.2.3, 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 A6:A0. If the link layer sends a DATA_END, the PHY shuts down transmission using the procedure described in 4.4.1.1 and returns to the Idle state.

4.4.2.4.2 Normal arbitration actions and conditions

Table 4.47—Normal arbitration actions and conditions (Sheet 1 of 4)

```

boolean child_request() { // true if a child is requesting access
int i;
for (i = 0; i < NPORT; i++)
    if (connected[i] && child[i] && (portR(i) == RX_REQUEST)) {
        // found a child that is requesting the bus
        requesting_child = i;
        // remember port for later
        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)) {
        // found a port that is sending a data_prefix signal
        receive_port = i;
        // remember port for later
        return true;
    }
return false;
}

void idle_actions() {
int i;
for (i = 0; i < NPORT; i++)
    portT(i, IDLE); // turn off all transmitters
}
void subaction_detect_actions () {
PH_DATA.ind (SUBACTION_GAP);
if (bus_initialize_active) {
    PH_EVENT.ind (BUS_RESET_COMPLETE);
    bus_initialize_active = FALSE;
}
}

```


Table 4.47—Normal arbitration actions and conditions (Sheet 2 of 4)

```

void request_delay_actions() {
    arb_timer = 0; // start delay timer
    link_req_active = true; // remember that this node made the request
}
void request_actions() {
    int i;
    for (i = 0; i < NPORT; i++)
        if (connected[i] && child [i] && (link_req_active || i != requesting child))
            portT(i, TX_DATA_PREFIX); // send data prefix to all nonrequesting children
    portT(parent_port, TX_REQUEST); // send request to parent
}
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 nonrequesting children
}

void receive_actions() {
    int i;
    int bit_count = 0;
    union {
        dataBit bits[64];
        struct {
            unsigned pkt_type:2;
            unsigned addr:6;
            unsigned R: 1; // force-root flag
            unsigned T: 1; // set gap flag
            unsigned gap_count:6; // new gap count
            unsigned reserved:16; // reserved
            dataBit check bits[32]:32;
        } phy_info;
    } phy_pkt;
    boolean test_end = false;
    boolean received_data;
    fair_req = false; // cancel requests
    cycle_master_req = false;
    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

    while (~test_end) {
        rx_bit(received_data, test_end);
        if (~test_end) { // normal data, send to link layer
            PH_DATA.ind(received_data);
            if (bit_count < 64) // accumulate first 64 bits
                phy_pkt.bits[bit_count++] = received_data;
        }
    }
}

```

Table 4.47—Normal arbitration actions and conditions (Sheet 3 of 4)

```

ending_data = portR(receive_port);
switch (ending_data) {
    // send appropriate end of packet indicator
    case RX_DATA_PREFIX : PH_DATA.ind(DATA_PREFIX);
        break;
    case RX_DATA_END : PH_DATA.ind(DATA_END);
        break;
}
stop_rx_packet(ending_data);
end_of_reception = true;
if (bit_count == 64) {
    // PHY packet received
    boolean good_phy_packet = TRUE; // check for good format
    for (i = 0; i < 32; i++)
        good_phy_packet =
            (phy_pkt.bits[i] == (~phy_pkt.phy_info.check_bits[i + 32]) &&
good_phy_packet));
    if (good_phy_packet) {
        phy_addr = received_bits
        switch (phy_pkt.phy_info.pkt_type) {
            // process packets differently based on phy packet type
            case 0b00 : // phy config packet case
                if (phy_pkt.phy_info.R)
                    // phy force-root, set if address match, clear otherwise
                    force_root = (phy_pkt.phy_info.address == physical_ID)
                if (phy_pkt.phy_info.T) {
                    // phy gap_count, set unconditionally and set reset_disable
                    gap_count = phy_pkt.phy_info.gap_count;
                    gap_count_reset_disable = TRUE;
                }
                break;
            case 0b01// : // link-on packet
                if (phy_pkt.phy_info.address == physical_ID)
                    // good address
                    PH_EVENT.ind (LINK_ON);
                break;
            // self-ID packet (0b10), and reserved packets (0b11) aren't handled
        }
    }
}
void transmit_actions() {
int i;
boolean test_end = false;
phyData data_to_transmit;
imm_req = false; // acknowledge any immediate requests
if (fair_req) {
    arb_enable = false; // reset arbitration enable flag
    fair_req = false; // cancel requests
}
cycle_master_req = false;
isoch_req = false;
receive_port = NPORT; // indicate that node is transmitting (no port has this number)
start_tx_packet(req_speed); // send data prefix and do speed negotiation
while (~test_end) {

```

Table 4.47—Normal arbitration actions and conditions (Sheet 4 of 4)

```

PH_CLOCK.ind; // let link know it is time to send data
while(~PH_DATA.req(data_to_transmit));
    // wait until we get data from the Link
switch(data_to_transmit) {
    case DATA_END:
        stop_tx_packet(DATA_END);
        test_end = true; //set end of packet indicator
        break;
    case DATA_PREFIX :
        stop_tx_packet(DATA_PREFIX);
        wait_time (MIN_PACKET_SEPARATION);
        // hold bus for more data (concatenated packet)
        break;
    case 0, 1 : // send data
        tx_bit (data_to_transmit) ;
        break;
}
}
end_of_transmission = true;
}

```

5. Backplane PHY specification

The backplane environment physical layer provides the interface from a physical device to the backplane media. It provides arbitration services to permit a node to gain access to the bus, and it performs the signal translations required to drive the bus and receive information from the bus.

Unlike the specification for the cable physical layer, the backplane PHY specification does not include a description of connectors or media. Such documentation is assumed to be part of a host backplane specification or to be included in the requirements for the application environment.

The term “application environment” refers to the physical environment of the bus, the nodes, and the system that contains them. This environment may be a standardized host backplane (e.g., a FutureBus+ profile) that provides the signal requirements, a detailed description of the transceivers, the mechanical arrangement of the modules, and the temperature range over which operation is guaranteed.

The backplane PHY shares some commonality with the cable PHY. Common functions include: bus state determination, bus access protocols, encoding and decoding functions, and synchronization of received data to a local clock.

5.1 Backplane PHY services

PHY layer services are provided at the interface between the PHY layer and higher layers; specifically, the link layer and the node controller, as illustrated in table 5.1. The method by which these services are communicated between the layers is not defined by this standard. PHY layer services may perform actions specified by the higher layer. PHY layer services may also communicate parameters that may or may not be associated with an action.

Table 5.1—Summary of backplane PHY layer services

Service	Layer communicated with	Purpose of service
Backplane PHY control request	From the node controller	Configure the backplane PHY layer
Backplane PHY control confirmation	To the node controller	Confirm backplane PHY control request
Backplane PHY event indication	To the node controller	Alert node controller to events detected in the backplane PHY layer
Backplane PHY arbitration request	From the link layer	Cause the backplane PHY layer to request control of the bus
Backplane PHY arbitration confirmation	To the link layer	Confirm backplane PHY arbitration request
Backplane PHY clock indication	To the link layer	Indicate when it is time for the link layer to make a backplane PHY data request
Backplane PHY data request	From the link layer	Cause the backplane PHY to send one clocked data symbol or to start or end a data packet
Backplane PHY data indication	To the link layer	Indicate the reception of one clocked data symbol or a bus status change

The method by which these services are communicated between the layers is not defined in this standard. PHY layer services may perform actions specified by the higher layer. PHY layer services may also communicate parameters that may or may not be associated with an action.

5.1.1 Backplane PHY bus management services for the management layer

These services are used by the node controller component of the Serial Bus management layer to control the bus level actions of the PHY layer. The PHY layer uses these services to communicate changes of state within the PHY layer or on the bus.

5.1.1.1 PHY control request (PH_CONTROL.request)

The node controller uses this service to request the PHY layer to perform specific actions and to specify PHY layer parameters. It may also be used to request status about the PHY layer. The PHY layer shall service the request immediately upon receipt by the PHY layer. This service is confirmed.

The following actions shall be provided by this service:

- a) Bus Reset. The PHY layer shall reset the bus and initialize itself.
- b) Disable Transmit. The PHY layer shall set all bus outputs to a high impedance state. This bus output state shall be maintained until the node controller requests an Enable Transmit action. Link layer service actions that would require a change in bus output state shall not be performed.
- c) Enable Transmit. The PHY layer shall allow link layer service actions to change the state of the bus outputs.
- d) Present Status. The PHY layer shall return status to the node controller. The PHY layer shall return status via the PHY control confirmation service.

NOTE 1— This Present Status service is expected to be used to communicate new parameters without causing any other action.

The following parameters are communicated via this service:

- **Physical_ID.** This 6-bit parameter is used as the arbitration number during the arbitration process. The value of this parameter is unique for each node on the bus. Refer to 5.4.2.1.
- **Priority.** This 4-bit parameter is used during the urgent arbitration process. Refer to 5.4.2.1.

NOTE 2— These parameters may be changed remotely by Transaction Layer “write” operations to CSRs within the node controller. Some register locations within the PHY, however, may not have corresponding CSRs or unit architectures. Such registers may be accessed locally via the LINK-PHY interface, as described in annex J. If an application environment requires remote access to such registers, then the application environment is expected to specify an architecture for doing this.

5.1.1.2 PHY control confirmation (PH_CONTROL.confirmation)

The PHY layer uses this service to confirm the results of a PHY control request service. The PHY layer shall communicate this service to the node controller upon completion of a PHY control request. There are no actions provided by this service. When the corresponding control request is “Present Status,” the following parameters are communicated via this service:

- **Physical_ID.** As described in 5.4.2.1.
- **Priority.** As described in 5.4.2.1.
- **Initiated_Reset.** The PHY control request action was completed successfully. See 5.5.1.

5.1.1.3 PHY event indication (PH_EVENT.indication)

The PHY layer uses this service to indicate PHY-level events to the node controller. There are no actions provided by this service. No response is defined for this indication. The following parameters are communicated via this service:

- **PHY Event.** This parameter shall contain the current state of the PHY layer. The following values are defined for this parameter:
 - **BUS_RESET_START.** The PHY layer has detected a bus reset. See 5.5.1.
 - **BUS_RESET_COMPLETE.** The PHY layer has detected a subaction gap after the bus reset process has started. See 5.5.1.

5.1.2 PHY layer arbitration services for the link layer

These services are used to communicate arbitration requests between the PHY layer and the link layer. See 5.4.1.

5.1.2.1 PHY arbitration request (PH_ARB.request)

The link layer uses this service to request the PHY layer to start arbitration for the bus. The PHY layer shall arbitrate for the bus using the method specified by the service parameters. The PHY layer shall service the request immediately upon receipt from the link layer. This service shall be confirmed when the arbitration process completes. If a node loses arbitration (PH_ARB.conf of LOST), it shall reissue the arbitration request.

The following parameters are communicated via this service:

- a) **Arbitration Class.** This parameter shall contain the method of arbitration performed by the PHY arbitration request. The method of arbitration shall be one of the following:
 - 1) **FAIR.** The PHY layer shall start arbitration at the next subaction gap if its arbitration_enable flag is set; otherwise, it shall start arbitration at the next arbitration reset gap. The link layer uses this arbitration class to send a fair asynchronous packet. The lowest priority (all zeros) is reserved for this arbitration class. Refer to 5.4.1.2.
 - 2) **URGENT.** The PHY layer shall begin arbitration at the next subaction gap if its urgent_count is not zero. Otherwise, it shall start arbitration at the next arbitration reset gap. The link layer uses this arbitration class to send an urgent asynchronous packet. The four-bit subparameter “pri” is used to set the priority level for the arbitration process, with the exception that the lowest priority and the highest priority are

reserved (and cannot be used for this arbitration class; refer to 5.4.2.1). This allows urgent requests to have priority access to the bus, but the number of requests that can be made each fairness interval is limited (see 5.4.1.3).

- 3) CYCLE_MASTER. The PHY layer shall begin arbitration at the next subaction gap. The link layer uses this arbitration class to send a cycle start packet. The highest priority (all ones) is reserved for this arbitration class (see 5.4.1.4).
 - 4) ISOCHRONOUS. The PHY layer shall begin arbitration as soon as an acknowledge gap is detected. The link layer uses this arbitration class to send an isochronous packet (see 5.4.1.3).
 - 5) IMMEDIATE. The PHY layer shall communicate a PHY arbitration confirmation with an Arbitration Request Status of WON to the link layer as soon as an acknowledge gap is detected. The link layer uses this arbitration class to send an acknowledge packet (see 5.4.1.3).
- b) Pri. This parameter shall contain the priority associated with the Urgent Access Method. This parameter shall be ignored if the arbitration class is ISOCHRONOUS or IMMEDIATE.

NOTE — The Pri parameter could be used to distinguish FAIR and CYCLE MASTER requests from URGENT requests.

5.1.2.2 PHY arbitration confirmation (PH_ARB.confirmation)

The PHY layer uses this service to confirm the results of a PHY arbitration request service. The PHY layer shall communicate this service to the link layer upon completion of a PHY arbitration request. There are no actions provided by this service. The following parameter is communicated via this service:

- Arbitration Request Status. This parameter shall contain the result of a PHY arbitration request action. The following values are defined for this parameter:
 - WON. The PHY arbitration request action was completed successfully. The PHY layer shall begin communicating PHY clock indications to the link layer. This confirmation is returned after the minimum length data_prefix (as described in 5.4.2.1) has been transmitted.
 - LOST. The PHY arbitration request action was not successful.

5.1.3 PHY layer data services for the link layer

These services are used to communicate data symbols and control information between the PHY layer and the link layer.

5.1.3.1 PHY clock indication (PH_CLOCK.indication)

The PHY layer uses this service to indicate to the link layer that a data symbol transmission is about to occur. The link layer shall respond to this indication with a PHY data request. There are no actions provided by this service. No parameters are communicated via this service.

This indication occurs once for each bit cell time (see 5.2.3).

The PHY layer shall begin communicating this indication to the link layer after it has communicated a PHY arbitration confirmation with an Arbitration Request Status of WON. The PHY layer shall stop communicating these indications to the link layer after the link layer communicates a PHY data request with data of DATA_END.

5.1.3.2 PHY data request (PH_DATA.request)

The link layer uses this service to control the transmission of clocked data symbols by the PHY layer: The link layer shall communicate one PHY data request for each PHY clock indication. The PHY layer shall service the request immediately upon receipt by the PHY layer.

The following parameter is communicated via this service:

- Data. This parameter shall contain the symbol to be transmitted on the bus. See 5.2.2.1 for a definition of the logic states associated with the symbols. The following values are defined for this parameter:
 - DATA_ONE. A symbol representing a data bit of one shall be transmitted on the bus.
 - DATA_ZERO. A symbol representing a data bit of zero shall be transmitted on the bus.
 - DATA_PREFIX. The PHY layer shall stop sending clocked data bits and leave the bus in the data_prefix state using the algorithm described in 5.4.2.1. This symbol shall be transmitted between acknowledge and response packets during concatenated subactions.
 - DATA_END. The link layer shall stop communicating PHY data requests to the PHY layer. The PHY layer shall stop communicating PHY clock indications to the link layer. The PHY layer shall set bus outputs to the data_end state using the algorithm described in 5.4.2.1.

NOTE — A DATA_PREFIX can also be initiated by the PHY layer. This occurs once a PHY arbitration request has been completed successfully and until transmission of the packet begins, and once an acknowledge gap is detected and until transmission of the acknowledge packet begins.

Once the link layer starts sending a DATA_PREFIX or a DATA_END, it shall continue sending it for at least four arbitration clock times (approximately 81.38 ns) before sending any other data symbols (see 5.4.2.1). This minimum value allows sufficient time for all nodes to detect that such a symbol has been transmitted. The duration of a DATA_PREFIX shall not exceed 160 arbitration clock times (approximately 3255.2 ns), and the duration of a DATA_END shall not exceed 16 arbitration clock times (approximately 325.52 ns). These maximum values ensure that a transaction will be completed within a limited amount of time.

Once the link layer starts sending a data bit (DATA_ONE or DATA_ZERO), it shall continue to send a whole packet unless the MAX_BUS_OCCUPANCY period is exceeded (see 5.4.2.1). If data requests continue to be received after this time, the PHY shall release the bus using the signal transitions for a DATA_END. In any event, the total number of continuous data bits shall be even.

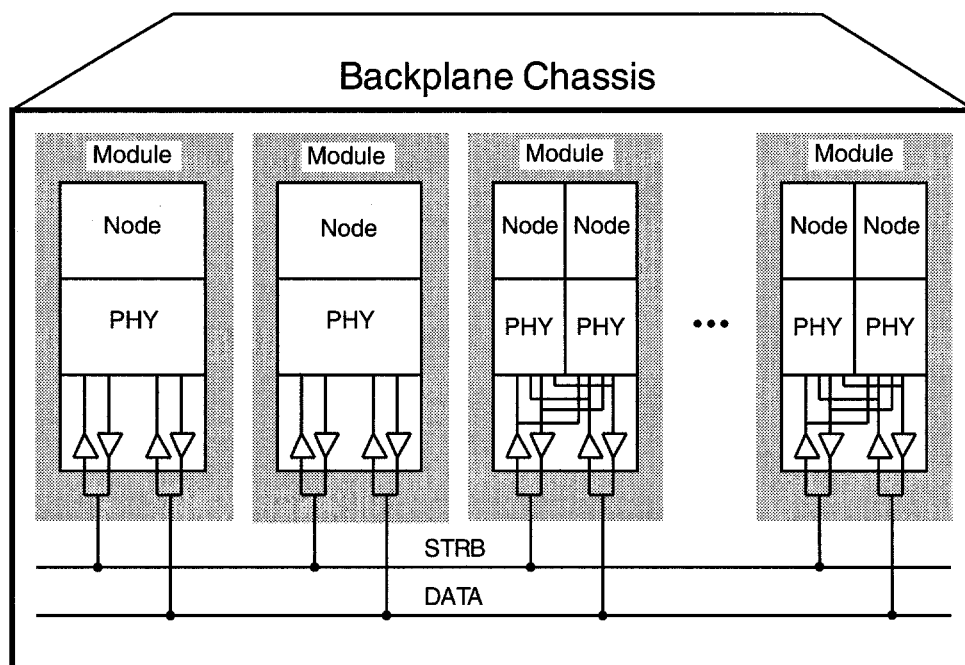
5.1.3.3 PHY data indication (PH_DATA.indication)

The PHY layer uses this service to indicate to the link layer changes in the state of the PHY layer. These changes can include received data and other bus events. The PHY layer shall communicate this indication to the link layer for each data bit received. If the node is not receiving data bits, the PHY layer shall use this indication to communicate events needed by the link layer. No response is defined for this indication. The following parameter is communicated via this service:

- Data. This parameter shall contain the information decoded by the PHY that is needed by the link layer. The following values are defined for this parameter:
 - DATA_START. The start of a packet has been detected on the bus.
 - DATA_ONE. A data bit of one has been received on the bus.
 - DATA_ZERO. A data bit of zero has been received on the bus.
 - DATA_PREFIX. A node has control of the bus and is preparing to transmit a packet.
 - DATA_END. The end of a packet has been detected on the bus, and the transmitting node is releasing control.
 - ARBITRATION_RESET_GAP. An arbitration reset gap event has been detected on the bus. This event is needed for link layers that implement the retry A/B protocol.
 - SUBACTION_GAP. A subaction gap event has been detected on the bus.

5.2 Backplane physical connection specification

Within the backplane environment, the Serial Bus is implemented with a pair of signals: Strb and Data. The topology is a simple pair of bused signals, as shown in figure 5.1.



NOTES

- 1—On a given bus, there may be as many as 63 nodes.
- 2—There is no restriction on the distribution of nodes throughout modules on the bus.
- 3—If more than one node occupies a module, they shall share the same transceivers.

Figure 5.1—Backplane topology

The backplane environment can be implemented with a number of different interface technologies. These include, but are not limited to: TTL for industry-standard transistor-transistor logic, BTL for backplane transceiver logic as defined by IEEE Std 1194.1-1991, and ECL for emitter-coupled logic.

In addition to the requirements specified by the application environment, the physical media of the Serial Bus shall meet the requirements defined for media attachment, media signal interface, and media signal timing. Timing requirements shall be met over the ranges specified in the application environment. These include temperature ranges, voltage ranges, and manufacturing tolerances.

5.2.1 Media attachment

5.2.1.1 Distribution of nodes

The number of modules on a backplane shall not exceed the maximum number specified for the application environment (i.e., the host backplane). The number of nodes on the backplane shall not exceed 63, although there is no restriction on the distribution of these nodes throughout the modules on the bus. If more than one node occupies a module, they shall share the same transceivers.

NOTE — Media signal timing is determined assuming that there is only one pair of Serial Bus transceivers per module on a given bus. If modules supported more than one pair of transceivers, the additional capacitive loading on the bus would result in higher bus propagation delays. Arbitration and data transmission, however, require that the bus propagation delay be within a certain fixed value.

As long as the arbitration and bus synchronization timing requirements are met (see 5.2.4.2), it is unnecessary to specify the number of modules on the backplane, the pitch of the modules, or the load that each module presents to the backplane. Nonetheless, these characteristics may still be defined within the specification (or profile) for a particular application environment.

All nodes shall have unique numbers or node addresses. Refer to 5.4.2.1.

5.2.1.2 Fault detection and isolation

It is recommended that modules be designed to support fault detection and that single points of failure between the serial bus and the host backplane bus be kept to a minimum. Within a module, it is recommended that the serial bus transceiver circuitry be packaged separately from the parallel backplane circuitry. Within the backplane chassis, it is recommended that termination impedances and voltage supplies be packaged separately where possible.

A serial bus backplane shall be capable of operation with modules unpowered or removed. A disabled or unpowered module shall not affect the operation of the bus.

5.2.1.3 Live insertion

Modules may support live insertion, minimizing glitches that might occur upon insertion of a board into an active backplane. The implementation of such insertion shall in no way introduce incompatibilities, under normal operating conditions, between interface circuits supporting live insertion and interface circuits that do not.

Depending upon the application environment, modules implementing Serial Bus may be required to support live insertion.

5.2.2 Media signal interface

The backplane media signal interface consists of two single-ended signal interfaces, one for Data and one for Strb.

If the Serial Bus is intended to accompany a standardized parallel bus, it is recommended that the media signal interface of the Serial Bus be similar to this bus. It is the intention of this standard that the transceivers, terminations, and other physical parameters be the same as those required for the control lines of the host bus (e.g., FutureBus+ profile characteristics or VMEbus electrical specifications). In this case, the specification for the host bus should be referenced for a description of the media signal interface.

If there is no host bus or the host bus does not adequately address the physical requirements for the serial bus, then the choice of interface technology is left to the implementor. Care should be taken with the engineering of the backplane to ensure proper performance of the bus. Requirements for bus synchronization and propagation delay (see 5.2.4.2) shall be met to ensure that nodes are able to arbitrate properly. Attention should be given to backplane and transceiver skew characteristics to ensure proper operation of the data-strobe encoding described in 5.3.1. Attention should also be given to signal transition times to minimize noise coupling.

This standard specifies parameters for the use of certain technologies (i.e., TTL, BTL, and ECL), but it does not prohibit the use of other technologies. It is not appropriate for this standard to specify a particular technology for a Serial Bus application.

5.2.2.1 Definition of logic states

Drivers shall assert the bus to indicate a “1” logic state or release the bus to indicate a “0” logic state. To assert the bus, a driver sinks current. To release the bus, drivers are tri-stated or turned off, allowing the bus signal to be pulled to the termination voltage of the bus.

NOTE — This typically results in a logical inversion of signals on TTL and BTL buses. Signals on ECL buses typically are not inverted.

All drivers shall operate in a “wired-or” or “open-collector” mode during arbitration. Drivers may operate in a “totem pole” mode during data packet and acknowledge transfers. In this mode, a driver may “drive” the bus into its released state in order to decrease the rise time of the bus signal (referred to as a “rescinding release” with TTL technology).

5.2.2.2 Bit rates

Data transmission and reception shall occur at 24.576 Mbit/s (± 100 ppm) for TTL, and 49.152 Mbit/s (± 100 ppm) for BTL and ECL. Regardless of the interface technology, arbitration occurs at the same rate (refer to 5.2.4.3).

5.2.2.3 Transition times

The transition times of the transceivers shall comply with the values indicated within the media signal timing (tables, and 5-6). The minimum detectable pulse width of receivers shall be less than the edge separation specified in table.

5.2.2.4 Noise rejection

Receivers may have noise filters that reject pulses. Filters should reject pulses that are 3 ns or less for TTL and 1.5 ns or less for BTL, measured at the center of the receiver input threshold.

5.2.3 Media signal timing

5.2.3.1 Backplane transmit data timing

The backplane bus signals, as they appear from the output of the transmitters and onto the backplane media, shall be within the constraints outlined in figure 5.2. Different data rates are supported for the different backplane technologies. Rise and fall times for the bus signals are measured from 10% to 90%. Slew rates are specified to ensure minimum highfrequency noise coupling during signal transitions. Edge separation is the minimum required time between any two consecutive transitions of the bus signals, whether they be transitions on the same signal or transitions on the two separate signals. Edges are measured at the center of the receiver input threshold. A minimum edge separation is required to ensure proper operation of the data-strobe bit level encoding mechanism.

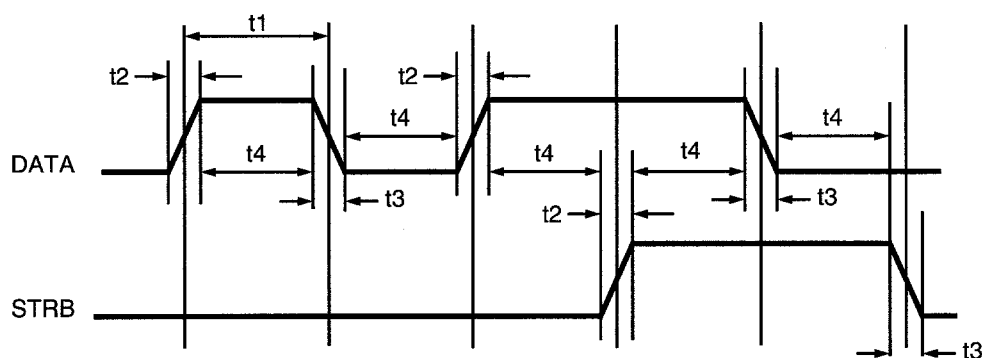


Figure 5.2—Backplane transmit data timing

Table 5.2—Backplane transmit data timing

		TTL	BTL	ECL
	Data rate	S25	S50	S50
t1	Bit cell period	Approximately 40.690 ns (1/24.576 Mbit/s ± 100 ppm)	Approximately 20.345 ns (1/49.152 Mbit/s ± 100 ppm)	Approximately 20.345 ns (1/49.152 Mbit/s ± 100 ppm)
t2	Rise time	1.2 ns min 3 ns max (from 1.0 to 2.0 V [*])	5 ns max	2 ns min 5 ns max
t3	Fall time	1.2 ns min 3 ns max (from 1.0 to 2.0 V [*])	5 ns max	2 ns min 5 ns max
	Slew rate		0.5 V/ns max (from 1.3 to 1.8 V [†])	
t4	Tx edge separation	33 ns min	15 ns min	15 ns min

*Measured with a $25 \Omega \pm 0.25 \Omega$ load terminated to $1.5 \text{ V} \pm 0.015 \text{ V}$.

†Measured with a 16.5Ω load to 2.1 V .

5.2.3.2 Backplane receive data timing

The receiver typically uses the transitions on the incoming bus signals Data_Rx and Strb_Rx to derive a clock at the code bit frequency that extracts the NRZ signal on Data_Rx. This clock may be derived by performing an exclusive-OR of Data_Rx and Strb_Rx.

The bus signals, as they appear from the backplane media and into the receivers, shall be within the constraints outlined by figure 5.3. The minimum required edge separation is reduced to allow for backplane and receiver skew.

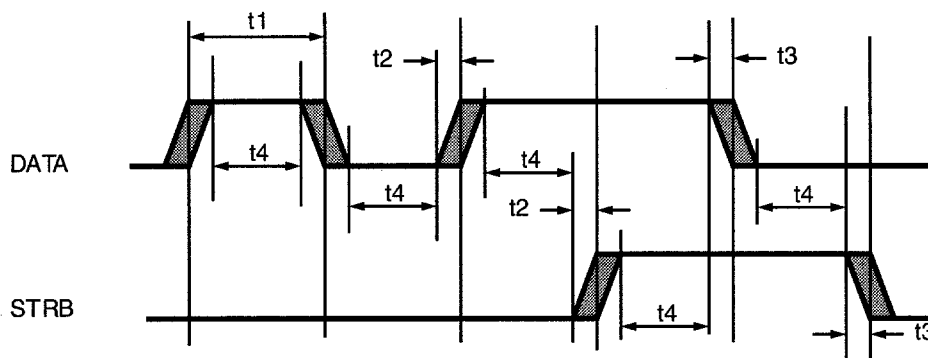
**Figure 5.3—Backplane receive data timing**

Table 5.3—Backplane receive data timing

		TTL	BTL	ECL
	Data rate	S25	S50	S50
t1	Bit cell period	Approximately 40.690 ns (1/24.576 Mbit/s ± 100 ppm)	Approximately 20.345 ns (1/49.152 Mbit/s ± 100 ppm)	Approximately 20.345 ns (1/49.152 Mbit/s ± 100 ppm)
t2	Rise time	1.2 ns min 3 ns max (from 1.0 to 2.0 V [*])	5 ns max	2 ns min 5 ns max
t3	Fall time	1.2 ns min 3 ns max (from 1.0 to 2.0 V [*])	5 ns max	2 ns min 5 ns max
	Slew rate		0.5 V/ns max (from 1.3 to 1.8 V [†])	
t4	Rx edge separation	26 ns min	9 ns min	9 ns min

*Measured with a $25 \Omega \pm 0.25 \Omega$ load terminated to $1.5 \text{ V} \pm 0.015 \text{ V}$.

†Measured with a 16.5Ω load to 2.1 V .

5.2.3.3 Backplane and transceiver skew

The skew introduced between Data_Rx and Strb_Rx by the Serial Bus backplane and transceiver packages shall be no greater than the values mentioned in table 5.4. This table gives the maximum allowable skew for each technology. Conformance to these requirements is necessary to ensure Tx and Rx edge separations. Calculations for edge separation and skew margin are contained in annex D

Table 5.4—Maximum transceiver package and bus skew

	TTL (in ns)	BTL (in ns)	ECL (in ns)
Tx package skew	5	3	3
Rx package skew	5	3	3
Backplane skew	7	6	6

5.2.4 Backplane PHY timing

Correct operation of the backplane PHY is dependent upon a number of timing requirements. Unlike the cable PHY specification (which defines a number of physical layer timing constants, see 4.3.5), the backplane PHY specification contains timing requirements but does not define them as constants. These requirements are indicated in table 5.5.

Table 5.5—Backplane physical layer timing requirements

Timing requirement	Cross-Reference
arbitration clock rate	5.2.4.1
node synchronization	5.2.4.2
bus propagation	5.2.4.2
arbitration bit timing	5.2.4.3
BUS_RESET time	5.3.3
BUS_IDLE time	5.3.3
DATA_PREFIX time	5.3.3
DATA_END time	5.3.3
gap timing	5.3.3
MAX_BUS_OCCUPANCY	5.4.2.1
ACK_RESPONSE_TIME	5.4.2.2
NOMINAL_CYCLE_TIME	4.3.5

NOTE — The cable PHY specification depends upon the use of C code (using C++ syntax) to describe certain operations. Consequently, it is necessary to define timing “constants” that can be referenced by the code. The backplane PHY specification does not use C-code descriptions. Consequently, it is not necessary to label timing requirements as constants. Nonetheless, many of the timing requirements that exist within the cable environment also exist within the backplane environment. Although similar timing requirements exist, they may have different values associated with them. The cable PHY specification should be referenced for values that are referenced in other clauses of this standard, but do not appear in the backplane PHY specification (in particular, values such as NOMINAL_CYCLE_TIME are defined in 4.3.5).

5.2.4.1 Arbitration clock rate

The arbitration clock rate is used to define a number of timing requirements within the backplane physical layer. It is 49.152 MHz \pm 100 ppm, regardless of the backplane interface technology. Note that this is not necessarily equal to the data bit rate (e.g., for TTL backplanes, the data bit rate is 24.576 Mbit/s \pm 100 ppm).

5.2.4.2 Bus synchronization and propagation delay

To ensure proper operation of the arbitration mechanism, all nodes participating in arbitration shall be synchronized to within a specified time period. This requirement allows all nodes to arbitrate at approximately the same time.

To achieve synchronization, nodes preparing to arbitrate shall sample the bus until an idle condition is detected. The bus becomes idle once four arbitration clock times (approximately 81.38 ns) have occurred without Data_Rx or Strb_Rx being asserted.

A node waiting to arbitrate for the bus shall detect an idle bus within 43.345 ns of the idle condition occurring at the receiver inputs. This time assumes a maximum propagation delay through the receiver (8 ns), a maximum input delay and setup time for the decision logic (15 ns), and a maximum synchronization delay of one arbitration clock period (approximately 20.345 ns). If Strb has already been asserted by another node beginning to arbitrate, all other nodes shall detect that the bus has been asserted within 43.345 ns of the arrival of the asserted signal at their receiver inputs.

Once a node that is waiting to arbitrate detects that the appropriate gap has occurred (see 5.3.3), it shall assert Strb_Tx within 43.345 ns. This time assumes a maximum output delay for the decision logic (15 ns), a maximum propagation delay through the driver (8 ns), and one state machine clock period (approximately 20.345 ns).

In order to guarantee proper arbitration timing, it is necessary to specify a maximum bus propagation delay. The maximum time required for a signal to propagate from one end of the backplane serial bus to the other end (a one-way propagation delay) shall be 18 ns or less.

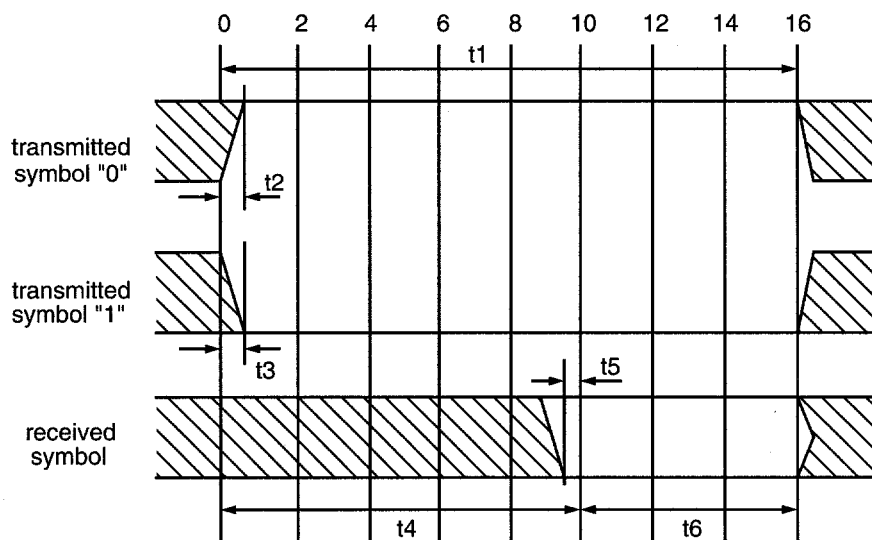
5.2.4.3 Arbitration bit timing

A node gains access to the serial bus by using the arbitration process. This is done in response to a PH_ARB.request from the link layer. To arbitrate for the bus, a node samples Strb_Rx and Data_Rx to determine if the bus is active. If the bus is inactive for a subaction gap or an arbitration reset gap (for an asynchronous transfer) or an acknowledge gap (for an isochronous transfer) the node can begin the arbitration process. It asserts Strb_Tx to indicate the beginning of the arbitration process. The assertion of Strb_Tx also ensures that long strings of “0” arbitration bits are not interpreted as gaps between packets.

At the same time that Strb_Tx is asserted, the node begins to transmit its arbitration sequence (see 5.4.2.1) on Data_Tx. The most significant bits are transmitted first (see 1.6.3). Each arbitration bit in the sequence has a duration of 16 arbitration clock times (approximately 325.6 ns). If the arbitration bit to be transmitted is a “1,” the node asserts Data_Tx. If the arbitration bit to be transmitted is a “0,” the node releases the Data_Tx, waits 10 arbitration clock times (approximately 203.4 ns), and then samples Data_Rx. If the bus is asserted at this time, the node has lost the arbitration contest. It sends a PH_ARB.confirmation(LOST) to the link layer, and then drops out of the arbitration process and wait for the next appropriate gap to compete for the bus. Regardless of whether arbitration is won or lost, the node shall not cause a transition on Strb or Data (i.e., release or assert the bus) until six more arbitration clock times have passed. This ensures that all participants in the arbitration process have had a chance to sample that arbitration bit in the arbitration sequence.

An arbitrating node continues to transmit its arbitration sequence until it loses arbitration to a higher priority node or it successfully completes its arbitration sequence. If a PHY layer has successfully transmitted each of its 10 arbitration bits and still holds the bus, it sends a PH_ARB.confirmation(WON) to the link layer. The PHY layer also initiates the transmission of a PH_DATA.request(DATA_PREFIX) on the bus (using the transitions in 5.4.2.1) for a minimum amount of time before sending the first PH_CLOCK.indication to the link layer. The PHY layer continues to assert PH_DATA.request(DATA_PREFIX) on the bus until the first PH_DATA.request is received from the link layer.

The backplane bus signals, as they appear from the output of the transmitters as well as on the inputs of the receivers, shall be within the constraints outlined in figure 5.4. The arbitration bit period is the same regardless of the backplane interface technology. The arbitration bit period (t_1) is divided into the sample time (t_4) and hold time (t_6). The sample time allows enough time for the bus to settle after arbitrating nodes have placed an arbitration bit on the bus. The hold time enables all nodes on the bus to sample the arbitration bit before transitions or glitches can occur on the bus. Rise and fall times for the bus signals are measured from 10% to 90%. Slew rates, if specified, ensure minimum high-frequency noise coupling during signal transitions.



NOTE— All signals are displayed with a “wired or” inversion. Signals are typically inverted on TTL and BTL buses. Signals on ECL buses typically are not inverted.

Figure 5.4—Arbitration bit timing

Table 5.6—Arbitration bit timing

		TTL	BTL	ECL
	Data rate	S25	S50	S50
t1	Arbitration bit period	16 arbitration clock times (approximately 325.6 ns)	16 arbitration clock times (approximately 325.6 ns)	16 arbitration clock times (approximately 325.6 ns)
t2	Rise time	4 ns min 30 ns max	5 ns max	2 ns min 5 ns max
t3	Fall time	4 ns min 10 ns max	5 ns max	2 ns min 5 ns max
	Slew rate		0.5 V/ns (from 1.3 to 1.8 V)*	
t4	Sample time	10 arbitration clock times (approximately 203.4 ns)	10 arbitration clock times (approximately 203.4 ns)	10 arbitration clock times (approximately 203.4 ns)
t5	Setup time	5 ns min	5 ns min	5 ns min
t6	Hold time	6 arbitration clock times (approximately 122.0 ns)	6 arbitration clock times (approximately 122.0 ns)	6 arbitration clock times (approximately 122.0 ns)

*Measured with a 16.5 Ω load to 2.1 V.

More information on arbitration bit timing is contained in annex D Refer to 5.3.4 for a description of the arbitration sequence.

5.3 Backplane PHY facilities

5.3.1 Coding

Peer physical layer entities on the bus communicate via NRZ data. The NRZ data is transmitted and received as Data and is accompanied by a strobe signal, Strb. This strobe signal changes state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on either Data or Strb. A clock that transitions every bit period can be extracted by performing an exclusive OR of Data_Rx and Strb_Rx, as is shown in figure 5.5.

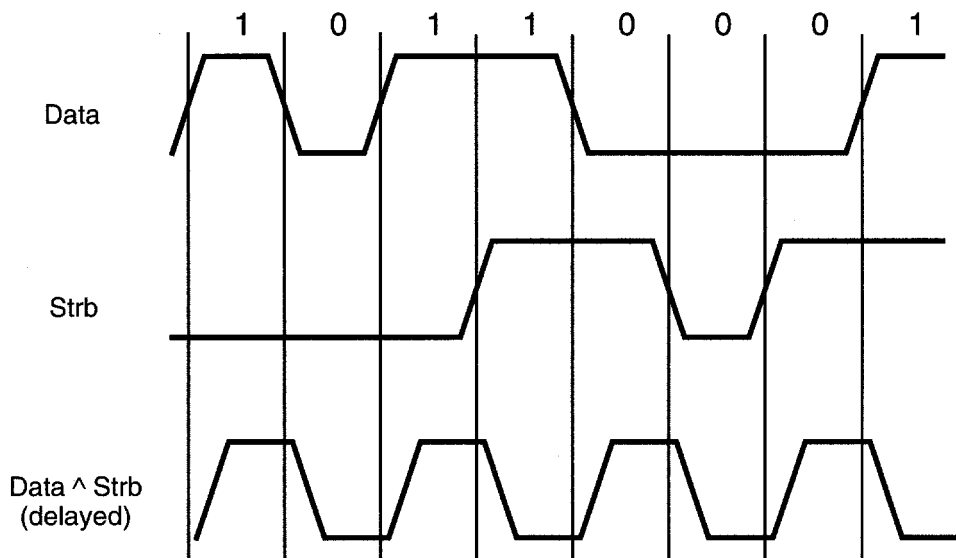


Figure 5.5—Data-strobe coding

The primary rationale for use of this transmission code is to improve the transmission characteristics of information to be transferred across the serial bus. In particular, the code ensures that transitions occurring on Data_Rx and Strb_Rx are approximately one bit period apart. This results in an increase in skew tolerance that could not be obtained with a clocked NRZ format.

The procedure for encoding data during transmission of a packet is described in 5.4.2.1, and decoding during reception of a packet is described in 5.4.2.2.

5.3.2 Backplane PHY signals

Backplane PHYs can send the following signals:

- | | |
|-----------|---|
| BUS_RESET | Both Data_Tx and Strb_Tx are asserted for at least 352 arbitration clock times (approximately 7161.4 ns), and at most 384 arbitration clock times (approximately 7812.48 ns). This signal is sent in response to a PH_CONTROL.request(BUS_RESET) from the link layer (see 5.1.1.1). |
| ARBITRATE | An arbitrating node asserts Strb_Tx while transmitting the arbitration sequence on Data_Tx. Arbitration bit timing is described in 5.2.4.3. Arbitration occurs in response to a PH_ARB.request from the link layer (see 5.1.2.1). |
| PACKET | A node transfers a request or a response packet by transmitting NRZ Data on Data_Tx and a strobe signal on Strb_Tx. This strobe signal changes state whenever two consecutive |

NRZ data bits are the same, ensuring that a transition occurs on either Data_Tx or Strb_Tx. This bit level encoding method is described in 5.3.1. Data bits occur in response to a PH_DATA.request(DATA_ONE or DATA_ZERO) from the link layer (see 5.1.3.2).

DATA_PREFIX	Before it transmits a packet, a node can hold the bus by asserting Data_Tx and deasserting (or driving into the unasserted state) Strb_Tx for at least four arbitration clock times (approximately 81.38 ns), and at most 160 arbitration clock times (approximately 3255.2 ns). The signal transitions required for this are defined in 5.4.2.1. This signal occurs in response to a PH_DATA.request(DATA_PREFIX) from the link layer (see 5.1.3.2). This signal can also be initiated by the PHY layer after arbitration is successfully completed (see 5.2.4.3).
DATA_END	After it transmits a packet, a node signals the release of the bus by asserting Strb_Tx and deasserting (or driving into the unasserted state) Data_Tx for at least four arbitration clock times (approximately 81.38 ns), and at most 16 arbitration clock times (approximately 325.52 ns). The signal transitions required for this are defined in 5.4.2.1. This signal occurs in response to a PH_DATA.request(DATA_END) from the link layer (see 5.1.3.2).

Backplane PHYs can receive the following signals:

BUS_RESET	Both Data_Rx and Strb_Rx are asserted for more than 320 arbitration clock times (approximately 6510.4 ns). The duration of the reset signal distinguishes it from a sequence of ones transmitted during arbitration.
BUS_IDLE	Both Data_Rx and Strb_Rx are unasserted for at least four arbitration clock times (approximately 81.4 ns). The amount of time that the bus is unasserted (idle) distinguishes an acknowledge gap, a subaction gap, and an arbitration reset gap.
ARBITRATE	Strb_Rx is asserted as the arbitration sequence is received on Data_Rx. Arbitration bit timing is described in 5.2.4.3.
PACKET	NRZ Data is received on Data_Rx and is accompanied by a strobe signal on Strb_Rx. This strobe signal changes state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on either Data_Rx or Strb_Rx. This bit level encoding method is described in 5.3.1. Each bit in the packet results in a PH_DATA.indication(DATA_ONE or DATA_ZERO) to the link layer (see 5.1.3.3).
DATA_PREFIX	Data_Rx is asserted and Strb_Rx is unasserted for at least four arbitration clock times. This indicates that another node is holding the bus before transmitting a packet. The receipt of this signal results in a PH_DATA.indication(DATA_PREFIX) to the link layer (see 5.1.3.3).
DATA_END	Strb_Rx is asserted and Data_Rx is unasserted for at least four arbitration clock times. This indicates that another node has released the bus after the transmission of a packet. The receipt of this signal results in a PH_DATA.indication(DATA_END) to the link layer (see 5.1.3.3).

5.3.3 Gap timing

A gap is a period of time during which the bus is idle (Data_Rx and Strb_Rx are unasserted). There are three types of gaps:

- a) Acknowledge gap—Appears between the end of a packet and an acknowledge, as well as between isochronous transfers. A node shall detect the occurrence of an acknowledge gap after the bus has been in an unasserted state for four arbitration clock times (approximately 81.38 ns), but it shall not assert the bus until a total of eight arbitration clock times (approximately 182.76 ns) have occurred. This requirement ensures that a node is given adequate time to detect the acknowledge gap before the bus is asserted by another node

(upon detecting an acknowledge gap). This includes the minimum time required to detect a BUS_IDLE (four arbitration clock times), as well as the maximum delay between the arbitration state machines within any two nodes on the bus (another four arbitration clock times).

- b) Subaction gap—Appears before asynchronous transfers within a fairness interval. A node shall detect the occurrence of a subaction gap after the bus has been in an unasserted state for at least 16 arbitration clock times (approximately 325.52 ns), but it shall not assert the bus until a total of 20 arbitration clock times (approximately 406.9 ns) have occurred. This requirement ensures that a node is given adequate time to detect the subaction gap before the bus is asserted by another node (upon detecting a subaction gap). The duration of the subaction gap ensures that another node asserting the bus after an acknowledge gap will have been detected by this time.
- c) Arbitration reset gap—Appears before asynchronous transfers when the fairness interval starts. A node shall detect the occurrence of an arbitration reset gap after the bus has been in an unasserted state for at least 28 arbitration clock times (approximately 569.66 ns), but it shall not assert the bus until a total of 32 arbitration clock times (approximately 651.04 ns) have occurred. This requirement ensures that a node is given adequate time to detect the arbitration reset gap before the bus is asserted by another node (upon detecting an arbitration reset gap). The duration of the arbitration reset gap ensures that another node asserting the bus after a subaction gap or an acknowledge gap will have been detected by this time.

If a node is waiting for the occurrence of a particular gap, and the bus has become idle for the specified time (e.g., 32 arbitration clock times for an arbitration reset gap), the node shall detect the gap and assert the bus within the time constraints described in 5.2.4.2. These constraints ensure that an asserted signal shall have propagated through the decision/transceiver circuitry of the node and onto the bus soon enough to allow arbitration to occur properly.

More information on gap timing is available in annex D

5.3.4 Arbitration sequence

5.3.4.1 Arbitration number

The arbitration sequence uses a unique arbitration number for each module. This 6-bit number is the same as the physical_ID of the node. If less than 6 bits are provided for the arbitration number, they shall occupy the most significant bits of the arbitration number. The remaining bits shall be zero-filled. The most significant bits are transmitted first (see 1.6.3).

NOTE — If the Serial Bus is contained within a host backplane, it is expected that the arbitration number (i.e., physical_ID) will be set by the host backplane at power-up (e.g., with a built-in slot identifier or configuration mechanism).

It is recommended that this number be software programmable to facilitate testing and to allow for consistent system operation and repeatability.

5.3.4.2 Priority

Within the arbitration sequence, the arbitration number is preceded by 4 bits that define a priority level. The method by which priority is assigned is to be determined by the system integrator, with two exceptions: the lowest priority (all zeros) is reserved for fair arbitration and the highest priority (all ones) is reserved for cycle start requests. This allows 14 priority levels to be used for the urgent arbitration process.

The use of an urgent priority class allows nodes to be granted a larger portion of the bandwidth on the bus. High-priority nodes are granted the bus before lower priority nodes during urgent allocation of the bus, allowing such nodes to be granted more bandwidth.

NOTE — If used properly, deterministic latency algorithms (e.g., rate monotonic scheduling) may be used to assign priority to transfers so that they would be made within a predetermined amount of time. It is beyond the scope of this standard to specify the use of such algorithms.

In order to guarantee forward progress, the lowest priority level is reserved for fair arbitration. This allows all nodes arbitrating with this priority level to be allowed one fair access to the bus each fairness interval. For fair arbitration, the value of the arbitration number has a minimal impact on the allocation of the bus. Although nodes with higher arbitration numbers will be granted the bus sooner, there is only a small decrease in latency.

The 4-bit priority field is not used in isochronous arbitration. When arbitrating for an isochronous transfer, the priority field is zero-filled.

5.3.4.3 Format of arbitration sequence

The format in figure 5.6 shall be used for the arbitration sequence.

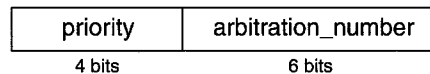


Figure 5.6—Arbitration sequence

- Each module on the backplane shall have a unique 6-bit arbitration number that is equal to the `physical_ID` of the node.
- The arbitration number shall be preceded by 4 bits of priority (refer to 5.4.2.1). The most significant bit of the priority field shall be transmitted first. The least significant bit of the priority field shall be followed by the most significant bit of the arbitration number (refer to 5.4.2.1).
- Dynamic assignment of priority shall be accommodated.
- The lowest priority level (all zeroes) shall be reserved for fair arbitration, and the highest priority level (all ones) shall be reserved for the identification of the cycle start packet.

5.4 Backplane PHY operation

The operation of the backplane PHY can best be understood with reference to the architectural diagram shown in figure 5.7. This diagram shows the two primary functions performed by the physical layer: arbitration and bit-level data transmission and reception.

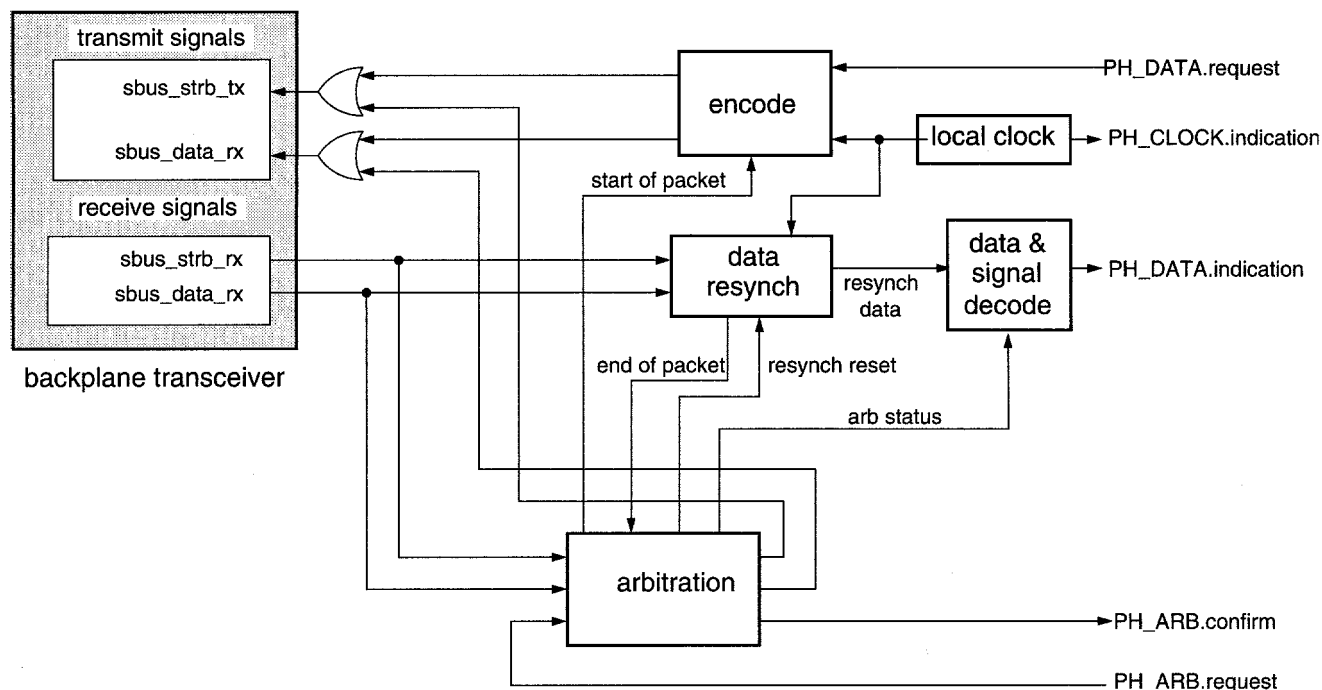


Figure 5.7—Backplane PHY architecture

The main controller of the backplane physical layer is the block labeled “arbitration,” which responds to arbitration requests from the link layer (PHY_ARB.request) and indicates changes in the state of the bus. It provides the management and timing signals for transmitting and receiving packets. It also provides the bus reset and configuration functions. The operation of this block is described in 5.4.1.

The “encode” block generates the appropriate strobe signal for the transmitted data. Its operation is described in 5.4.2.1.

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 has to be able to compensate for a difference of 200 ppm over the maximum packet length of 87.24 μ s (256 byte isochronous packet at 24.576 Mbit/s). The operation of this block is described in 5.4.2.2.

The “data and signal decode” block provides a common interface to the link layer for both packet data and arbitration signals (data, gaps, and bus reset indicators). The operation of this block is also described in 5.4.2.2.

An example of a backplane physical implementation is contained in annex F. This annex describes how the physical layer of Serial Bus *may* be implemented in the backplane environment.

5.4.1 Arbitration

Unless a node is using immediate arbitration to access the bus (in which case there is no contention for the bus), it is possible that more than one node may attempt to access the bus at a given time. Consequently, it is necessary for a node to arbitrate for the bus in order to gain access to the bus.

NOTE — A node uses immediate arbitration to send an acknowledge. Since there is no contention for the bus in this case, arbitration is not necessary (a node that is transmitting an acknowledge does not arbitrate for the bus, but merely waits for an acknowledge gap to be detected before it begins transmission). If a node is attempting to gain access to the bus without using immediate access, it has to first arbitrate for the bus.

Arbitration occurs in response to a PHY arbitration request from the link layer. Nodes begin arbitrating once the bus has become idle for a predetermined amount of time (the appropriate gap indication occurs). Once this happens, nodes begin a bit-by-bit transmission of their arbitration sequence. Refer to 5.3.4 for a description of the arbitration sequence. Refer to 5.4.2.1 for a description of the bit-by-bit process of asserting and sampling the bus during arbitration.

A node can obtain access to the bus in a limited number of ways. Because some arbitration classes allow nodes to begin arbitration before others, nodes arbitrating with certain arbitration classes may detect that the bus is busy before they can begin to arbitrate. In this way, certain arbitration classes can be bypassed (e.g., fair and urgent nodes will not get a chance to arbitrate if another node is sending an acknowledge or if it is arbitrating for an isochronous transfer).

The backplane environment supports the fair, urgent, cycle_master, isochronous, and immediate arbitration classes. These are described in the following clauses. Refer to 5.1.2.1 for a description of the PH_ARB.request.

5.4.1.1 Fairness intervals

NOTE — This clause is a partial reiteration of 3.7.2.

The fairness protocol is based on the concept of a fairness interval. A fairness interval consists of one or more periods of bus activity separated by short idle periods called subaction gaps and is followed by a longer idle period known as an arbitration reset gap. At the end of each gap, bus arbitration is used to determine the next bus owner. This concept is shown in figure 5.8.

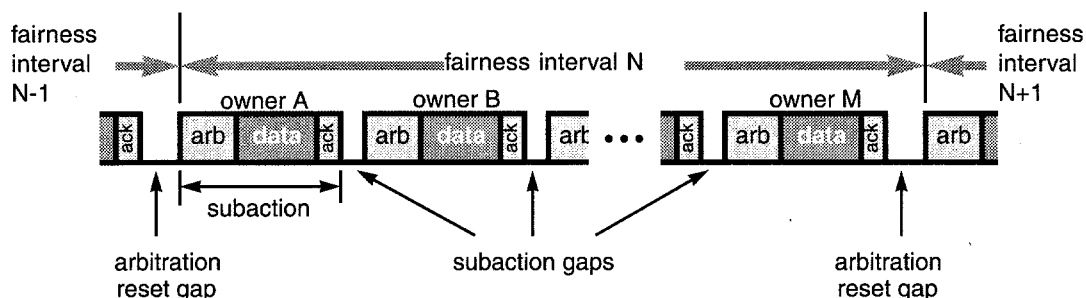


Figure 5.8—Fairness interval

The implementation of the fair arbitration protocol is defined in terms of these fairness intervals, as is discussed in the following clauses.

5.4.1.2 Fair arbitration

NOTE — This clause is a partial reiteration of 3.7.2.

When using this arbitration class, an active node can send an asynchronous packet exactly once each fairness interval. Once a subaction gap is detected, a node can begin arbitration if its arbitration_enable signal is set. The arbitration_enable signal is set at the beginning of the fairness interval and is cleared when the node successfully accesses the bus through fair arbitration. This disables further fair arbitration attempts by that node for the remainder of the fairness interval. In the absence of urgent nodes, a fairness interval ends once all of the nodes attempting fair arbitration have successfully accessed the bus. At this time, all of the fair nodes have their arbitration_enable signals

reset and cannot arbitrate for the bus. The bus remains idle until an arbitration reset gap occurs. Once this happens, the next fairness interval begins. All of the nodes set their arbitration_enable signal and can begin to arbitrate for the bus. This process is illustrated in figure 5.9.

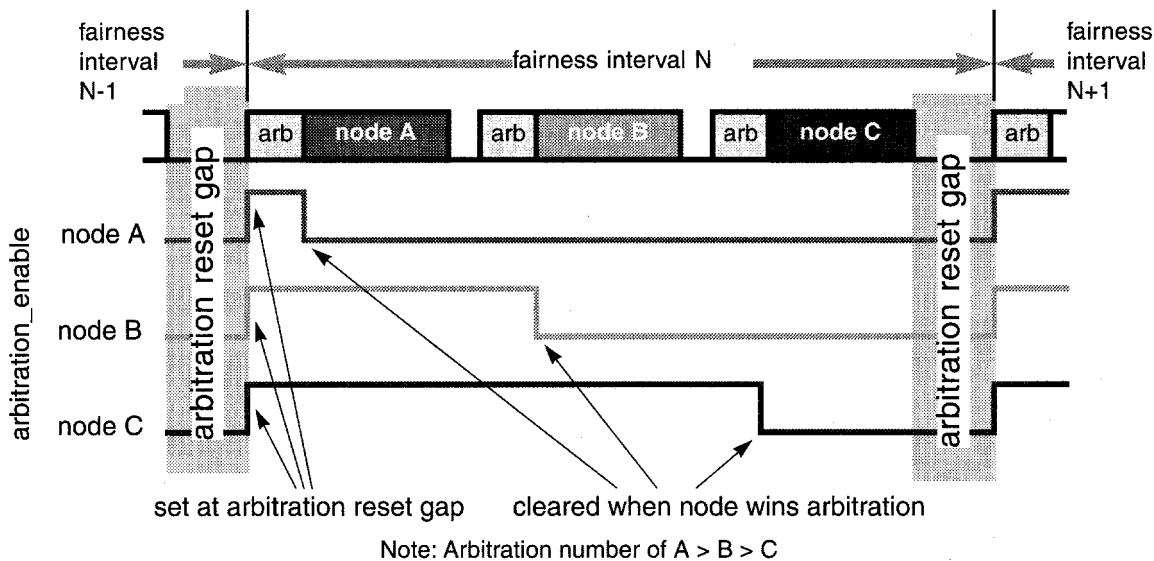


Figure 5.9—Fair arbitration

Note that a node sending a concatenated subaction (see 3.6.2.3) does not reset its arb_enable bit.

5.4.1.3 Urgent arbitration

NOTE — This clause is a reiteration of 3.7.4.2.

The backplane environment enhances the fair algorithm by splitting access opportunities among nodes based on two priority classes: “fair” and “urgent.” Nodes using an “urgent” priority may use up to three-fourths of the access opportunities, with the remaining equally shared among nodes using the “fair” priority. All nodes are required to implement the fair priority class, while the urgent priority class is optional. Packets are labeled as “urgent” if that priority class was used.

The fair/urgent allocation uses the same fairness interval described in 3.7.2, but it accompanies the arbitration_enable flag with an “urgent_count.” The fair/urgent method works as follows:

- If the bus is idle for longer than an arbitration reset gap, a fairness interval begins and all nodes shall set their “arbitration_enable” flags, while nodes implementing urgent priority shall set their “urgent_count” to three.
- A node that is waiting to send a packet using the fair priority class shall begin arbitrating after detecting a subaction gap as long as its arbitration_enable flag is set. If its arbitration_enable flag is cleared, it shall wait for an arbitration reset gap before it begins arbitrating. When such a node wins an arbitration contest, it sends a packet without the “urgent” label and its arbitration_enable flag is cleared.
- A node that is waiting to send a packet with urgent priority shall begin arbitrating after detecting a subaction gap if its urgent_count is nonzero. If its urgent_count is zero, it shall wait for an arbitration reset gap before it begins arbitrating. Whenever such a node wins an arbitration contest, it sends a packet with the “urgent” label.
- A node implementing urgent priority shall set its urgent_count to three whenever an unlabeled (i.e., fair) packet is transmitted or received. This includes received packets that are addressed to other nodes.

- e) A node shall decrement its `urgent_count` whenever a packet with the “urgent” label is transmitted or received. This includes received packets that are addressed to other nodes. This ensures that there will be at most three “urgent” packets for every “fair” packet. (This does not ensure that every node using urgent priority will obtain the bus three times each fairness interval. The node arbitrating with the highest priority will always obtain the bus before other nodes arbitrating with an urgent, but lower, priority.)

In the presence of urgent nodes, a fairness interval ends after the final fair node and up to three remaining urgent nodes have successfully accessed the bus. Since all fair nodes now have their `arbitration_enable` signals reset and all urgent nodes have their `urgent_count` decremented to zero, none of the nodes can access the bus. The bus remains idle until an arbitration reset gap has occurred, re-enabling arbitration on all nodes and starting the next fairness interval. This process is illustrated in figure 5.10, which illustrates a situation where there are three nodes arbitrating for the bus with `physical_ID`s such that A has the highest, B is in the middle, and C has the lowest, with nodes A and C using fair priority and B using urgent.

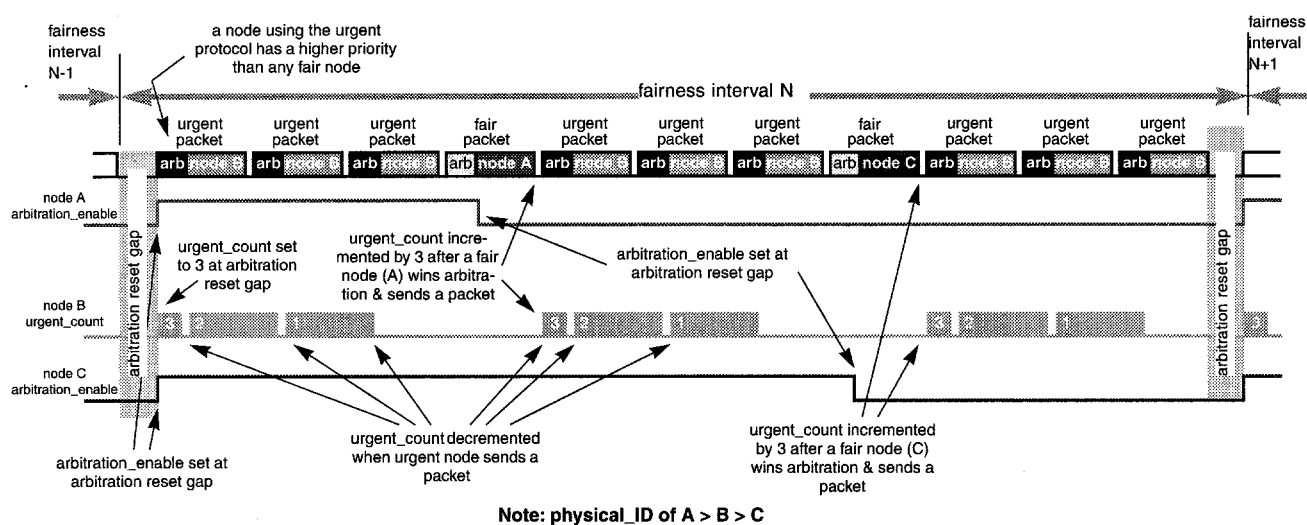


Figure 5.10—Urgent arbitration

In the backplane environment, the natural priority is the concatenation of the 4-bit urgent priority level with the `physical_ID`. This results in the following:

- A node using the urgent priority will always win an arbitration contest over all nodes using the fair priority.
- The node using the highest priority level will win the arbitration contest.
- If more than one node uses the highest priority level, then the one with the highest `physical_ID` will win.

5.4.1.4 Arbitration by the cycle_master

This arbitration class is used by the `cycle_master` when it needs to arbitrate for the transmission of a `cycle_start` packet. It is similar to the urgent arbitration class, except that the priority field is defined to be all ones (see 5.4.2.1). Arbitration begins once a subaction gap is detected, regardless of the state of the `arbitration_enable` signal or the `urgent_count`.

5.4.1.5 Isochronous arbitration

This arbitration class is used by nodes arbitrating to send isochronous packets. Arbitration begins once an acknowledge gap is detected, regardless of the state of the `arbitration_enable` signal or the `urgent_count`. Because an

acknowledge gap is shorter than an arbitration reset gap and a subaction gap, nodes arbitrating with this class will win the bus before nodes waiting to send fair, urgent, or cycle_master packets. The priority field is not used in this arbitration class (see 5.4.2.1). See 3.6.4 for more information regarding isochronous arbitration.

5.4.1.6 Immediate arbitration

This arbitration class is used by nodes sending an acknowledge to a received packet. Transmission of the acknowledge (beginning with a DATA_PREFIX; refer to 5.4.2.1) occurs as soon as an acknowledge gap is detected. This arbitration class is referred to as “immediate” because an arbitration sequence is not transmitted to obtain access to the bus (i.e., the node does not actually arbitrate for the bus).

5.4.2 Backplane environment packet transmission and reception

Packet transmission and reception are synchronized to a local clock that shall be accurate within 100 ppm. The nominal data rates are either 49.152 Mbit/s or 24.576 Mbit/s for the backplane environment, depending on the particular backplane technology.

5.4.2.1 Backplane environment packet transmission

Protocols and timing for packet transmission are different than those for arbitration. Packet transmission uses the data-strobe bit level encoding method described in 5.3.1, and it is delimited by PH_DATA.request(DATA_PREFIX and DATA_END) symbols. Timing of the signals used to implement the encoding method is described in 5.2.3.1.

Once a PHY layer completes an arbitration sequence, it signals the beginning of packet transmission by putting the bus into the PH_DATA.request(DATA_PREFIX) state. Because data can be either asserted or unasserted at the end of the arbitration sequence, the PHY layer first asserts Data_Tx for one arbitration clock time (approximately 20.35 ns) to put the bus into a known state (Strb_Tx is already in a known state because it is asserted during the arbitration sequence). Strb_Tx is then deasserted to transmit the DATA_PREFIX.

If the PHY layer uses immediate arbitration to access the bus, the DATA_PREFIX does not follow an arbitration sequence (the bus has been idle for an acknowledge gap). In this case, Strb_Tx remains unasserted while Data_Tx is asserted to transmit the DATA_PREFIX.

Once the PHY layer begins transmitting a DATA_PREFIX, it shall continue its transmission for at least four arbitration clock times (approximately 81.38), and it shall begin transmission of a packet within 160 arbitration clock times (approximately 3255.2 ns).

The transmission of data within a packet occurs in response to PHY Data Requests. The link layer transmits each data bit in the packet as a PH_DATA.request(DATA_ONE or DATA_ZERO) in response to PH_CLOCK.indications from the PHY layer. The PHY layer sends PH_CLOCK.indications to the link layer for each bit to indicate the rate at which data bits are to be transmitted. The PHY layer encodes the data bits and produces the transmit signals Data_Tx and Strb_Tx. In order to provide a transition from the DATA_PREFIX state once the first bit of data is transmitted, Strb_Tx is asserted if the first bit is a DATA_ONE, or Data_Tx is deasserted if the first bit is a DATA_ZERO. This ensures that a transition will occur on either Strb or Data, and that a clock indication will be generated for the first bit of data once it is received by other nodes.

The completion of a packet transmission requires two additional bits after the last bit in the packet. The first bit is required to flush the last bit in the packet through the receiving circuit. The second bit is required to put Data_Tx and Strb_Tx into the proper state to transmit a PH_DATA.request(DATA_PREFIX or DATA_END). A node transmits a DATA_PREFIX at the end of a packet in order to hold onto the bus (with a concatenated response, this would happen after a node transmits the acknowledge and before it transmits the response packet). A node transmits a DATA_END at the end of a packet in order to signal the release of the bus. Refer to 5.3.3 for further description of these signals.

In order to remain in accordance with the data-strobe bit level encoding algorithm (see 5.3.1), only one of the two signals Data_Tx and Strb_Tx can make a transition at one time. Tables 5.7 and 5.8 indicate the proper series of transitions required to: provide the extra transition at the end of a packet; leave the bus in the proper state (DATA_PREFIX or DATA_END); and comply with the data-strobe bit level encoding algorithm. Note that the Data_Tx and Strb_Tx cannot have the same values at the end of the packet since

- a) The starting state of the bus is DATA_PREFIX (Strb is 1 and Data is 0)
- b) The number of bits in a packet *must* be even.

Table 5.7—DATA_PREFIX signal transitions after packet transmission

Signal	Signal state after last bit in packet	First transition	Second transition
Data_Tx Strb_Tx	0 1	1 1	1 0
Data_Tx Strb_Tx	1 0	1 1	1 0
NOTE — These signals typically undergo a “wire or” inversion at the output of TTL and BTL bus transceivers. The signals typically do not undergo an inversion at the output of ECL transceivers.			

Table 5.8—DATA_END signal transitions after packet transmission

Signal	Signal state after last bit in packet	First transition	Second transition
Data_Tx Strb_Tx	0 1	1 1	0 1
Data_Tx Strb_Tx	1 0	1 1	0 1
NOTE — These signals typically undergo a “wire or” inversion at the output of TTL and BTL bus transceivers. The signals typically do not undergo an inversion at the output of ECL transceivers.			

Once the transmission of a DATA_PREFIX or a DATA_END begins, it shall continue for at least four arbitration clock times (approximately 81.38 ns). This allows sufficient time for all nodes to detect that such a symbol has been transmitted. The duration of a DATA_PREFIX shall not exceed 160 arbitration clock times (approximately 3255.2 ns), and the duration of a DATA_END shall not exceed 16 arbitration clock times (approximately 325.52 ns). Within this time, the PHY layer shall either begin transmission of a packet or release the bus.

In order to ensure that transactions are completed within a certain amount of time, the link layer shall stop transmitting within a MAX_BUS_OCCUPANCY period of 100 μ s after receiving the PHY Arbitration WON confirmation. If data requests continue to be received after this time, then the PHY shall release the bus using the signal transitions for a DATA_END. In such an event, the total number of transmitted data bits shall be even.

NOTE — This assumes the longest period of bus occupancy would occur with an isochronous transmission on a TTL backplane. In such a case, this would include: a maximum-length data prefix (160 arbitration clock times), followed by a maximum-length isochronous transfer (4288 arbitration clock times on a TTL backplane), and finally a data end symbol (16 arbitration clock times). This corresponds to a total of 4464 arbitration clock times (approximately 90.820 μ s).

5.4.2.2 Backplane environment packet reception

Upon receipt of a packet, the receive signals Data_Rx and Strb_Rx are decoded using the method described in 5.3.1. The timing of these signals shall meet the requirements described in 5.2.3.2.

Each recovered data bit is sent to the link layer as a PHY Data Indication. Data bits within the packet are indicated as a PH_DATA.indication(DATA_ONE or DATA_ZERO). DATA_PREFIX and DATA_END indications are also communicated to the link layer, as well as DATA_START indications that are generated upon the reception of a packet. Indications of gap events may be communicated to the link layer, but they are used primarily within the PHY layer.

When transmitting an acknowledge, the link layer shall have communicated a PHY arbitration request with an Arbitration Class of IMMEDIATE within a finite time after the end of a packet has been reported to the link layer (i.e., once the PHY layer indicates to the link layer that the transmitting node has completed the transmission of DATA_END, and has released the bus). This duration is called the ACK_RESPONSE_TIME, and shall not exceed 32 arbitration clock times (approximately 651.04 ns).

5.5 Backplane initialization and reset

5.5.1 Backplane PHY reset

The following clauses describe reset operations within the PHY layer. Refer to 8.3.1.3 for more information regarding bus reset events and bus management.

Upon a power_reset event (i.e., power up or live insertion), registers and CSRs associated with the operation of the PHY layer shall be initialized to their default values. State machines associated with PHY layer operations may be initialized. The BUS_RESET signal (as described in 5.3.2) shall *not* be transmitted on the bus by the PHY layer. PHY event indications of BUS_RESET_START and BUS_RESET_COMPLETE (as described in 5.1.1.3) shall *not* be communicated to the node controller.

5.5.1.1 Command reset

Upon a command_reset event, CSRs associated with the operation of the PHY layer shall be initialized to their default values. The BUS_RESET signal shall *not* be transmitted on the bus by the PHY layer. PHY event indications of BUS_RESET_START and BUS_RESET_COMPLETE shall *not* be communicated to the node controller.

5.5.1.2 Bus reset

Once a PHY control request of Bus Reset (as described in 5.1.1.1) is communicated from the node controller to the PHY layer, the registers and CSRs associated with the operation of that PHY layer shall be initialized to their default values. State machines associated with PHY layer operations may be initialized. The PHY layer shall communicate BUS_RESET onto the bus. Once this signal is initiated, a PHY control confirmation of Initiated_reset (as described in 5.1.1.2) shall be communicated to the node controller. After the BUS_RESET event is “detected” by the node transmitting it (i.e., approximately 320 arbitration clock times after the signal is initiated), the PHY layer shall initialize itself.

NOTE — Since a PHY layer transmitting a BUS_RESET also has to react accordingly once the BUS_RESET event is detected, its state machines will not have had the opportunity to “advance” beyond those of other nodes. This ensures that all nodes are in somewhat similar states after such a bus reset event. Obviously, the logic (e.g., counters) used within the PHY layer to generate the BUS_RESET signal shall not be reset once that PHY simultaneously detects its “own” BUS_RESET.

Once a PHY layer detects that a BUS_RESET event has occurred on the bus, it shall initialize itself. PHY event indications of BUS_RESET_START and BUS_RESET_COMPLETE shall be communicated to the node controller.

5.5.2 Backplane PHY initialization

Upon entering the bus (i.e., power up, live insertion, or bus reset), a module shall wait for an arbitration reset gap before arbitrating for the bus. A node that is live inserted *should not* initiate a BUS_RESET.

6. Link layer specification

The link layer of the Serial Bus provides connectionless acknowledged data transfer services between a source node and a destination node. In the link layer the basic PDU is the packet and its corresponding acknowledge, defined in this standard as a “subaction.” There are two types of subactions:

- a) Asynchronous—The source node transmits data in turn, using the appropriate access method.
- b) Isochronous—The source node transmits data every NOMINAL_CYCLE_TIME with a guaranteed maximum latency of NOMINAL_CYCLE_TIME. The maximum size of each of these packets is set by a bus management entity.

Isochronous subactions have “guaranteed” access to the available bus bandwidth, while asynchronous subactions use the remaining bandwidth. It is the responsibility of the isochronous resource management process to guarantee that adequate bus bandwidth remains for asynchronous use.

The link layer makes use of PHY layer services (see 4.1 and 5.1) to perform actions on the bus. Note that the PHY layer services may impose additional requirements on the link layer.

6.1 Link layer services

Link layer services are provided at the interface between the link layer and higher layers, as illustrated in table 6.1.

Table 6.1—Summary of link layer services

Service	Layer communicated with	Purpose of service
Link control request	From the node controller	Configure the link layer
Link control confirmation	To the node controller	Confirm link control request
Link event indication	To the node controller	Alert node controller to events detected in the link layer
Link remote configuration request	From the node controller	Configure remote cable PHY layer
Link remote configuration indication	To the node controller	Report remote cable PHY configuration information
Link data request	From the transaction layer	Cause the link layer to send one primary asynchronous packet
Link data confirmation	To the transaction layer	Confirm link data request
Link data indication	To the transaction layer	Indicate the reception of one primary asynchronous packet
Link data response	From the transaction layer	Respond to link data indication
Link bus indication	To the transaction layer	Indicate bus event
Link isochronous control request	From an application	Select receive channels
Link cycle synch indication	To an application	Indicate the start of the local isochronous cycle
Link isochronous request	From an application	Cause the link layer to send one primary isochronous packet
Link isochronous indication	To an application	Indicate the reception of one primary isochronous packet

The method by which these services are communicated between the layers is not defined by this standard. Link layer services may perform actions specified by the higher layer. Link layer services may also communicate parameters that may or may not be associated with an action.

6.1.1 Link layer bus management services for the node controller

These services are used by the node controller to control the resources of the link layer. The link layer uses these services to communicate to the node controller changes of state within the link layer or on the bus.

6.1.1.1 Link control request (LK_CONTROL.request)

The node controller uses this service to request the link layer to perform specific actions and to specify link layer parameters. The link layer shall service the request immediately upon receipt by the link layer. This service is confirmed.

The following actions defined for all link layer implementations shall be provided by this service:

- Reset. The link layer shall discard all pending transactions and subactions, disable reception of all nonbroadcast packets, and disable transmission of all packets.
- Initialize. The link layer shall discard all pending transactions and subactions, enable reception of all nonbroadcast packets, and enable transmission of all packets.

If the link layer implementation is capable of being the cycle master, then the following actions shall also be provided by this service:

- Enable cycle master. The link layer shall send cycle start packets, as defined in 6.2.2.2.3.
- Disable cycle master. The link layer shall not send cycle start packets.

The following parameter is communicated to the link layer via this service:

- Node_ID. This parameter shall contain the current node_ID.

If the node is an isochronous resource manager, then the following **optional** parameter may be communicated to the link layer via this service:

- Expected channel list. This parameter shall contain the list of isochronous channels that are expected each isochronous cycle.

NOTE — This parameter is only present in nodes that have the isochronous resource manager function and is used for error detection. Note also that the isochronous resource manager shall be able to receive at all bus speeds in use on the bus, or it may not be able to recognize all channel transfers.

Table 6.2 summarizes the actions and parameters provided by this service.

Table 6.2—Summary of link control request actions and parameters

Action or parameter	Supported by...
Reset action	All link layers
Initialize action	
Node_ID parameter	
Enable cycle master action	Link layers that are capable of being the cycle master.
Disable cycle master action	
Expected channel list parameter (optional)	Link layers that are capable of being isochronous resource managers

6.1.1.2 Link control confirmation (LK_CONTROL.confirmation)

The link layer uses this service to confirm the results of a link control request service. The link layer shall communicate this service to the node controller upon completion of a link control request. There are no actions provided by this service. No parameters are communicated to the node controller via this service.

6.1.1.3 Link event indication (LK_EVENT.indication)

The link layer uses this service to indicate to the node controller events detected by the link layer. There are no actions provided by this service. No response is defined for this indication. The following parameters are communicated to the node controller via this service:

- Link event. This parameter shall contain the event detected by the link layer. The following values are defined for this parameter:
 - UNEXPECTED CHANNEL DETECTED (Optional). An unallocated channel was detected during the last isochronous cycle, as specified by the expected channel list. This value may only be used if the node is isochronous resource manager capable.
 - HEADER CRC ERROR DETECTED. The CRC check (see 6.2.4.15) for the header of a received primary packet failed.
 - UNKNOWN TRANSACTION CODE DETECTED. The header of a primary packet addressed to this node, or to an isochronous channel on the channel receive list, contained an invalid or unknown transaction code.
 - BUS OCCUPANCY VIOLATION DETECTED. A node on the bus held the bus longer than a time of MAX_BUS_OCCUPANCY. This value is optional.
 - CYCLE TOO LONG. The last isochronous cycle was too long. A cycle start packet was received, an ISOCHRONOUS_CYCLE_TIME passed, and a subaction gap was not yet detected.
 - DUPLICATE CHANNEL DETECTED. Two isochronous packets with the same channel number were detected during the last isochronous cycle. This value shall only be returned if the node is isochronous resource manager capable.

NOTE — The link event indication service is provided to report events that are detected (or detectable) by the link layer and that are not reportable through the data services. The node controller may (or may not) wish to log these events, or it may perform other implementation-specific actions.

6.1.1.4 Link remote configuration request (LK_CONFIG.request) (cable environment only)

The node controller uses this service to request the link layer to send PHY control packets to remote PHYs. No confirmation is defined for this request. The following actions shall be provided by this service:

- Send PHY configuration packet. A PHY configuration packet as described in 4.3.4.3 will be sent using the parameters listed below:

- Set force root. If true, the force_root bit will be set for the remote node with its physical_ID matching the remote physical ID. The force_root bit for all other nodes will be cleared.
- Remote physical ID. (Optional, only present when the set force root parameter is true.) The physical_ID of the remote node will have its force_root bit set.
- Set gap count. If true, the gap_count of all remote nodes will be set to the gap count parameter.
- Gap count. (Optional, only present when the set gap count parameter is true.) The new value of gap_count for all remote nodes.

IMPORTANT—The PHY control request is used to set the parameters for the local PHY. The link remote configuration request is only used for setting parameters of remote PHYs.

- Send Link-on packet. A link-on packet as described in 4.3.4.2 will be sent using the parameter listed below:
 - Remote physical ID. The physical ID of the node that is to receive the link-on packet.

NOTE — This request is part of the link layer specification since implementations will typically use link layer facilities to send PHY control packets. PHY hardware will typically only send self-ID packets.

6.1.1.5 Link remote configuration indication (LK_CONFIG.indication) (cable environment only)

The link layer uses this service to indicate to the node controller configuration information received during the bus initialization process from the self-ID packets. No response is defined for this indication. The following parameters are communicated via this service:

- a) Remote physical ID. This parameter shall contain the physical_ID of the node currently sending configuration information as described in 4.3.8.
- b) Remote gap_count. As described in 4.3.8.
- c) Remote link_active. As described in 4.3.8.
- d) Remote PHY_SPEED. As described in 4.3.7.
- e) Remote PHY_DELAY. As described in 4.3.7.
- f) Remote CONTENDER. As described in 4.3.7.
- g) Remote NPORT. As described in 4.3.9.
- h) Remote Child [one for each port]. As described in 4.3.9.
- i) Remote Connected [one for each port]. As described in 4.3.9.
- j) Remote Initiated_reset. As described in 4.3.8.

NOTE — This indication is part of the link layer specification since implementations will typically use link layer facilities to receive PHY self-ID packets. PHY hardware will typically only receive and process PHY control packets (configuration and link-on packets).

6.1.2 Link layer asynchronous data services for the transaction layer

These services are used to communicate asynchronous data packets between the link layer and the transaction layer. See 6.3.3.

6.1.2.1 Link data request (LK_DATA.request)

The transaction layer uses this service to request the link layer to transmit one primary asynchronous packet on the bus. This service shall be confirmed.

The following parameters are communicated to the link layer via this service. Some of the parameters may not be communicated because they are not required by the packet format specified by the transaction code.

- a) Destination address. As defined in 6.2.4.2.
- b) Transaction code. As defined in 6.2.4.5.
- c) Extended transaction code. As defined in 6.2.4.9.

- d) Transaction label. As defined in 6.2.4.3.
- e) Retry code. As defined in 6.2.4.4.
- f) Response code. As defined in 6.2.4.10.
- g) Data length. As defined in 6.2.4.8.
- h) Data. This is the data to be sent.
- i) Priority (*backplane only*). As defined in 6.2.4.6.
- j) Speed (*cable only*). As defined in table 4.2.

6.1.2.2 Link data confirmation (LK_DATA.confirmation)

The link layer uses this service to confirm the transmission of a packet. The link layer shall communicate this service to the transaction layer upon completion of a link data request. There are no actions provided by this service. The following parameters are communicated to the transaction layer via this service:

- a) Request status. This parameter shall contain the result of a link data request. The following values are defined for this value:
 - 1) ACKNOWLEDGE_RECEIVED. An expected acknowledge packet was received from the destination node.
 - 2) ACKNOWLEDGE_MISSING. An expected acknowledge packet was not received from the destination node.
 - 3) BROADCAST_SENT. The broadcast packet was transmitted.
- b) Acknowledge. This parameter shall be set to one of the values defined in table 6.13. This parameter is valid only when the value of request status is ACKNOWLEDGE_RECEIVED.

6.1.2.3 Link data indication (LK_DATA.indication)

The link layer uses this service to indicate to the transaction layer that an asynchronous packet has been received. There are no actions provided by this service. The transaction layer shall respond to this indication. The link layer shall communicate this indication to the transaction layer whenever an asynchronous packet has been received.

The following parameters are communicated to the transaction layer via this service. Some of the parameters may not be communicated because they are not included in the packet format indicated by the transaction code.

- a) Source ID. As defined in 6.2.4.7.
- b) Destination address. As defined in 6.2.4.2.
- c) Transaction code. As defined in 6.2.4.5.
- d) Extended transaction code. As defined in 6.2.4.9.
- e) Transaction label. As defined in 6.2.4.3.
- f) Retry code. As defined in 6.2.4.4.
- g) Response code. As defined in 6.2.4.10.
- h) Data length. As defined in 6.2.4.8.
- i) Data. This is the data received.
- j) Priority (*backplane only*). As defined in 6.2.4.6.
- k) Speed (*cable only*). As defined in table 4.2.
- l) Packet status. This parameter shall contain the result of the receive packet operation performed by the link layer. The following values are defined for this parameter:
 - 1) GOOD. The packet was received with no errors detected by the link layer.
 - 2) BROADCAST. A broadcast packet was received with no errors detected by the link layer.
 - 3) DATA_CRC_ERROR. The CRC check (see 6.2.4.15) for the data portion of a block packet failed.
 - 4) FORMAT_ERROR. The received packet had an unrecognized field value, or a reserved field was nonzero.

6.1.2.4 Link data response (LK_DATA.response)

The transaction layer uses this service to respond to a received packet; i.e., complete the subaction by sending an acknowledge packet. The transaction layer shall communicate this response to the link layer after receiving a link data indication.

The following parameters are communicated to the link layer via this service:

- a) Acknowledge. This parameter shall be set to one of the values defined in 6.2.5.2.2. The link layer shall use this parameter for the response.
- b) Bus Occupancy Control. This parameter controls whether the link layer will release control of the bus after sending the acknowledge. The following values are defined for this parameter:
 - 1) RELEASE. The link layer shall release control of the bus.
 - 2) HOLD. The transaction layer shall immediately follow this response with a link data request containing the response data for the current transaction. The link layer shall not give up control of the bus. The value of the acknowledge parameter shall be set to `ack_pending`.
 - 3) NO_OPERATION. The link layer shall do nothing. This parameter value shall be used only to respond to a link data indication with packet status of BROADCAST. The acknowledge parameter is undefined.
- c) Speed (*cable only*). As defined in table 4.2.

6.1.2.5 Link bus indication (LK_BUS.indication)

The link layer uses this service to indicate to the transaction layer events on the Serial Bus. There are no actions provided by this service. No response is defined for this indication. The following parameters are communicated to the transaction layer via this service:

- Bus event. This parameter shall contain the most recent event on the Serial Bus. The following values are defined for this parameter:
 - ARB_RESET_GAP. An arbitration reset gap has been detected on the Serial Bus.

6.1.3 Link layer isochronous data services for application layers

These services are used to communicate isochronous data packets between the link layer and an application layer. See 6.3.1.5 through 6.3.1.7.

6.1.3.1 Link isochronous control request (LK_ISO_CONTROL.request)

The application layer uses this service to communicate the following parameter to the link layer:

- Channel receive list. This parameter shall contain the list of isochronous channels that the link layer shall recognize. The link layer shall generate a link isochronous data indication whenever an isochronous packet is received with the channel field set to the value of one of the channels on this list.

6.1.3.2 Link cycle synch indication (LK_CYCLE.indication)

The link layer uses this service to indicate to an application layer that the cycle synch event has occurred. See 6.3.2.

There are no actions provided by this service. There is no response to this indication. The link layer shall communicate this indication to the application layer whenever the cycle synch event occurs.

The following parameters are communicated to the application layer via this service:

- a) Current cycle count. This parameter shall contain the current `CYCLE_TIME.cycle_count` value as defined in 8.3.2.2.8.

- b) Current Seconds count. This parameter shall contain the current value of the BUS_TIME register as defined in 8.3.2.2.9.

6.1.3.3 Link isochronous request (LK_ISO.request)

The application layer uses this service to request the link layer to transmit one isochronous packet on the bus. This service is not confirmed. The application layer shall make at most only one link isochronous request per channel for each link cycle synch indication received by the application layer.

The following parameters are communicated to the link layer via this service:

- a) Tag. As defined in 6.2.4.12.
- b) Channel. As defined in 6.2.4.13.
- c) Synchronization code. As defined in 6.2.4.14.
- d) Data length. As defined in 6.2.4.8.
- e) Data. This is the data to be sent.
- f) Speed (*cable only*). As defined in table 4.2.

6.1.3.4 Link isochronous indication (LK_ISO.indication)

The link layer uses this service to indicate to the application layer that an isochronous packet has been received. There are no actions provided by this service. No response is defined for this indication. The link layer shall communicate this indication to the application layer whenever an isochronous packet has been received.

The following parameters are communicated to the application layer via this service:

- a) Tag. As defined in 6.2.4.12.
- b) Channel. As defined in 6.2.4.13.
- c) Synchronization code. As defined in 6.2.4.14.
- d) Data length. As defined in 6.2.4.8.
- e) Data. This is the data received.
- f) Speed (*cable only*). As defined in table 4.2.
- g) Packet status. This parameter shall contain the result of the receive packet operation performed by the link layer, as defined in 6.1.2.3.

NOTE — The transaction code is implied by this service. See 6.2.3.1.

6.2 Link layer facilities

The link layer facilities use two of the three basic packet types: primary packets and acknowledge packets. Figure 6.1 shows the relationship between the different packet types.

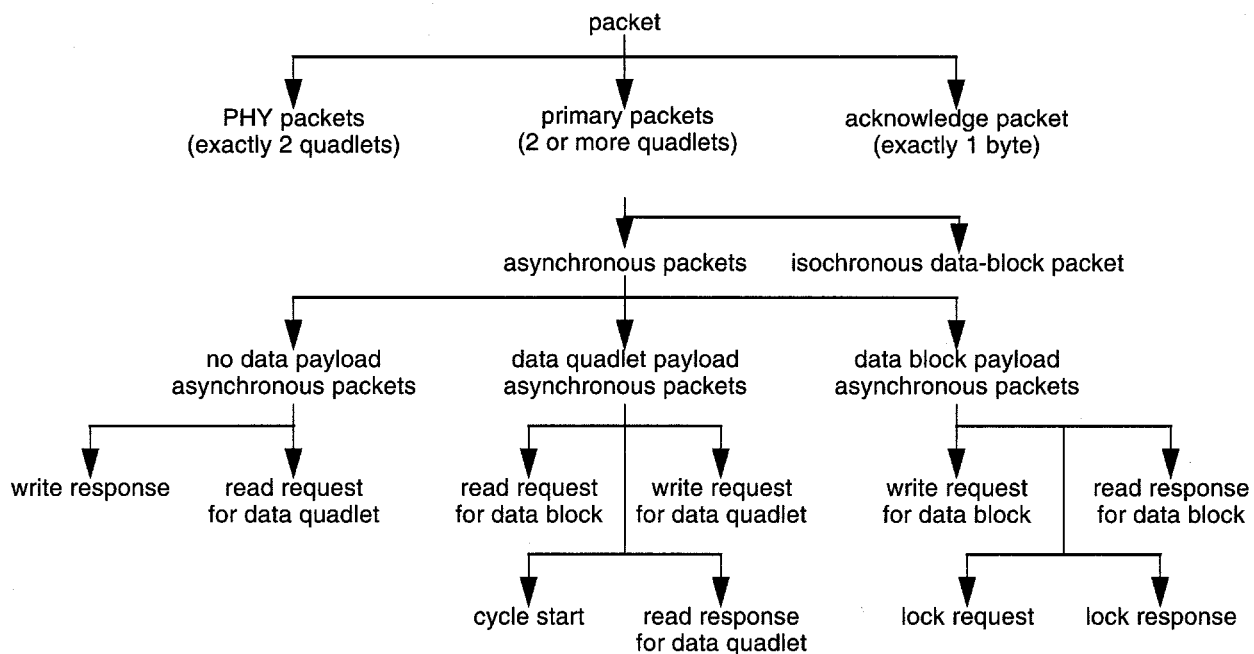


Figure 6.1—Serial Bus packets

6.2.1 Primary packets

All primary Serial Bus packets share the format shown in figure 6.2.

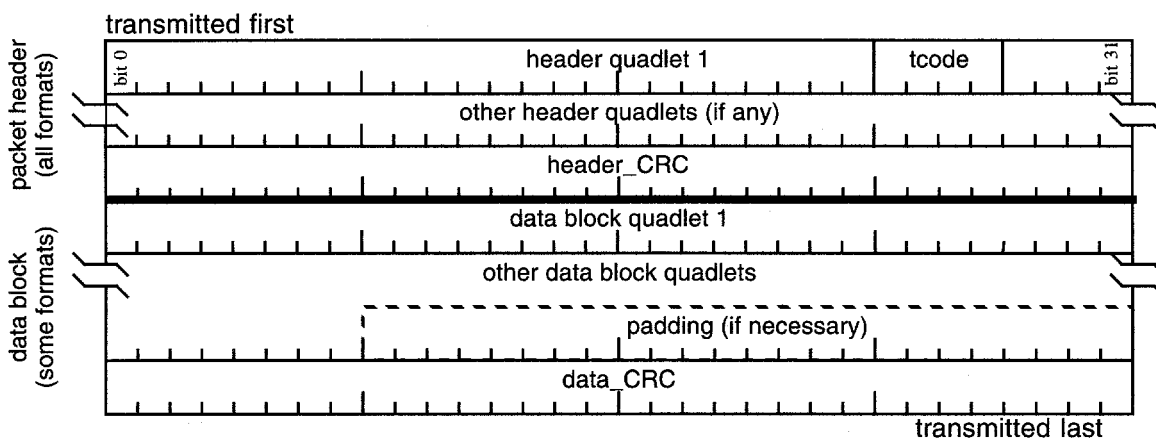


Figure 6.2—Primary packet format

Every valid primary packet is a sequence of aligned quadlets. A primary packet is distinguished from an acknowledge packet or a PHY packet by its length and/or encoding. The length of a primary packet shall be at least two quadlets (a zero-data isochronous packet). A primary packet shall consist of a packet header, and it may also include a data block.

All packet headers shall contain one or more quadlets followed by a header_CRC. The header_CRC shall be calculated only on the packet header. A node shall not act on or generate a response to a packet in which the packet header does not pass the header_CRC check (see 6.2.4.15).

The first quadlet of the packet header of every primary packet shall contain the transaction code (tcode in figure 6.5) in bits 24 to 27. The transaction code defines the packet type of a primary packet (see 6.2.4.5).

Primary packet types with a data block payload shall contain one or more quadlets of data followed by a data_CRC. The data_CRC shall be calculated only on the data block quadlets and shall not include any of the packet header quadlets.

Two basic varieties of primary packet are defined for the Serial Bus: the asynchronous packet and the isochronous packet, which are distinguishable by the transaction code value.

6.2.2 Asynchronous packets

All asynchronous Serial Bus packets share the format shown in figure 6.3.

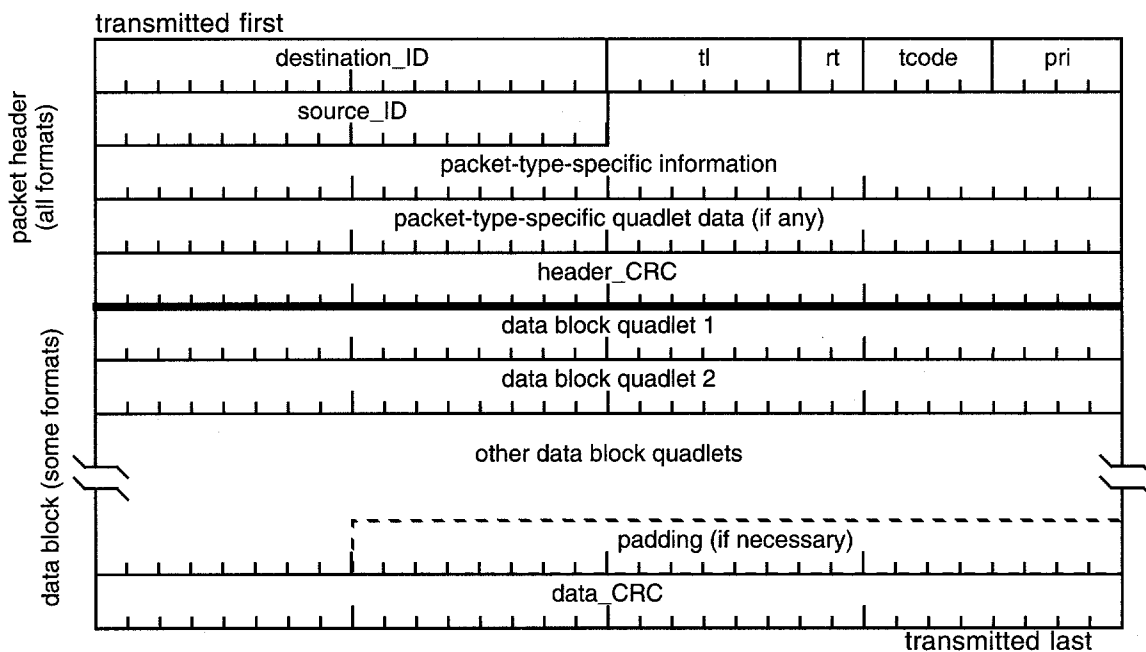


Figure 6.3—Asynchronous packet format

Every valid asynchronous packet is a sequence of aligned quadlets. An asynchronous packet conforms to the rules for a primary packet. The length of an asynchronous packet shall be at least four quadlets. An asynchronous packet shall consist of a packet header, and it may also include a data block. The fourth quadlet of the header may be required to hold packet-type-specific data.

Table 6.3 summarizes the asynchronous packet components and abbreviations used in this clause. The packet components are fully defined in clause 6.2.4.

Table 6.3—Summary of asynchronous packet components

Packet component	Name(s)	Size (bits)	Subclause	Comment
Destination identifier	destination_ID	16	6.2.4.2.1	All asynchronous packets
Transaction label	tl	6	6.2.4.3	All asynchronous packets
Retry code	rt	2	6.2.4.4	All asynchronous packets
Transaction code	tcode	4	6.2.4.5	All primary packets
Priority	pri	4	6.2.4.6	All asynchronous packets
Source identifier	source_ID	16	6.2.4.7	All asynchronous packets
Packet-type-specific information	destination_offset	48	6.2.4.2.2	Request packet: destination offset
	rcode & reserved	4 & 44	6.2.4.10	Response packet: response code and reserved field
Packet-type-specific quadlet data	quadlet data	32	6.2.4.11	Quadlet packets: quadlet payload
	data_length and extended_tcode		6.2.4.8, 6.2.4.9	Data-block packets: data length and extended transaction code
Header CRC	header_CRC	32	6.2.4.15	All primary packets
Data block payload	data field	—	6.2.4.11	All data block packets; has special meaning in lock-request and lock-response packets
	arg_value	32 or 64	6.2.2.3.2	Special for lock-request packets
	data_value	32 or 64	6.2.2.3.2	Special for lock-request packets
	old_value	32 Or 64	6.2.2.3.4	Special for lock-response packets
Data block CRC	data_CRC	32	6.2.4.15	All data block packets

6.2.2.1 Asynchronous packets with no-data payload

This packet format contains no payload. A no-data packet contains a packet header only; no data block follows the header. The generic format of a no-data packet is shown in figure 6.4. The “packet-specific information” field differs based on the packet type.

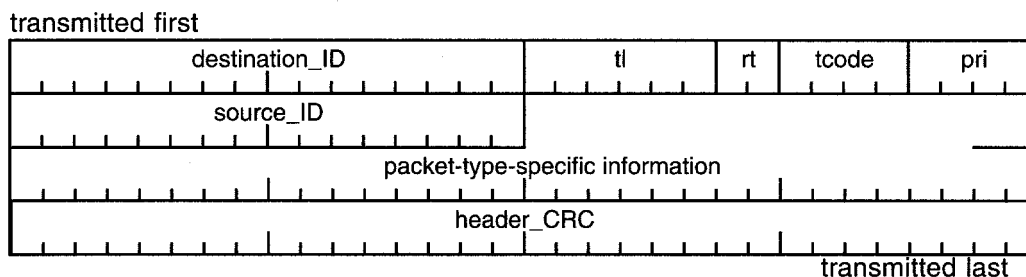


Figure 6.4—No-data payload primary packet format

6.2.2.1.1 Read request for data quadlet

This packet type requests a single data quadlet from the specified destination address (see 6.2.4.2). The packet-specific information field of this packet type specifies the destination_offset for the read request.

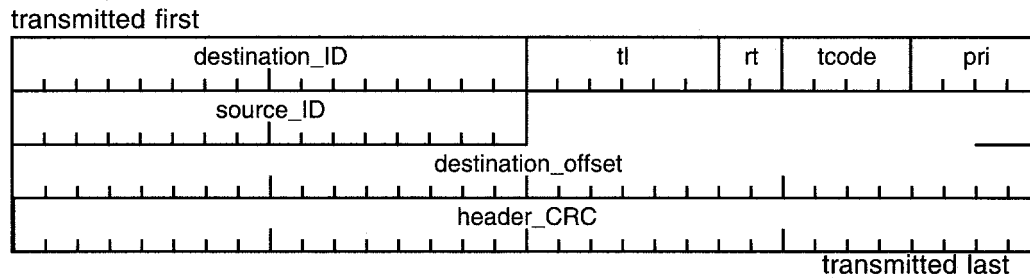


Figure 6.5—Read request for data quadlet packet format

6.2.2.1.2 Write response

This packet type shall be sent only in response to a write request for data quadlet or a write request for data block. This packet type shall not be sent if the transaction was a unified transaction (see 7.3.2.1). The packet-specific information field of this packet type specifies the response code (rcode) for the corresponding write request.

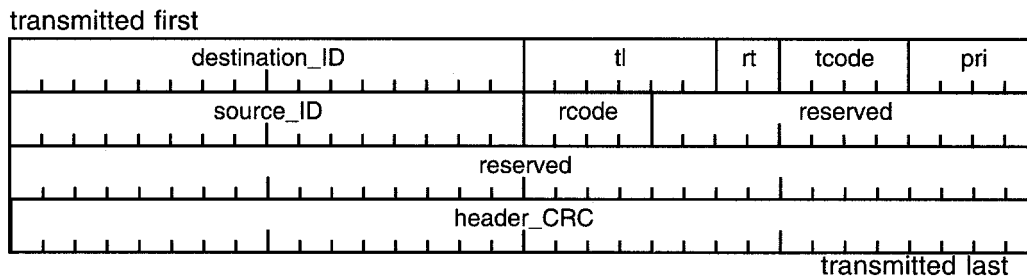


Figure 6.6—Write response packet format

6.2.2.2 Asynchronous packet formats with data quadlet payload

This packet format contains a single quadlet as a payload, which is contained within the packet header. A data quadlet payload packet contains a packet header only; no data block follows the header. The generic format of a data quadlet payload packet is shown in figure 6.7. The “packet-specific information” and “packet-type-specific quadlet data” fields differ based on the packet type.

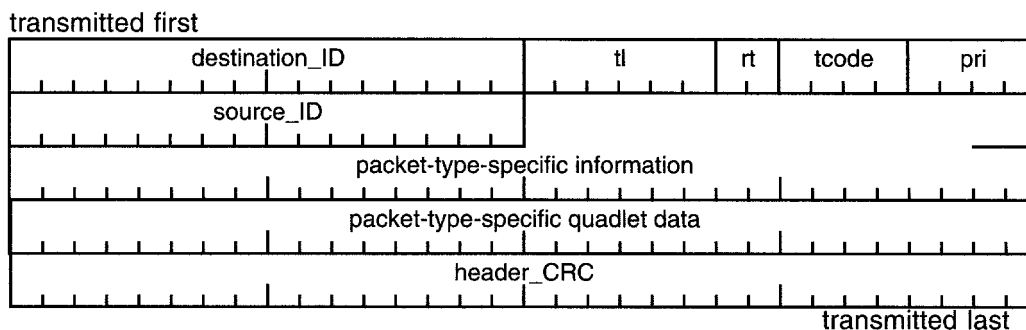


Figure 6.7—Data quadlet payload packet format

6.2.2.2.1 Read request for data block

This packet type requests a data block from the specified destination address (see 6.2.4.2). The packet-specific information field of this packet type specifies the destination_offset for the read request. The packet-specific quadlet data field specifies the data_length of the expected response data, and the extended transaction code (extended_tcode). The extended_tcode field is reserved for this packet type, and it shall be set to 0000_{16} .

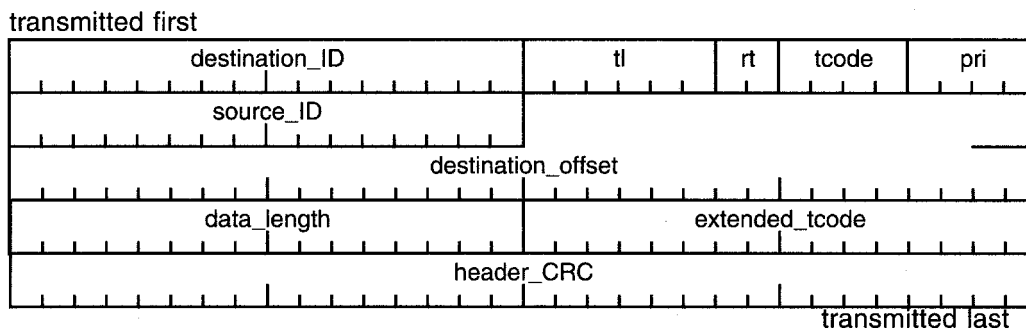


Figure 6.8—Read request for data block packet format

6.2.2.2.2 Write request for data quadlet

This packet type requests a data quadlet be written to the specified destination address (see 6.2.4.2). The packet-specific information field of this packet type specifies the destination_offset for the write request. The packet-specific quadlet data field specifies the data quadlet to be written as a result of the request.

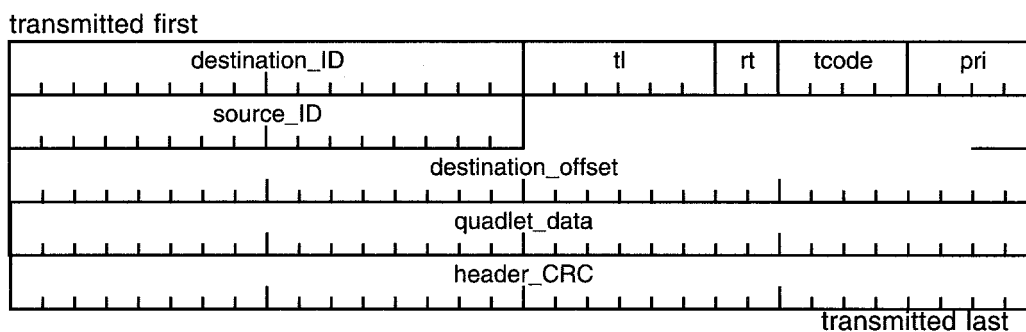


Figure 6.9—Write request for data quadlet packet format

6.2.2.2.3 Cycle start

This packet type indicates the start of an isochronous cycle on the bus (see 6.3.1.5).

NOTE — The cycle start packet is a quadlet payload packet with special values that can be interpreted as a write request to the CYCLE_TIME register (see 8.3.2.2.8). While this can be handled in an implementation by the transaction layer and node controller, this standard assumes that the link layer directly handles the generation and detection of the start of an isochronous cycle.

This packet shall be sent only by a cycle master.

The packet-specific information field of this packet type specifies the destination_offset for the request. The packet-specific quadlet data field specifies the data quadlet to be written as a result of this request.

The cycle start packet type has special requirements for packet field settings. The fields within this packet type shall be set as follows:

- The destination_ID field shall be set to $FFFF_{16}$. This specifies a broadcast on the local bus (see 6.2.4.2.1).
- The transaction label (tl) field shall be set to 000000_2 .
- The retry code (rt) field shall be set to 00_2 .
- The transaction code field shall be set to the transaction code for this packet type (see 6.2.4.5).
- The priority field shall be set to 1111_2 . This indicates the highest priority for this subaction.
NOTE — The priority field is set to this value for compatibility with the backplane PHY.
- The source_ID field shall be set to the node_ID of the cycle master node sending this packet.
- The destination_offset field shall be set to the standard address of the CYCLE_TIME register (see 8.3.2.2.8.)
- The cycle_time_data payload shall contain the contents of the CYCLE_TIME register at the cycle master node (see 8.3.2.2.8).
- The header_CRC is calculated normally.

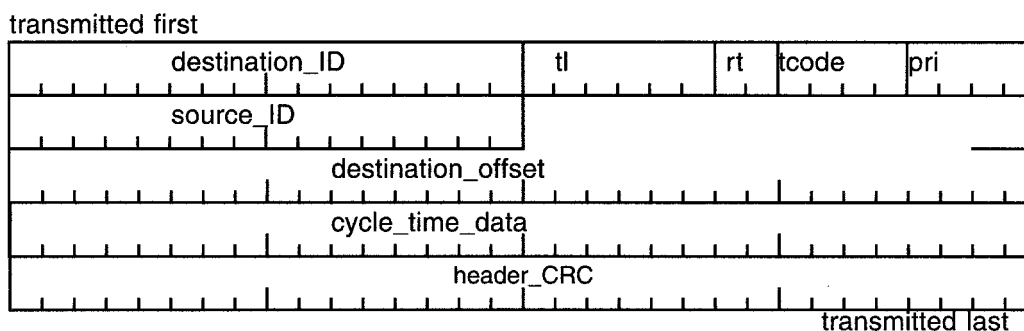


Figure 6.10—Cycle start primary packet format

6.2.2.2.4 Read response for data quadlet

This packet type shall be sent only in response to a read request for a data quadlet. The packet-specific information field of this packet type specifies the response code (rcode) for the corresponding read request. The packet-specific quadlet data field contains the requested read data.

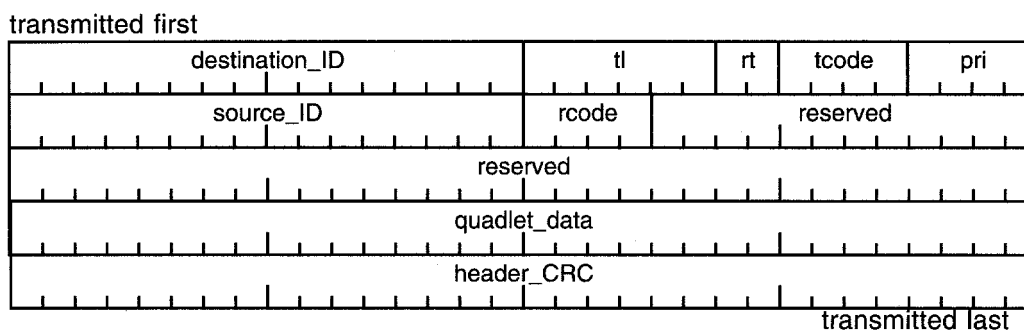


Figure 6.11—Read response for data quadlet packet format

6.2.2.3 Asynchronous packet formats with data block payload

This packet format contains a data block as a payload, which is transferred after the packet header. A data block payload packet can contain zero or more bytes of data. If the data_length is not a multiple of four, the originating node shall pad the data field to a quadlet boundary with 00₁₆ fill data. If the data_length is zero, no data field shall be sent, and no data_CRC shall be sent. The generic format of a data block payload packet is shown in figure 6.12. The “packet-specific information” and “packet-specific quadlet data” fields differ based on the packet type.

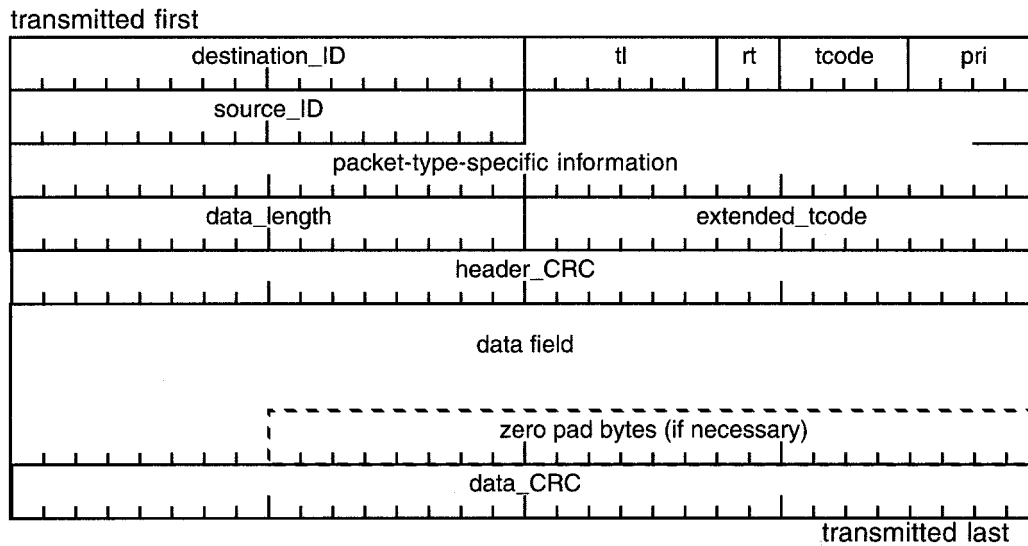


Figure 6.12—Data block payload packet format

The number of bytes in the data field shall not exceed an absolute maximum value based on the data rate used to transmit the packet, as shown in table 6.4.

Table 6.4—Maximum payload size for primary packets with data block payload

Data rate	Maximum payload size (bytes)	Comment
S25	128	TTL backplane
S50	256	BTL and ECL backplane
S100	512	Cable base rate
S200	1024	
S400	2048	

6.2.2.3.1 Write request for data block

This packet type requests a data block be written to the specified destination address (see 6.2.4.2). The packet-specific information field of this packet type specifies the destination_offset for the write request. The extended_tcode field is reserved for this packet type, and it shall be set to 0000₁₆.

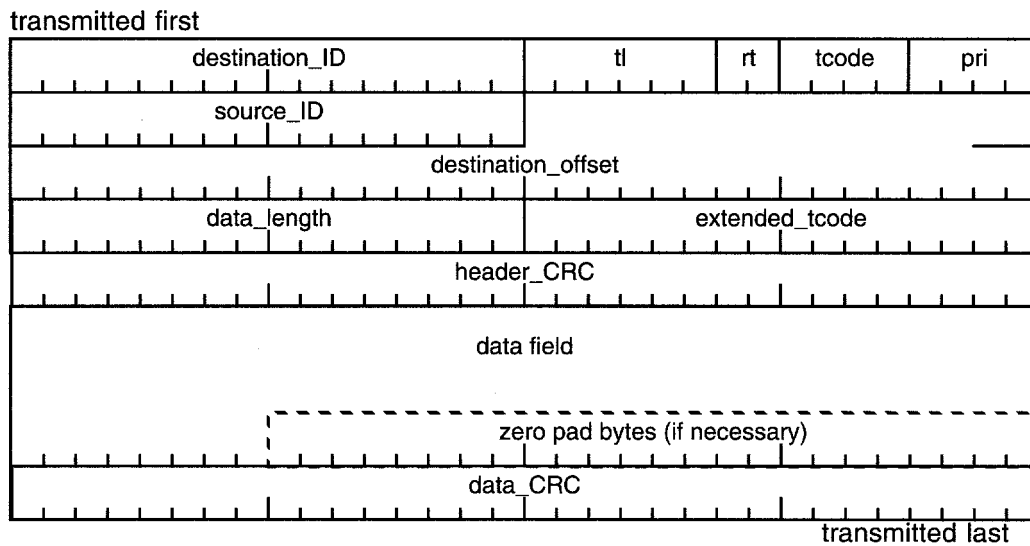


Figure 6.13—Write request for data block packet format

6.2.2.3.2 Lock request

This packet type requests a locked access of the specified destination address (see 6.2.4.2). The packet-specific information field of this packet type specifies the destination_offset for the lock request packet. The extended_tcode field is set to the lock function requested (see 6.2.4.9).

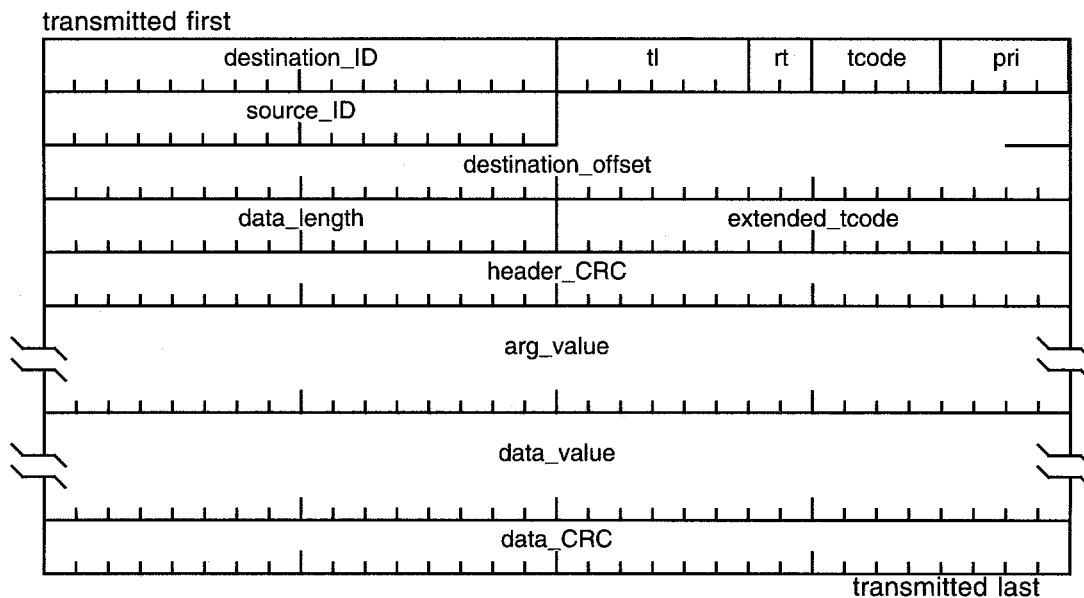


Figure 6.14—Lock-request packet format

The data_length field shall be set only to the values shown in table 6.5. All other values for data_length are reserved. The values specified in data_length and extended tcode indicate the size of the arg_value and data_value fields.

Table 6.5—Data_length values for lock requests

Data_length (bytes)	Size of arg_value (bytes)	Size of data_value (bytes)	Comment
4	0	4	Single argument 32-bit little_add/fetch_add
8	4	4	Two argument 32-bit function
8	0	8	Single argument 64-bit little_add/fetch_add
16	8	8	Two argument 64-bit function

6.2.2.3.3 Read response for data block

This packet type shall be sent only in response to a read request for data block. The packet-specific information field of this packet type specifies the response code (rcode) for the corresponding read request. The extended_tcode field is reserved for this packet type, and it shall be set to 0000₁₆. If the value of rcode is not resp_complete, then the data_length shall be set to 0000₁₆, and no data block (data field and data_CRC) shall be transferred.

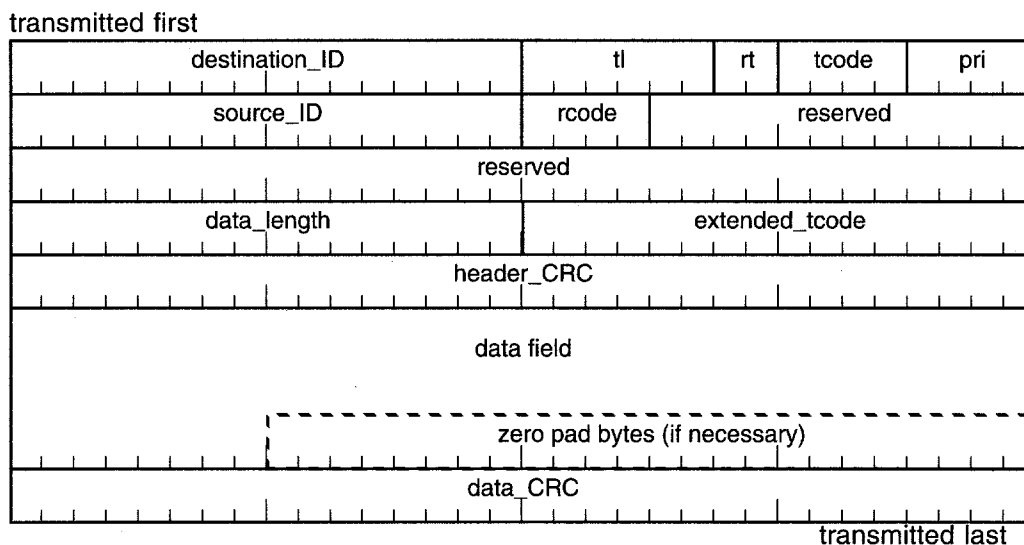


Figure 6.15—Read response for data block packet format

6.2.2.3.4 Lock response

This packet type shall be sent only in response to a lock-request packet. The packet-specific information field of this packet type specifies the response code (rcode) for the corresponding lock request. The extended_tcode field shall be set to the lock function specified in the lock-request packet. The data_length field shall be set only to a value of 0004₁₆ or 0008₁₆, corresponding to the size of old_value. If the value of rcode is not resp_complete, then the data_length shall

be set to 0000_{16} , and no data block (old_value and data_CRC) shall be transferred. All other values for data_length are reserved.

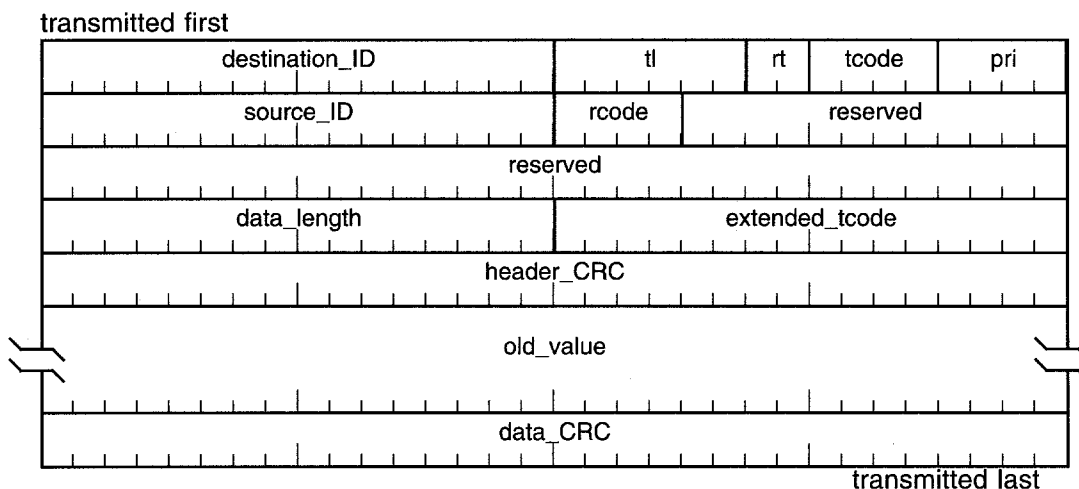


Figure 6.16—Lock-response packet format

6.2.3 Isochronous packets

Every valid isochronous packet is a sequence of aligned quadlets. An isochronous packet conforms to the rules for a primary packet. The length of an isochronous packet shall be at least two quadlets. An isochronous packet shall consist of a packet header, and it may also include a data block. Only one type of isochronous packet is defined for the Serial Bus, as described in 6.2.3.1.

Table 6.6 summarizes the isochronous packet components and abbreviations used in this clause. The packet components are fully defined in 6.2.4.

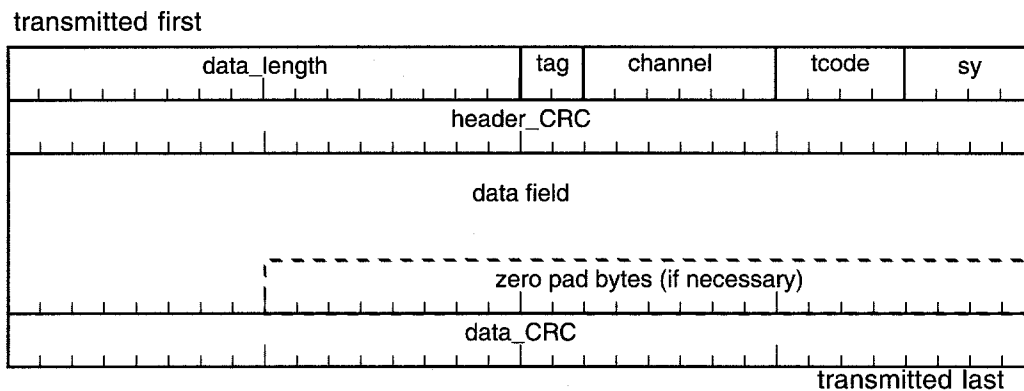
Table 6.6—Summary of isochronous packet components

Packet component	Abbreviation	Size (bits)	Comment
Data length	data_length	16	All data-block packets
Isochronous data format tag	tag	2	Isochronous data block packet only
Isochronous channel	channel	6	Isochronous data block packet only
Transaction code	tcode	4	All primary packets
Synchronization code	sy	4	Isochronous data block packet only
Header CRC	header_CRC	32	All primary packets
Data block payload	data_field	—	All data block packets
Data block CRC	data_CRC	32	All data block packets

6.2.3.1 Isochronous data block packet format

This packet format contains a data block as a payload, which is transferred after the packet header. The data block may contain zero or more bytes of data. If the data_length is not a multiple of four, the talking node shall pad the data field to a quadlet boundary with 00(16) fill data. If the data_length is zero, no data field shall be sent and no data_CRC shall be sent.

Isochronous data block primary packets shall be sent only during an isochronous cycle (see 6.3.1.5).

**Figure 6.17—Isochronous data block packet format**

The number of bytes in the data field shall not result in an isochronous packet that exceeds the bandwidth allocated for this channel (see 8.4.2.6).

6.2.4 Primary packet components

This clause describes the component fields and codes of primary packets.

6.2.4.1 Reserved fields, codes, and values

Reserved codes and reserved fields within a packet are reserved by this standard for future expansion. All reserved fields shall be set to zero by the sender of the packet. Reserved or invalid values for a field shall not be used by the sender of a packet.

The appropriate responses to a reserved or unimplemented transaction code value, a reserved or unimplemented extended transaction code value, a reserved response code value, or an unimplemented or invalid length value are defined in the transaction layer (see 7.3.3.1).

The receiving link shall ignore any nonzero reserved fields.

NOTE — Reserved fields only occur in response packets

6.2.4.2 Destination address

The destination address is a concatenation of two fields, the destination ID and the destination offset.

6.2.4.2.1 Destination ID (`destination_ID`)

The destination ID field shall specify the node ID of the receiving node, and it is also the concatenation of two fields. The upper 10 bits of the destination ID specify the `destination_bus_ID`. The lower 6 bits of the destination ID specify the `destination_physical_ID`. Table 6.7 describes the encoding of the destination ID field and special meanings of certain `bus_ID` and `physical_ID` values.

Table 6.7—Destination ID encoding

<code>destination_bus_ID</code>	<code>destination_physical_ID</code>	Comment
0 to $3FE_{16}$	0 to $3E_{16}$	The <code>destination_ID</code> is addressed to a specific node on a specific bus. Only the node with a <code>node_ID</code> equal to the <code>destination_ID</code> shall respond to or acknowledge the packet.
$3FF_{16}$	0 to $3E_{16}$	The <code>destination_ID</code> is addressed to a specific node on the local bus. Only the node with an <code>physical_ID</code> equal to the <code>destination_physical_ID</code> shall respond to or acknowledge the packet.
0 to $3FE_{16}$	$3F_{16}$	The <code>destination_ID</code> specifies a broadcast to a specific bus. Only nodes with a <code>bus_ID</code> equal to the <code>destination_bus_ID</code> shall recognize the packet.
$3FF_{16}$	$3F_{16}$	The <code>destination_ID</code> specifies a broadcast to the local bus. All nodes on the local bus shall recognize the packet.

6.2.4.2.2 Destination offset (`destination_offset`)

The destination offset field shall specify the lower 48 bits of the destination node address of a request packet. For the read request for data quadlet and the write request for data quadlet packets, the `destination_offset` value shall be a quadlet aligned address.

6.2.4.3 Transaction label (tl)

The transaction label shall specify a unique tag for each outstanding transaction from a node. The transaction label sent in a request subaction shall be used as the transaction label returned in the corresponding response subaction. All transaction label values are valid, and all values shall be accepted by all devices.

A node shall not originate a request with a transaction label of N to another node with the node_ID of M if the requesting node has another request outstanding to node M that uses the transaction label N value. A transaction is considered outstanding if a transaction response has not been received within a transaction timeout period as discussed in 7.2.1 and if a bus reset has not occurred.

NOTE — Each transaction is uniquely identified by the source ID of the requesting node, the destination ID of the responding node, and the transaction label.

During a bus reset, all nodes shall discard currently queued asynchronous requests and responses, so that all transaction label values can be safely reused.

The behavior of the responding node is not defined by this standard when the requesting node violates these restrictions.

6.2.4.4 Retry code (rt)

The rt (retry code) field shall specify whether this packet is a retry attempt and the retry protocol to be followed by the destination node. See 7.3.3.2.2 for a complete description of the retry protocol.

Table 6.8—Retry code encoding

Code (binary)	Name	Comment
00	retry_1	retry_A/retry_B supported and a new reservation is requested.
01	retry_X	retry_A/retry_B not supported or (if supported) reservation not requested.
10	retry_A	retry_A/retry_B supported and a reservation has been assigned (ack_busy_A previously returned).
11	retry_B	retry_A/retry_B supported and a reservation has been assigned (ack_busy_B previously returned).

If the (optional) retry_A/retry_B capability is not supported, the retry_X code shall be used *every* time the subaction is sent. If the retry_A/retry_B capability is supported, the retry_X code is only used until this becomes the oldest subaction (of the same type, request or response) being retried.

The retry_1 code shall be used if the retry_A/retry_B capability is supported when this becomes the oldest subaction (of the same type, request, or response) being retried. The retry_1 code shall also be used if the previous retry contained a retry_1, retry_A, or retry_B code and was acknowledged with ack_busy_X.

The retry_A code shall be used if the retry_A/retry_B capability is supported and if the previous retry contained a retry_1, retry_A, or retry_B code and was acknowledged with ack_busy_A.

The retry_B code shall be used if the retry_A/retry_B capability is supported and if the previous retry contained a retry_1, retry_A, or retry_B code and was acknowledged with ack_busy_B.

Simple, fully capable nodes are expected to limit their currently active retries to their oldest request and their oldest response. Sophisticated nodes may concurrently retry newer requests or responses, as long as they meet the minimum

retry-delay requirements for their oldest subactions, and all but the oldest subaction of each type (request or response) is sent using the `retry_X` (no reservation requested) retry code.

6.2.4.5 Transaction code (tcode)

The transaction code shall specify the packet format and the type of transaction that shall be performed. If the addressed node detects a reserved or unsupported transaction code, it shall ignore the packet.

Table 6.9—Transaction code encoding

Code	Name	Comment
0	Write request for data quadlet	Request subaction, quadlet payload
1	Write request for data block	Request subaction, block payload
2	Write response	Response subaction for both write requests types, no payload
3	Reserved	
4	Read request for data quadlet	Request subaction, no payload
5	Read request for data block	Request subaction, quadlet payload
6	Read response for data quadlet	Response subaction to request for quadlet, quadlet payload
7	Read response for data block	Response subaction to request for block, block payload
8	Cycle start	Request to start isochronous cycle, quadlet payload
9	Lock request	Request subaction, block payload
A ₁₆	Isochronous data block	Isochronous subaction, block payload
B ₁₆	Lock response	Response subaction for lock request, block payload
C ₁₆	Reserved	
D ₁₆	Reserved	
E ₁₆	Reserved	
F ₁₆	Reserved	

6.2.4.6 Priority (pri)

The priority field is a 4-bit value that only has meaning for the backplane PHY. See 5.4.2.1 and 5.4.1.3 for more details. All priority values are valid, and all values shall be accepted by all devices. Link layers attached to cable PHY devices shall ignore this field when receiving, and they shall set this field to 0000₂ when originating any asynchronous packet except cycle start (see 6.2.2.2.3).

For all environments, a priority value of all zeros corresponds to the fair arbitration mechanism and all ones to the highest priority with fairness disabled. The link layer shall not alter the priority value. All priority values shall be passed through the link layer unaltered in order to facilitate bridging between the cable and the backplane environments.

6.2.4.7 Source ID (source_ID)

The source ID field shall specify the node ID of the sending node. The source ID value of a request subaction shall be used as the destination ID value for the corresponding response subaction, if any.

6.2.4.8 Data length (`data_length`)

The data length field shall specify the length of the data field of data block payload packets and isochronous data block packets. The data length field may have specific limits on allowed values (see 6.2.2.3, 6.2.3.1, and 6.2.2.3.2). If the receiving link detects an invalid or unsupported data length field in a nonbroadcast primary packet, it shall either immediately return a `type_error` acknowledge code or return a `type_error` response code during a later response subaction.

6.2.4.9 Extended transaction code (`extended_tcode`)

The `extended_tcode` is a 16-bit field that is only meaningful if the transaction code indicates a lock-request or lock-response primary packet type. For all other packet types, this field is reserved and shall be set to 0000_{16} . See 7.3.3.2.1 for the definition of lock transactions. This field uses the encoding shown in table 6.10.

Table 6.10—Extended transaction code encoding

Code	Name
0000	Reserved
0001	mask_swap
0002	compare_swap
0003	fetch_add
0004	little_add
0005	bounded_add
0006	wrap_add
0007	vendor-dependent
0008-FFFF ₁₆	Reserved

6.2.4.10 Response code (`rcode`)

The response code shall specify the response to an earlier corresponding request subaction.

Table 6.11—Response code encoding

Code	Name	Comment
0	resp_complete	The node has successfully completed the command.
1	Reserved	
2	Reserved	
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 header 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 field in the request was set to an address not accessible in the destination node.
8 to F ₁₆	Reserved	

6.2.4.11 Data field

The data field shall contain the data to be transferred in a primary packet with a data block payload.

If the data length field specifies a length that is not a multiple of four, the sender shall pad the data field with 00₁₆ bytes such that the data field is composed of full quadlets.

6.2.4.12 Tag

The tag field provides a high-level label for the format of data carried by the isochronous packet.

Table 6.12—Tag field encoding

Value (binary)	Meaning
00	Data field unformatted
01	Reserved
10	Reserved
11	Reserved

6.2.4.13 Channel

The channel field shall specify the isochronous channel number for the packet. A channel field value shall not be repeated within a single isochronous cycle. The channel number is assigned to a node for talking or listening by the isochronous resource manager. Channel numbers 0 through 63 are available for assignment.

6.2.4.14 Synchronization code (sy)

The synchronization code is an application-specific control field. The details of its use are beyond the scope of this standard.

NOTE — A synchronization point may be defined as boundary between video or audio frames, or any other point in the data stream the application may consider appropriate.

6.2.4.15 CRCs

Both the header CRC and the data CRC are generated and checked using a standard algorithm.¹⁴ The most significant bit [the x^{31} term of $R(x)$] of the CRC shall be transmitted first.

6.2.4.15.1 Definitions

- $F(x)$ A degree $k-1$ polynomial that is used to represent the k bits of the packet covered by the CRC. For the purposes of the CRC, the coefficient of the highest order term shall be the first bit transmitted.
- $L(x)$ A degree 31 polynomial with all of the coefficients equal to one, i.e.,
 $L(x) = x^{31} + x^{30} + x^{29} + \dots + x^2 + x^1 + 1$
- $G(x)$ The standard generator polynomial:
 $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- $R(x)$ The remainder polynomial that is of degree less than 32.
- $P(x)$ The remainder polynomial on the receive checking side that is of degree less than 32.
- CRC The CRC polynomial that is of degree less than 32.
- $Q(x)$ The greatest common multiple of $G(x)$ in $[x^{32}F(x) + x^kL(x)]$.
- $Q^*(x)$ $x^{32}Q(x)$.
- $M(x)$ The sequence that is transmitted.
- $M^*(x)$ The sequence that is received.
- $C(x)$ A unique polynomial remainder produced by the receiver upon reception of an error-free sequence. This polynomial has the value:
 $C(x) = x^{32}L(x)/G(x)$
 $C(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$

6.2.4.15.2 CRC generation equations

The equations that are used to generate the CRC sequence from $F(x)$ are

$$\text{CRC} = L(x) + R(x) = R\$(x) \tag{1}$$

where

$R\$(x)$ is the one's complement of $R(x)$.

$$[x^{32}F(x) + x^kL(x)]/G(x) = Q(x) + R(x)/G(x) \tag{2}$$

$$M(x) = [x^{32}F(x)] + \text{CRC} \tag{3}$$

NOTE — All arithmetic is modulo 2. Equation (1), adding $L(x)$ (all ones) to $R(x)$, simply produces the one's complement of $R(x)$; thus, this equation is specifying that the $R(x)$ is inverted before it is sent out. Equation (3) simply specifies that the CRC is appended to the end of $F(x)$.

¹⁴This is the same CRC used by the IEEE 802 LANs and FDDI. The remainder of this clause was extracted from chapter 7 of the ANSI FDDI Media Access Control specification.

6.2.4.15.3 CRC checking

The received sequence $M^*(x)$ may differ from the transmitted sequence $M(x)$ if there are transmission errors. The process of checking the sequence for validity involves dividing the received sequence by $G(x)$ and testing the remainder. Direct division, however, does not yield a unique remainder because of the possibility of leading zeros. Thus, a term $L(x)$ is prepended to $M^*(x)$ before it is divided. Mathematically, the received checking is shown in the following equation:

$$x^{32}[M(x) + x^k L(x)]/G(x) = Q^*(x) + P(x)/G(x) \quad (4)$$

In the absence of errors, the unique remainder is the remainder of the division:

$$P(x)/G(x) = x^{32}L(x)/G(x) = C(x) \quad (5)$$

Table 6.15 includes a C program of the generating and checking algorithms.

6.2.5 Acknowledge packets

An acknowledge packet shall be sent by a destination node as the immediate response to a nonbroadcast or nonisochronous primary packet. An acknowledge packet is distinguished from a primary packet by its length, which shall be exactly 1 byte.

6.2.5.1 Acknowledge packet format

Every acknowledge packet shall consist of a single byte of data, where the first 4 bits transmitted shall contain the acknowledge code (ack_code), and the 4 bits transmitted shall contain the acknowledge parity (ack_parity).

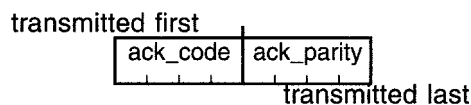


Figure 6.18—Acknowledge packet format

6.2.5.2 ACK packet components

This clause describes the component fields and codes of acknowledge packets. Note that in tables that give the code values for a field, the leftmost bit of the code is transmitted first.

6.2.5.2.1 Reserved acknowledge codes

Reserved codes within an acknowledge packet are reserved by this standard for future expansion. Reserved ack_code values shall not be used by a transmitting link. The receiving link shall check all fields for reserved code values.

If the receiving link detects a reserved ack_code, the acknowledge packet shall be ignored, and a request status of ACKNOWLEDGE MISSING shall be presumed (see 6.1.2.2).

6.2.5.2.2 Acknowledge code (ack_code)

The acknowledge code shall specify the immediate response to a nonbroadcast primary packet. The allowable acknowledge codes are listed in table 6.13.

Table 6.13—Acknowledge codes

Code	Name	Meaning
0	Reserved	
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 7.3.3.2.2).
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 7.3.3.2.2).
7	Reserved	
8	Reserved	
9	Reserved	
A ₁₆	Reserved	
B ₁₆	Reserved	
C ₁₆	Reserved	
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	

6.2.5.2.3 Acknowledge parity (ack_parity)

The acknowledge parity field shall specify the parity check bits for an acknowledge packet. The value of the acknowledge parity field shall be the one's complement of the acknowledge code.

If the receiving link detects an acknowledge parity error (the ack_parity field is not the one's complement of the ack_code), the acknowledge packet shall be ignored, and a request status of ACKNOWLEDGE MISSING shall be presumed (see 6.1.2.2).

6.3 Link layer operation

This subclause defines how the link layer attempts to initiate the transmission of a packet, and how the link layer receives a packet. This clause makes extensive reference to the PHY layer services described in clauses 4. and 5.

6.3.1 Overview of link layer operation

This subclause gives an informal overview of link layer operations. The descriptions in this subclause are informative and nonbinding. The formal description of behavior is in 6.3.3.

6.3.1.1 Communication with the PHY layer

The link layer communicates via a set of abstract services. These abstract services provide a means of documenting the information that moves between the link layer and PHY layer. Two subsets of services are defined: the PHY arbitration services and the PHY data services.

The PHY arbitration services allow the link layer to gain control of the bus. The PHY arbitration request service is communicated from the link layer to the PHY layer to request control of the bus. The request contains information regarding the type of arbitration to perform. The PHY layer communicates back a PHY arbitration response service, which contains information regarding the success or failure of the request. Arbitration services are communicated in a manner that is independent of the PHY data services.

The PHY data services are used to communicate data symbols between the PHY and link layers. The PHY layer controls when data will be presented to the link layer, and it also controls when the link layer may present data to the PHY layer. The PHY layer communicates data to the link layer via the PHY data indication service. Each communication of this service presents one data symbol (a data bit or other bus symbol) to the link layer.

The flow of information in the other direction is a two-step process. First, the PHY layer allows the link layer to send one data symbol by communicating the PHY Clock indication service to the link layer. The link layer responds by communicating a PHY data request to the PHY layer, which communicates the data symbol to the PHY layer. The PHY Clock indication carries no information other than the permission to the link layer to send the data symbol.

An informative example of a real PHY-link interface is given in Annex J

6.3.1.2 Sending an asynchronous packet

The link layer sends an asynchronous packet when the transaction layer requests it via the link data request service. The link layer constructs the appropriate packet format and requests the PHY layer to arbitrate for the bus via the PHY arbitration request service. If the arbitration is lost, the link layer tries again at a later time; note that a lost arbitration is usually followed by an incoming packet to be received. After the arbitration is won, the link layer communicates the packet to be sent to the PHY layer via the PHY data request service. The packet ends when the link layer communicates a data end symbol to the PHY layer.

After the packet is sent, the link layer will usually expect an acknowledge to be returned. The acknowledge is communicated from the link layer to the transaction layer via the link data confirmation service. If the acknowledge is not valid, or if it is not received within a time-out period, the link layer considers the acknowledge missing. The link layer does not expect an acknowledge if the packet was a broadcast packet.

6.3.1.3 Receiving an asynchronous packet

The link layer receives an asynchronous packet when the PHY layer starts communicating data to the link layer via the PHY data indication service. The link layer detects the end of packet when the PHY layer communicates an ending symbol to the link layer. The link layer examines the packet length and packet components to determine whether the packet is valid and is addressed to this node. If the packet is not valid (e.g., the header CRC check failed) or is not

addressed to this node, the link layer ignores the packet. If the packet is valid and is addressed to this node, the link layer communicates the packet components to the transaction layer via the link data indication service.

After receiving a valid packet, the link layer will usually return an acknowledge to the sending node. The acknowledge code is specified to the link layer by the transaction layer via the link data response service. The link layer sends the acknowledge by requesting that the PHY layer do an immediate arbitration. The link layer then communicates the acknowledge to the PHY layer. The acknowledge ends when the link layer communicates a data end symbol to the PHY layer. The link layer does not send an acknowledge if the packet was a broadcast packet.

6.3.1.4 Sending an acknowledge concatenated to an asynchronous packet

The link layer may concatenate a response packet to the acknowledge described 6.3.1.3. This occurs when the transaction layer asks the link layer to hold the bus by setting Bus Occupancy Control in the link data response service to HOLD. In this case, the link layer ends the acknowledge by sending a data prefix to the PHY layer. This causes the PHY layer to hold the bus for sending another packet. After a delay time, and after the transaction layer communicates the response packet to the link layer via the link data request service, the link layer begins communicating the packet to the PHY layer, ending with a data end symbol. The link layer expects an acknowledge as described above.

6.3.1.5 Isochronous cycles

An isochronous cycle begins when the cycle master notes that a NOMINAL_CYCLE_TIME has elapsed since the last isochronous cycle began. The cycle master link layer then arbitrates and sends a cycle start packet as described in 6.3.3.3. The end of the cycle start packet begins an isochronous cycle.

During an isochronous cycle only isochronous packets shall be transmitted. Note that the arbitration rules prevent the transmission of any asynchronous packets during an isochronous cycle. Furthermore, between the time an isochronous cycle ends and the next isochronous cycle begins, no isochronous packets shall be transmitted.

An isochronous cycle ends when a subaction gap is detected by the PHY layer. Note that the transaction layer does not manage isochronous transfers.

6.3.1.6 Sending isochronous packets

An application should queue the transmission of isochronous packets when the link cycle synch indication is generated by the link layer. When there is data in this queue and a cycle start packet is received, the bus is requested using a PHY arbitration request with an arbitration class of ISOCHRONOUS. If the arbitration is lost, the link layer tries again at a later time; note that a lost arbitration is usually followed by an incoming packet to be received.

After the arbitration is won, the actual sending of the packet is the same as for an asynchronous packet: the packet is constructed in the appropriate format and is communicated to the PHY layer via the PHY data request service. The packet is ended when the link layer communicates a data end symbol to the PHY layer.

When the link layer has more than one isochronous packet to send during an isochronous cycle (sent to different channels), it shall arbitrate and send the first packet, and it then should concatenate subsequent isochronous packets until all channels are sent. This avoids the need to re-arbitrate for each packet. To concatenate, the link layer ends an isochronous packet by sending data prefix symbols to the PHY layer for a delay time; the link layer does not send a data end symbol. The link layer then communicates the next isochronous packet to the PHY layer. The last isochronous packet ends normally with a data end symbol.

There is no requirement to queue data packets during bus reset (between PHY events of BUS_RESET_START and BUS_RESET_COMPLETE). Data may be lost during these events, since this will always take more than a single 125 μ s cycle period (typically about 200 μ s). The application has to be able to tolerate missing data during this interval.

6.3.1.7 Receiving an isochronous packet

The link layer receives an isochronous packet when the PHY layer starts communicating data (via the PHY data indication service) to the link layer during an isochronous cycle. The link layer will receive isochronous packets if it has been configured as a listener by an application. The link layer detects the end of packet when the PHY layer communicates an ending symbol to the link layer. The link layer examines the packet length and packet components to determine whether the packet is valid and is addressed to a channel to which this node is listening. If the packet is not valid (e.g., the header CRC check failed) or is not addressed to an active channel on this node, the link layer ignores the packet. If the packet is valid and is addressed to a channel on this node, the link layer communicates the packet components to the appropriate application via the link isochronous indication service.

After receiving a packet, the link layer does not return an acknowledge to the sending node.

6.3.2 Cycle synch event

If the link layer is capable of sending or receiving isochronous packets or is capable of being the cycle master, it shall generate a local cycle synch event. The generation of the cycle synch event is done using the procedure described by the following c-like code:

Table 6.14—Cycle synch event code

```

boolean cycle_synch_queued; // the cycle synch event has occurred

struct
{
    seconds_count : 7, // count of seconds
    cycle_count : 13, // count of cycles (1/8000) second
    cycle_offset : 12 // count of 1/24.576 microsecond periods after last event
} CYCLE_TIME;

generate_cycle_synch_event(); // invoked every 1/24.576 microseconds
{
    if (cycle_offset == 3071) // 125 microseconds have elapsed
    {
        cycle_offset = 0;
        if (cycle_count == 7999)
        {
            seconds_count = seconds_count + 1;
            cycle_count == 0;
        }
    }
    else
    {
        ++cycle_count;
    }
    LK_CYCLE.ind(); // notify the application of a cycle synch
    if (STATE_SET.cmstr) // if the node is the cycle master
        cycle_synch_queued = TRUE ; // queue the cycle synch event
}
else
{
    ++cycle_offset;
}
} // end generate_cycle_synch_event

```


The procedure `generate_cycle_synch_events` is invoked every $1/24.576 \mu\text{s}$ (± 100 ppm). The variable `cycle_synch_queued` is used to queue the cycle start for the link layer packet transmit/receive state machine described in 6.3.3. The structured variable `CYCLE_TIME` is the `CYCLE_TIME` register described in 8.3.2.2.8 and the `STATE_SET.cmstr` bit is described in 8.3.1.6.

The `CYCLE_TIME` register is also set by reception of a cycle start packet as described in 6.2.2.2.3.

6.3.3 Details of link layer operation

The operation of the link layer packet transmitter and receiver is described by the state machine in figure 6.19.

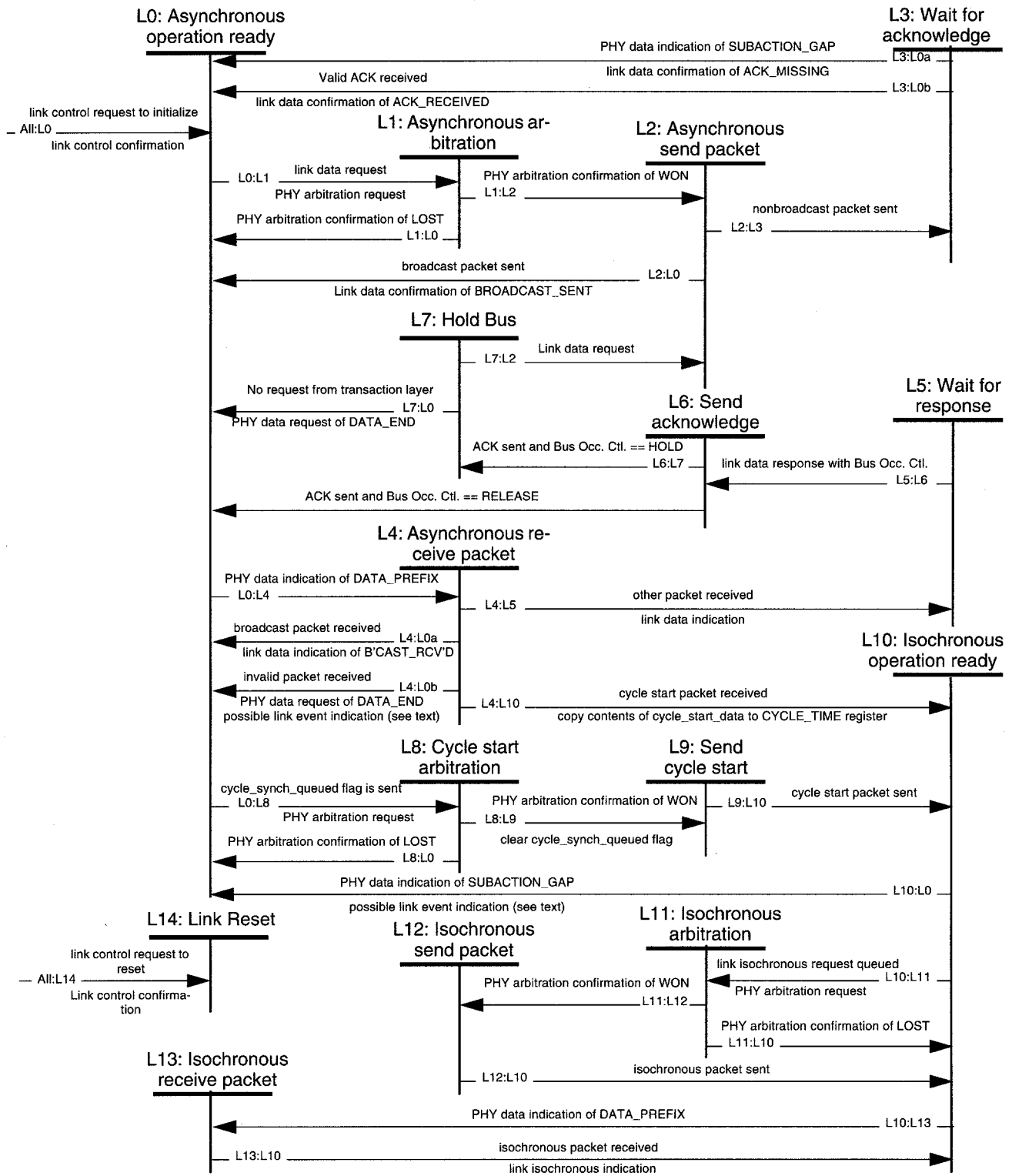


Figure 6.19—Link layer packet transmit/receive state machine

6.3.3.1 Link initialization

Transition A11:L0. The link layer shall transition from any other link layer state to state L0 when a link Control request with an action of Initialize is communicated from the node controller.

Transition A11:L14. The link layer shall transition from any other link layer state to state L14 when a link Control request with an action of Reset is communicated from the node controller.

State L14. The link layer is suspended. The link layer shall only leave state L14 via transition A11:L0.

6.3.3.2 Asynchronous operation

State L0: Asynchronous operation ready. The link layer is ready to send or receive asynchronous packets. The link layer shall not send or receive isochronous packets. Any link isochronous requests received by the link layer shall be deferred until the link layer enters state L10.

Transition L0:L1. The link layer shall construct the appropriate asynchronous packet format (as described in 6.2) based on parameters specified by the transaction layer in the link data request. The link layer shall communicate a PHY arbitration request to the PHY layer, using the arbitration class specified by the transaction layer.

Transition L0:L4. The link layer shall begin receiving data from the PHY layer when it receives a PHY data indication with data of DATA_START.

Transition L0:L8. An isochronous cycle shall only be requested by a cycle master. When the link layer of the cycle master queues a local cycle synch event (see 6.3.2), the link layer shall communicate a PHY arbitration request to the PHY layer, using an arbitration class of CYCLE_MASTER

State L1: Asynchronous arbitration. The link layer shall wait for a PHY arbitration confirmation from the PHY layer.

Transition L1:L0. The link layer receives a PHY arbitration confirmation. If the arbitration request status is LOST, the link layer may re-arbitrate at a later time, subject to the rules for the arbitration method used.

NOTE — A lost arbitration is usually followed by an incoming packet to be received by the link layer, as described in Transition L0:L4.

Transition L1:L2. The link layer receives a PHY arbitration confirmation. If the arbitration request status is WON, the link layer shall immediately begin communicating PHY data requests to the PHY layer following each PHY Clock indication received by the link layer.

State L2: Asynchronous send packet. The link layer shall continue sending PHY data requests until all of the packet is transmitted.

Transition L2:L0. The link layer shall end the broadcast packet by communicating a PHY data request with data of DATA_END to the PHY layer. The time between the first PHY arbitration confirmation of WON and the final PHY data request of DATA_END shall be no greater than a time of MAX_BUS_OCCUPANCY. When the transaction layer specifies the asynchronous packet as a broadcast packet, the link layer shall not expect an acknowledge packet to be returned by a destination node. The link layer shall communicate a link data confirmation with a request status of BROADCAST_SENT to the transaction layer.

Transition L2:L3. The link layer shall end the nonbroadcast packet by communicating a PHY data request with data of DATA_END to the PHY layer. The time between the first PHY arbitration confirmation of WON and the final PHY data request of DATA_END shall be no greater than a time of MAX_BUS_OCCUPANCY. When the transaction layer

specified the asynchronous packet was not a broadcast packet, then the link layer shall expect an acknowledge packet to be returned by the destination node.

State L3: Wait for acknowledge. The link layer shall wait for either a PHY data indication of SUBACTION_GAP or a PHY data indication of DATA_START. The link layer shall begin receiving the acknowledge packet from the PHY layer when it receives a PHY data indication with data of DATA_START. The link layer shall detect the end of the acknowledge packet when it receives a PHY data indication with data of DATA_END or DATA_PREFIX. If the packet length was one byte, and the ack_code field EXCLUSIVE-ORed with the ack_parity field equal to 11112, the packet is an acknowledge packet. Otherwise, the acknowledge packet shall be ignored by the link layer.

Transition L3:L0a. If the link layer receives a PHY data indication with a data of SUBACTION_GAP before receiving an acknowledge packet, the link layer shall communicate a link data confirmation with request status of ACKNOWLEDGE_MISSING to the transaction layer.

Transition L3:L0b. If the link layer receives an acknowledge packet, it shall communicate a link data confirmation with a request status of ACKNOWLEDGE_RECEIVED, and it shall communicate the appropriate acknowledge to the transaction layer (see table 6.13).

State L4: Asynchronous Receive packet. The link layer shall continue to receive the packet until it detects the end of the packet. The end of the packet shall be detected when the link layer receives a PHY data indication with data of DATA_END or DATA_PREFIX. The link layer shall examine the length of the packet, the transaction code field, and the data length field to determine the packet format so that the header and data CRC can be calculated and checked.

The link layer should communicate the PHY arbitration request (with an arbitration class of IMMEDIATE) as soon as possible after it verifies that the destination_ID in the packet header addresses this node (see transition L4:L5 below), in anticipation of sending an acknowledge packet in state L6. The PHY layer shall respond with a PHY arbitration confirmation of WON.

NOTE — The arbitration should be done while the packet is being received to reduce the latency between the end of a packet and the acknowledge, and to meet the acknowledge_delay requirement.

Transition L4:L0a. If the destination_ID field indicates that this is a broadcast packet (see 6.2.4.2.1), the link layer shall consider the transaction complete and perform the requested action. If the destination_ID indicates the node shall not accept the packet, the link layer shall ignore the packet.

Transition L4:L0b. If the packet length is 1 byte, the packet is an acknowledge packet and shall be ignored by the link layer. If the transaction code indicates that the packet is an isochronous data block packet, or that the header CRC check failed, the packet shall be ignored by the link layer.

If the header CRC check failed, or an unknown or unexpected transaction code (such as isochronous data block) is detected, the link layer shall communicate the appropriate link event indication to the transaction layer. If the received packet is longer than allowed by the MAX_BUS_OCCUPANCY time, and if the link layer is capable of detecting this, the link layer shall communicate the appropriate link event indication to the transaction layer.

If the link layer had requested an immediate arbitration (see state L4 above), it shall communicate a PHY data request of DATA_END to cause a release of the bus.

NOTE — In state L4, the link layer requested the PHY layer to gain immediate control of the bus in anticipation of sending an acknowledge. Since the link layer cannot send an acknowledge in this transition, it shall direct the PHY layer to release the bus immediately.

If the destination_ID indicates that the node shall not accept the packet (see 6.2.4.2.1), the packet shall be ignored by the link layer.

Transition L4:L5. If the packet length is a multiple of one quadlet, and the header CRC checks, the link layer shall interpret the packet as a primary packet. If the destination_ID indicates that the node shall accept the packet (see 6.2.4.2.1), and if the transaction code indicates that the packet is not a cycle start packet or an isochronous data block packet, the link layer shall communicate a link data indication to the transaction layer.

Transition L4:L10. If the packet is a cycle start packet, the link layer shall enter an isochronous cycle. The link layer shall immediately update the value of the CYCLE_TIME register to the quadlet contained in the data field of the received packet.

State L5: Wait for response (from transaction layer). The link layer shall wait for a link data response to be communicated from the transaction layer. The transaction layer shall communicate the link data response such that the time from the end of the received packet to the data prefix of the acknowledge packet on the bus is no greater than an ACK_RESPONSE_TIME plus the appropriate arb_delay. The link layer shall continue to hold the bus by communicating PHY data requests with data of DATA_PREFIX to the PHY layer following each PHY Clock indication received by the link layer.

Transition L5:L6. When the transaction layer communicates a link data response, the link layer shall immediately send an acknowledge packet with the acknowledge code specified in the service.

State L6: Send acknowledge. When the link layer receives a link data response from the transaction layer, the link layer shall send the acknowledge by immediately communicating PHY data requests to the PHY layer following each PHY Clock indication received by the link layer. The link layer shall continue sending PHY data requests until all of the acknowledge packet is transmitted.

The link layer may hold the bus by communicating PHY data requests with data of DATA_PREFIX prior to sending the acknowledge packet data.

Transition L6:L0. If the Bus Occupancy Control parameter of the link data response is RELEASE, the link layer shall end the acknowledge packet by communicating a PHY data request with data of DATA_END to the PHY layer. The time between the end of the received packet and the final PHY data request of DATA_END of the acknowledge shall be no greater than a time of DATA_END_TIME. The link layer shall consider the subaction complete.

Transition L6:L7. If the Bus Occupancy Control parameter of the link data response is HOLD, the link layer shall end the acknowledge packet by communicating a PHY data request with data of DATA_PREFIX to the PHY layer. The time between the first PHY arbitration confirmation of WON and the final PHY data request of DATA_PREFIX shall be no greater than a time of DATA_PREFIX_TIME. The link layer shall consider the subaction complete.

State L7: Hold Bus (for Concatenated response). The link layer shall wait for a link data request to be communicated from the transaction layer. The transaction layer should communicate the link data request such that the time from the end of acknowledge packet to the first data symbol of the response packet on the bus is no greater than a data_prefix_time.

An acknowledge packet may be concatenated to a single following asynchronous response packet type, provided the response packet is the response associated with the acknowledge packet. In other words, a response packet for a transaction shall not be concatenated with an acknowledge packet for a different transaction. Another packet shall not be concatenated following the response packet.

Transition L7:L2. When the transaction layer communicates a link data request, the link layer shall immediately send a response packet.

Transition L7:L0. When the transaction layer fails to communicate link data request within a data_prefix_time, the link layer shall release the bus by communicating a PHY data request with data of DATA END to the PHY layer.

NOTE — When the link data request finally arrives, the link layer treats it as a new subaction that completes a split transaction.

6.3.3.3 Isochronous operation

State L8: Cycle start arbitration. The link layer shall wait for a PHY arbitration confirmation from the PHY layer.

Transition L8:L0. The link layer receives a PHY arbitration confirmation of LOST; the link layer may rearbitrate at a later time.

NOTE — If arbitration were instantaneous, the cycle master would never lose cycle start arbitration. However, since arbitration does take a finite amount of time, this state transition allows implementations to present the LOST status to the link if a grant occurs at the same time the link asks for a cycle start. The lost arbitration is usually followed by an incoming packet to be received by the link layer.

Transition L8:L9. The link layer receives a PHY arbitration confirmation of WON and shall immediately begin communicating PHY data requests to the PHY layer following each PHY Clock indication received by the link layer.

State L9: Send cycle start. The link layer shall continue sending PHY data requests until all of the cycle start packet is transmitted. The `cycle_time_data` field transmitted shall be the value of the `CYCLE_TIME` register at the instant that transition L8:L9 is taken.

Transition L9:L10. The link layer shall end the cycle start packet by communicating a PHY data request with data of DATA END to the PHY layer. The time between the first PHY arbitration confirmation of WON and the final PHY data request of DATA_END shall be no greater than a time of `MAX_BUS_OCCUPANCY`. The link layer shall not expect an acknowledge packet to be returned by a destination node. The link layer shall not confirm to any layer that the packet was sent.

State L10: Isochronous operation ready. The link layer shall consider that an isochronous cycle has begun immediately after the request for cycle start packet is sent (if the node is a cycle master) or when the request for cycle start packet is received (all other nodes). The link layer is ready to send or receive isochronous packets. The link layer shall not send or receive asynchronous packets. Any link data requests received by the link layer shall be deferred until the link layer enters the L0 state.

Transition L10:L0. When the link layer receives a PHY data indication with a data of `SUBACTION_GAP`, the link layer shall consider the isochronous cycle ended. If the time between transition L9:L10 or L4:L10 and L10:L0 is greater than `NOMINAL_CYCLE_TIME`, then a link event indication of `CYCLE_TOO_LONG` shall be generated.

Transition L10:L11. When any link isochronous requests are pending, the link layer shall construct the appropriate isochronous packet format (as described in 6.2) based on parameters specified by the application layer in the request. The link layer shall communicate a PHY arbitration request to the PHY layer with an arbitration class of `ISOCRONOUS`.

NOTE — Implementations are free to do any kind of queuing of isochronous requests that makes sense for the application, as long as the bandwidth allocation (see 8.4.2.6) is not exceeded. A fully general implementation might queue requests between link cycle synch indications for transmission during the following isochronous cycle. Since the cycle start can be delayed almost a full `NOMINAL_CYCLE_TIME` from the preceding cycle synch, this implies that isochronous data requests may have to be queued for up to two cycles.

Transition L10:L13. The link layer shall begin receiving data from the PHY layer when it receives a PHY data indication with data of `DATA_START`.

State L11: Isochronous arbitration. The link layer shall wait for a PHY arbitration confirmation from the PHY layer.

Transition L11:L10. The link layer receives a PHY arbitration confirmation. If the arbitration request status is LOST, the link layer may rearbitrate at a later time within the isochronous cycle.

Transition L11:L12. The link layer receives a PHY arbitration confirmation. If the arbitration request status is WON, the link layer shall immediately begin communicating PHY data requests to the PHY layer following each PHY Clock indication received by the link layer.

State L12: Isochronous send packet(s). The link layer shall continue sending PHY data requests until all of the isochronous packet is transmitted. If the link layer has another isochronous packet to send, the link layer shall end the current packet by communicating a PHY data request with data of DATA_PREFIX to the PHY layer. The link layer shall communicate DATA_PREFIX for a data_prefix_delay. The link layer shall then send PHY data requests until all of the concatenated isochronous packet is transmitted. This procedure is repeated until all isochronous packets are sent.

Transition L12:L10. If the current isochronous packet is the last packet to be sent, then the link layer shall end the packet by communicating a PHY data request with data of DATA_END to the PHY layer. The link layer shall not expect an acknowledge packet to be returned by a destination node. The link layer shall not confirm to the application layer that the packet was sent.

State L13: Isochronous Receive packet. The link layer shall continue to receive the packet until it detects the end of the packet. The end of the packet shall be detected when the link layer receives a PHY data indication with data of DATA END or DATA PREFIX. The link layer shall examine the length of the packet, the transaction code field, and the data length field for purposes of calculating header and data CRC.

Transition L13:L10. If the transaction code was not the value for an isochronous data block packet, the packet shall be ignored by the link layer. The rules of 6.2.4.1 also apply here. If the channel value does not match one of the values on the channel receive list (see 6.1.1.1), the packet shall be ignored by the link layer. When the link layer receives an isochronous packet with a channel value that matches one of the values on the channel receive list during an isochronous cycle, the link layer shall communicate a link isochronous indication to the application layer. If the channel value is not in the expected channel list (see 6.1.1.1) *or* is a duplicate of a channel number that has already been received this cycle, and if the node is the isochronous resource manager, the link layer shall communicate the appropriate link event indication to the node controller.

6.4 Link layer reference code

The following code is an example implementation of the CRC generation and checking algorithm used by the link layer. It is informative in nature only.

Table 6.15—CRC generation and checking

```

#define MSB32          ((unsigned) 1<<31)
#define ONES32        0xFFFFFFFF
#define CRC_COMPUTE   ((Quadlet) 0X04C11DB7)
#define CRC_RESULTS   ((Quadlet) 0XC704DD7B)

typedef unsigned long Quadlet // where "long" is a 32-bit value
// Generate the CRC for a packet containing "sizeInQuads" quadlet data values
Quadlet GenerateCrc (Quadlet * inputs, int sizeInQuads)
{
    Quadlet crcSum;

    assert (sizeInQuads >= 1); // size of packet including CRC

    crcSum = CalculateCrc(inputs, sizeInQuads - 1);
                                // compute CRC on just the data
    return (~crcSum);
}
// Validate the CRC for a packet containing "size" quadlet data values
int ValidateCrc (Quadlet * inputs, int sizeInQuads)
{
    Quadlet crcSum;

    assert (sizeInQuads >= 1); // size of packet including CRC

    crcSum = CalculateCrc (inputs, sizeInQuads);
                                // compute CRC on the data *and* the received CRC
    return (crcSum != CRC_RESULTS);
}

// The GenerateCrc() function points to protected values,
// it checks these values and return a final 32-bit result
Quadlet CalculateCrc (Quadlet * inputs, int sizeInQuads)
{
    Quadlet inQuad, crcSum, newMask;
    int i, oldBit, newBit, sumBit;

    // The crcSum value is initialized to all ones
    crcSum = (Quadlet) 0xFFFFFFFF;

    // Process each of the quadlets covered by the CRC value
    for (i = 0, i < sizeInQuads; i += 1)
    {
        inQuad = inputs[i];

        // Process each of the bits within the input quadlet value
        for (newMask = MSB32; newMask != 0; newMask >>= 1)
        {
            newBit = ((inQuad & newMask) != 0); // The next input bit
            oldBit = ((crcSum & MSB32) != 0); // and MSB of crcSum
            sumBit = oldBit ^ newBit; // are EXOR'd together
            // Shift the old crcSum left and exclusive-OR the new newBit values
            crcSum = ((crcSum < 1) & ONES32) ^ (sumBit ? CRC_COMPUTE : 0); // "^" is X-OR
        }
    }

    return (crcSum);
}

```


7. Transaction layer specification

The transaction layer provides three operations for the transfer of data between nodes:

- a) Write transaction—Data is transferred to an address in a different node.
- b) Read transaction—Data is retrieved from an address in a different node.
- c) Lock transaction—Data is sent to a different node, used to perform an indivisible function, and the results returned.

The transaction layer services may be multithreaded at the option of the implementor. This means that more than one transaction may be in process (pending) at the same time.

7.1 Transaction layer services

Transaction layer services are provided at the interface between the transaction layer and higher layers, as illustrated in table 7.1.

Table 7.1—Summary of transaction layer services

Service	Layer communicated with	Purpose of service
Transaction control request	From the node controller	Configure the transaction layer
Transaction control confirmation	To the node controller	Confirm transaction control request
Transaction event indication	To the node controller	Alert node controller to events detected in the transaction layer
Transaction data request	From the application, node controller, or bus manager	Cause the transaction layer to initiate a transaction
Transaction data confirmation	To the application, node controller, or bus manager	Confirm transaction data request
Transaction data indication	To the application, node controller, or bus manager	Indicate the reception of a transaction request
Transaction data response	From the application, node controller, or bus manager	Respond to transaction data indication

The method by which these services are communicated between the layers is not defined by this standard. Transaction layer services may perform actions specified by the higher layer. Transaction layer services may also communicate parameters that may or may not be associated with an action.

7.1.1 Transaction layer bus management services for Serial Bus management

These services are used by the node controller to control the resources of the transaction layer. The transaction layer uses these services to communicate events within the transaction layer or on the bus to the node controller.

7.1.1.1 Transaction control request (TR_CONTROL.request)

The node controller uses this service to request the transaction layer to perform specific actions and to specify transaction layer parameters. The transaction layer shall service the request immediately upon receipt by the transaction layer. This service is confirmed.

The following actions defined for all transaction layer implementations shall be provided by this service:

- Reset. The transaction layer shall discard all pending transactions. No transaction requests shall be accepted from the link layer or from any applications.
- Initialize. The transaction layer shall discard all pending transactions. Transaction requests shall be accepted from the link layer and from applications.

The following parameters are communicated to the transaction layer via this service:

- Split transaction timeout value. This parameter shall contain the maximum time after sending a transaction request to the link layer (via a link data request) that a response will be accepted from the link layer (via a link data indication).
- Transaction retry limit. This parameter shall contain the maximum number of times transmission of a transaction request packet or transaction response packet (via a link data request) will be retried upon receiving a busy acknowledge (via a link data confirmation).

7.1.1.2 Transaction control confirmation (TR_CONTROL.confirmation)

The transaction layer uses this service to confirm the results of a transaction control request service. The transaction layer shall communicate this service to the node controller upon completion of a transaction control request. There are no actions provided by this service. No parameters are communicated to the node controller via this service.

7.1.1.3 Transaction event indication (TR_EVENT.indication)

The transaction layer uses this service to indicate to the node controller events detected by the transaction layer. There are no actions provided by this service. No response is defined for this indication. The following parameters are communicated to the node controller via this service:

- Transaction event. This parameter shall contain an event detected by the transaction layer. The following values are defined for this parameter:
 - RESPONSE DATA ERROR. A data error acknowledge was returned to a transaction response sent by this node.
 - RESPONSE FORMAT ERROR. A type error acknowledge was returned to a transaction response sent by this node.
 - REQUEST DATA ERROR. A data error was detected in a transaction request received by this node, and a data error response was returned.
 - ACKNOWLEDGE MISSING. No acknowledge was returned for a transaction response sent by this node.
 - UNSOLICITED RESPONSE. A transaction response was received, but there was no request pending for the transaction label and destination contained in the response (see 6.2.4.3).
 - RESPONSE RETRY LIMIT. A transaction response could not be delivered within the number of retries specified by the transaction retry limit.

7.1.2 Transaction layer data services for applications and bus management

These services are used to communicate data transactions between the transaction layer and applications or bus management. See clause 7.3.5.2.1.

7.1.2.1 Transaction data request (TR_DATA.request)

Applications or bus management use this service to initiate a data transaction on the bus. This service shall be confirmed.

The following parameters are communicated to the transaction layer via this service:

- a) Transaction type. This parameter shall contain the type of transaction to be performed. The following values are defined for this parameter:
 - 1) WRITE. A write transaction shall be performed to the destination address. See 7.3.3.2.
 - 2) READ. A read transaction shall be performed from the destination address. See 7.3.3.1.3.
 - 3) LOCK. A lock transaction shall be performed to the destination address. See 7.3.3.2.1.
- b) Extended transaction code. As defined in 6.2.4.9. This parameter is only defined for lock transactions. This parameter is reserved for read transactions and write transactions.
- c) Destination address. As defined in 6.2.4.2.
- d) Data length. As defined in 6.2.4.8.
- e) Data. This is the data to be sent for a write transaction or a lock transaction.
- f) Priority. As defined in 6.2.4.6.
- g) Speed (CABLE ONLY). As defined in table 4.2.

7.1.2.2 Transaction data confirmation (TR_DATA.confirmation)

The transaction layer uses this service to confirm the completion of a transaction. The transaction layer shall communicate this service to the requesting application, node controller, or bus manager upon completion of a transaction data request. There are no actions provided by this service. The following parameters are communicated to the appropriate layer via this service:

- a) Request status. This parameter shall contain the result of the requested transaction. The following values are defined for this value:
 - 1) COMPLETE. The transaction was successfully completed.
 - 2) TIMEOUT. The transaction request was sent, but a transaction response was not received within the period specified by the transaction timeout value (see 7.1.1.1).
 - 3) ACKNOWLEDGE MISSING. No acknowledge was received to the transaction request.
 - 4) RETRY LIMIT. The transaction request could not be delivered within the number of retries specified by the transaction retry limit.
 - 5) DATA ERROR. A data error was detected in data received during the transaction.
- b) Response code. As defined in 6.2.4.10. The response code is valid only if the request status is COMPLETE or DATA_ERROR.
- c) Data. This is the data received if the transaction was a read transaction or a lock transaction.
- d) Data length. As defined in 6.2.4.8, only if data is present.

NOTE — It is the responsibility of the application (or bus management) to associate the transaction data request with the appropriate transaction data response.

7.1.2.3 Transaction data indication (TR_DATA.indication)

The transaction layer uses this service to indicate to the application, node controller, or bus manager that a transaction request has been received. There are no actions provided by this service. The application, node controller, or bus manager shall respond to this indication. The transaction layer shall communicate this indication to the application, node controller, or bus manager whenever a transaction request has been received.

The following parameters are communicated to the application, node controller, or bus manager via this service:

- a) Transaction type. This parameter shall contain the type of transaction to be performed. The following values are defined for this parameter:
 - 1) WRITE. A write transaction shall be performed to the destination address. See 7.3.3.2.
 - 2) BROADCAST WRITE. A write transaction shall be performed to the destination address. See 7.3.3.2 and 7.3.2.4.
 - 3) READ. A read transaction shall be performed from the destination address. See 7.3.3.1.3.
 - 4) LOCK. A lock transaction shall be performed to the destination address. See 7.3.3.2.1.
- b) Destination address. As defined in 6.2.4.2.

- c) Requester ID. This parameter gives the node_ID of the requesting node.
- d) Extended transaction code. As defined in 6.2.4.5. This parameter is only defined for lock transactions. This parameter is reserved for read transactions and write transactions.
- e) Transaction label. As defined in 6.2.4.3.
- f) Data length. As defined in 6.2.4.8.
- g) Data. This is the data received.
- h) Priority. As defined in 6.2.4.6.
- i) Speed (*cable only*). As defined in table 4.2.

7.1.2.4 Transaction data response (TR_DATA.response)

The transaction layer uses this service to respond to a received packet; i.e., to complete the transaction. The application or bus management shall communicate this response to the transaction layer after receiving a transaction data indication.

The following parameters are communicated to the transaction layer via this service:

- a) Transaction type. This parameter shall contain the type of transaction to be completed. The following values are defined for this parameter:
 - 1) WRITE. A write transaction was performed to the destination address. See 7.3.3.2.
 - 2) BROADCAST WRITE. A write transaction was performed to the destination address. See 7.3.3.2 and 7.3.2.4. Note that no response packet is expected. All other parameters communicated by this service are invalid for this transaction type.
 - 3) READ. A read transaction was performed from the destination address. See 7.3.3.1.3.
 - 4) LOCK. A lock transaction was performed to the destination address. See 7.3.3.2.1.
- b) Transaction label. As defined in 6.2.4.3.
- c) Extended transaction code. As defined in 6.2.4.9. This parameter is only defined for lock transactions. This parameter is reserved for read transactions and write transactions.
- d) Requester ID. This parameter gives the node_ID of the requesting node.
- e) Response code. As defined in 6.2.4.10.
- f) Data length. As defined in 6.2.4.8, only if data is present.
- g) Data. This is the data to be sent if the transaction was a read transaction or a lock transaction.
- h) Priority. As defined in 6.2.4.6.
- i) Speed (*cable only*). As defined in table 4.2.

7.2 Transaction facilities

Transactions are constructed of link layer packets, with one request packet and (in many cases) one response packet. These packets have many fields in common: a request/response header, requester address, responder address, and optional data.

7.2.1 Split transaction timer

The split transaction timer keeps track of each pending transaction. All transactions shall have a default split transaction timeout value of 100 ms. The split transaction timeout value may be changed in those nodes that implement the SPLIT_TIMEOUT control register in the node controller. The timeout period is measured from the last data symbol on the bus of the request subaction to the first data symbol on the bus for the response subaction. If the response subaction has not been received by the node when the timeout period is exceeded, the transaction fails.

NOTE — The initiating node should make allowances for any header decode delay following the receipt of the first response subaction data symbol.

7.2.2 Transaction retry limit

The BUSY_TIMEOUT control register limits the number of times each request subaction is sent to its destination node. If the single-phase retry protocols are being used, a field within this register limits the number of times the subaction is retried; the retries use the same retry_X label, and no reservations are assigned. The transaction fails if the retry limit is exceeded, although recovery at a higher level is expected.

If the (optional) dual-phase retry protocols are being used, distinct fields within this register limit the time interval during which the retries are performed. If the retry time interval is exceeded, the transaction fails, and no higher level recovery is expected.

7.3 Transaction operation

This subclause defines how the transaction layer initiates and (if necessary) retries transaction requests, and how the transaction layer reacts to transaction requests. This subclause makes reference to the link layer services described in clause 6..

7.3.1 Overview of transaction layer operations

This subclause gives an informal overview of transaction layer operations. The descriptions in this subclause are informative and nonbinding. The formal descriptions of behavior are in 7.3.2 and 7.3.3.2.2.

7.3.1.1 Read transactions

A node originates a read transaction when the bus management or the application layer above the transaction layer communicates a transaction data request with a transaction type of READ. The request contains the destination address and the data length for the read. The transaction layer begins the transaction by communicating a link data request to the link layer. The link data confirmation completes the request subaction of the read transaction at the source node.

The request subaction causes a link data indication to be communicated to the transaction layer in the destination node. The transaction layer communicates a link data response, as appropriate for how the transaction is to be completed (see 7.3.2), to complete the request subaction at the destination node. The transaction layer communicates a transaction data indication to the destination application layer, announcing the arrival of a read request.

The response subaction of the read transaction begins when the application layer at the destination node completes the requested read operation and communicates a transaction data response. The response contains the read data, the data length, the node_ID of the source node, the transaction label of the corresponding request, and the response code. The transaction layer communicates a link data request to the link layer. The link data confirmation completes the response subaction of the read transaction at the destination node.

The response subaction causes a link data indication to be communicated to the transaction layer in the source node. The transaction layer communicates a link data response, as appropriate for the type of transaction, to complete the request subaction at the source node. The transaction layer communicates a transaction data confirmation to the appropriate source application, completing the read transaction.

A read transaction may be completed without a response subaction, but this only occurs if there was an error in the request subaction.

7.3.1.2 Write transactions

A node originates a write transaction when the bus management or the application layer above the transaction layer communicates a transaction data request with a transaction type of WRITE. The request contains the destination

address, the write data, and the data length for the write. The transaction layer begins the transaction by communicating a link data request to the link layer. The link data confirmation completes the request subaction of the write transaction at the source node.

The request subaction causes a link data indication to be communicated to the transaction layer in the destination node. The transaction layer communicates a link data response, as appropriate for how the transaction is to be completed (see 7.3.2), to complete the request subaction at the destination node. The transaction layer communicates a transaction data indication to the destination application layer, announcing the arrival of a write request.

The response subaction of the write transaction begins when the application layer at the destination node completes the requested write operation and communicates a transaction data response. The response contains the node_ID of the source node, the transaction label of the corresponding request, and the response code. The transaction layer communicates a link data request to the link layer. The link data confirmation completes the response subaction of the write transaction at the destination node.

The response subaction causes a link data indication to be communicated to the transaction layer in the source node. The transaction layer communicates a link data response, as appropriate for the type of transaction, to complete the request subaction at the source node. The transaction layer communicates a transaction data confirmation to the appropriate source application, completing the write transaction.

A write transaction may also be completed without a response subaction. The write transaction begins as described above with a request subaction. This time, however, the transaction data response at the destination node is communicated before the transaction layer communicates the link data response. The response code in the transaction data response is used to create the acknowledge code for the link data response. The link data response completes the write transaction at the destination node. The value of the acknowledge code received by the source node causes the source transaction layer to communicate a transaction data confirmation to the source application layer, completing the write transaction.

7.3.1.3 Lock transactions

A node originates a lock transaction when the bus management or the application layer above the transaction layer communicates a transaction data request with a transaction type of LOCK. The request contains the destination address, data, any argument required for the lock operation, the data length of the arguments and data, and the lock function to be performed. The transaction layer begins the transaction by communicating a link data request to the link layer. The link data confirmation completes the request subaction of the lock transaction at the source node.

The request subaction causes a link data indication to be communicated to the transaction layer in the destination node. The transaction layer communicates a link data response, as appropriate for how the transaction is to be completed (see 7.3.2), to complete the request subaction at the destination node. The transaction layer communicates a transaction data indication to the destination application layer, announcing the arrival of a lock request.

The response subaction of the lock transaction begins when the application layer at the destination node completes the requested lock operation and communicates a transaction data response. The response contains the old data value, the old data length, the node_ID of the source node, the transaction label of the corresponding request, and the response code. The transaction layer communicates a link data request to the link layer. The link data confirmation completes the response subaction of the lock transaction at the destination node.

The response subaction causes a link data indication to be communicated to the transaction layer in the source node. The transaction layer communicates a link data response, as appropriate for the type of transaction, to complete the request subaction at the source node. The transaction layer communicates a transaction data confirmation to the source application layer, completing the lock transaction.

A lock transaction may be completed without a response subaction, but this only occurs if there was an error in the request subaction.

7.3.1.4 Error handling

The transaction layer handles error conditions in certain cases, and expects the application to handle errors in other cases. This subclause is intended as a brief survey of error conditions and how they are handled.

Some of the errors handled by the transaction layer are enumerated in the definition of the transaction event indication (see 7.1.1.3). These errors are handled by the transaction layer because the transaction layer should not pass the bad information from an inbound request to the application (REQUEST DATA ERROR), the transaction layer has no active context for an inbound response (UNSOLICITED RESPONSE), or the error occurs after transaction layer has completed an inbound transaction with the application (RESPONSE DATA ERROR, RESPONSE FORMAT ERROR, ACKNOWLEDGE MISSING, RESPONSE RETRY TIMEOUT).

All other errors on inbound transaction requests are expected to be detected and handled by the application via the response code in the transaction data response.

Errors on outbound transaction requests are reported to the application via the transaction data confirmation in the request status and response code parameters. These errors are handled by the application, perhaps by retrying the transaction request, or via some other higher level recovery protocol.

7.3.2 Transaction completion definitions

This subclause defines the various ways in which a transaction can be completed. Note that the responding node determines how a transaction is completed. The method by which a responding transaction layer determines how to complete a given transaction is not defined by this standard.

NOTE — The method of completing a transaction requires agreement between the transaction layer and application layers. One method is to hardwire the decision based on the destination address. For example, a write to a hardware control register can complete as a unified transaction with minimum bus overhead. On the other hand, a large read of a ROM space that is seldom accessed may be best achieved via a split transaction, allowing the application the opportunity to schedule the transaction with a lower priority. Another method is to implement a timer, which can trigger the completion decision, depending on how fast the application can respond.

7.3.2.1 Unified transaction

A unified transaction is defined as a transaction that begins with an acknowledged request subaction but is not followed by a response subaction. Only write transactions may normally complete as unified transactions. Read transactions and lock transactions do not normally complete as unified transactions.

7.3.2.2 Split transaction

A split transaction is defined as a transaction that begins with an acknowledged request subaction and is followed by an acknowledged response subaction. Read transactions, write transactions, and lock transactions may be split transactions. Other subactions may occur between the request subaction and the response subaction.

7.3.2.3 Concatenated transaction

A concatenated transaction is defined as a transaction that begins with an acknowledged request subaction and is followed immediately by the corresponding acknowledged response subaction, with no subaction gap between. Read transactions, write transactions, and lock transactions may be concatenated transactions. No other subactions shall occur between the request subaction and the response subaction of a concatenated transaction.

7.3.2.4 Broadcast transaction

A broadcast transaction is defined as a transaction that contains only an unacknowledged request subaction. Only write transactions may be broadcast transactions. Read transactions and lock transactions shall not be broadcast transactions.

7.3.2.5 Pending transaction

A pending transaction is defined as a transaction that is not yet completed. For a unified transaction, the transaction completes when the request subaction has been acknowledged. For a broadcast transaction, the transaction completes when the request subaction has been sent. For all other transactions, the transaction completes when the response subaction has been acknowledged. A transaction is also considered complete when the response has not been received within a period specified by the SPLIT_TIMEOUT register.

The transaction layer may optionally initiate other transactions and react to transactions while a transaction is pending.

7.3.3 Details of transaction layer operation

The operation of the transaction layer with respect to remote nodes is described by the state machines in figures 7.1 and 7.2. Also, the reader may also wish to refer back to the discussion of transaction types in clause 1.3. The two state machines described operate in concert to originate outbound transactions and process inbound transaction requests.

This standard does not require that transaction requests to the local node force the generation of link-layer packets; the transaction layer may directly handle read, write, and lock transactions to local addresses. In other words, if an application requests a write to the local STATE_SET register, then that action can take place without any data packets appearing on the physical bus.

7.3.3.1 Outbound transaction state machine

The state machine in figure 7.1 describes the operation of the transaction layer when originating an outbound transaction request.

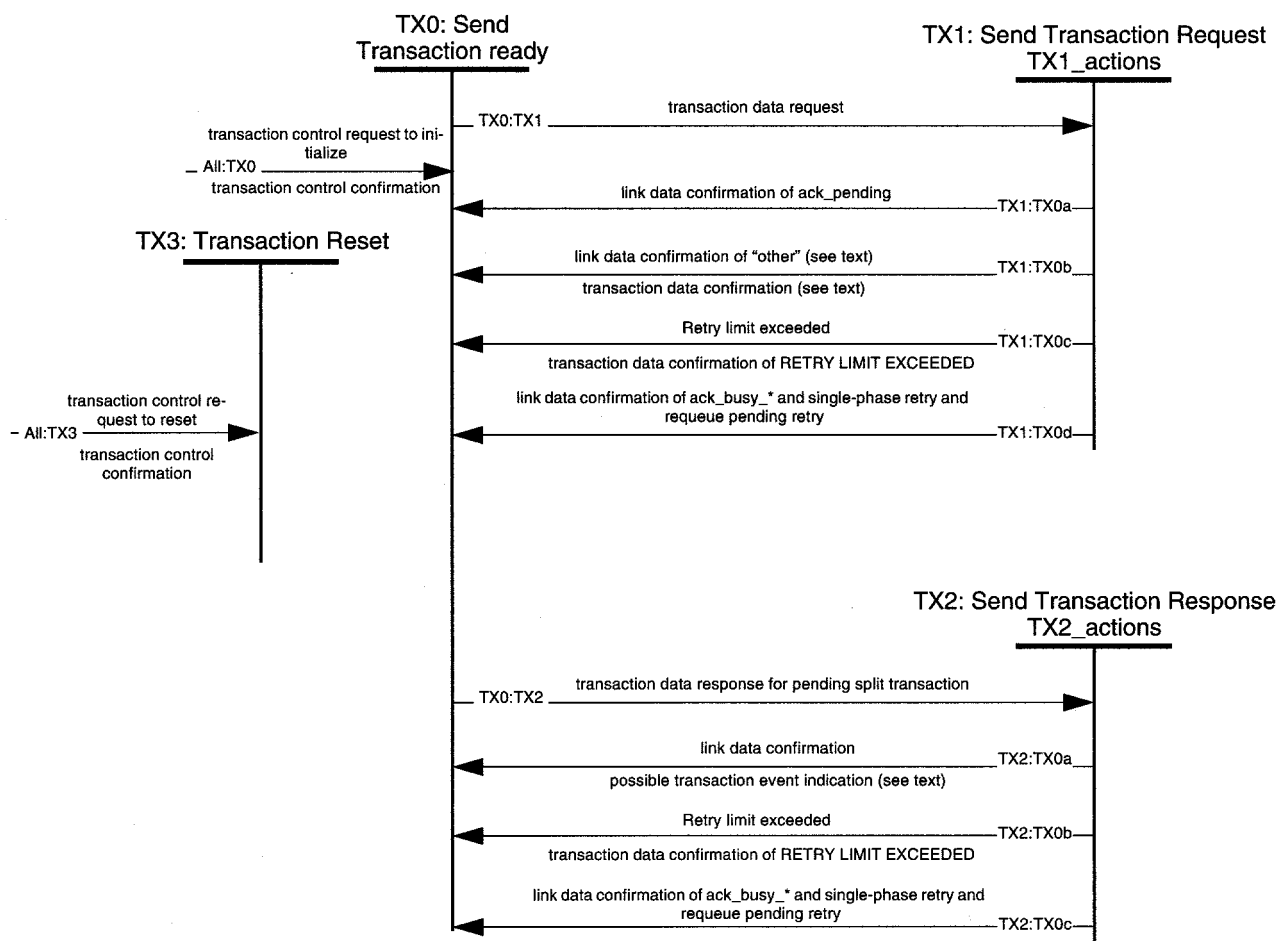


Figure 7.1—Outbound transaction state machine

7.3.3.1.1 Outbound transaction state machine initialization

Transition All:TX0. The transaction layer shall transition from any other transaction layer state to the TX0 state when a transaction control request with an action of Initialize is communicated from the node controller.

State TX0: Send Transaction Ready. The transaction layer is ready to send transaction requests and responses. The transaction layer shall initiate a new transaction only when in state TX0. If the transaction layer receives a transaction data request or a transaction data response when not in state TX0, it shall hold the request or response until it returns to state TX0. The method by which the transaction layer manages pending transaction data requests and transaction data responses is not defined by this standard.

Transition All:TX3. The transaction layer shall transition from any other transaction layer state to the TX3 state when a transaction control request with an action of Reset is communicated from the node controller

State TX3. The transaction layer is suspended. The transaction layer shall only leave state TX3 via transition All:TX0.

7.3.3.1.2 Sending a transaction request

Transition TX0:TX1. The transaction layer receives a transaction data request. The transaction layer shall transition to state TX1.

State TX1: Send Transaction Request. The transaction layer shall initiate a transaction by communicating a link data request to the link layer to cause the request subaction of the transaction to be sent to the specified destination node. The retry code of the request shall be set to `retry_1` for the first attempt to send the subaction. A transaction label that uniquely identifies the transaction to the destination node (see 6.2.4.3) shall be specified by the transaction layer. The destination address, extended transaction code, speed, priority, data length, and data parameters of the link data request shall be set to the values of the corresponding fields contained in the transaction data request. The transaction code parameter value shall be set to

- A write request for a data quadlet, if the transaction type value in the transaction data response is `WRITE` and the data length is 4.
- A write request for a data block, if the transaction type value in the transaction data response is `WRITE` and the data length is not 4.
- A read request for a data quadlet, if the transaction type value in the transaction data response is `READ` and the data length is 4.
- A read request for a data block, if the transaction type value in the transaction data response is `READ` and the data length is not 4.
- A lock request, if the transaction type value in the transaction data response is `LOCK`.

The retry count for the transaction shall be initialized.

After issuing the link data request, the transaction layer waits until it receives a link data confirmation. If the request status is `ACKNOWLEDGE_RECEIVED`, and an acknowledge of `ack_busy_A`, `ack_busy_B`, or `ack_busy_X` is received, then the destination node is busy; the transaction layer shall retry using the appropriate retry method. The method by which the transaction layer performs the retries is defined in 7.3.3.2.2.

Transition TX1:TX0a. The transaction layer receives a link data confirmation, with a request status of `ACKNOWLEDGE_RECEIVED`, and an acknowledge of `ack_pending`. The transaction will complete as a split transaction or a concatenated transaction. The destination node will return a response at some later time. The split transaction timer for the transaction shall be initialized and begin advancing.

Transition TX1:TX0b. The transaction layer receives a link data confirmation, with a request status and/or an acknowledge set to values other than listed for transition TX1:TX0a or transition TX1:TX0d. The transaction has completed as either a unified transaction or a broadcast transaction. The transaction layer shall communicate a transaction data confirmation. Table 7.2 summarizes the parameters that shall be returned by the transaction data confirmation based on the contents of the link data confirmation.

Table 7.2—Summary of transaction data confirmation during transition TX1:TX0b

Value of request status in link data confirmation	Value of acknowledge in link data confirmation	Value of request status in transaction data confirmation	Value of response code in transaction data confirmation
ACKNOWLEDGE_MISSING	None	ACKNOWLEDGE_MISSING	None
BROADCAST_SENT	None	COMPLETE	resp_complete
ACKNOWLEDGE_RECEIVED	ack_type_error	COMPLETE	resp_type_error
ACKNOWLEDGE_RECEIVED	ack_data_error	COMPLETE	resp_data_error
ACKNOWLEDGE_RECEIVED	ack_complete	COMPLETE	resp_complete

Transition TX1:TX0c. The transaction layer detects that the retry count for the transaction has exceeded the transaction retry limit. The transaction layer shall communicate a transaction data confirmation, with a request status of RETRY LIMIT. The transaction shall be considered completed.

Transition TX1:TX0d. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_A, ack_busy_B, or ack_busy_X, and single-phase retry is to be used. The request subaction will continue to be retried using the single-phase retry algorithm (see 7.3.4). The transaction layer may transition back to state TX0 for purposes of sending a different request or response. The transaction layer may also stay in state TX1 and continue retrying the request. The transaction layer shall maintain retry counts for each subaction that is currently busy.

NOTE — This transition is provided to allow a transaction layer implementation to retry busy subactions at a later time, without delaying other queued requests and responses. This implementation is allowed only for single-phase retry.

7.3.3.1.3 Sending a transaction response

Transition TX0:TX2. The transaction layer receives a transaction data response for either a split transaction (see transition RX1:RX0c in 7.3.2.5) or a concatenated transaction (see transition RX1:RX0d in 7.3.2.5). The transaction layer shall transition to state TX2.

State TX2: Send Transaction Response. The transaction layer shall communicate a link data request to the link layer to cause the response subaction of the transaction to be sent to the specified destination node. The retry code of the response shall be set to retry_1 for the first attempt to send the subaction. The response code, extended transaction code, transaction label, speed, priority, data length, and data parameters of the link data request shall be set to the values of the corresponding fields contained in the transaction data response. The destination address parameter shall be set to the value of the Requester ID field contained in the transaction data response. The transaction code parameter value shall be set to

- A write response, if the transaction type value in the transaction data response is WRITE.
- A read response for a data quadlet, if the transaction type value in the transaction data response is READ and the data length is 4.
- A read response for a data block, if the transaction type value in the transaction data response is READ and the data length is not 4.
- A lock response, if the transaction type value in the transaction data response is LOCK.

The retry count for the transaction shall be initialized.

After issuing the link data request, the transaction layer waits until it receives a link data confirmation. If the request status is ACKNOWLEDGE_RECEIVED, and an acknowledge of ack_busy_A, ack_busy_B, or ack_busy_X is

received, then the destination node is busy; the transaction layer shall retry using the appropriate retry method. The method by which the transaction layer performs the retries is defined in 7.3.3.2.2.

Transition TX2:TX0a. The transaction layer receives a link data confirmation. The transaction layer may communicate a transaction event indication, if necessary. The conditions that require the transaction layer to communicate a transaction event indication are summarized in table 7.3.

Table 7.3—Summary of transaction event indication during transition TX2:TX0a

Value of request status in link data confirmation	Value of acknowledge in link data confirmation	Value of transaction event in transaction event indication
ACKNOWLEDGE_MISSING	None	ACKNOWLEDGE_MISSING
BROADCAST_SENT	None	Not valid
ACKNOWLEDGE_RECEIVED	ack_complete	No indication necessary
ACKNOWLEDGE_RECEIVED	ack_data_error	RESPONSE_ERROR
ACKNOWLEDGE_RECEIVED	Any other value	Not valid

Transition TX2:TX0b. The transaction layer detects that the retry count for the subaction has exceeded the transaction retry limit. The transaction layer shall communicate a transaction event indication, with a transaction event of RESPONSE_RETRY_TIMEOUT. The transaction shall be considered completed.

Transition TX2:TX0c. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_A, ack_busy_B, or ack_busy_X, and single-phase retry is to be used. The response subaction will continue to be retried using the single-phase retry algorithm (see 7.3.4). The transaction layer may transition back to state TX0 for purposes of sending a different request or response. The transaction layer may also stay in state TX1 and continue retrying the response. The transaction layer shall maintain retry counts for each subaction that is currently busy.

NOTE — This transition is provided to allow a transaction layer implementation to retry busy subactions at a later time, without delaying other queued requests and responses. This implementation is allowed only for single-phase retry.

7.3.3.2 Inbound transaction state machine

The state machine in figure 7.2 describes the operation of the transaction layer when receiving an inbound transaction request or response.

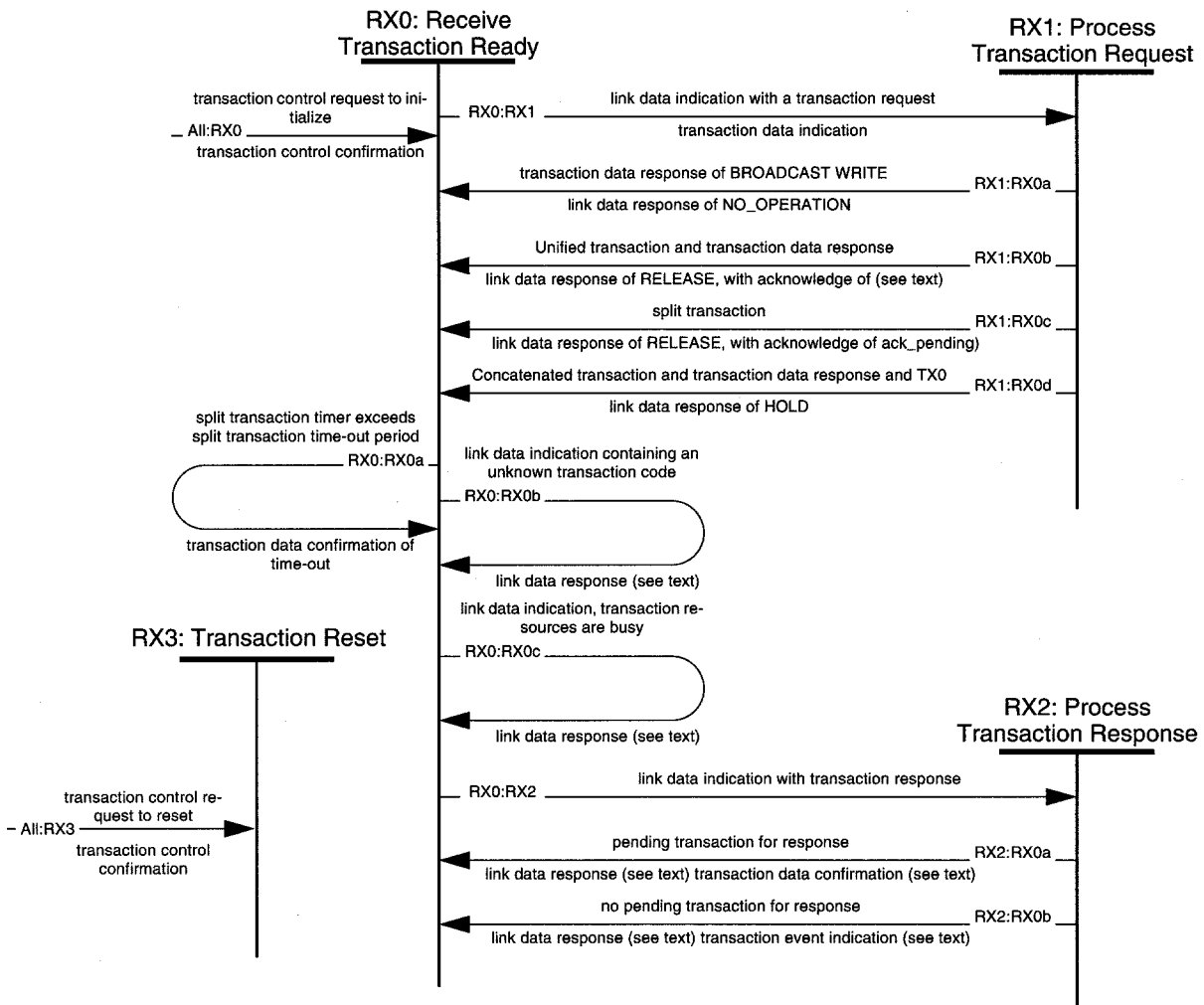


Figure 7.2—Inbound transaction state machine

7.3.3.2.1 Inbound transaction state machine initialization

Transition All:RX0. The transaction layer shall transition from any other transaction layer state to the RX0 state when a transaction control request with an action of Initialize is communicated from the node controller.

State RX0: Receive Transaction Ready. The transaction layer is ready to receive transaction requests and responses.

Transition All:RX3. The transaction layer shall transition from any other transaction layer state to the TX3 state when a transaction control request with an action of Reset is communicated from the node controller.

State RX3. The transaction layer is suspended. The transaction layer shall only leave state RX3 via transition All:RX0.

7.3.3.2.2 Responding to a transaction request

Transition RX0:RX1. The transaction layer receives a link data indication, which contains a transaction code value for a transaction request; specifically, a write request for a data quadlet, a write request for a data block, a read request

for a data quadlet, a read request for a data block, or a lock request. The transaction layer shall communicate a transaction data indication to the application. The destination address, extended transaction code, transaction label, speed, priority, data length, and data parameters shall be set to the values of the corresponding fields contained in the link data indication. The Requester ID parameter shall be set to the value of the source ID field contained in the link data indication. The transaction type parameter value shall be set to

- WRITE, if the transaction code value in the link data indication is either a write request for a data quadlet or a write request for a data block, and the packet status value is not BROADCAST.
- BROADCAST_WRITE, if the transaction code value in the link data indication is either a write request for a data quadlet or a write request for a data block, and the packet status value is BROADCAST.
- READ, if the transaction code value in the link data indication is either a read request for a data quadlet or a read request for a data block.
- LOCK, if the transaction code value in the link data indication is a lock request.

NOTE — It is the responsibility of the application layer (or bus management) to associate the transaction data indication with the appropriate transaction data response.

Transition RX0:RX0a. The split transaction timer for a pending transaction has exceeded the split transaction timeout period. The transaction layer shall communicate a transaction data confirmation. The request status shall be set to ACKNOWLEDGE_MISSING. The pending transaction shall be considered completed.

Transition RX0:RX0b. The transaction layer receives a link data indication, which contains a packet that has a transaction code and/or an extended transaction code that is unknown to or unimplemented by the node. The transaction layer shall communicate a link data response, the bus occupancy control shall be set to RELEASE, and the acknowledge shall be set to ack_type_error.

Transition RX0:RX0c. The transaction layer is busy and shall return a busy acknowledge (the transaction has to be completed as a unified transaction). The transaction layer shall communicate a link data response, and the bus occupancy control shall be set to RELEASE. The acknowledge shall be set as defined in 7.3.3.2.2. Note that a transaction data indication shall not be communicated to the application.

State RX1: Process Transaction Request. The transaction layer has received the transaction request. The next step is to complete the request subaction and then complete the transaction. Unless the transaction is to be completed as a split transaction, the transaction layer waits for a transaction data response.

NOTE — Recall that the method by which a responding transaction layer determines how to complete a given transaction is not defined by this standard.

Transition RX1:RX0a. The transaction layer receives a transaction data response with the transaction type set to BROADCAST_WRITE. The transaction layer shall communicate a link data response, and the bus occupancy control shall be set to NO_OPERATION.

Transition RX1:RX0b. The transaction layer receives a transaction data response and the transaction is to be completed as a unified transaction. The transaction layer shall communicate a link data response, and the bus occupancy control shall be set to RELEASE.

The acknowledge shall be set based on the value of the response code contained in the transaction data response, as summarized in table 7.4.

Table 7.4—Acknowledge value in link data response during transition RX1:RXOb

Value of response code in transaction data response	Value of acknowledge in link data response
resp_complete	ack_complete
resp_conflict_error	NOT ALLOWED
resp_data_error	ack_data_error
resp_type_error	ack_type_error
resp_address_error	NOT ALLOWED

The application layer, node controller, or bus manager shall not communicate a transaction data response that contains a response code of resp_conflict_error or resp_address_error with the intent to complete a unified transaction.

Transition RX1:RX0c. The transaction is to be completed as a split transaction. The transaction layer shall communicate a link data response, and the bus occupancy control shall be set to RELEASE. The acknowledge shall be set to ack_pending.

Transition RX1:RX0d. The transaction is to be completed as a concatenated transaction, and the transaction layer receives a transaction data response. The transaction layer shall communicate a link data response, and the bus occupancy control shall be set to HOLD. The acknowledge shall be set to ack_pending. The transaction layer shall not make this transition if the outbound transaction state machine is not in state TX0.

NOTE — If the outbound transaction state machine is not in state TX0, it is currently attempting a dual-phase retry. The inbound transaction state machine has to complete all transactions that would normally complete as concatenated transactions as split transactions instead.

7.3.3.2.3 Responding to a transaction response

Transition RX0:RX2. The transaction layer receives a link data indication, which contains a transaction code value for a transaction response; specifically, a write response, a read response for a data quadlet, a read response for a data block, or a lock response. The transaction layer shall not communicate a transaction data indication to the application.

State RX2: Process Transaction Response. The transaction layer has received the transaction response. The next step is to complete the response subaction and then complete the transaction.

NOTE — It is the responsibility of the application layer (or node controller or bus manager) to associate the transaction data request with the appropriate transaction data confirmation.

Transition RX2:X0a. The transaction layer determines that it has received the response for a pending request. The transaction layer determines this by comparing the source ID, transaction label, and transaction code from the link data indication with its pending requests. The pending transaction is completed as either a split transaction or a concatenated transaction. The transaction layer shall communicate a link data response, the bus occupancy control shall be set to RELEASE, and, if the packet status in the link data indication was set to DATA_CRC_ERROR, the acknowledge shall be set to ack_data_error; otherwise, the acknowledge shall be set to ack_complete.

The transaction layer shall also communicate a transaction data confirmation to the application. If the link data indication contained valid data and data length values, these parameters shall be valid in the transaction data confirmation. The response code shall be set to the value contained in the link data indication. If the packet status in the link data indication was set to DATA_CRC_ERROR, the request status shall be set to DATA_ERROR; otherwise, the request status shall be set to COMPLETE.

If the packet status in the link data indication was set to `DATA_CRC_ERROR`, the transaction layer shall also communicate a transaction event indication, with the transaction event set to `RESPONSE_DATA_ERROR`. If the packet status in the link data indication was set to `FORMAT_ERROR`, the transaction layer shall also communicate a transaction event indication, with the transaction event set to `RESPONSE_FORMAT_ERROR`.

Transition RX2:X0b. The transaction layer determines that it has received a response for which there is no pending request. The transaction layer determines this by comparing the source ID, transaction label, and transaction code from the link data indication with its pending requests. The transaction layer shall communicate a link data response, the bus occupancy control shall be set to `RELEASE`, and the acknowledge shall be set to `ack_complete`.

The transaction layer shall also communicate a transaction event indication, with the transaction event set to `UNSOLICITED_RESPONSE`.

7.3.4 Transaction types

This subclause defines the types of transactions that can be performed.

7.3.4.1 Read transactions

A read transaction shall be initiated as defined in 7.3.2.5. The request subaction shall contain the address of the data to be read.

A read transaction shall be reacted to as defined in 7.3.3.1. If the request subaction is received correctly, and if the address is a valid address readable by the requesting node, the read data shall be returned in the response subaction. A read transaction may only be completed as a split transaction or as a concatenated transaction. A read transaction shall not be completed as a unified transaction, except when the request subaction cannot be received correctly. A read transaction shall not be completed as a broadcast transaction.

7.3.4.2 Write transactions

A write transaction shall be initiated as defined in 7.3.2.5. The request subaction shall contain the address of the data to be written, and the data to be written.

A write transaction shall be reacted to as defined in 7.3.3.1. If the request subaction is received correctly, and if the address is a valid address writable by the requesting node, the write data shall be written to the requested address. A write transaction may be completed as a split transaction, as a concatenated transaction, as a unified transaction, or as a broadcast transaction.

7.3.4.3 Lock transactions

A lock transaction shall be initiated as defined in 7.3.2.5. The request subaction shall contain the address of the location in the destination node to be lock accessed, the type of lock access to be performed, and the data and argument to be used in the lock access.

A lock transaction shall be reacted to as defined in 7.3.3.1. If the request subaction is received correctly, and if the address is a valid address accessible by the requesting node, the lock access shall be performed to the requested address. A lock transaction may only be completed as a split transaction or as a concatenated transaction. A lock transaction shall not be completed as a unified transaction except when the request subaction cannot be received correctly. A lock transaction shall not be completed as a broadcast transaction.

If a node receives any other transaction request while a lock transaction is pending, and if the address specified in the new transaction overlaps in any way the address range of the pending lock transaction, the node shall either complete the entire function to be performed by the lock transaction before completing the new transaction, or it shall complete the new transaction before performing any part of the entire function to be performed by the lock transaction.

Lock functions can be performed on 32-bit or 64-bit values. The value at the destination address immediately prior to performing the lock function is always returned in the data of the lock response. The following table describes the lock functions defined for lock transactions. Both big- and little-endian fetch and add lock subcommands are defined (fetch_add and little_add). For the big-endian and little-endian adds, the byte with the smallest address within the addressed integer is assumed to be the most or least significant, respectively. Only big-endian unequal_add and wrap_add lock subcommands, which are expected to be used less frequently, are defined.

Table 7.5—Summary of lock transaction functions

Lock function	Update action
mask_swap	$\text{new_value} = \text{data_value} \mid (\text{old_value} \& \sim\text{arg_value});$
compare_swap	$\text{if } (\text{old_value} == \text{arg_value}) \text{ new_value} = \text{data_value};$
fetch_add	$\text{new_value} = \text{old_value} + \text{data_value};$
little_add	$(\text{little}) \text{ new_value} = (\text{little}) \text{ old_value} + (\text{little}) \text{ data_value};$
bounded_add	$\text{if } (\text{old_value} != \text{arg_value}) \text{ new_value} = \text{old_value} + \text{data_value};$
wrap_add	$\text{new_value} = (\text{old_value} != \text{arg_value}) ? \text{old_value} + \text{data_value} : \text{data_value};$

7.3.5 Retry protocols

The retry protocols define the procedure that the transaction layer shall use to respond to an inbound packet when the node is unable to service the packet; i.e., when the node is “busy.” A retry protocol may be applied to any inbound packet that may be acknowledged, including both request packets and response packets. The transaction layer shall implement one and only one of the two available retry protocols, with a minimum retry count of one.

7.3.5.1 Single-phase retry protocol

The transaction layer single-phase retry protocol is described by the state machines in figures 7.3 and 7.4. The state machine in figure 7.3 describes the behavior of the transaction layer when reacting to an inbound primary packet from another node. The state machine in figure 7.4 describes the behavior of the transaction layer when reacting to a received busy acknowledge after an outbound primary packet to another node.

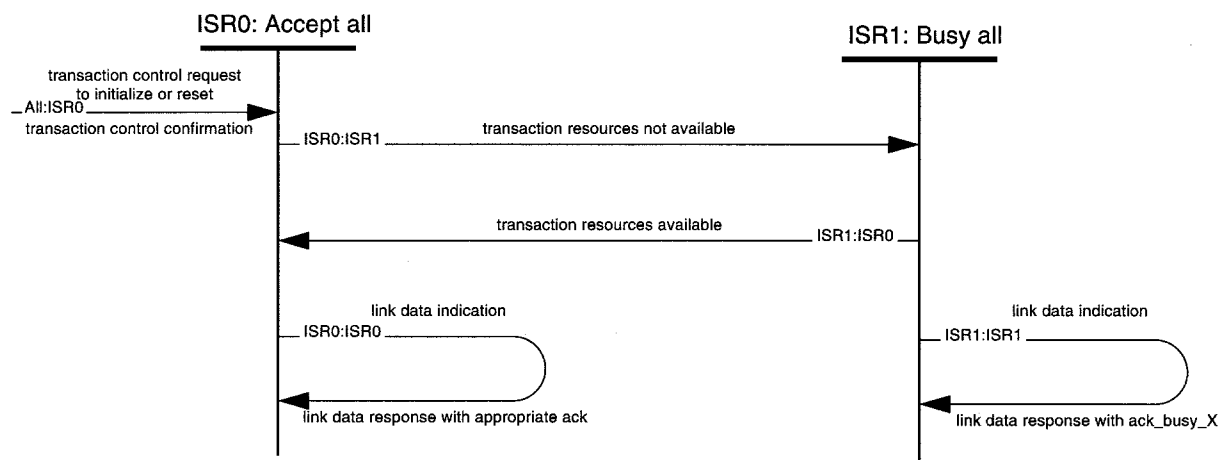


Figure 7.3—Transaction single-phase inbound retry state machine

7.3.5.1.1 Initialization of inbound retry

Transition All:ISR0. The transaction layer shall transition from any other transaction layer inbound retry state to the ISR0 state when a transaction control request with an action of Initialize or an action of Reset is communicated from the node controller.

7.3.5.1.2 Inbound-retry operation

State ISR0: Accept All Transactions. The transaction layer is ready to accept primary packets from other nodes.

Transition ISR0:ISR0. The transaction layer receives a link data indication. The transaction resources are available; therefore, the transaction layer shall communicate a link data response with acknowledge set to the appropriate value (as dictated by the transaction state machine).

Transition ISR0:ISR1. The transaction resources have become unavailable. The transaction layer is now busy.

State ISR1: Busy All Transactions. The transaction layer cannot accept any primary packets from other nodes.

Transition ISR1:ISR1. The transaction layer receives a link data indication. The transaction resources are not available, so the transaction layer shall communicate a link data response with acknowledge set to `ack_busy_X`.

Transition ISR1:ISR0. The transaction resources have become available. The transaction layer is no longer busy.

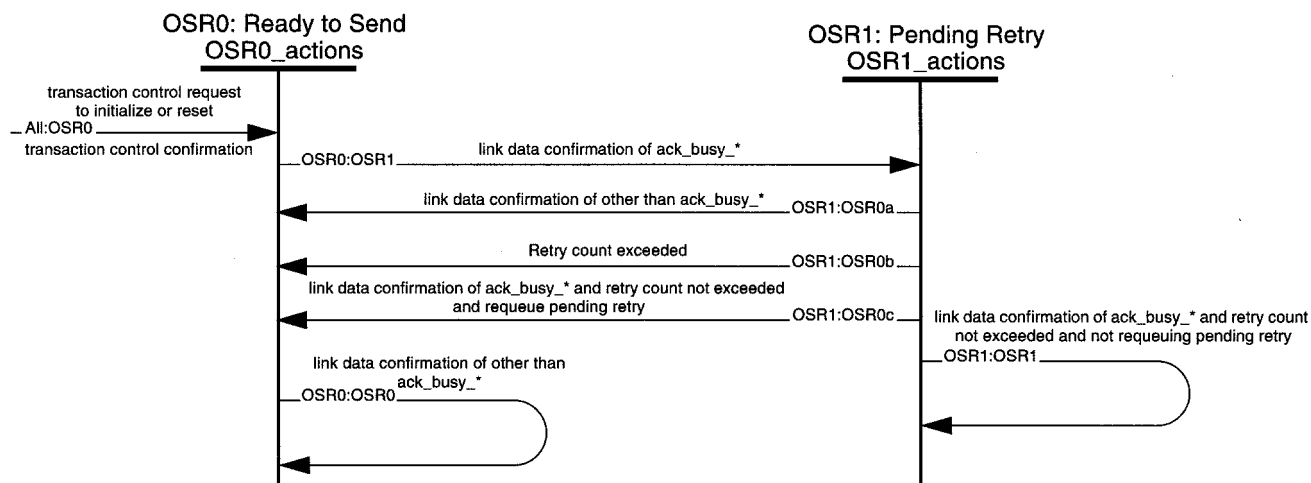


Figure 7.4—Transaction single-phase outbound retry state machine

7.3.5.1.3 Initialization of outbound retry

Transition All:OSR0. The transaction layer shall transition from any other transaction layer outbound retry state to the OSR0 state when a transaction control request with an action of Initialize or an action of Reset is communicated from the node controller.

7.3.5.1.4 Outbound-retry operation

State OSR0: Ready to Send. The transaction layer is ready to send primary packets to other nodes. The transaction layer communicates a link data request to the link layer, with the retry code set to `retry_X`.

Transition OSR0:OSR0. The transaction layer receives a link data confirmation, with an acknowledge other than `ack_busy_A`, `ack_busy_B`, or `ack_busy_X`. The packet has been sent with no need to retry. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition OSR0:OSR1. The transaction layer receives a link data confirmation, with a request status of `ACKNOWLEDGE_RECEIVED` and an acknowledge of `ack_busy_A`, `ack_busy_B`, or `ack_busy_X`. The destination node is busy; the transaction layer shall retry the packet.

State OSR1: Pending Retry. The transaction layer has a pending retry to resolve, and it shall resolve it before sending any other packets. The transaction layer communicates a link data request to the link layer, with the retry code set to `retry_X`.

Transition OSR1:OSR1. The transaction layer receives a link data confirmation, with a request status of `ACKNOWLEDGE_RECEIVED` and an acknowledge of `ack_busy_A`, `ack_busy_B`, or `ack_busy_X`. The retry count (see 7.2.2) has not yet been exceeded. The transaction layer chooses not to requeue the pending retry. The destination node is still busy; the transaction layer shall retry the packet again.

Transition OSR1:OSR0a. The transaction layer receives a link data confirmation, with an acknowledge other than `ack_busy_A`, `ack_busy_B`, or `ack_busy_X`. The packet has been sent. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition OSR1:OSR0b. The retry count has been exceeded. The transaction layer shall not continue to attempt to deliver the packet. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition OSR1:OSR0c. The transaction layer receives a link data confirmation, with a request status of `ACKNOWLEDGE_RECEIVED` and an acknowledge of `ack_busy_A`, `ack_busy_B`, or `ack_busy_X`. The retry count (see 7.2.2) has not yet been exceeded. The transaction layer chooses to requeue the pending retry. The destination node is still busy; the transaction layer shall retry the packet again.

7.3.5.2 Dual-phase retry protocol

The transaction layer dual-phase retry protocol is described by the state machines in figures 7.5 and 7.6. The state machine in figure 7.5 describes the behavior of the transaction layer when reacting to an inbound primary packet from another node. The state machine in figure 7.6 describes the behavior of the transaction layer when reacting to a received busy acknowledge after an outbound primary packet to another node.

A more efficient version of the dual-phase retry (which is not specified) could use counters, in addition to timeouts, to determine when all of the expected `busy_A` or `busy_B` retries have occurred. In the typical case, this would allow acceptance of the `busy_B` retries immediately after the `busy_A` retries had completed, and vice versa. Note that a timeout is still required, since (due to a reset or a retry-interval timeout) the expected `busy_A` may never occur.

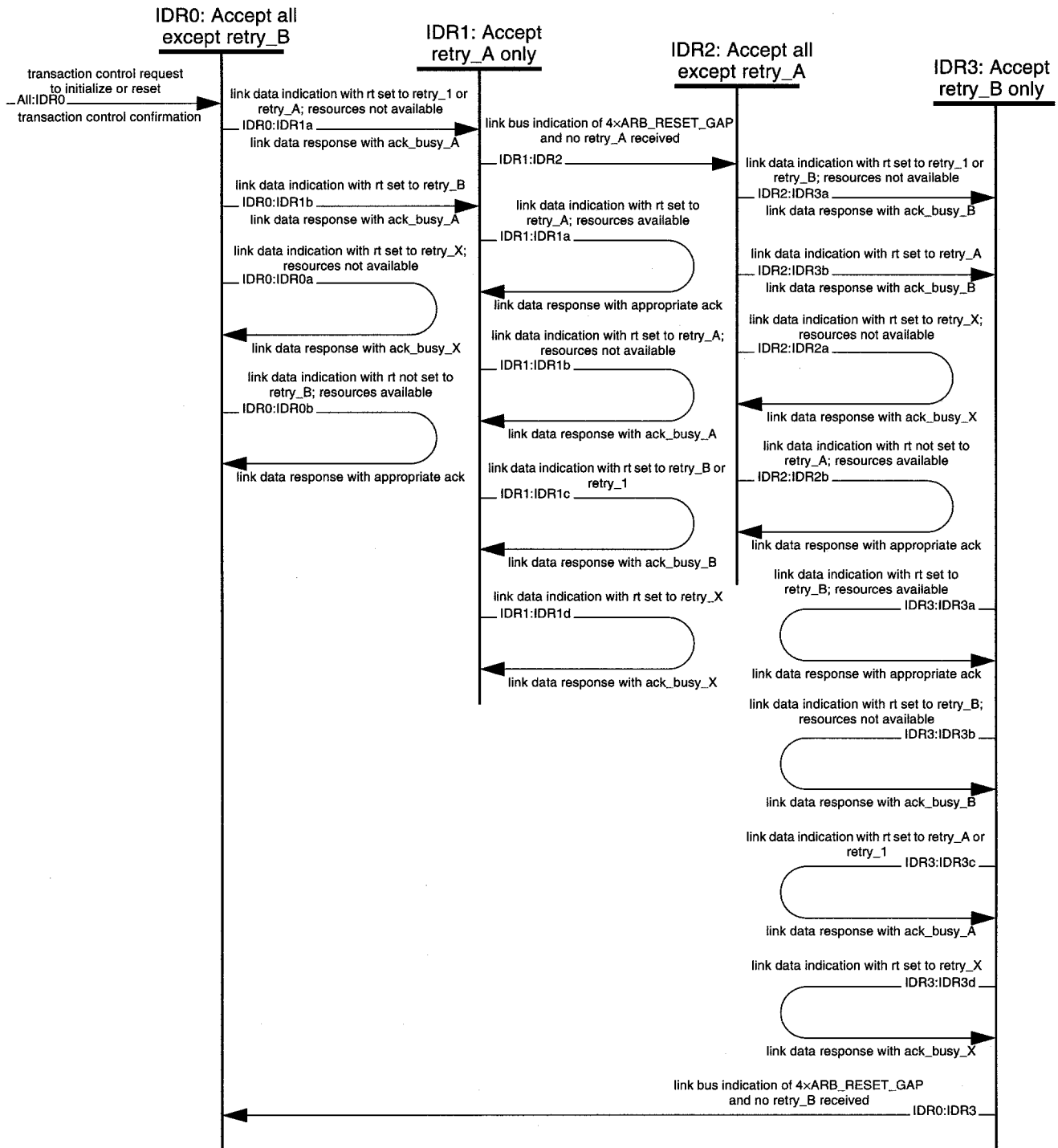


Figure 7.5—Transaction dual-phase inbound retry state machine

7.3.5.2.1 Initialization of inbound retry

Transition All:IDR0. The transaction layer shall transition from any other transaction layer retry state to the IDR0 state when a transaction control request with an action of Initialize is communicated from the node controller.

7.3.5.2.2 Inbound-retry operation

State IDR0: Accept All Primary Packets Except retry_B. The transaction layer is ready to accept primary packets from other nodes. The current retry phase is A. Table 7.6 summarizes the operation of the transaction layer while in this state.

Table 7.6—Summary of operation in state IDR0

Value of retry code in link data indication	Transaction resources are available	Transaction resources are not available
retry_1	Service the packet; stay in state IDR0 (IDR0:IDR0b)	Link data response with acknowledge set to ack_busy_A; transition to state IDR1 (IDR0:IDR1a)
retry_X	Service the packet; stay in state IDR0 (IDR0:IDR0b)	Link data response with acknowledge set to ack_busy_X; stay in state IDR0 (IDR0:IDR0a)
retry_A	Service the packet; stay in the state IDR0 (IDR0:IDR0b)	Link data response with acknowledge set to ack_busy_A; transition to state IDR1 (IDR0:IDR1a)
retry_B	Link data response with acknowledge set to ack_busy_A; transition to state IDR1 (IDR0:IDR1b)	Link data response with acknowledge set to ack_busy_A; transition to state IDR1 (IDR0:IDR1b)

Transition IDR0:IDR0a. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_X, the transaction layer shall communicate a link data response with acknowledge set to ack_busy X.

Transition IDR0:IDR0b. The transaction layer receives a link data indication. If transaction resources are available, and if the retry code is set to retry_1 (initial attempt), retry_X, or retry_A, the transaction layer shall service the request normally, as defined in 7.3.2.

Transition IDR0:IDR1a. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_1 or retry_A, the transaction layer shall communicate a link data response with acknowledge set to ack_busy_A.

Transition IDR0:IDR1b. The transaction layer receives a link data indication. If the retry code is set to retry B (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy A.

NOTE — This transition should never occur in normal operation. It is included for completeness, and to ensure that the anomaly is converted to a retry_A and serviced quickly.

State IDR1: Accept Only retry_A Primary Packets. The transaction layer is ready to accept only retry_A packets from other nodes. The current retry phase is A. Table 7.7 summarizes the operation of the transaction layer while in this state.

Table 7.7—Summary of operation in state IDR1

Value of retry code in link data indication	Transaction resources are available	Transaction resources are not available
retry_1	Link data response with acknowledge set to ack_busy_B; stay in state IDR1 (IDR1:IDR1c)	Link data response with acknowledge set to ack_busy_B; stay in state IDR1 (IDR1:IDR1c)
retry_X	Link data response with acknowledge set to ack_busy_X; stay in state IDR1 (IDR1:IDR1d)	Link data response with acknowledge set to ack_busy_X; stay in state IDR1 (IDR1:IDR1d)
retry_A	Service the transaction request; stay in state IDR1 (IDR1:IDR1a)	Link data response with acknowledge set to ack_busy_A; stay in state IDR1 (IDR1:IDR1b)
retry_B	Link data response with acknowledge set to ack_busy_B; stay in state IDR1 (IDR1:IDR1c)	Link data response with acknowledge set to ack_busy_B; stay in state IDR1 (IDR1:IDR1c)

Transition IDR1:IDR1a. The transaction layer receives a link data indication. If transaction resources are available, and if the retry code is set to retry_A, the transaction layer shall service the packet normally, as defined in 7.3.2.

Transition IDR1:IDR1b. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_A, the transaction layer shall communicate a link data response with acknowledge set to ack_busy A.

Transition IDR1:IDR1c. The transaction layer receives a link data indication. If the retry code is set to retry_1 or retry_B (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy_B.

Transition IDR1:IDR1d. The transaction layer receives a link data indication. If the retry code is set to retry_X (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy X.

Transition IDR1:IDR2. The transaction layer receives a link bus indication. A timeout condition is true when

- a) The bus event is set to ARB_RESET_GAP and the transaction layer has received no primary packets with a retry code of retry_A since the last four link bus indications of ARB_RESET_GAP
- b) The bus event is set to ARB_RESET_GAP4 (an arbitration gap four times longer than the minimal value)

When the timeout condition is true, the transaction layer shall transition to state IDR2.

NOTE — If no retry_A packets have been received for four fairness intervals, the transaction layer services the retry_B packets. This initiates servicing of retry_B packets after all retry_A packets have been serviced. Although this transition could safely occur earlier (when the number of accepted retry_A packets equals the number of retry_1 packets that had an acknowledge of ack_busy_A), this timeout would still be needed because an expected retry_A packet can never return.

State IDR2: Accept All Primary Packets Except retry_A. The transaction layer is ready to accept packets from other nodes. The current retry phase is B. Table 7.8 summarizes the operation of the transaction layer while in this state.

Table 7.8—Summary of operation in state IDR2

Value of retry code in link data indication	Transaction resources are available	Transaction resources are not available
retry_1	Service the packet; stay in state IDR2 (IDR2:IDR2b)	Link data response with acknowledge set to ack_busy_B; transition to state IDR3 (IDR2:IDR3a)
retry_X	Service the packet; stay in state IDR2 (IDR2:IDR2b)	Link data response with acknowledge set to ack_busy_X; stay in state IDR2 (IDR2:IDR2a)
retry_A	Link data response with acknowledge set to ack_busy_B; transition to state IDR3 (IDR2:IDR3b)	Link data response with acknowledge set to ack_busy_B; transition to state IDR3 (IDR2:IDR3a)
retry_B	Service the packet; stay in state IDR2 (IDR2:IDR2b)	Link data response with acknowledge set to ack_busy_B; transition to state IDR3 (IDR2:IDR3b)
NOTE — Retry phase A and retry phase B handling are identical, with the exception of the retry and acknowledge code values.		

Transition IDR2:IDR2a. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_X, the transaction layer shall communicate a link data response with acknowledge set to ack_busy_X.

Transition IDR2:IDR2b. The transaction layer receives a link data indication. If transaction resources are available, and if the retry code is set to retry_1 (initial attempt), retry_X, or retry_B, the transaction layer shall service the packet normally, as defined in 7.3.2.

Transition IDR2:IDR3a. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_1 or retry_B, the transaction layer shall communicate a link data response with acknowledge set to ack_busy_B.

Transition IDR2:IDR3b. The transaction layer receives a link data indication. If the retry code is set to retry_A (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy_B.

NOTE — This transition should never occur in normal operation. It is included for completeness, and to ensure that the anomaly is converted to a retry_B and serviced quickly.

State IDR3: Accept Only retry_B Primary Packets. The transaction layer is ready to accept only retry_B packets from other nodes. The current retry phase is B. Table 7.9 summarizes the operation of the transaction layer while in this state.

Table 7.9—Summary of operation in state IDR3

Value of retry code in link data indication	Transaction resources are available	Transaction resources are not available
retry_1	Link data response with acknowledge set to ack_busy_B; stay in state IDR3 (IDR3:IDR3c)	Link data response with acknowledge set to ack_busy_B; stay in state IDR3 (IDR3:IDR3c)
retry_X	Link data response with acknowledge set to ack_busy_X; stay in state IDR3 (IDR3:IDR3d)	Link data response with acknowledge set to ack_busy_X; stay in state IDR3 (IDR3:IDR3d)
retry_A	Link data response with acknowledge set to ack_busy_A; stay in state IDR3 (IDR3:IDR3c)	Link data response with acknowledge set to ack_busy_A; stay in state IDR3 (IDR3:IDR3c)
retry_B	Service the packet; stay in state IDR3 (IDR3:IDR3a)	Link data response with acknowledge set to ack_busy_B; stay in state IDR3 (IDR3:IDR3b)

Transition IDR3:IDR3a. The transaction layer receives a link data indication. If transaction resources are available, and if the retry code is set to retry_B, the transaction layer shall service the packet normally, as defined in 7.3.2.

Transition IDR3:IDR3b. The transaction layer receives a link data indication. If transaction resources are not available, and if the retry code is set to retry_B, the transaction layer shall communicate a link data response with acknowledge set to ack_busy_B.

Transition IDR3:IDR3c. The transaction layer receives a link data indication. If the retry code is set to retry_1 or retry_A (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy_A.

Transition IDR3:IDR3d. The transaction layer receives a link data indication. If the retry code is set to retry_X (independent of the availability of transaction resources), the transaction layer shall communicate a link data response with acknowledge set to ack_busy_X.

Transition IDR3:IDR0. The transaction layer receives a link bus indication. A timeout condition is true when

- a) The bus event is set to ARB_RESET_GAP and the transaction layer has received no primary packets with a retry code of retry_B since the last four link bus indications of ARB_RESET_GAP
- b) The bus event is set to ARB_RESET_GAP4 (an arbitration gap four times longer than the minimal value)

When the timeout condition is true, the transaction layer shall transition to state IDR0.

NOTE — If no retry_B packets have been received for four fairness intervals, the transaction layer services the retry_A packets. This initiates servicing of retry_A packets after all retry_B packets have been serviced. Although this transition could safely occur earlier (when the number of accepted retry_B packets equals the number of retry_1 packets previously sent an acknowledge of ack_busy_B), this timeout would still be needed because an expected retry_B packet can never return.

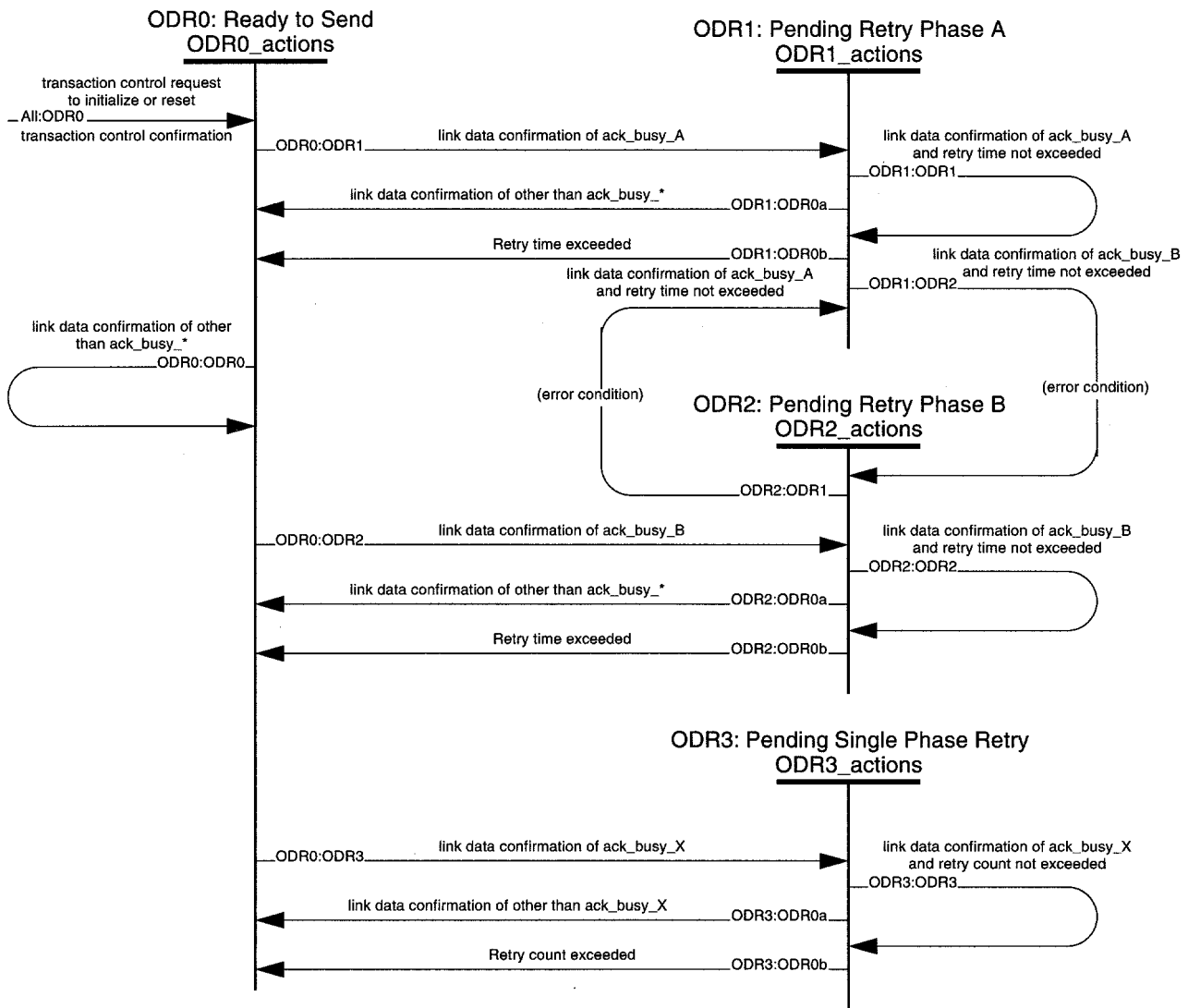


Figure 7.6—Transaction dual-phase outbound retry state machine

7.3.5.2.3 Initialization of outbound retry

Transition All:ODR0. The transaction layer shall transition from any other transaction layer outbound retry state to the ODR0 state when a transaction control request with an action of Initialize or an action of Reset is communicated from the node controller.

7.3.5.2.4 Outbound-retry operation

State ODR0: Ready to Send. The transaction layer is ready to send primary packets to other nodes. The transaction layer communicates a link data request to the link layer, with the retry code set to retry_1.

Transition ODR0:ODR0. The transaction layer receives a link data confirmation, with an acknowledge other than ack_busy_A, ack_busy_B, or ack_busy_X. The packet has been sent with no need to retry. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition ODR0:ODR1. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_A. The destination node is busy and is a dual-phase retry node; the transaction layer shall retry the packet as a dual-phase retry, in retry phase A.

Transition ODR0:ODR2. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_B. The destination node is busy and is a dual-phase retry node; the transaction layer shall retry the packet as a dual-phase retry, in retry phase B.

Transition ODR0:ODR3. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_X. The destination node is busy and is a single-phase retry node; the transaction layer shall retry the packet as a single-phase retry.

State ODR1: Pending Retry Phase A. The transaction layer has a pending retry to resolve and shall resolve it before sending any other packets. The transaction layer communicates a link data request to the link layer, with the retry code set to retry_A. This request has to be made immediately to guarantee that the packet is sent during the next fairness interval.

Transition ODR1:ODR1. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_A. The retry time (see 7.2.2) has not yet been exceeded. The destination node is still busy; the transaction layer shall retry the packet again.

Transition ODR1:ODR0a. The transaction layer receives a link data confirmation, with an acknowledge other than ack_busy_A or ack_busy_B. The packet has been sent. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition ODR1:ODR0b. The retry time has been exceeded. The transaction layer shall not continue to attempt to deliver the packet. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition ODR1:ODR2. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_B. The retry time (see 7.2.2) has not yet been exceeded. The destination node is still busy; the transaction layer shall retry the packet again. The destination node has changed the phase for which the retry shall occur; it is now retry phase B.

NOTE — This transition should never happen in normal operation. It is included for completeness.

State ODR2: Pending Retry Phase B. The transaction layer has a pending retry to resolve and shall resolve it before sending any other packets. The transaction layer communicates a link data request to the link layer, with the retry code set to retry_B. This request has to be made immediately to guarantee that the packet is sent during the next fairness interval.

Transition ODR2:ODR2. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_B. The retry time (see 7.2.2) has not yet been exceeded. The destination node is still busy; the transaction layer shall retry the packet again.

Transition ODR2:ODR0a. The transaction layer receives a link data confirmation, with an acknowledge other than ack_busy_A or ack_busy_B. The packet has been sent. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition ODR2:ODR0b. The retry time has been exceeded. The transaction layer shall not continue to attempt to deliver the packet. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition :ODR2:ODR1. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_A. The retry time (see 7.2.2) has not yet been exceeded. The destination node is still busy; the transaction layer shall retry the packet again. The destination node has changed the phase for which the retry shall occur; it is now retry phase A.

NOTE — This transition should never happen in normal operation. It is included for completeness.

State ODR3: Pending Single Phase Retry. The transaction layer has a pending retry to resolve and shall resolve it before sending any other packets. The transaction layer communicates a link data request to the link layer, with the retry code set to retry_X.

Transition ODR3:ODR3. The transaction layer receives a link data confirmation, with a request status of ACKNOWLEDGE_RECEIVED and an acknowledge of ack_busy_X. The retry count (see 7.2.2) has not yet been exceeded. The destination node is still busy; the transaction layer shall retry the packet again.

Transition ODR3:ODR0a. The transaction layer receives a link data confirmation, with an acknowledge other than ack_busy_X. The packet has been sent. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

Transition ODR3:ODR0b. The retry count has been exceeded. The transaction layer shall not continue to attempt to deliver the packet. The transaction state machine defines the method by which the transaction layer reports this occurrence to the higher layers.

7.4 CSR Architecture transactions mapped to Serial Bus

The CSR-Architecture specified transactions are mapped into Serial Bus transactions using table 7.10.

Table 7.10—CSR Architecture/Serial Bus transaction mapping

CSR Architecture transaction	Serial Bus transaction
read n^*	Transaction data request, transaction type of READ, length of n
write n^*	Transaction data request, transaction type of WRITE, length of n
lock n^\dagger	Transaction data request, transaction type of LOCK, length of n

* n is 1, 2, 4, 8, 16 or 64 for CSR-Architecture-specified read and write transactions.

† n shall be 4 or 8 for CSR-Architecture-specified lock transactions.

Unlike the CSR Architecture, the Serial Bus does not require data alignment nor are lengths restricted to powers-of-two for read and write transactions. However, implementations are allowed to make such restrictions as long as the quadlet aligned read4 and write4 are also implemented.

8. Serial Bus management specification

8.1 Serial Bus management summary

This clause describes the functions and capabilities of the Serial Bus management layer. Figure 3.4 graphically illustrates the relationships between the hardware and software entities located within a Serial Bus node: the PHY layer, the link layer, the transaction layer, the Serial Bus management layer and, implicitly, application elements at the same node. The functions of Serial Bus management include bus management, isochronous resource management,

and node control. Of these three, only node control is required at all nodes with an active link and transaction layer. At least one isochronous-resource-manager-capable node is required on any Serial Bus on which there is isochronous traffic.

8.1.1 Node control

The node controller implements the CSRs required by all Serial Bus nodes and communicates with the PHY, link, and transaction layers and any applications, as shown in figure 3.4.

8.1.2 Isochronous resource manager (cable environment)

Within the cable environment, the isochronous resource manager is a part of the Serial Bus management layer. The isochronous resource manager provides the resources necessary for Serial Bus nodes to allocate and deallocate cooperatively the isochronous resources, channels, and bandwidth required for orderly isochronous operations. The isochronous resource manager, whose location is known immediately upon completion of the self-identify process, also provides a common location where Serial Bus nodes may determine the identity of the bus manager, if one is present.

In the absence of a bus manager, the isochronous resource manager may assume some bus management functions, such as a limited style of power management or the activation of a cycle master.

8.1.3 Isochronous resource manager (backplane environment)

There is no requirement for isochronous resource management within the backplane environment, only a requirement for the basic services provided by the node controller. After a power-on reset or command reset has been completed within the backplane environment, nodes on the Serial Bus may begin to arbitrate for asynchronous transfers once an arbitration reset gap has been detected.

If a Serial Bus in the backplane environment supports isochronous operations, the isochronous facilities should be provided and managed in essentially the same fashion as in the cable environment. Adherence to this guideline will facilitate the development of common management software for use in both environments. Equally important, it simplifies the interconnection of both backplane and cable Serial Buses when isochronous data crosses the bridge.

8.1.4 Bus manager (cable environment)

The bus manager, if present, is also a part of the Serial Bus management layer. The bus manager provides management services to other nodes on the Serial Bus. These include activation of a cycle master, performance optimization, power management, speed management, and topology management. There are procedures described later that govern the orderly selection of a bus manager from potentially many candidates for the role of bus manager.

8.2 Serial Bus management services

These services exist at the interface between the Serial Bus management layer and application(s) executing at the same node. Some services originate within an application and control actions within the Serial Bus management layer; others come from the Serial Bus management layer and communicate changes of state within this layer or on the bus.

8.2.1 Serial Bus control request (SB_CONTROL.request)

An application uses this service to request the Serial Bus management layer to perform specific actions or to specify bus management and node control parameters. Applications also use it to request status about the Serial Bus management layer. The Serial Bus management layer shall process the request immediately upon receipt. The Serial Bus management layer confirms this service.

This service shall provide the following actions:

- **Reset.** The Serial Bus management layer shall request the PHY layer to reset the bus and initialize itself and shall request the link and transaction layers to discard all pending transactions and subactions. The link layer shall disable reception of all nonbroadcast packets and disable transmission of all packets. The transaction layer shall not accept data requests from applications.
- **Initialize.** The Serial Bus management layer shall request the link and transaction layers to discard all pending transactions and subactions. The link layer shall enable reception of all nonbroadcast packets and enable transmission of all packets. The transaction layer shall accept data requests from applications.
- **Link-on.** The Serial Bus management layer shall request the PHY layer to transmit a link-on packet to the designated node. (Cable environment only: required for bus manager nodes, optional for isochronous resource manager nodes. Only to be used by active bus manager, or, if there is no bus manager, the active isochronous resource manager.)
- **Present Status.** The Serial Bus management layer shall return status to the application via the Serial Bus control confirmation service.
- **PHY configuration.** The Serial Bus management layer shall request the PHY layer to transmit a PHY configuration packet. (Cable environment only: required for bus manager and isochronous resource manager nodes. Only to be used by active bus manager, or, if there is no bus manager, the active isochronous resource manager.)

NOTE — The PHY configuration action should never be used in single-node buses.

If the action is Reset or Initialize, the following parameters are communicated to the Serial Bus management layer via this service:

- **Bandwidth Set-Aside.** When a bus reset occurs, the bus manager shall reduce the available isochronous bandwidth by the quantity of bandwidth allocation units specified by this parameter. (Cable environment only: required for bus manager and isochronous resource manager nodes. Only to be used by active bus manager, or, if there is no bus manager, the active isochronous resource manager.)
- **Enable Isochronous Resource Manager.** (Valid only in the backplane environment.) If this Boolean parameter is TRUE, the Serial Bus management layer shall enable the isochronous resource manager capabilities. At most one application element within the backplane environment shall set this parameter to TRUE; all other application elements shall communicate FALSE to the Serial Bus management layers at their own nodes.

If the action is Link-on, the Serial Bus management layer receives the following parameter via this service:

- **Physical ID.** This parameter shall specify the 6-bit physical ID of the Serial Bus node that is to receive the link-on packet.

If the action is PHY configuration, the Serial Bus management layer receives the following parameters via this service:

- **Set force root.** If this flag is set, the Physical ID parameter shall specify the 6-bit physical ID of the Serial Bus node that is to set its force_root bit, and all other nodes shall clear their force_root bit.
- **Physical ID.** See set force root.
- **Set gap count.** If this flag is set, the gap count parameter shall specify the 6-bit gap_cnt value to be used for all nodes on the bus.
- **Gap count.** See set gap count.

8.2.2 Serial Bus control confirmation (SB_CONTROL.confirmation)

The Serial Bus management layer uses this service to confirm to an application the results of a Serial Bus control request service. The Serial Bus management layer shall communicate this service to the application upon completion of a Serial Bus control request. There are no actions provided by this service. This service shall communicate the following parameters after a control request of Present Status:

- a) Bandwidth Set-aside. The actual amount of bandwidth the bus manager was able to subtract from the BANDWIDTH_AVAILABLE register at the isochronous resource manager. This parameter is available only in the cable environment and if the node is the active bus manager or, if there is no bus manager, the active isochronous resource manager.
- b) Bus Manager ID. The 6-bit physical ID of the bus manager. If no bus manager is active, this parameter shall have a value of 3F₁₆. This parameter is available only in the cable environment and if the node is bus-manager or isochronous-resource-manager capable.
- c) Cycle Master ID. The 6-bit physical ID of the cycle master. If no cycle master is active, this parameter shall have a value of 3F₁₆. This parameter is available only if the node is bus-manager or isochronous-resource-manager capable.
- d) Force Root. The force_root variable of the PHY of this node. Cable environment only, as described in 4.3.8.
- e) Gap Count. The gap_cnt variable of the PHY of this node. Cable environment only.
- f) Isochronous Resource Manager. The 6-bit physical ID of the isochronous resource manager. If no isochronous resource manager is active, this parameter shall have a value of 3F₁₆. This parameter is available only if the node is bus-manager or isochronous-resource-manager capable.
- g) Physical ID. The 6-bit physical ID of this node, as described in 4.3.8 for the cable environment and 5.4.2.1 for the backplane environment.
- h) Root ID. The 6-bit physical ID of the root. If no cycle master is active, this parameter shall have a value of 3F₁₆. Cable environment only.

8.2.3 Serial Bus event indication (SB_EVENT. indication)

The Serial Bus management layer uses this service to indicate bus manager, node controller, or bus events to an application. There are no actions provided by this service. No response is defined for this indication. This service shall communicate the following parameters:

- BUS EVENT. This parameter shall contain the event detected on the bus. The following values are defined for this parameter:
 - BUS OCCUPANCY VIOLATION DETECTED. A node on the Serial Bus held the bus longer than a time of MAX_BUS_OCCUPANCY.
 - BUS RESET START. A bus reset has started.
 - BUS RESET COMPLETE. A bus reset process has completed. In the cable environment, this is indicated by the first subtraction gap after the bus reset has started.
 - CYCLE TOO LONG. The last isochronous cycle was too long. The Serial Bus management layer received this indication from the link layer. Only the bus manager and isochronous resource manager shall indicate this event and then only if the node is isochronous capable.
 - CABLE POWER FAIL. In the cable environment, a loss of cable power has occurred as described in 4.2.2.7.
 - DUPLICATE CHANNEL DETECTED. An isochronous packet was received that had the same channel number that is used by one of the transmitted isochronous channels of this node.
 - HEADER CRC ERROR DETECTED. The CRC check, as described in 6.2.4.15, of the header of a received Primary Packet failed.
 - REQUEST DATA ERROR. A transaction request received by this node contained a data error, and the node returned an ack_data_error acknowledgment.
 - RESPONSE ACKNOWLEDGE MISSING. This node did not observe an acknowledgment after transmitting a transaction response packet.
 - RESPONSE DATA ERROR. This node received ack_data_error in acknowledgment of a transaction response packet.
 - RESPONSE FORMAT ERROR. This node received ack_type_error in acknowledgment of a transaction response packet.
 - RESPONSE RETRY FAILED. This node failed to transmit successfully a transaction response within the retry limits specified by Transaction Retry Limit or Transaction Retry Time-out.
 - UNEXPECTED CHANNEL DETECTED (Optional, available only at active isochronous resource manager). The isochronous resource manager detected an isochronous packet that contained an

unallocated channel number during the last isochronous cycle, as specified by the Expected Channel List.

- UNKNOWN TRANSACTION CODE DETECTED. The header of a Primary Packet addressed to this node or the header of an isochronous packet contained an invalid or unknown Transaction Code.
- UNSOLICITED RESPONSE. This node received a transaction response for which there was no request pending, as determined by the transaction label and the destination contained in the response, as described in 6.2.4.3.

All of the following parameters communicated by SB_EVENT.indication are meaningful only in the cable environment when the BUS EVENT is BUS RESET:

- a) Bandwidth Set-aside. The actual amount of bandwidth the bus manager was able to subtract from the BANDWIDTH_AVAILABLE register at the isochronous resource manager. This parameter is available only in the cable environment and if the node is the active bus manager or, if there is no bus manager, the active isochronous resource manager
- b) Bus Manager ID. The 6-bit physical ID of the bus manager. If no bus manager is active, this parameter shall have a value of 3F₁₆.
- c) Configuration Time-out. Physical layer initialization has not completed because of a timeout in the tree identify process. This parameter may indicate a loop in the Serial Bus topology.
- d) Cycle Master ID. The 6-bit physical ID of the cycle master. If no cycle master is active, this parameter shall have a value of 3F₁₆.
- e) Gap Count. The value of the *gap_cnt* field observed in all self-ID packets transmitted during the self-identify process. This parameter is meaningful only if the bus manager is active at this node.
- f) Gap Count Error. Either the bus manager or the isochronous resource manager, if active at this node, may return this event. It indicates that the *gap_cnt* field does not have the same value in all the self-ID packets observed.
- g) Insufficient Cable Power. The bus manager has performed a consistency check of self-ID packets received after a bus reset and the sum of the current power requirements is greater than the sum of the power available.
- h) Isochronous Resource Manager ID. The 6-bit physical ID of the isochronous resource manager. If no isochronous resource manager is active, this parameter shall have a value of 3F₁₆.
- i) Physical ID. This parameter shall contain a unique 6-bit number, as determined by the self-identify process.
- j) Root ID. This parameter shall contain the 6-bit physical ID of the root, as determined by the tree identify process.
- k) Self-ID Error. Either the bus manager or the isochronous resource manager, if active at this node, may return this event. It indicates that a malformed self-ID packet was received, that the self-ID packets were not received in the correct sequence, or that the last self-ID packet received does not describe a root.
- l) Topology Error. Only the bus manager, if active at this node, may return this event. It is an indication that the Serial Bus topology described by the self-ID packets received at the bus manager describe an impossible configuration. For example, the total number of Child ports does not equal the total number of Parent ports.

8.3 Serial Bus management facilities

8.3.1 Node capabilities taxonomy

This subclause describes a hierarchy of progressively increasing node capabilities, from the least capable repeater nodes to the most fully capable bus manager nodes. For some of the capabilities, namely that of cycle master, isochronous resource manager, and bus manager, it is a Serial Bus requirement that only one instance of each of these capabilities shall be active on the Serial Bus. That is, there shall be zero or one cycle master, zero or one isochronous resource manager, and zero or one bus manager active on the Serial Bus at a given moment. The configuration procedures used to determine which nodes among the candidate nodes actually assume these roles are described in 8.4.1 for the backplane environment and 8.4.2 for the cable environment.

8.3.1.1 Repeater (cable environment)

All multiple port nodes present on a Serial Bus in the cable environment are, by definition, repeater nodes. This is the minimum capability required and consists of an active PHY layer. The PHY may be powered from the bus (via the power/ground pair in the Serial Bus cable) or from some other source. Repeater nodes shall

- a) Have an active physical layer
- b) Function as an accurate signal repeater to propagate the signal state from the PHY port conditioned for reception to all other PHY ports conditioned for transmission
- c) Participate in the cable initialization and normal arbitration phases, and be capable of functioning as the root of a Serial Bus
- d) Reconfigure their operational characteristics in response to PHY configuration packets

8.3.1.2 Transaction capable

Any node on a Serial Bus with an enabled link layer is transaction capable, i.e., it can participate in asynchronous transactions with other transaction-capable nodes on the bus. In the cable environment, a repeater node that contains an unpowered or inactive link layer may be transformed into a transaction-capable node by means of a PHY link-on packet. Transaction-capable nodes shall

- a) In the backplane environment, have an active PHY layer
- b) In the cable environment, implement all the capabilities of repeater nodes
- c) Have an active link layer
- d) Implement the `STATE_CLEAR`, `STATE_SET`, `NODE_IDS`, `RESET_START`, and `SPLIT_TIMEOUT` registers

If a transaction-capable node implements one or more units that draw power from the bus, the node shall implement a `Unit_Power_Requirements` entry in the `Unit_Directory` of each unit that draws power from the bus. This imposes the additional requirement that the node shall implement configuration ROM in the general ROM format.

8.3.1.3 Isochronous capable

Any node on a Serial Bus that can either send or receive isochronous packets is isochronous capable. Isochronous-capable nodes shall

- a) Implement all the capabilities of transaction-capable nodes
- b) Implement the `CYCLE_TIME` register and a free-running 24.576 MHz clock that updates the `CYCLE_TIME` register
- c) Implement configuration ROM, in the general ROM format

The requirement for configuration ROM is necessary in order for the node to describe its isochronous capabilities in the `Bus_Info_Block`.

8.3.1.4 Cycle master capable

For any Serial Bus capable of isochronous operations, there shall be a cycle master. Through a process described for the backplane environment in 8.4 and for the cable environment in 8.4.2.2, a single-cycle master shall be enabled from possible candidates after a bus reset. A node that is capable of becoming the cycle master shall

- a) Implement all the capabilities of isochronous-capable nodes
- b) Be able to generate cycle start events triggered by the 8 kHz clock synchronized to the `CYCLE_TIME` register and broadcast the corresponding cycle start packets
- c) Implement the `BUS_TIME` register

8.3.1.5 Isochronous resource manager capable

For any Serial Bus capable of isochronous operations, there shall be an isochronous resource manager. Through a process described for the backplane environment in 8.4.1.2 and for the cable environment in 8.4.2.3, a single isochronous resource manager shall be selected from possible candidates after a bus reset. A node that is capable of becoming the isochronous resource manager shall

- a) Implement all the capabilities of transaction-capable nodes
- b) Implement the `BUS_MANAGER_ID`, `BANDWIDTH_AVAILABLE` and `CHANNELS_AVAILABLE` registers
- c) In the cable environment, be able to analyze received self-ID packets in order to determine correctly the physical ID of the isochronous resource manager node from all the candidates for this role
- d) Implement configuration ROM, in the general ROM format

The requirement for configuration ROM is necessary in order for the node to describe its isochronous resource manager capabilities in the `Bus_Info_Block`.

NOTE — In a sense, the name **isochronous resource manager** is misleading since such a node does not directly “manage” isochronous resources such as bandwidth and channels. In actual fact, the isochronous resource manager provides a single location where other Serial Bus nodes may cooperatively record their usage of isochronous resources.

Only in the absence of a bus manager may the isochronous resource manager perform the following bus management functions:

- a) Limited power management
- b) Setting the `force_root` flag in a node

8.3.1.6 Bus manager capable (cable environment)

The most fully capable nodes on a Serial Bus are those that may become bus managers. Bus-manager-capable nodes shall be isochronous resource manager capable, provide advanced power management, provide facilities to optimize Serial Bus performance, describe the topology of the bus, and cross-reference the maximum speed for data transmission between any two nodes on the bus. Through a process described in 8.4.2.1, a single bus manager shall be selected from all bus manager capable nodes; this occurs any time there is a bus reset. A node that is capable of becoming the bus manager shall

- a) Implement all the capabilities of isochronous-resource-manager-capable nodes
- b) Implement the `SPEED_MAP` and `TOPOLOGY_MAP` registers

NOTE — Just as the term isochronous resource manager is somewhat misleading, so is the description *bus manager* with respect to some of its functions. With respect to the topology and speed information, the bus manager does not “manage” this information as much as it publishes it. However, for Serial Bus optimization and power management, the bus manager takes a more active role and may actually configure nodes on the Serial Bus.

8.3.2 Command and status registers

The Serial Bus uses the CSR Architecture, whose formal standard designations are IEEE Std 1212 and ISO/IEC 13213, to define the framework and selected contents of its command and status registers (CSRs) and for configuration ROM entries. This specification includes the minimal CSR and configuration ROM requirements needed for compliant Serial Bus implementations as well as optional CSR and ROM entries for expanded functionality. The CSR Architecture is the reference for more details on the use of optional parts of required CSRs, as well as optional CSR and ROM entries needed for any purpose, including bus-dependent and unit-dependent applications. One example of an optional bus-dependent application is the SBP, which describes a command and status transfer protocol supporting a serial version of Small Computer Systems Interface (SCSI[®]) implemented on the Serial Bus. Consult the SBP document for details relating to CSRs and configuration ROM specific to the Serial Bus Protocol.

8.3.2.1 Reset conditions

The CSR Architecture defines two types of reset events for nodes: power reset and command reset. Serial Bus defines an additional reset type: bus reset. Table 8.1 describes the conditions that generate these resets.

Table 8.1—Reset types

Reset Type	Description
Power reset	<p>Restoration of primary power to link, PHY, and bus management (all environments): Upon a power reset, all CSR registers shall return to their initial values, as defined by the CSR Architecture and this standard. In both the backplane and cable environments, the PHY layer shall also be reset. In the cable environment, if the PHY layer is powered locally it shall initiate a bus reset. In the backplane environment, a node that is “live inserted” shall undergo a power reset.</p>
Bus reset	<p>Bus reconfiguration (cable environment): A change in the topology of Serial Bus shall cause a bus reset. Two events can change Serial Bus topology: the physical addition or removal of a node or a change between the powered and unpowered states of the PHY layer of a node. No bus reset is required when the power state of other layers, e.g., the link layer, changes, as long as the physical layer remains continuously powered. In addition to these requirements, any Serial Bus node may initiate a bus reset at its option.</p> <p>Bus reinitialization (backplane environment): A bus reset may be initiated by the action of any node on the Serial Bus. This is done with communication of the BUS_RESET signal defined in 5.3.3. A node that is “live inserted” shall <i>not</i> initiate a bus reset.</p>
Command reset	<p>Write to RESET_START register (all environments): In the backplane environment, a command reset shall cause a reset of the PHY layer. In the cable environment, a command reset shall not cause a reset of the PHY layer nor a bus reset.</p>

In general, most CSRs return to their initial values upon either a power reset, bus reset, or command reset. However, individual CSRs are defined with exceptions to this rule. The clauses that follow and describe individual CSRs specify the special reset rules that are exceptions to the general rule.

8.3.2.2 CSR Architecture core registers

The CSR Architecture standardizes the function of the core CSRs and their location within the initial register space. The addresses of these core registers are specified in terms of offsets within the initial register space, where the offset of the start of initial register space (from the beginning of the initial node space) is $FFFF\ F000\ 0000_{16}$.

Table 8.2 defines the core registers defined by the CSR architecture. The offset value is the byte offset measured from the beginning of the initial register space.

Table 8.2—Core CSR locations

Offset	Name	Description
000 ₁₆	STATE_CLEAR*	State and control information
004 ₁₆	STATE_SET*	Sets STATE_CLEAR bits
008 ₁₆	NODE_IDS*	Specifies 16-bit node ID value
00C ₁₆	RESET_START*	Resets state of node
010 ₁₆ –014 ₁₆	INDIRECT_ADDRESS, INDIRECT_DATA	Indirectly access ROMs > one kilobyte
018 ₁₆ –01C ₁₆	SPLIT_TIMEOUT_HI, SPLIT_TIMEOUT_LO	Split-request timeout
020 ₁₆ –02C ₁₆	ARGUMENT_HI, ARGUMENT_LO, TEST_START, TEST_STATUS	Optional diagnostic-test interface
030 ₁₆ –04C ₁₆	UNITS_BASE, UNITS_BOUND, MEMORY_BASE, MEMORY_BOUND	Never implemented
050 ₁₆ –054 ₁₆	INTERRUPT_TARGET, INTERRUPT_MASK	Optional broadcast/nodectast interrupt
058 ₁₆ –07C ₁₆	CLOCK_VALUE, CLOCK_TICK_PERIOD, CLOCK_STROBE_ARRIVED, CLOCK_INFO	Synchronized time-of-day value and control
080 ₁₆ –0FC ₁₆	MESSAGE_REQUEST, MESSAGE_RESPONSE	Optional message passing registers
100 ₁₆ –17C ₁₆		Reserved for CSR Architecture
180 ₁₆ –1FC ₁₆	ERROR_LOG_BUFFER	Reserved for Serial Bus
200 ₁₆ –3FC ₁₆	Serial bus dependent	See table 8.3

*Required in whole or in part

8.3.2.2.1 STATE_CLEAR register

The STATE_CLEAR register provides standard fields and an 8-bit bus-dependent field, as illustrated in figure 8.1.

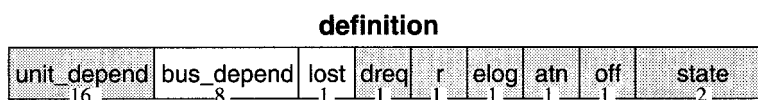


Figure 8.1—STATE_CLEAR format

The CSR Architecture fully defines the shaded fields (*unit_depend*, *dreq*, *r*, *elog*, *atn*, *off*, and *state*).

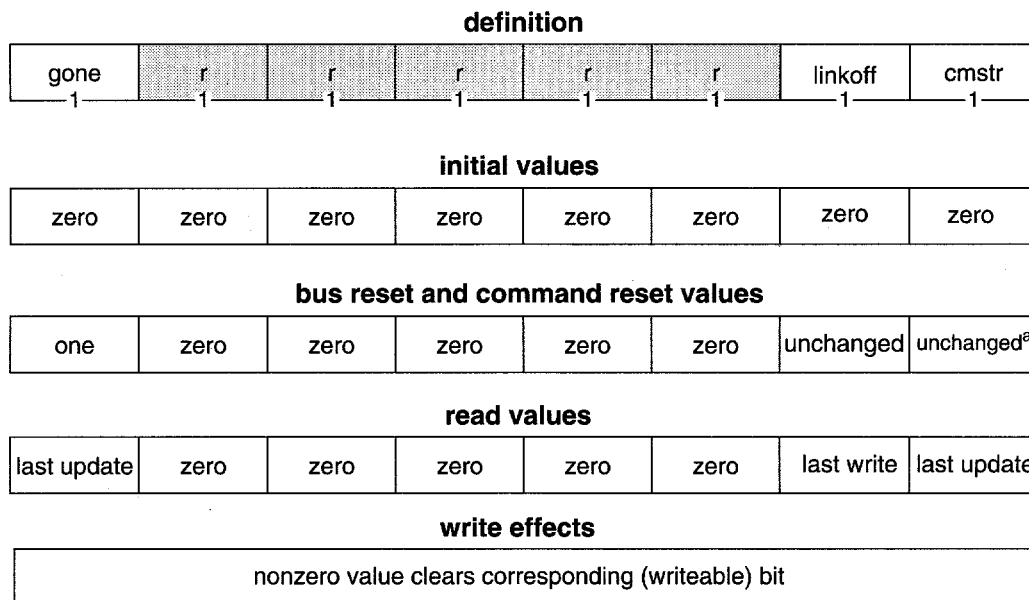
The optional 16-bit *unit_depend* field is available for unit-dependent applications, as defined within the CSR Architecture.

The *bus_depend* field provides bus-dependent information; for Serial Bus, figure 8.2 shows the format of the *bus_depend* field.

Although the *lost* bit is optional within the CSR Architecture, Serial Bus nodes shall implement the *lost* bit. In the cable environment, the *lost* bit is not directly affected by a bus reset. However, the *lost* bit shall be set to 1 during a bus reset if a power reset or transition to the dead state (as defined by the CSR Architecture) has occurred.

Although the *dreq* bit is optional within the CSR Architecture, Serial Bus nodes capable of originating transaction requests shall implement the *dreq* bit. In addition to the other requirements of the CSR Architecture, these request-capable nodes shall be unaffected by a bus reset. If the *dreq* bit is subsequently set to one, the node shall not transmit transaction requests. Serial Bus nodes that are not request capable, i.e., nodes that only respond to requests originated by other nodes, shall implement this bit as one.

The *r*, *elog*, *atn*, and *off* bits are optional within the CSR Architecture. In addition to their properties defined within the CSR Architecture, the *elog* bit shall be unchanged by a bus reset. Serial Bus reserves the *atn* and *off* bits.



^a The value of the *cmstr* bit after a bus reset is defined below.

Figure 8.2—STATE_CLEAR.bus_depend field

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

The *gone* bit shall be set to one on a power reset, command reset, or bus reset. Units on a node have the option of implementing an equivalent *gone* bit (within their unit-dependent registers) and ignoring the value of the STATE_CLEAR.*gone* bit. Multiple-unit nodes are expected to have unit-dependent *gone* bits, if some of the units (such as a processor, which initializes other nodes, and a DMA adapter, which is initialized by others) are expected to be reactivated (after a bus reset) at different times.

If a unit does not have an equivalent *gone* bit, then that unit shall not transmit transaction requests while STATE_CLEAR.*gone* is set. However, a unit that independently resumes operation after a bus reset may clear its own STATE_CLEAR.*gone* bit before accessing the bus. To other nodes, the STATE_CLEAR.*gone* bit of this node may be cleared before becoming visible to other nodes (thus providing the appearance that the *gone* bit was never set).

In the cable environment, nodes whose link layer draws power from the bus shall implement the *linkoff* bit. If the value of the *linkoff* bit is zero, the link layer of the node is active. For nodes whose link layer draws power from the bus, a write of one to the *linkoff* bit in the STATE_SET register shall cause the link layer to become inactive and unpowered. The link layer of the node may be subsequently reactivated by a link-on packet.

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. For the cable environment, the following additional requirements shall be met:

- a) If the bus manager or isochronous resource manager sets the value of the *cmstr* bit to one, it shall also set the Force_Root variable of the node to one prior to the write to the STATE_CLEAR.*cmstr* bit (done via the PHY control request if the new cycle master is the also the manager node; otherwise, it is done by sending a PHY configuration packet). **Important**—The force_root bit shall *not* be set in single-node buses, since that will interfere with the automatic restart of isochronous data when that single node is attached to an operating bus.
- b) If a node is not the root, it shall ignore any attempt to set *cmstr* to one, and *cmstr* shall retain a zero value.

Special bus reset rule—In the cable environment, any node that is not the root after the tree identify process that follows a bus reset shall clear the *cmstr* bit to zero. Otherwise, the *cmstr* bit shall retain its value prior to the bus reset.

8.3.2.2.2 STATE_SET register

A write of one to a bit in the STATE_CLEAR register clears the corresponding bit position in the STATE_SET register (if that bit is writable in the STATE_SET register).

8.3.2.2.3 NODE_IDS register

The NODE_IDS register is used to identify and modify the current bus ID and physical ID values, which directly affect the initial node address. Serial Bus reserves the 16-bit bus-dependent field, as indicated by the shaded field within figure 8.3.

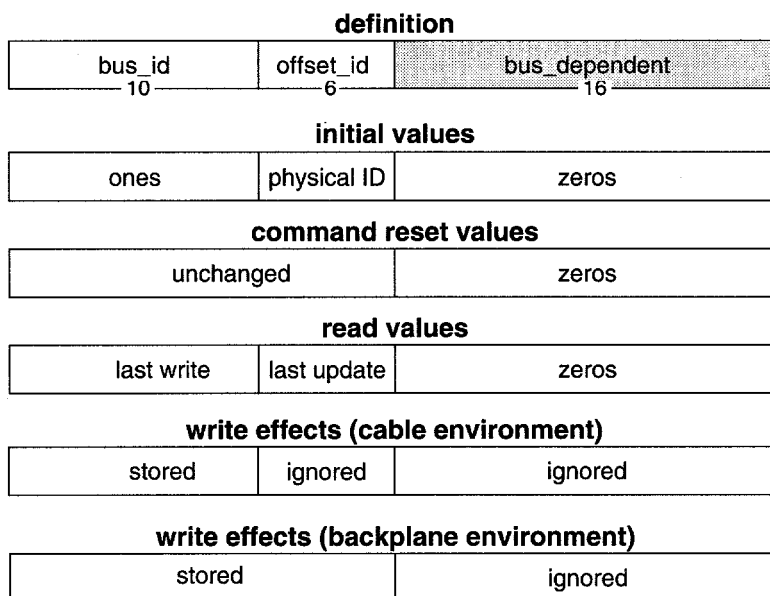


Figure 8.3—NODE_IDS format

The 10-bit read/write *bus_id* field provides software with a mechanism for reconfiguration of the initial node address space. The *bus_id* field provides a means for multiple-bus configurations to distinguish node addresses on one bus from those on another.

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

The 16-bit read-only *bus_dependent* field shall be reserved.

NOTE — If there are any side-effects of a write transaction to a register, a node shall delay the return of a transaction response until all effects of the write are complete. In the case of the `NODE_IDS` register, a return of `resp_complete` indicates that the node recognizes transactions to the newly assigned `NODE_IDS` value. The contents of the `source_ID` field of the response packet shall reflect the updated value of the `NODE_IDS` register.

8.3.2.2.4 RESET_START register

A write to the `RESET_START` register performs an immediate command reset, as defined in the CSR Architecture.

8.3.2.2.5 INDIRECT_ADDRESS and INDIRECT_DATA registers

The `INDIRECT_ADDRESS` and `INDIRECT_DATA` registers are reserved. All access to the configuration ROM shall be done with directly addressed reads. The Serial Bus configuration ROM may be longer than 1 kilobyte by extending into the initial units space.

8.3.2.2.6 SPLIT_TIMEOUT register

The `SPLIT_TIMEOUT` registers set the default timeout value for detecting split-transaction errors. The value of `SPLIT_TIMEOUT` sets the maximum time for the receipt of a response subaction after the transmission of a request subaction. After this time, a requester should terminate the transaction with a `response_timeout` status. Figure 8.4 illustrates the portions of the `SPLIT_TIMEOUT` register implemented on Serial Bus.

definition	
integers of a second	3
13	fraction of a second 19
initial values	
zeros	zero
800	zeros
command reset and bus reset values	
zeros	unch
unchanged	zeros
read values	
zeros	w
last write	zeros
write effects	
ignored	s
stored	ignored

Figure 8.4—`SPLIT_TIMEOUT` format

Since the Serial Bus SPLIT_TIMEOUT_HI register implements only the three least-significant bits, the timeout can be no longer than 8 s.

The Serial Bus SPLIT_TIMEOUT_LO register implements only the 13 most-significant bits. These bits specify a fractional value of a second in units of 1/8000 s, rather than 1/8192 s as specified by the CSR Architecture. The timeout resolution is nominally 125 μ s.

NOTE — The Serial Bus definition of the SPLIT_TIMEOUT register is different from that defined within the CSR Architecture. Serial Bus interprets the most significant 13 bits of the SPLIT_TIMEOUT_LO register as units of 1/8000 s, rather than a true binary fraction of a second with units of 1/8192 s. Since precise timeouts are not necessary, applications may ignore this difference when calculating values for use within the SPLIT_TIMEOUT_LO register.

8.3.2.2.7 ARGUMENT, TEST_START, and TEST_STATUS registers

The ARGUMENT, TEST_START, and TEST_STATUS registers are optional. If implemented, they are used to initiate or monitor the built-in test capabilities of a node, as defined in the CSR Architecture.

8.3.2.2.8 UNITS_BASE, UNITS_BOUND, MEMORY_BASE, and MEMORY_BOUND registers

The UNITS_BASE, UNITS_BOUND, MEMORY_BASE, and MEMORY_BOUND registers are reserved. The CSR Architecture defines these registers only for the 32-bit or 64-bit extended address-space models; Serial Bus uses the 64-bit fixed address-space model.

8.3.2.2.9 INTERRUPT_TARGET and INTERRUPT_MASK registers

The INTERRUPT_TARGET and INTERRUPT_MASK registers are optional. If implemented, they provide a way for sending interrupts to all units on the node, as defined in the CSR Architecture.

8.3.2.2.10 CLOCK_VALUE, CLOCK_TICK_PERIOD, CLOCK_STROBE_ARRIVED, and CLOCK_INFO registers

The CLOCK_VALUE, CLOCK_TICK_PERIOD, CLOCK_STROBE_ARRIVED, and CLOCK_INFO registers are optional. Since the CYCLE_TIME register provides a nearly equivalent functionality, Serial Bus nodes usually do not implement these registers.

8.3.2.2.11 MESSAGE_REQUEST and MESSAGE_RESPONSE registers

The MESSAGE_REQUEST and MESSAGE_RESPONSE registers are optional. If implemented, they provide target addresses for message transactions broadcast to all units on the bus or all units within the node, as defined by the CSR Architecture.

8.3.2.3 Serial-Bus-dependent 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 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 0000₁₆. Table 8.3 summarizes these bus-dependent registers.

Table 8.3—Serial-Bus-dependent registers

Offset	Name	Notes
200 ₁₆	CYCLE_TIME	For nodes providing isochronous services.
204 ₁₆	BUS_TIME	For nodes requiring synchronized bus time.
208 ₁₆	POWER_FAIL_IMMINENT	For nodes needing power-fail warning.
20C ₁₆	POWER_SOURCE	
210 ₁₆	BUSY_TIMEOUT	For transaction-capable nodes.
214 ₁₆ –218 ₁₆		Reserved.
21C ₁₆	BUS_MANAGER_ID*	For selecting or locating the bus manager.
220 ₁₆	BANDWIDTH_AVAILABLE*	For bandwidth allocation of isochronous-resource-manager-capable nodes.
224 ₁₆ –228 ₁₆	CHANNELS_AVAILABLE*	For channel allocation of isochronous-resource-manager-capable nodes.
22C ₁₆	MAINT_CONTROL	For introducing diagnostic error conditions.
230 ₁₆	MAINT_UTILITY	
234 ₁₆ –3FC ₁₆		Reserved.

*Only quadlet read and quadlet lock (compare and swap) transactions supported.

The following subclauses provide detailed definitions of the Serial Bus-dependent registers.

8.3.2.3.1 CYCLE_TIME register

Optional. All nodes capable of providing isochronous services shall implement this register.

Nodes that are isochronous capable shall implement a 24.576 MHz clock that shall run freely and update the contents of the CYCLE_TIME register. A write to the CYCLE_TIME register shall initialize the clock hardware to the value contained in the write transaction. Neither a bus reset nor a command reset shall affect the time maintained by clock hardware.

The CYCLE_TIME register provides fields that specify the current time value. Figure 8.5 illustrates the formats of these fields.

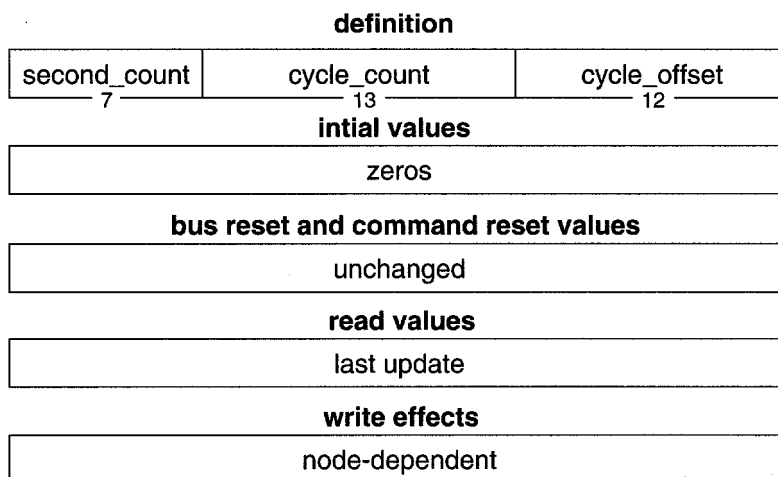


Figure 8.5—CYCLE_TIME format

The 7-bit read/write *second_count* field shall increment on each carry from the *cycle_count* field, with the exception that an increment from the value 127 shall cause a wraparound to zero and shall carry into the *BUS_TIME.second_count_hi* field.

The 13-bit read/write *cycle_count* field shall increment on each carry from the *cycle_offset* field, with the exception that an increment from the value 7999 shall cause a wraparound to zero and shall carry into the *second_count* field. The value is the fractional part of the second of the current time, in units of 125 μ s.

The 12-bit read/write *cycle_offset* field shall be updated on each tick of the local 24.576 MHz PHY clock, with the exception that an increment from the value 3071 shall cause a wraparound to zero and shall carry into the *cycle_count* field. The value is the fractional part of the isochronous cycle of the current time, in units that are counts of the 24.576 MHz clock.

Isochronous-capable nodes whose *STATE_CLEAR.cmstr* bit is zero are called cycle slaves. Cycle slaves shall update the value of the *CYCLE_TIME* register based upon the time value received in each cycle start packet. A cycle slave shall implement a synchronization mechanism between the cycle start packets and the *CYCLE_TIME* register such that time, as observed by the values of the *CYCLE_TIME* register, never appears to move backwards. The details of cycle slave synchronization mechanisms (as well as their sensitivity to jitter and noise) are implementation dependent and beyond the scope of this standard.

Cycle-master-capable nodes shall implement hardware that can arbitrate for the bus in order to transmit a cycle start packet each time the *cycle_count* field increments. The cycle master (i.e., a cycle master capable node that is the root with its *STATE_CLEAR.cmstr* bit set to one), shall insert the current *CYCLE_TIME* value into each cycle start packet transmitted. The *CYCLE_TIME* value inserted into the cycle start packet shall be the time at which the cycle start packet was transmitted and not the time of the cycle start event.

8.3.2.3.2 BUS_TIME register

Optional. Cycle-master-capable nodes shall implement this register. May be initialized by the bus manager or, in the absence of a bus manager, the isochronous resource manager. A broadcast write transaction shall be used to initialize the *BUS_TIME* register.

The *BUS_TIME* register provides a measure of the current time in seconds and has a range of 2^{32} s or approximately 136 years. Figure 8.6 shows the format of this register.

definition	
second_count_hi 25	second_count_lo 7
initial values	
zeros	
bus reset and command reset values	
unchanged	
read values	
last write	last update
write effects	
stored	ignored

Figure 8.6—BUS_TIME format

The 25-bit read/write *second_count_hi* field shall increment on each carry from the CYCLE_TIME.*second_count* field.

The 7-bit read-only *second count_lo* field is the current value of the CYCLE_TIME.*second_count* field.

8.3.2.3.3 POWER_FAIL_IMMINEENT register

Optional. If implemented, the POWER_SOURCE register shall also be implemented.

The POWER_FAIL_IMMINEENT register provides a means of notifying Serial Bus nodes that a power failure is about to occur.

Within a backplane environment, the node responsible for the power supply, upon sensing an imminent failure, should perform a broadcast write (using node address $3F_{16}$ to the POWER_FAIL_IMMINEENT register. The data written into this register includes the node ID of the power-supply node and a time value that specifies the time remaining before the power supply is expected to go out of regulation. Figure 8.7 shows the format of this register.

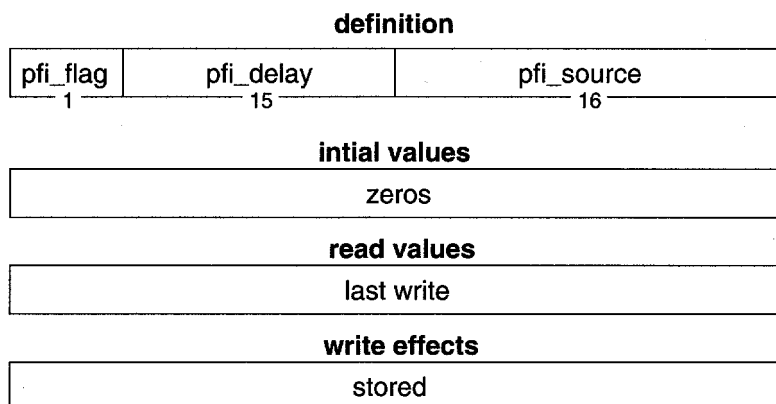


Figure 8.7—POWER_FAIL_IMMINEENT format

The read/write *pfi_flag* bit indicates that a power failure is imminent. If the *pfi_flag* bit is one, a power failure is imminent; otherwise, the values within this register shall be ignored.

The 15-bit read/write *pfi_delay* field defines the approximate time that remains before the power supply goes out of regulation, in hundredths of milliseconds. A value of zero represents the minimum allowable amount of time (at least 100 μ s) before the power supply is expected to go out of regulation.

The 16-bit read/write *pfi_source* field provides the 16-bit node ID of the source of the power-fail-imminent broadcast. To validate the applicability of the POWER_FAIL_IMMINEENT field, the node shall compare the values contained within its POWER_FAIL_IMMINEENT.*pfi_source* and POWER_SOURCE.*power_source* fields. Unless these two values are the same, the value of the *pfi_delay* field shall be ignored.

8.3.2.3.4 POWER_SOURCE register

Optional. If implemented, the POWER_FAIL_IMMINEENT register shall also be implemented.

The POWER_SOURCE register is used to screen the writes to the POWER_FAIL_IMMINEENT register of the node, so that only those from the power supply of the node are accepted. The bus manager shall initialize this register during the configuration process. The techniques used by the bus manager to identify the affiliation between power supply nodes and their power consuming nodes are beyond the scope of this standard. Figure 8.8 shows the format of this register.

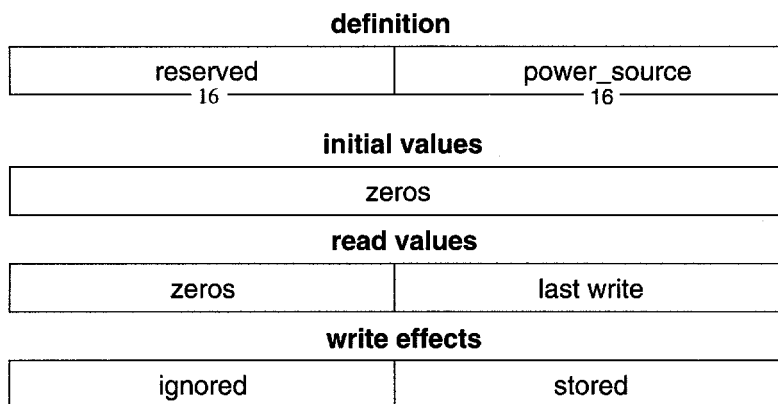


Figure 8.8—POWER_SOURCE format

The 16-bit read/write *power_source* field is used to select which of the writes to the POWER_FAIL_IMMINENT register shall be ignored. See 8.3.2.2.10 for details.

8.3.2.3.5 BUSY_TIMEOUT register

Optional. If not implemented, the node does not support retries.

The BUSY_TIMEOUT register provides fields that make visible transaction layer state variables that control the behavior of retry protocols. Figure 8.5 illustrates the format of these fields.

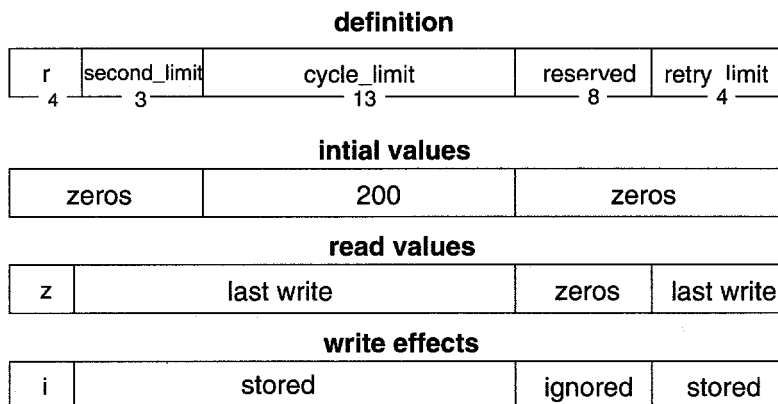


Figure 8.9—BUSY_TIMEOUT format

The *r* and *reserved* bits are reserved for future definition by this standard.

The *second_limit* and *cycle_limit* fields control the retry behavior utilized by the transaction layer when dual-phase retry protocol is in use, as described in 7.3.5.1. A zero value in these fields disables dual-phase retry protocol. A node whose transaction layer does not implement dual-phase retry protocol shall ignore writes to the *second_limit* and *cycle_limit* fields, and a read of the BUSY_TIMEOUT register shall return zeros. Together, the *second_limit* and *cycle_limit* fields define a time limit for retry attempts. The format of these fields and the units used are identical to the

second_count and *cycle_count* fields in the CYCLE_TIME register. Nodes that implement the dual-phase retry protocol shall not retransmit a packet after this time limit has elapsed. Time counts from the first transmission of the packet.

The *retry_limit* field controls the retry behavior utilized by the transaction layer when the single-phase retry protocol is in use, as described in 7.3.4. A node engaged in single-phase retry protocol shall not attempt retransmission of the busied packet if the *retry_limit* field is zero. Otherwise, the node shall retransmit the packet *retry_limit* times or until the receipt of something other than a busy acknowledgment.

8.3.2.3.6 BUS_MANAGER_ID register

Reserved, backplane environment.

Optional, cable environment. This register shall be implemented on isochronous-resource-manager-capable and bus-manager-capable nodes.

Special transaction limitations: Only the quadlet read and lock compare swap transactions shall be supported.

The BUS_MANAGER_ID register provides a “well-known” data storage location for the purpose of identifying the physical ID of the bus manager. During bus configuration, the process that determines which node, from perhaps more than one bus-manager-capable node, assumes the role of bus manager utilizes this location. Figure 8.10 shows the format of this register.

definition	
reserved 26	bus_mngr_id 6
initial values	
zeros	3F ₁₆
command reset values	
zeros	unchanged
read values	
zeros	last successful lock
lock effects	
ignored	conditionally written

Figure 8.10—BUS_MANAGER_ID format

The 6-bit *bus_mngr_id* field provides storage accessible through the quadlet read and lock (compare and swap) transactions.

The *bus_mngr_id* field provides a location that bus-manager-capable nodes shall use in order to determine which one among them contends successfully for the role of bus manager. The initial value of this field, 3F₁₆, is not a permissible value for a node on the Serial Bus since it is the broadcast address. The initial value indicates that no bus manager, as yet, has been selected. The procedure that bus-manager-capable nodes shall use to determine the bus manager is described in 8.4.2.5.

8.3.2.3.7 BANDWIDTH_AVAILABLE register

Optional. This register shall be implemented on isochronous-resource-manager-capable and bus-manager-capable nodes.

Special transaction limitations: Only the quadlet read and lock compare swap transactions shall be supported.

The BANDWIDTH_AVAILABLE register provides a “well-known” data storage location for the purpose of allocating and deallocating isochronous bandwidth by owners of isochronous resource. Although this register shall be implemented on every bus-manager-capable and isochronous-resource-manager-capable node, only the one on the active isochronous resource manager has valid data. Figure 8.11 shows the format of the BANDWIDTH_AVAILABLE register.

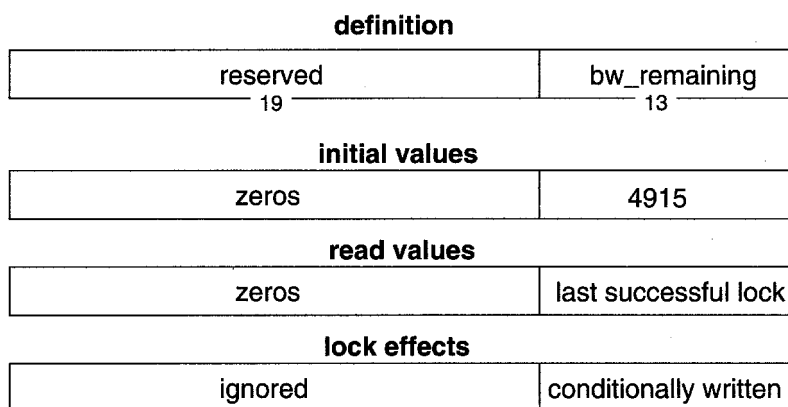


Figure 8.11—BANDWIDTH_AVAILABLE format

The read/write 13-bit *bw_remaining* field provides storage accessible through the quadlet read and lock (compare and swap) transactions.

The contents of the *bw_remaining* field represent the amount of isochronous bandwidth currently available for allocation. This bandwidth quantity is expressed in terms of time units referred to as bandwidth allocation units, where one unit represents the time to send one quadlet of data at a future (larger than present maximum definition) S1600 data rate, roughly 20 ns. The maximum possible value of this register derives from the highest possible value of allowed bandwidth at the S1600 data rate for one isochronous cycle of 125 μ s, namely 6144 bandwidth allocation units. In order to permit a small amount of asynchronous traffic to be interleaved with each isochronous cycle, a fixed 25 μ s shall be subtracted from the theoretical maximum of 6144 bandwidth allocation units. The initial value of *bw_remaining* shall be equivalent to a maximum of 100 μ s of isochronous traffic, or 4915 bandwidth allocation units. Such a reservation is necessary even in dedicated isochronous communication environments to support command and control operations that proceed by means of asynchronous traffic on the bus.

8.3.2.3.8 CHANNELS_AVAILABLE register

Optional. This register shall be implemented on isochronous-resource-manager-capable and bus-manager-capable nodes.

Special transaction limitations: Only the quadlet read and lock compare swap transactions shall be supported.

The CHANNELS_AVAILABLE register provides a “well-known” data storage location through which Serial Bus nodes may cooperate in the allocation and deallocation of isochronous channel numbers in the range zero to 63. Although this register shall be implemented on every bus-manager-capable and isochronous-resource-manager-capable node, only the one on the active isochronous resource manager has valid data. Bits allocated in the *channels_available_hi* field of this register shall start at bit zero (channel number zero), and additional channel numbers shall be represented in a monotonically increasing sequence of bit numbers up to a maximum of bit 31 (channel number 31). Bits allocated in the *channels_available_lo* field of this register shall start at bit zero (channel number 32), and additional channel numbers shall be represented in a monotonically increasing sequence of bit numbers up to a maximum of bit 31 (channel number 63). Figure 8.12 shows the format of the CHANNELS_AVAILABLE register.

definition	
channels_available_hi	32
channels_available_lo	32
initial values	
ones	
ones	
read values	
last successful lock	
last successful lock	
lock effects	
conditionally written	
conditionally written	

Figure 8.12—CHANNELS_AVAILABLE format

The read/write 32-bit *channels_available_hi* and *channels_available_lo* fields provide storage accessible through the quadlet read and lock (compare and swap) transactions.

Bits within this register indicate which of the (zero through 63) isochronous channel numbers are in use. For each bit, a zero value indicates this channel is owned and therefore unavailable for use. Otherwise, the channel is available, and a node that wishes to obtain ownership of the channel shall use the procedure described in 8.4.2.7.

8.3.2.3.9 MAINT_CONTROL register

Optional. This register may be implemented as a means of diagnosing and verifying the error detection logic within a Serial Bus system.

The MAINT_CONTROL register provides fields that enable the generation of Serial Bus errors. Figure 8.13 shows the format of this register.

The read/write *e_hcrc* bit affects the generation of packet header CRC value. If *e_hcrc* is one, the packet header CRC component of the next primary packet generated by this node shall be in error or shall be invalid; otherwise, the bit has no effect. This bit shall be cleared to zero immediately after the next packet for this node is generated.

The read/write *e_dcrc* bit affects the generation of the data CRC value. If *e_dcrc* is one, the data CRC component of the next primary packet generated by this node shall be in error or shall be invalid; otherwise, the bit has no effect. This bit shall be cleared to zero immediately upon transmission of the erroneous CRC.

The read/write *no_pkt* bit affects the generation of the next primary packet. If *no_pkt* is one, the next primary packet to be generated by this node shall be discarded. This bit shall be cleared to zero immediately after the next packet for this node is discarded.

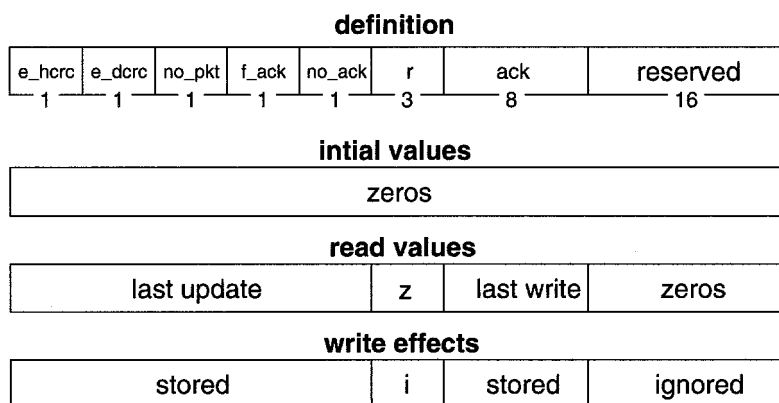


Figure 8.13—MAINT_CONTROL format

The read/write *f_ack* bit affects the generation of the acknowledge packet. If *f_ack* is one, the *ack* field shall be used within the next acknowledge packet generated by this node. This bit shall be cleared to zero immediately after the next acknowledge packet for this node is generated.

The read/write *no_ack* bit affects the generation of the acknowledge packet. If *no_ack* is one, the next acknowledge packet (that would normally have been generated by this node) is not sent. This bit shall be immediately cleared to zero when the next acknowledge packet for this node is discarded.

The 8-bit read/write *ack* field contains the 8-bit acknowledge packet (*ack_code* and *ack_parity*) to be supplied when the *f_ack* bit indicates a modified acknowledge packet is to be generated.

These control bits allow errors to be inserted into various places in the packets generated by this node. After the completion of the error insertion, enabled error-insertion controls are disabled.

NOTE — If the control bits are set via a Serial Bus write transaction into this node, then the error insertion shall not affect this write; the acknowledgment of the write request and the following response subaction (if present) are not affected. The next primary packet or acknowledge packet generated after this write response shall be affected.

8.3.2.3.10 MAINT_UTILITY register

Optional. This register may be implemented for diagnostic purposes.

The MAINT_UTILITY register is a safe target for diagnostic read and write transactions. The 32-bit register is read/write, and when it is read it returns the value last written. The register otherwise has no effect on the state of the node. Figure 8.14 shows the contents of the register.

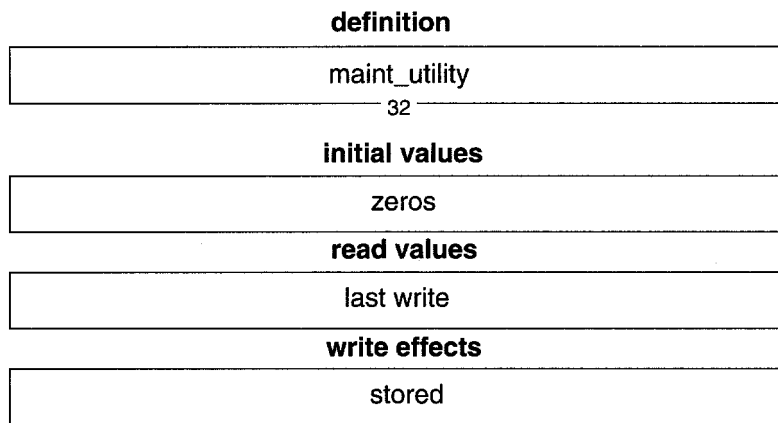


Figure 8.14—MAINT_UTILITY format

8.3.2.4 Unit registers

The CSR Architecture reserves address space for node-dependent resources within its initial units space, defined as the space above FFFF F000 0800₁₆. Serial Bus reserves a portion of initial units space for bus-dependent use, summarized in table 8.4. The addresses of these registers are specified in terms of offsets within the initial register space, where the base of the initial register space (from the beginning of the initial node space) is FFFF F00 0000₁₆. Any other CSRs specific to units (peripheral devices) implemented within the node are expected to share the initial units space and shall not be located within the range reserved by the Serial Bus.

Table 8.4—Serial-Bus-dependent registers in the initial units space

Offset	Name	Notes
800 ₁₆ –FFC ₁₆		Reserved for Serial Bus.
1000 ₁₆ –13FC ₁₆	TOPOLOGY_MAP	Present at the bus manager only.
1400 ₁₆ –1FFC ₁₆		Reserved for Serial Bus.
2000 ₁₆ –2FFC ₁₆	SPEED_MAP	Present at the bus manager only.
3000 ₁₆ –FFFC ₁₆		Reserved for Serial Bus.

Except as specified in this or future Serial Bus standards, unit architectures shall not implement any CSRs that fall within this address space.

The following clauses provide detailed definitions of the Serial-Bus-dependent registers located within the initial units space.

8.3.2.4.1 TOPOLOGY_MAP registers (cable environment)

Not implemented, backplane environment.

Optional, cable environment. These registers shall be implemented on bus manager capable nodes. These registers shall be read-only.

The bus manager shall generate a record of the self-ID packets observed plus the self-ID packet transmitted by the bus manager during the most recent self-identify process; this record is externally accessible through the

TOPOLOGY_MAP registers. This map shall be the resource used to deduce topological information about the given bus configuration. Although this register can be implemented on every node, only the one on the active bus manager is guaranteed to have valid data. Figure 8.15 shows the format of these TOPOLOGY_MAP registers.

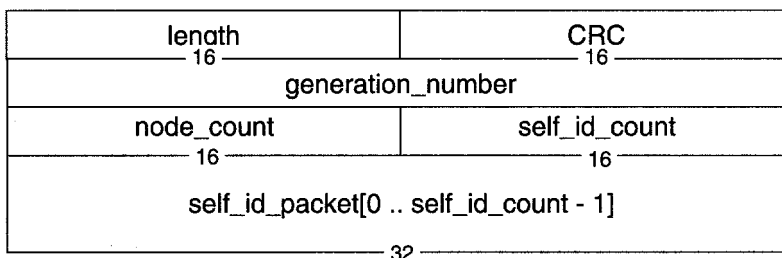


Figure 8.15—TOPOLOGY_MAP format

The 16-bit *length* field specifies the number of following quadlets in the topology map. The 16-bit *CRC* field covers all of the following quadlets. The *CRC* value is the CRC-16 algorithm described in clause 8.1.5 of ISO/IEC 13213 : 1994.

The 32-bit *generation_number* value specifies how many times this node has generated the topology map since its last power reset.

The 16-bit *node_count* value specifies the number of nodes present on the local Serial Bus. The *node_count* is redundant, since it can be derived from the self-ID packets available in the topology map.

The 16-bit *self_id_count* value specifies the number of self-ID packets stored at the following quadlet addresses.

The 32-bit *self_id_packet* fields in the following quadlets are a copy of the first quadlet of all self-ID packets observed by the bus manager, with the addition of a copy of the self-ID packet transmitted by the bus manager. The self-ID packets shall be stored in the order observed on the Serial Bus. The bus manager shall insert a copy of its own transmitted self-ID packets in the proper sequence location in the topology map.

NOTE — The values of these registers may be in an inconsistent state if the bus manager updates the topology map at the same time another node reads it. In order to guarantee accurate topology map information, 8.4.4.5 defines a procedure to access the topology map.

8.3.2.4.2 SPEED_MAP registers (cable environment)

Not implemented, backplane environment.

Optional, cable environment. These registers shall be implemented on bus-manager-capable nodes. These registers shall be read-only.

The bus manager shall generate a speed map for use by other nodes that need to know the maximum transfer rate supported between pairs of local Serial Bus nodes. The speed map is an array of vectors, where each vector entry indicates the maximum data transfer rate supported between two nodes. Although this register can be implemented on every node, only the one on the active bus manager is guaranteed to have valid data. Figure 8.16 shows the format of these SPEED_MAP registers.

length 16		CRC 16	
generation_number			
speed_code[0]	speed_code[1]	speed_code[2]	speed_code[3]
(speed_code[4] through speed_code[4027])			
speed_code[4028]	speed_code[4029]	undefined	undefined

Figure 8.16—SPEED_MAP format

The 16-bit *length* field specifies the number of following quadlets in the speed map; its value shall be $3F1_{16}$. The 16-bit *CRC* field covers all of the following quadlets within the SPEED_MAP. The *CRC* value shall be calculated by the CRC-16 algorithm described in clause 8.1.5 of ISO/IEC 13213 : 1994.

The 32-bit *generation_number* value specifies how many times this node has generated the speed map since its last power reset.

The two least-significant bits of the *speed_code* bytes specify one of the data transfer speeds S100, S200, or S400. The remaining most-significant six bits are reserved for future standardization and may not be relied upon to be read as zeros.

The 4030 *speed_code[i]* bytes, for values of $i = 64 \cdot m + n$, specify the highest data transfer speed usable between Serial Bus nodes with the physical IDs m and n . When m equals n , the value of *speed_code[i]* represents the speed capabilities of the node. The value of *speed_code[i]* is undefined for any i calculated where either m or n is a physical ID not assigned to any node on the Serial Bus. For any two nodes with the physical IDs m and n , the value of *speed_code[i]* and *speed_code[j]* shall be the same where $i = 64 \cdot m + n$ and $j = 64 \cdot n + m$.

NOTE — The values of these registers may be in an inconsistent state if the bus manager updates the speed map at the same time another node reads it. In order to guarantee accurate speed map information, 8.4.4.3 defines a procedure to access the speed map.

8.3.2.5 Configuration ROM

Transaction-capable Serial Bus nodes shall implement configuration ROM, as defined in clause 8. of ISO/IEC 13213 : 1994. Two configuration ROM formats are supported: minimal and general. The minimal ROM format provides a 24-bit company identifier. The general ROM format provides additional information in a Bus_Info_Block and a Root_Directory. Entries within the Root_Directory may provide information or may provide a pointer to another directory (which has the same structure as the Root_Directory) or a leaf (which contains information). The Unit_Directories contain information about the units, such as their software version number and their location within the address space of the node. Figure 8.17 illustrates the structure of these directories and leaves.

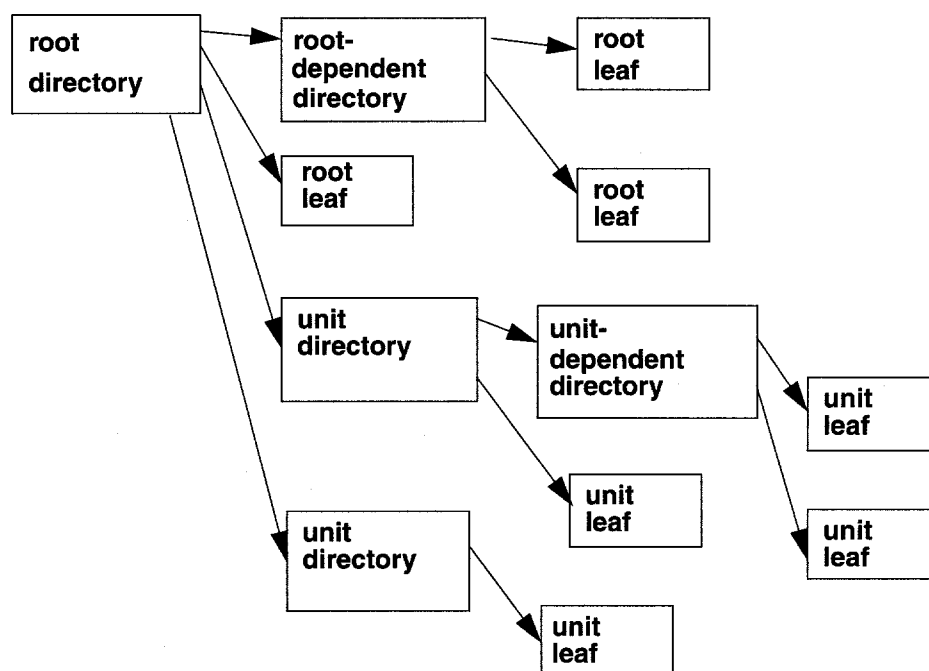


Figure 8.17—Configuration ROM hierarchy

Either the minimal ROM format or the general ROM format may be implemented, but bus-manager-capable, cycle-master-capable, and isochronous-capable nodes shall implement the general ROM format. It is expected that many unit architectures, e.g., the Serial Bus Protocol, will also use the general ROM format, since it provides a standard and extensible access mechanism to unit-dependent parameters.

8.3.2.5.1 Company ID

The term “company ID” is used throughout this clause to identify uniquely vendors that manufacture or specify components. To obtain a company ID (or an Organizationally Unique Identifier, OUI), contact:

Registration Authority Committee (RAC)
 The Institute of Electrical and Electronic Engineers, Inc.
 445 Hoes Lane
 Piscataway, NJ 08855-1331
 USA
 (908) 562-3813

Ask for a company ID for your organization. See 8.3.2.4.1, 8.3.2.5, 8.3.2.5.5.1, and 8.3.2.5.7.1 for descriptions of how the IEEE/RAC-administered company ID value is used in the identifier values defined by this standard.

Your company need not obtain a company ID if it has been previously assigned an IEEE **48-bit Globally Assigned Address Block** or an IEEE-assigned **Organizationally Unique Identifier (OUI)** for use in network applications. However, be aware that the (left through right) order of the bits within the company ID value is not the same as the (first through last) network-transmission order of the bits within these other identifiers. Consult the IEEE Registration Authority for clarifying documentation.

8.3.2.5.2 Minimal ROM format

The minimal ROM implementation consists of a single quadlet, which provides a 24-bit company ID value as illustrated in figure 8.18.

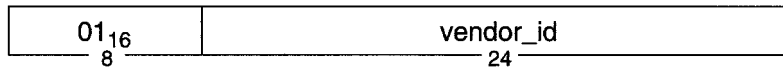


Figure 8.18—Minimal ROM format

The *vendor_id* value shall be the 24-bit company ID of the vendor that manufactured the module. The IEEE/RAC uniquely assigns a company ID to each module vendor, as described in clause 8.7 of ISO/IEC 13213 : 1994. Note that the minimal ROM format does not preclude the presence of additional vendor-dependent information in the ROM.

8.3.2.5.3 General ROM format

In its more general form, the ROM directory structure is a hierarchy of information blocks; the blocks higher in the hierarchy point to the blocks beneath them. The locations of the initial blocks (which include the Bus_Info_Block and Root_Directory) are fixed. The location of other entries (unit directories, root, and unit leaves) is vendor-dependent but shall be specified by entries within the Root_Directory or its associated directories.

Figure 8.19 shows a general ROM implementation. Note that there may be multiple Unit_Directories, one for each unit at the node.

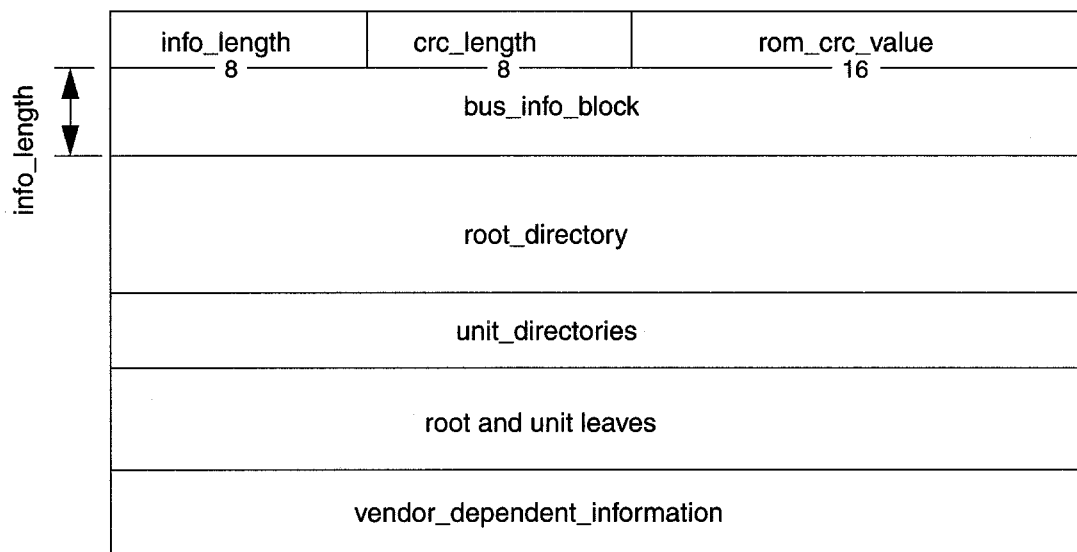


Figure 8.19—General ROM format

The *info_length* field shall have a value greater than one to distinguish the general ROM format from the minimal ROM format (whose *info_length* field has a value of one). The *info_length* value specifies the number of quadlets contained within the following Bus_Info_Block data structure.

The *crc_length* field specifies how many of the following quadlets are protected by the *rom_crc_value*. The minimum number of quadlets that shall be protected is the number within the Bus_Info_Block. The maximum number of quadlets that may be protected, 255, provides protection for the largest configuration ROM size permitted for Serial Bus, 1024 bytes.

The *rom_crc_value* shall be calculated according to the CRC-16 algorithm described in clause 8.1.5 of ISO/IEC 13213 : 1994.

8.3.2.5.4 Bus_Info_Block

The Serial Bus Bus_Info_Block shall have the format shown in figure 8.20.

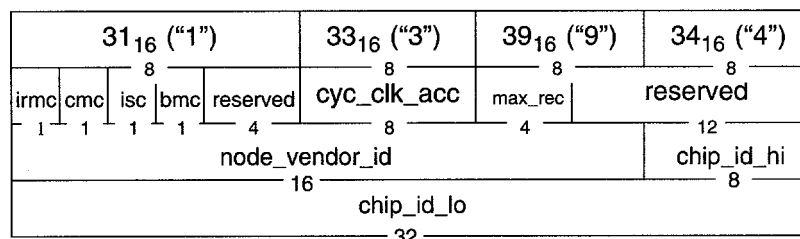


Figure 8.20—Bus_Info_Block format

The first quadlet contains the string "1394" in ASCII as the bus_name value required by the CSR Architecture.

The *irmc* bit shall be one if the node is isochronous resource manager capable; otherwise, the *irmc* value shall be zero.

The *cmc* bit shall be one if the node is cycle master capable; otherwise, this value shall be zero.

The *isc* bit shall be one if the node supports isochronous operations; otherwise, this value shall be zero.

The *bmc* bit shall be one if the node is bus manager capable; otherwise, this value shall be zero.

The *reserved* fields are reserved for future definition by Serial Bus and shall be zeros.

The *cyc_clk_acc* field specifies the cycle master clock accuracy of the node in parts per million. If the *cmc* bit is set, the value in this field shall be between zero and 100. If the *cmc* bit is cleared, this field shall be all ones.

The *max_rec* field defines the maximum payload size of an asynchronous block write transaction addressed to the node. The range of the maximum payload size is from 4 bytes to 2048 bytes. Within the range of defined payload sizes, the maximum size is equal to $2^{max-rec+1}$, as specified by table 8.5.

Table 8.5—Encoding of max_rec field

max_rec (binary encoding)	Maximum size in bytes (decimal values)
0000	Not specified
0001	4
0010	8
0011	16
0100	32
0101	64
0110	128
0111	256
1000	512
1001	1024
1010	2048
1011 to 1111	Reserved

This *max_rec* field does not place any limits on the maximum payload size in asynchronous data packets, either requests or responses, that the node may transmit.

The *node_vendor_id* field is a copy of the 24-bit node company ID present in the Node_Unique_Id leaf of configuration ROM, described in 8.3.2.5.7.1.

The *chip_id_hi* and *chip_id_lo* fields are copies of the 40-bit chip ID present in the Node_Unique_Id leaf of configuration ROM, described in 8.3.2.5.7.1.

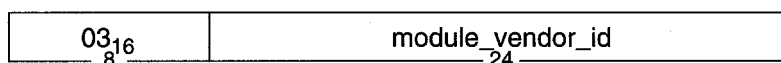
Together the *node_vendor_id*, *chip_id_hi*, and *chip_id_lo* fields form a 64-bit node unique identifier. Because physical addresses on the Serial Bus may change after a bus reset, this unique identifier is the only secure method of node identification.

8.3.2.5.5 Root_Directory

The configuration ROM for each Serial Bus node that implements the general ROM format shall contain a Root_Directory. The Root_Directory shall contain Module_Vendor_Id, Node_Capabilities, and Node_Unique_Id entries.

8.3.2.5.5.1 Module_Vendor_Id entry

The Module_Vendor_Id entry is an immediate entry in the Root_Directory that provides the company ID of the vendor that manufactured the module. Figure 8.21 shows the format of this entry.

**Figure 8.21—Module_Vendor_Id entry format**

03₁₆ is the CSR Architecture key_type and key_value for the Module_Vendor_Id entry.

The IEEE/RAC uniquely assigns the 24-bit *module_vendor_id* to each module vendor, as defined in clause 8.7 of ISO/IEC 13213 : 1994.

8.3.2.5.2 Node_Capabilities entry

The Node_Capabilities entry is an immediate entry in the Root_Directory that describes node capabilities. Figure 8.22 shows the format of this entry.

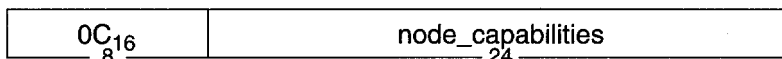


Figure 8.22—Node_Capabilities entry format

0C₁₆ is the CSR Architecture key_type and key_value for the Node_Capabilities entry.

The 24-bit *node_capabilities* field contains subfields defined within clause 8.4.11 of ISO/IEC 13213 : 1994. Serial Bus nodes shall implement the *spt*, *64*, *fix*, *1st*, and *drq* bits.

Transaction-capable nodes shall set the *spt* bit to one to indicate that the SPLIT_TIMEOUT register is implemented.

Transaction-capable nodes that initiate Serial Bus transactions shall set the *drq* bit to one. This indicates that the STATE_CLEAR.*dreq* bit is implemented. The STATE_CLEAR.*dreq* bit, if cleared by another node, inhibits the initiation of Serial Bus transactions, either asynchronous or isochronous.

All Serial Bus nodes shall set the *64*, *fix*, and *1st* bits to one. These indicate that the node implements the 64-bit fixed addressing scheme and that the STATE_CLEAR.*lost* bit is implemented.

8.3.2.5.3 Node_Unique_Id entry

The Node_Unique_Id entry is a leaf entry in the Root_Directory that describes the location of the Node_Unique_Id leaf within the configuration ROM. Figure 8.23 shows the format of this entry.

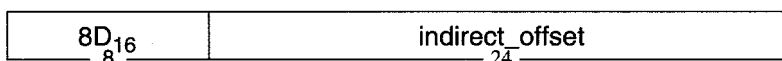


Figure 8.23—Node_Unique_Id entry format

8D₁₆ is the CSR Architecture key_type and key_value for the Node_Unique_ID entry.

The 24-bit *indirect_offset* field specifies the number of quadlets from the address of the Node_Unique_Id entry to the address of the Node_Unique_Id leaf within the configuration ROM.

8.3.2.5.6 Unit_Directories

With the exception of the Unit_Power_Requirements entry, the format of Unit_Directories within a Serial Bus node that implements unit architectures is beyond the scope of this standard. The format of Unit_Directories are described in clause 8.4.5.1 of ISO/IEC 13213 : 1994.

8.3.2.5.6.1 Unit_Power_Requirements entry (cable environment)

The `Unit_Power_Requirements` entry shall be implemented in the `Unit_Directory` of any unit that can consume bus power in excess of the level specified by the `pwr` field in the self-ID packet transmitted by the node. This entry shall not be implemented for any units that are locally powered. Figure 8.24 shows the format of this entry.

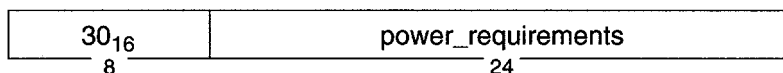


Figure 8.24—Unit_Power_Requirements entry format

30_{16} is the Serial Bus `key_type` and `key_value` for the `Unit_Power_Requirements` entry.

The 24-bit *power_requirements* field specifies the maximum additional bus power requirements of the unit, in deciwatts, beyond the power consumption enabled by the receipt of a link-on packet. This entry shall apply to units whose architecture implements a unit-dependent means of enabling bus power consumption in addition to that indicated by the `pwr` field of the self-ID packet.

8.3.2.5.7 Root_Leaves

`Root_Leaves` are located within the configuration ROM, as described in 8.3.2.4.2. The Serial Bus uses the `Root_Leaves` within the configuration ROM to store bus-dependent information referenced by `Root_Directory` entries required by the Serial Bus, namely the `Node_Unique_Id` leaf.

8.3.2.5.7.1 Node_Unique_Id leaf

Within the CSR Architecture, the node unique ID is a 64-bit number appended to a company ID value to create a globally unique 88-bit number. While conforming to the CSR Architecture standard, the Serial Bus additionally constrains the definition of the 64-bit node unique ID values so that they are unique within the global context of all Serial Bus nodes. Figure 8.25 shows the format of the `Node_Unique_Id` leaf.

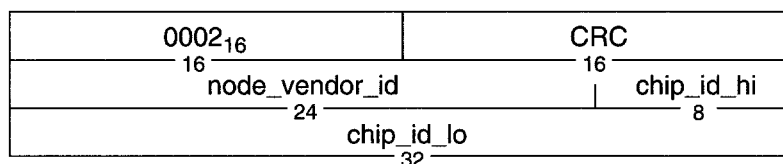


Figure 8.25—Node_Unique_Id leaf format

The 24-bit *node_vendor_id* value shall be the node company ID value from the `Root_Directory`.

The 8-bit *chip_id_hi* field is concatenated with the 32-bit *chip_id_lo* field to create a 40-bit chip ID value. The vendor specified by the *node_vendor_id* value shall administer the chip ID values. When appended to the *node_vendor_id* value, these shall form a unique 64-bit value called EUI-64 (Extended Unique Identifier, 64 bits). These EUI-64 values are, by definition, unique from other EUI-64 identifiers derived from the IEEE/RAC-provided company ID value.

8.3.3 Serial Bus management variables

Each node has a set of variables in the Serial Bus management layer that an application may query via Serial Bus services described in 8.2. Table 8.6 summarizes these variables.

Table 8.6—Serial Bus management variables

Variable name	Comment
Incumbent	Used only if the node is bus manager capable. Set to TRUE if the node is the bus manager; FALSE if the node did not win the role of bus manager.
Isochronous Resource Manager ID	Used only if the node is isochronous resource manager capable. Upon completion of the self-identify process, it shall be set to the physical ID of the isochronous resource manager.
NOTE — Other Serial Bus management variables, specifically Bus Time, Split-Transaction Timeout, Transaction Retry Limit, and Transaction Retry Timeout, are defined in 8.3.1.2.	

8.4 Serial Bus management operations

8.4.1 Bus configuration procedures (backplane environment)

The Serial Bus in the backplane environment requires no reconfiguration after a power reset, command reset, or the live insertion of a node. As soon as a reset event is complete, any node on the bus may begin to arbitrate for asynchronous access upon detection of an arbitration reset gap. It is only in the case of isochronous operations that Serial Bus configuration is necessary in the backplane environment.

Although this standard places no requirement on a Serial Bus in a backplane environment as to how it implements the functions needed to support isochronous operations, this standard strongly recommends that analogs of the cable environment be used. That is, one of the nodes on the bus in the backplane environment should function as the cycle master and another, potentially the same node, should function as the isochronous resource manager. Serial Bus nodes in the backplane environment that wish to allocate or deallocate isochronous bandwidth or channel number resources should use the same procedures used in the cable environment, namely those described in 8.4.2.5.

The subclauses that follow describe the procedures to be executed after a power reset in the backplane environment.

8.4.1.1 Unmanaged bus (backplane environment)

If there are no nodes on the Serial Bus that are isochronous resource manager capable, the Serial Bus is unmanaged and capable only of asynchronous transactions between nodes with enabled link layers upon completion of the power reset. The remainder of 8.3.2.5.7 is not applicable to an unmanaged Serial Bus.

8.4.1.2 Determination of the isochronous resource manager (backplane environment)

The process used in the backplane environment to select the isochronous resource manager from the contending isochronous-resource-manager-capable nodes is not specified by this standard. Since no procedures exist for the isochronous-resource-manager-capable nodes to resolve this contention by themselves, the application element at each node shall communicate to the bus manager layer, via an SB_CONTROL.request, whether or not the isochronous resource manager is to be enabled. Of necessity, the application elements at the isochronous-resource-manager-capable nodes shall have either *a priori* knowledge as to which node is to be the isochronous resource manager or else a procedure to select the isochronous resource manager. Although a procedure for selecting an isochronous resource manager is not specified, an example of such a procedure is given in annex G

Isochronous-resource-manager-capable nodes are identifiable by the *irmc* bit in the *Bus_Info_Block* as described in 8.3.2.5.

8.4.1.3 Determination of the cycle master (backplane environment)

As soon as the isochronous resource manager is determined, it shall activate a cycle-master-capable node to become the cycle master. Cycle-master-capable nodes are identifiable by the *cmc* bit in the *Bus_Info_Block*, as described in 8.3.2.5.

8.4.2 Bus configuration procedures (cable environment)

Since some node roles on the Serial Bus (e.g., root, isochronous resource manager, and bus manager) may change whenever the bus topology changes, a bus reset is the trigger that causes a redetermination of which nodes are to perform these functions. It is also possible that a node, in its management role as either an isochronous resource manager or bus manager, needs to change which nodes perform these functions. In this case, a bus reset initiated by the manager forces this process, even though the bus topology is unaltered.

When a bus reset occurs, all asynchronous and isochronous traffic on the Serial Bus ceases. Asynchronous transfers may resume as soon as the self-identify process that follows a bus reset has completed. Previously established isochronous data streams, either talker or listener, are to resume as soon as possible after the self-identify process completes. In general, the roles of cycle master, isochronous resource manager, and bus manager shall be redetermined before new allocation of isochronous resources can be performed and before the new topology and speed maps can be made available.

This subclause describes the interaction of Serial Bus nodes as they contend for the roles of isochronous resource manager and bus manager after a bus reset. The resolution of these conflicts defines the points in time at which each successive Serial Bus facility becomes available.

In order for this process to function properly, each Serial Bus node shall implement the appropriate state machines described in 8.4.5.1. These state machines describe specific, observable behavior from the vantage point of an imaginary, transaction-capable observer on the Serial Bus. The state machines also assume the existence of certain variables internal to each node implementation. However, the internal implementation of these state machines is beyond the scope of this standard.

8.4.2.1 Unmanaged bus (cable environment)

If there are no nodes on the Serial Bus that are either isochronous resource manager capable or bus manager capable, the Serial Bus is unmanaged and capable only of asynchronous transactions between nodes with already enabled link layers at the time of the bus reset. The remainder of 8.4.2 is not applicable to an unmanaged Serial Bus.

8.4.2.2 Prior isochronous traffic (cable environment)

If the previous cycle master remains the root¹⁵ (as determined immediately upon completion of the self-identify process), it shall resume cycle master operations. If the cycle master remains the root and therefore is able to resume broadcast of the cycle start packet, any isochronous talkers that had been enabled prior to the bus reset may resume isochronous operations as well.

If the node that was cycle master before the bus reset is no longer the root,¹⁶ isochronous talkers are unable to resume isochronous operations until a new cycle master is selected at a later step in this process.

¹⁵In normal operation, the previous cycle master will remain the root since it will have its *force_root* flag set.

¹⁶This condition is possible if two operating buses (both having cycle masters with their *force_root* flag set) are joined together.

8.4.2.3 Determination of the isochronous resource manager (cable environment)

After a bus reset, the PHY at each node participates in the self-identify process, as described in 4.4.2.3. During this process, the PHY layer of each node shall observe all self-ID packets from other nodes in order to generate its own self-ID packet. The link and higher layers of all nodes that are candidates to become either the isochronous resource manager or the bus manager shall observe all self-ID packets in order to determine the identity of the isochronous resource manager.

From all the nodes that are capable of becoming either the isochronous resource manager or the bus manager, one is selected as follows. Each of these candidate nodes shall transmit its own self-ID packet with both the *c* bit and the *I* bit set to one. Each of these candidate nodes shall also monitor all received self-ID packets in order to observe the largest physical ID from a packet with both the *c* and *I* bits set. The candidate node with the largest physical ID wins the role of the isochronous resource manager.

Candidate isochronous resource managers shall apply consistency checks to the observed self-ID packets to insure that

- a) The second quadlet of each self-ID packet is the logical inverse of the first quadlet
- b) The *phy_id* fields of self-ID packet #0 are in monotonically increasing sequence and the *bphy_id* fields of all self-ID packets other than #0 are equal to the *phy_id* field of the preceding self-ID packet #0
- c) The last self-ID packet set indicates that all its connected PHY ports are connected to children

If any of these requirements is not met, the candidate isochronous resource manager shall perform a bus reset.

NOTE — A candidate for the role of isochronous resource manager that is also bus manager capable may become the bus manager at a later step in the bus reset process, whether or not it won the role of the isochronous resource manager.

8.4.2.4 Reallocation of prior isochronous resources (cable environment)

In order to promote stability of Serial Bus configurations across a bus reset, the owners of isochronous resources established prior to the bus reset, i.e., channels and bandwidth, are required to reallocate these resources as soon as possible after a bus reset. As soon as the self-identify process has completed, the identity of the isochronous resource manager is determined, and all the owners of prior isochronous resources shall access the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers at the isochronous resource manager in order to reallocate these resources. The procedures described in 8.4.2.5 shall be used.

If an isochronous resource, either channel or bandwidth, cannot be reallocated, the Serial Bus node that owned the resources prior to the bus reset shall cause the corresponding isochronous talker to cease broadcast of isochronous packets.

8.4.2.5 Determination of the bus manager (cable environment)

One of the goals of the resolution process described in 8.4.2 is to promote stability of the Serial Bus configuration across bus resets. To this end, a node that had the role of bus manager prior to the bus reset may attempt to reestablish its role earlier than new candidates for this role. Specifically, immediately upon completion of the self-identify process, the incumbent bus manager (if any) shall make a lock request with an extended transaction code of compare and swap to the BUS_MANAGER_ID register at the isochronous resource manager node. The lock packet shall have an *arg_value* of 3F(16) and a *data_value* equal to the physical ID of the incumbent bus manager. The incumbent bus manager shall issue this lock request until it obtains a Request Status of COMPLETE and a Response Code of resp_complete. If the *old_value* received in the lock response packet is 3F₁₆, the incumbent bus manager has reestablished itself as the bus manager. If any other value is returned, it is the physical ID of the node that has won the role of bus manager; the incumbent manager has lost and shall not perform any of the functions of the bus manager.

Challengers for the role of bus manager shall follow a similar process. These are all the nodes on the Serial Bus that are bus manager capable but are not incumbent. These challengers shall wait a minimum of 125 ms before making a

lock request with an extended transaction code of compare and swap to the `BUS_MANAGER_ID` register at the isochronous resource manager node. Once the 125 ms delay period has elapsed, the challenger nodes shall implement the same procedure described for the incumbent bus manager node.

At the earliest point in this process that a node is selected as the bus manager, it shall perform the power management procedure described in 8.4.4.4, shall make available the `SPEED_MAP` and `TOPOLOGY_MAP` registers described in 8.3.2.4.2 and 8.3.2.4.1, respectively, and shall attempt to enable a cycle master as described in 8.4.2.6.

8.4.2.6 Determination of the cycle master (cable environment)

As described in 8.4.1.2, a cycle master node that remains the root node of the Serial Bus shall resume the broadcast of cycle start packets as soon as the self-identify process has completed. When the prior cycle master is no longer the root node after a bus reset, it shall not broadcast cycle start packets. In this case, it is necessary to activate a new cycle master. If a bus manager is present on the Serial Bus, it shall select and activate a cycle master as described in the following paragraphs. If there is no bus manager, the isochronous resource manager shall select and activate a cycle master as described in the following paragraphs.

The bus manager or, if a bus manager is not present, the isochronous resource manager shall examine the `Bus_Info_Block` of the root node of the Serial Bus. If this node is capable of becoming the cycle master, the bus manager or the isochronous resource manager shall

- a) If the bus has more than one node, set the `force_root` flag of the root node¹⁷ and clear the `force_root` flag of all other nodes. If the bus has only a single node, the `force_root` flag shall be cleared.¹⁸
- b) Set the `cmstr` bit in the `STATE_CLEAR` register of the root node.

If the root node does not have cycle master capabilities and the bus has more than one node, the bus manager or isochronous resource manager shall select one of the nodes on the Serial Bus that has cycle master capabilities as a candidate to become the new cycle master. Because the selected node is not, at present, the root, it is necessary that another bus reset be initiated to modify the topology of the Serial Bus. Prior to the bus reset, the bus manager or isochronous resource manager shall transmit a PHY configuration packet with the `r` bit set to one and the `root_id` field set to the physical ID of the candidate cycle master node. Because of this, the candidate cycle master node is guaranteed to become the root of the new Serial Bus configuration. Another application of the procedures described in 8.4.1 will permit the resumption of isochronous traffic with the new cycle master.

8.4.2.7 Power management by the isochronous resource manager (cable environment)

Although the identity of the isochronous resource manager is established as soon as the self-identify process has completed, the isochronous resource manager shall not engage in power management operations until it has determined that no bus manager is present on the Serial Bus.

In order to determine whether or not a bus manager is present, the isochronous resource manager shall wait a minimum of 625 ms after the self-identify process completes. When this delay has elapsed, the isochronous resource manager examines its own `BUS_MANAGER_ID` register. If the `BUS_MANAGER_ID` register is equal to `3F16`, there is no bus manager present, and the isochronous resource manager may transmit link-on packets to enable the link layer at other nodes. If the `BUS_MANAGER_ID` register contains a value other than `3F16`, it is the physical ID of the bus manager, and the isochronous resource manager shall not engage in any power management activities.

¹⁷This guarantees that the root node will stay root after a bus reset, providing that two operating buses (each having a node with the `force_root` flag set) are not connected together.

¹⁸Isolated nodes shall not set their `force_root` flag, since this might cause the root to shift in any operating bus to which the isolated node is attached.

8.4.2.8 Allocation of new isochronous resources (cable environment)

Nodes that wish to acquire isochronous resources, either bandwidth or channels, not allocated prior to the bus reset shall wait a minimum of 1000 ms before attempting the allocation protocols described in 8.4.3. Note that an increase in bandwidth allocation by a node that owned bandwidth prior to a bus reset is also considered an allocation of new resources. In this case, the node that wishes to increase its bandwidth allocation may reestablish the original allocation immediately upon completion of the self-identify process but shall wait until 1000 ms have elapsed before making any attempt to gain additional bandwidth.

8.4.3 Isochronous management (cable environment)

The isochronous resource manager, through the means of the `BANDWIDTH_AVAILABLE` and `CHANNELS_AVAILABLE` registers, provides the means for the “owners” of isochronous resources to share the total isochronous resources available on the Serial Bus. Owners of isochronous resources, either bandwidth or channel assignments, gain ownership of these resources through the procedures described in 8.4.3.1 and 8.4.3.2. Note that the owner of an isochronous resource need be neither the talker nor a listener for the channel in question.

In the cable environment, a Serial Bus node that wishes to acquire ownership of either bandwidth or a channel shall have the capability to monitor self-ID packets after a bus reset, as described in 8.4.2.3. This is essential in order that the node know the location of the isochronous resource manager. Among the self-ID packets that have both the *l* and the *c* bits set, the one with the largest physical ID specifies the isochronous resource manager.

NOTE — Clause 8.4.1 recommends that isochronous management in the backplane environment occur in a manner similar to that described in this clause. It is beyond the scope of this standard to specify a method for determining the location of the isochronous resource manager in the backplane environment, although an example of such a method is given in annex G

Once the physical ID of the isochronous resource manager has been determined, the node may attempt the allocation of bandwidth or channels. An isochronous-capable node shall not transmit isochronous packets unless the required bandwidth and channel number have been allocated.

8.4.3.1 Bandwidth allocation

The `BANDWIDTH_AVAILABLE` register, located at the isochronous resource manager, reflects the aggregate amount of isochronous bandwidth available on the Serial Bus at a given point in time. Since more than one entity may wish to allocate bandwidth at essentially the same time, a compare and swap protocol shall be used to change the `BANDWIDTH_AVAILABLE` register. The use of compare and swap guarantees atomicity of access to the value of this register in the distributed environment of the Serial Bus.

A transaction-capable node that wishes to allocate isochronous bandwidth shall perform the following actions:

- a) The node shall use a quadlet read request to obtain the value of the `BANDWIDTH_AVAILABLE` from the isochronous resource manager. The value returned represents the maximum bandwidth the node may allocate, as described in 8.3.2.3.7.
- b) The bandwidth desired shall be calculated to include the bandwidth needed by the application plus all overhead associated with isochronous data transfer, e.g., isochronous gap, arbitration, data prefix, and data end. The bandwidth desired shall not exceed the current bandwidth available. If a node desires more bandwidth than is available, the node shall either reduce its request for bandwidth or shall delay some period of time and retry the allocation later.
- c) The bandwidth allocation shall be attempted by a lock request with an extended transaction code of compare and swap to the `BANDWIDTH_AVAILABLE` register at the isochronous resource manager. The lock packet shall have an *arg_value* equal to the value of the current bandwidth available and a *data_value* equal to the bandwidth available less the bandwidth desired.
- d) If the lock transaction fails to complete, i.e., the Request Status returned is not `COMPLETE` or the Response Code is not `resp_complete`, the node may retry the entire bandwidth allocation procedure.

- e) If the lock transaction is successful and the *old_value* received is equal to the *arg_value* transmitted in the lock request, the allocation of isochronous bandwidth is successful. In all other cases, the bandwidth allocation has failed and may be retried as appropriate. A subsequent read of the BANDWIDTH_AVAILABLE register is not necessary, since the *old_value* returned by the failing lock request reflects the current bandwidth available.

When this procedure succeeds, the requesting node becomes the owner of the isochronous bandwidth. Isochronous bandwidth shall not be deallocated by any node other than the owner of the bandwidth unless the owner of the bandwidth has requested, by means beyond the scope of this standard, another node to deallocate the bandwidth on behalf of the owner.

Bandwidth deallocation is performed by an essentially similar protocol. The owner of the bandwidth shall first read the value of the BANDWIDTH_AVAILABLE register at the isochronous resource manager in order to determine how much bandwidth is currently available. A subsequent lock transaction shall be used to attempt to increase the value of the BANDWIDTH_AVAILABLE register by the amount of bandwidth returned. Lock requests that fail should be retried until the bandwidth is successfully deallocated.

8.4.3.2 Channel allocation

The CHANNELS_AVAILABLE register, located at the isochronous resource manager, reflects (in the form of a bit map) which isochronous channels are in use on the Serial Bus at a given point in time. Since more than one entity may wish to allocate channels at essentially the same time, a compare and swap protocol shall be used to change this register. The use of compare and swap guarantees atomicity of access to these register values in the distributed environment of the Serial Bus.

A transaction-capable node that wishes to allocate isochronous channels shall perform the following actions:

- a) The node shall use a quadlet read request to obtain the value of the CHANNELS_AVAILABLE register from the isochronous resource manager. The value returned represents the availability of each channel by the value of the bit that corresponds to the ordinal position of the channel, as described in 8.3.2.3.8. Zero bits represent channels in use and one bits represent available channels.
- b) If unused channels are available, the request for a channel is made with a lock request with an extended transaction code of compare and swap to the CHANNELS_AVAILABLE register at the isochronous resource manager. The lock packet shall have an *arg_value* equal to the bit mask that represents the current state of channel availability and a *data_value* equal to the same value except that the bit(s) that represents the desired channel(s) shall be cleared to zero. A node shall not attempt to allocate a channel that is in use; in such a circumstance, the node shall either select a different, unused channel or shall delay some period of time and retry the allocation later.
- c) If the lock transaction fails to complete, i.e., the Request Status returned is not COMPLETE or the Response Code is not resp_complete, the node may retry the entire channel allocation procedure.
- d) If the lock transaction is successful and the *old_value* received is equal to the *arg_value* transmitted in the lock request, the allocation of isochronous channel(s) is successful. In all other cases, the channel allocation has failed and may be retried as appropriate. A subsequent read of the CHANNELS_AVAILABLE register is not necessary, since the *old_value* returned by the failing lock request reflects the current availability of isochronous channels.

When this procedure succeeds, the requesting node becomes the owner of the isochronous channel(s). Isochronous channel(s) shall not be deallocated by any node other than the owner of the channel(s) unless the owner of the channel(s) has requested, by means beyond the scope of this standard, another node to deallocate the channel(s) on behalf of the owner.

Channel deallocation is performed by an essentially similar protocol. The owner of the channel shall first read the value of the CHANNELS_AVAILABLE register at the isochronous resource manager in order to obtain an accurate picture of which channels are in use. A subsequent lock transaction shall be used to attempt to set the bit that

corresponds to the channel being released. Lock requests that fail should be retried until the channel is successfully deallocated.

8.4.3.3 Bandwidth set-aside

If the SB_CONTROL.request service has been used to communicate a nonzero value for the Bandwidth Set-aside parameter to the bus manager, the bus manager shall use the bandwidth allocation procedure described in 8.4.2.6 to reduce the total amount of isochronous bandwidth by the set-aside amount. Note that it is possible for the bus manager to fail in the attempt to reduce the bandwidth available if other isochronous resource owners have already reduced the bandwidth available to an amount less than the Bandwidth Set-aside. If this is the case, the bus manager shall set the bandwidth available to zero.

8.4.4 Power management (cable environment)

One of the benefits of the Serial Bus in the cable environment is that the cable itself can supply modest amounts of power to connected nodes. Along with this benefit comes the requirement that the distribution of power be managed, in either a sophisticated or a simple fashion. The bus manager shall implement a power management scheme that performs consistency checks and the isochronous resource manager, in the absence of a bus manager, may implement an elementary power management scheme.

Each node on the Serial Bus has at least three components that require power to function: the physical layer, the link layer, and the Serial Bus management layer. There may be additional application components, visible as units, that also require power. The necessary power may originate from a power source associated with the node or it may be taken from the cable.

8.4.4.1 PHY power management

It is a minimum requirement that the physical layer of a node be powered and active while the node is connected to the Serial Bus. The self-ID packet generated by the PHY during the self-identify processes advertises whether the PHY consumes power from the bus or is self-powered.

It is also a minimum requirement that any Serial Bus node with an active physical layer shall have an active and powered node controller component within the Serial Bus management layer. This is necessary so that the PH_EVENT.indication with an event of LINK ON may be processed and in turn generate the appropriate LK_CONTROL.requests to initialize and activate the link layer.

Nodes whose physical layer is not active are effectively absent from the Serial Bus. If such a node is a leaf, no problems result; if power is subsequently available and the PHY becomes active, it generates a bus reset and the node identifies itself as part of the new topology. If such a node is in the middle of a Serial Bus it effectively breaks the bus in two; the separate Serial Buses function normally. If power is subsequently supplied and the PHY becomes active, the two Serial Buses would be reconfigured as one after the bus reset.

8.4.4.2 Link power management

The power source for the link layer determines the state of the link layer after a power-on reset. If the link layer obtains its power from the cable, the link layer shall be inactive and unpowered after a power-on reset until the receipt of a link-on packet. If the link layer is independently powered, it may be active or inactive after a power-on reset, at the option of the Serial Bus management layer. An independently powered but inactive link layer may become active either through the receipt of a link-on packet or as the result of an action taken by the node controller.

8.4.4.3 Unit power management

The power requirements of one or more units within a Serial Bus node are visible in two ways: the self-ID packet transmitted by the node during the self-identify process and the Unit_Power_Requirements entries in the configuration

ROM. The self-ID packet describes the maximum power the node may consume once it is enabled by a link-on packet. If one or more units implement architectures that can consume additional power once they are enabled by unit-dependent means, these additional power requirements shall be specified by a `Unit_Power_Requirements` entry for each such unit. The maximum power consumption of a node may be calculated by adding the value of the `power_requirements` fields from all the `Unit_Power_Requirements` entries to the power requirements in the `pwr` field of the self-ID packet.

8.4.4.4 Power management by the bus manager

When a bus manager is selected from among the candidate bus-manager-capable nodes, it shall perform power management functions as follows:

- a) The total power requirements of the bus shall be calculated from the `pwr` fields in the self-ID packets received during the self identify process
- b) The total power available on the bus shall be calculated from the self-ID packets
- c) If the power requirements exceed the power available, an `SB_EVENT.indication` with a parameter that indicates Insufficient Cable Power shall be provided to the application at the bus manager node
- d) If the power requirements are less than or equal to the power available, link-on packets shall be transmitted to all nodes whose self-ID packet indicated an inactive link layer

If there is insufficient power to activate the link layers of all nodes on the Serial Bus, it shall be the responsibility of an application at the bus manager node to determine which, if any, of the nodes should receive link-on packets. The application shall use the `SB_CONTROL.request` service to cause the bus manager to transmit the necessary link-on packets.

8.4.4.5 Power management by the isochronous resource manager

The isochronous resource manager, in the absence of a bus manager, may issue link-on packets to all nodes whose self-ID packet indicated an inactive link layer. There is no requirement for the isochronous resource manager to perform any power requirement calculations prior to the transmission of link-on packets.

8.4.5 Speed management (cable environment)

A bus manager simplifies the operation of mixed-speed Serial Bus configurations by making `SPEED_MAP` registers available to describe the maximum data transfer speed usable between any two nodes. Clause 8.3.2.4.2 describes the format of the `SPEED_MAP` registers.

As soon as a bus manager is selected from the candidates for this role, the bus manager shall compute the `SPEED_MAP` registers from the self-ID packets.

Before the bus manager is changes the speed map for any reason, it shall first set the `length` field of the `SPEED_MAP` registers to zero. After the changes are complete, the `generation_number` shall be incremented, a new CRC calculated, and the `length` field restored.

8.4.5.1 Accessing the speed map

Applications at transaction-capable nodes that need to obtain accurate information from the `SPEED_MAP` registers shall use the following procedure, where m and n are the physical IDs of the nodes of interest:

- a) Read the first quadlet of the `SPEED_MAP` registers, which contains a `length` field that describes the data that follows and a `CRC` field whose value has been calculated over that data length
 - 1) If the `length` field is zero, the `SPEED_MAP` is not valid, and the application shall not utilize any data contained within the `SPEED_MAP`

- 2) If the *length* field is nonzero, the application shall read the SPEED_MAP quadlet that contains *speed_code[i]*, where *i* is calculated from *m* and *n* as described in 8.3.2.4.2
- b) Read the first quadlet of the SPEED_MAP registers once again and compare its value to the value obtained the first time that the *length* and *CRC* fields were read
 - 1) If they are identical, the *speed_code[i]* value obtained is accurate and shall be the upper bound for data transfer speed between this node and the destination node
 - 2) If the values are not identical, the *speed_code[i]* value obtained is not guaranteed to be accurate and shall not be used

If this procedure fails to return a usable *speed_code[i]* value, either because the *length* field was zero or because the values of the *length* and *CRC* fields changed between the two read transactions, the entire procedure may be retried by the application.

8.4.6 Topology management (cable environment)

The bus manager shall use topology information collected from self-ID packets received after a bus reset to construct the TOPOLOGY_MAP registers visible to other Serial Bus nodes. As soon as a bus manager is selected from the candidates for this role, the bus manager shall compute the TOPOLOGY_MAP registers from the self-ID packets.

The bus manager shall insure that the observed self-ID packets, including its own transmitted self-ID packet, meet the following requirement:

— The total of all connected ports that are connected to parents equals the total of all connected child ports.

If the observed self-ID packets are not consistent, the bus manager shall not make a topology map available, i.e., the bus manager shall set the *length* field of the TOPOLOGY_MAP registers to zero, but the bus manager shall not initiate a bus reset. The inconsistent condition shall be reported to the application layer by means of an SB_EVENT.indication that indicates Topology Error.

The bus manager may use consistent topology information to optimize performance of the Serial Bus, as described in 8.4.6.2.

Before the bus manager changes the topology map for any reason, it shall first set the *length* field of the TOPOLOGY_MAP registers to zero. After the changes are complete, the *generation_number* shall be incremented, a new CRC calculated, and the *length* field restored.

8.4.6.1 Accessing the topology map

Applications at transaction-capable nodes that need to obtain accurate information from the TOPOLOGY_MAP registers shall use the following procedure:

- a) Read the first quadlet of the TOPOLOGY_MAP registers, which contains a *length* field that describes the data that follows and a *CRC* field whose value has been calculated over that data length
 - 1) If the *length* field is zero, the TOPOLOGY_MAP is not valid, and the application shall not utilize any data contained within the TOPOLOGY_MAP
 - 2) If the *length* field is nonzero, the application shall read the TOPOLOGY_MAP quadlets
- b) Read the first quadlet of the TOPOLOGY_MAP registers once again and compare its value to the value obtained the first time that the *length* and *CRC* fields were read
 - 1) If they are identical, the values of the TOPOLOGY_MAP quadlets are accurate
 - 2) If the values are not identical, the values of the TOPOLOGY_MAP quadlets are not guaranteed to be accurate and shall not be used

If this procedure fails to return usable TOPOLOGY_MAP quadlets, either because the *length* field was zero or because the values of the *length* and *CRC* fields changed between the two read transactions, the entire procedure may be retried by the application.

8.4.6.2 Gap count optimization

Serial Bus optimization by the bus manager is optional. Serial Bus performance may be optimized in three principal ways:

- a) Reconfigure the cable topology in order to reduce the number of hops
- b) Reconfigure the cable topology in order to arrange nodes of the same speed capability adjacent to one another
- c) Optimize the gap count for the current cable topology

The first two methods, while desirable, are beyond the scope of this standard. The third method is the only one available to the bus manager to optimize Serial Bus performance for a given cable topology. Although this optimization is optional, it is strongly recommended because Serial Bus performance is seriously degraded when the gap count is left at its default value.

Table 8.7 gives the values for *gap_count* for each maximum number of cable hops for a particular topology. The formulas used to calculate this table, and their derivations are given in E.1.

Table 8.7—Calculated gap counts

Max_hops	Total delay	Gap_count	Subaction_gap (μ s)	Arb_delay (μ s)	Total (μ s)
1	0.3295	1	0.6002	0.0814	0.6816
2	0.6589	4	0.9257	0.1628	1.0885
3	0.9884	6	1.2512	0.2441	1.4954
4	1.3178	9	1.5767	0.3255	1.9023
5	1.6473	12	1.9023	0.4069	2.3092
6	1.9767	14	2.2278	0.4883	2.7161
7	2.3062	17	2.5533	0.5697	3.1230
8	2.6356	20	2.8788	0.6510	3.5299
9	2.9651	23	3.2043	0.7324	3.9368
10	3.2945	25	3.5299	0.8138	4.3437
11	3.6240	28	3.8554	0.8952	4.7506
12	3.9534	31	4.1809	0.9766	5.1575
13	4.2829	33	4.5064	1.0579	5.5644
14	4.6123	36	4.8319	1.1393	5.9713
15	4.9418	39	5.1574	1.2207	6.3782
16	5.2712	42	5.4829	1.3021	6.7851

If the bus manager performs gap count optimization, it shall first calculate an upper bound for the maximum number of hops of the current cable topology. This is done by examining the self-ID packets stored in the topology map in order to determine the total number of nodes and the number of leaf nodes. Once the value of *Max_hops* is calculated, the bus manager shall broadcast a PHY configuration packet with the *t* bit set to one and the *gap_cnt* field set to no less

than the value obtained from table 8.7. After the bus manager has broadcast the PHY configuration packet, it may initiate a bus reset in order to confirm that all Serial Bus nodes are configured to operate with the new gap count.

NOTE — A bus manager or, in the absence of a bus manager, an isochronous resource manager, may effect significant Serial Bus performance improvements without the need to analyze bus topology and calculate the maximum number of hops. A gap count of 33 is sufficient for the worst-case 16 hops permitted by Serial Bus. This value is a substantial optimization of the default gap count of 63 in effect after a power reset.

8.5 Bus configuration state machines (cable environment)

The text provided in 8.4.2, which describes the bus configuration procedures that shall occur after a bus reset or power reset, can also be specified more precisely by state machines. The state machines described in the remainder of this clause are assumed to be executing at all of the Serial Bus nodes that have the relevant capabilities: cycle master, isochronous resource manager, or bus manager. Note that all of the state machines arrive at states such that there is one and only one node on the Serial Bus that has assumed each of these roles. A single node of course may assume more than one role, as permitted by the rules of the state machines. For example, the root node could become the cycle master, the isochronous resource manager, and the bus manager.

8.5.1 Candidate cycle master states

Each Serial Bus node that is cycle master capable shall execute the state machine described in figure 8.26. If the node has other capabilities, other state machines may be executing concurrently. In this case, the Serial Bus management variables, described in 8.3.2.5.6, are the only means by which the state machines explicitly share information.

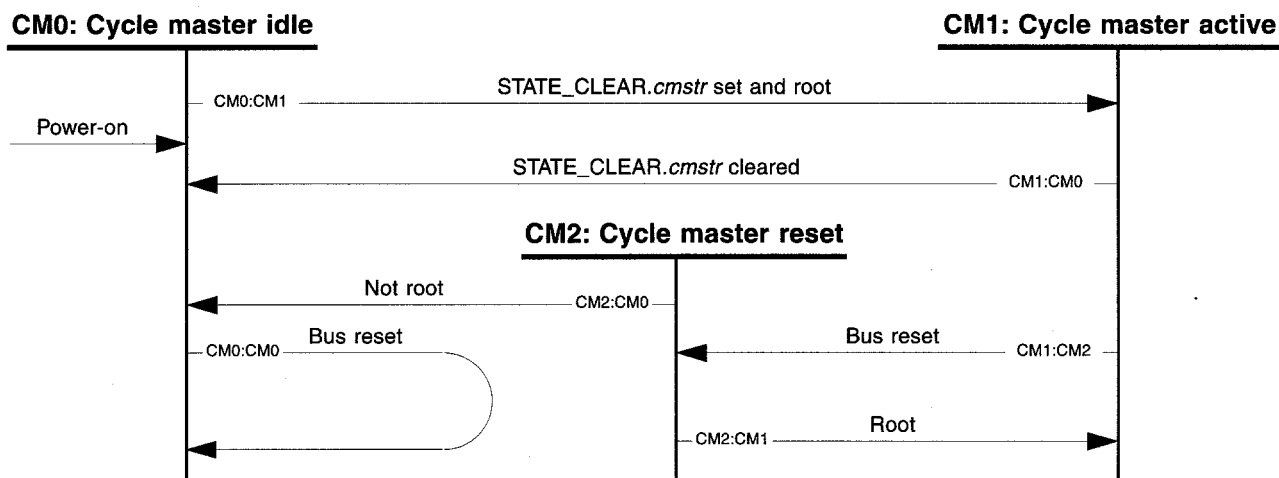


Figure 8.26—Candidate cycle master state machine

State CM0: Cycle master idle. The STATE_CLEAR.cmstr bit is clear and the node shall not broadcast any cycle start packets. Automatic updates of the CYCLE_TIME register by the hardware of the node shall be enabled, but the CYCLE_TIME register shall be synchronized to the time value contained in any cycle start packet received.

Transition CM0:CM1. Another node, either the bus manager or the isochronous resource manager, has enabled cycle master activity at this node by a write to STATE_SET.cmstr with a value of one. If this node is not the root, it shall ignore the write and STATE_CLEAR.cmstr shall retain a zero value. Otherwise, this node shall become an active cycle master.

Transition CM0:CM0. The cycle master shall remain in the idle state if a bus reset is detected.

State CM1: Cycle master active. The `STATE_CLEAR.cmstr` bit is set and automatic hardware update of the `CYCLE_TIME` register is enabled, as described in 8.3.2.2.8. A cycle start packet shall be broadcast each time the `cycle_count` field in the `CYCLE_TIME` register increments. The cycle master shall remain active until either a bus reset occurs or the `STATE_CLEAR.cmstr` bit is cleared.

Transition CM1:CM0. Either the bus manager or the isochronous resource manager has written a zero to the `STATE_CLEAR.cmstr` bit. The automatic hardware update of `CYCLE_TIME` shall be disabled. The cycle master shall not broadcast cycle start packets and shall transition to the idle state.

Transition CM1:CM2. A bus reset has been detected, as notified by a `PH_EVENT.indication`. The automatic hardware update of `CYCLE_TIME` shall be disabled. The cycle master shall not broadcast cycle start packets and shall transition to the reset state.

State CM2: Cycle master reset. The cycle master shall wait for the completion of the self-identify process that follows a bus reset. If the cycle master is still the root node, it shall revert to the active state. If the cycle master is no longer the root, it shall clear the `STATE_CLEAR.cmstr` bit and proceed to the idle state.

Transition CM2:CM0. The cycle master is no longer the root and shall transition to the idle state after clearing the `STATE_CLEAR.cmstr` bit.

Transition CM2:CM1. The cycle master is still the root and shall transition to the active state in order to resume the broadcast of cycle start packets.

8.5.2 Candidate isochronous resource manager states

Each Serial Bus node that is isochronous resource manager capable shall execute the state machine described in figure 8.27. If the node has other capabilities, other state machines may be executing concurrently. In this case, the Serial Bus management variables, described in 8.3.2.5.6, are the only means by which the state machines explicitly share information.

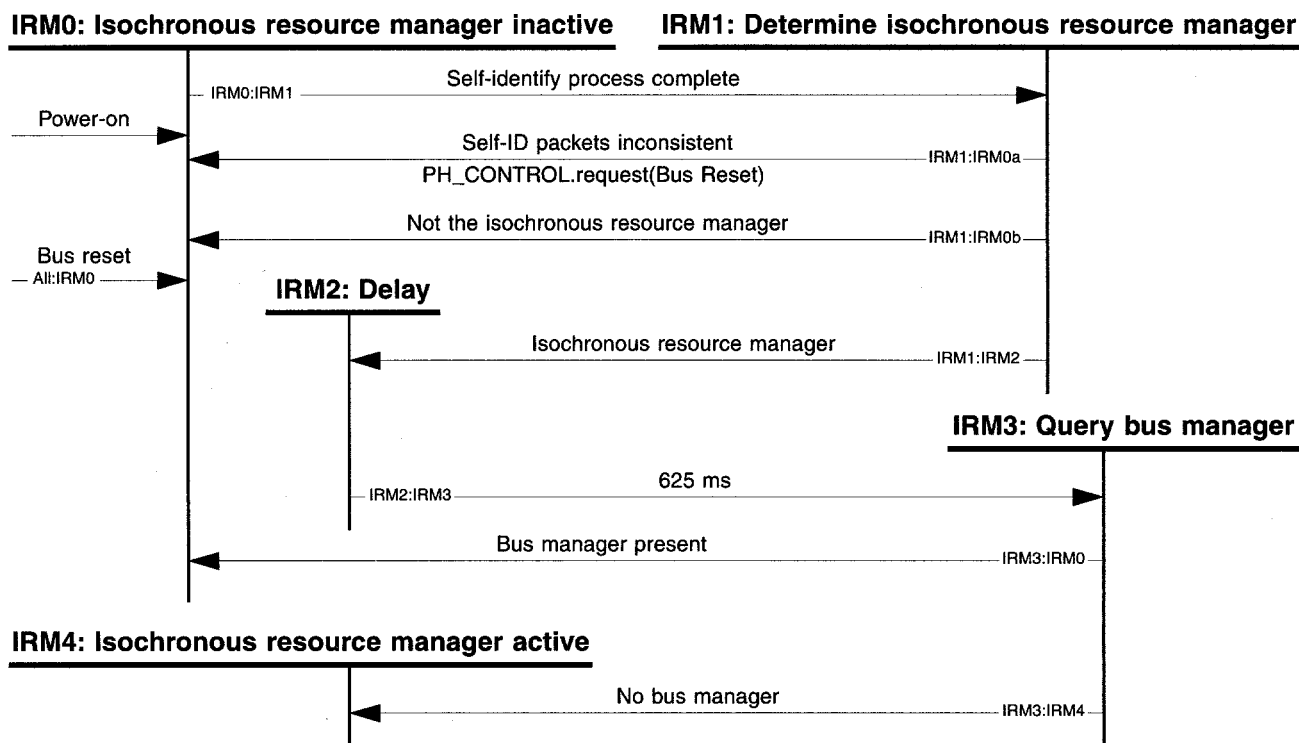


Figure 8.27—Candidate isochronous resource manager state machine

Transition All:IRM0. A bus reset detected in any state shall cause the candidate isochronous resource manager to transition to the inactive state. All isochronous resources, the `BANDWIDTH_AVAILABLE` register, and the `CHANNELS_AVAILABLE` registers are reset to their initial values upon entry to this state. The `BUS_MANAGER_ID` register is also reset to its initial value, $3F_{16}$.

State IRM0: Isochronous resource manager inactive. The isochronous resource manager is inactive, either as a result of a power on reset, a bus reset, or a determination that another candidate has become the isochronous resource manager. The candidate isochronous resource manager awaits the completion of the self-identify process that follows a bus reset. Read and lock (compare and swap) transactions to the `BANDWIDTH_AVAILABLE`, `CHANNELS_AVAILABLE`, and `BUS_MANAGER_ID` registers are accepted by isochronous-resource-manager-capable nodes in this and all other states.

Transition IRM0:IRM1. The candidate isochronous resource manager has received a `PH_CONTROL.indication` with an event of `SELF ID COMPLETE`. The candidate isochronous resource manager shall have stored all received self-ID packets, which at this time are available for inspection.

State IRM1: Determine isochronous resource manager. The candidate isochronous resource manager shall examine the self-ID packets in order to determine the physical ID of the isochronous resource manager, as described in 8.4.1.3. The Serial Bus management variable Isochronous Resource Manager ID is set to the value determined by this process.

Transition IRM1:IRM0a. The self-ID packets observed by this node are inconsistent, as described in 8.3.3. The candidate isochronous resource manager shall issue a `PH_CONTROL.request` with an action of Bus Reset and transition to the inactive state.

Transition IRM1:IRM0b. The candidate isochronous resource manager has determined that another candidate has assumed the role of isochronous resource manager, as described in 8.4.1.3. The candidate isochronous resource manager shall transition to the inactive state.

Transition IRM1:IRM2. The candidate isochronous resource manager has determined that it has assumed the role of isochronous resource manager, as described in 8.4.1.3. The candidate isochronous resource manager shall transition to the delay state.

State IRM2: Delay. The isochronous resource manager shall wait 625 ms before proceeding to the query bus manager state.

Transition IRM2:IRM3. After 625 ms have elapsed, the isochronous resource manager shall transition to the query bus manager state.

State IRM3: Query bus manager. The isochronous resource manager shall determine whether or not a bus manager is active on the Serial Bus, as described in 8.4.2.3. If the value of the BUS_MANAGER_ID register is $3F_{16}$, there is no bus manager, and the isochronous resource manager shall continue to the active state. If the value of the BUS_MANAGER_ID register is other than $3F_{16}$, a bus manager is active, and the isochronous resource manager shall not perform any bus management but shall proceed to the inactive state.

Transition IRM3:IRM0. The BUS_MANAGER_ID register contains a value other than $3F_{16}$. A bus manager is active, and the isochronous resource manager shall transition to the inactive state.

Transition IRM3:IRM4. The BUS_MANAGER_ID register contains the value $3F_{16}$; the isochronous resource manager shall transition to the active state.

State IRM4: Isochronous resource manager active. The isochronous resource manager shall determine whether or not a cycle master is active. If a cycle master is not active, the isochronous resource manager shall select a cycle-master-capable node and make it active as the cycle master, as described in 8.4.2.6. The isochronous resource manager may initialize the BUS_TIME and CYCLE_TIME registers by means of a broadcast write. The isochronous resource manager may issue link-on packets to any nodes whose self-ID packets indicated an inactive link layer, as described in 8.4.4.5. The isochronous resource manager may also optimize the gap_count setting.

8.5.3 Candidate bus manager states

Each Serial Bus node that is bus manager capable shall execute the state machine described in figure 8.28. If the node has other capabilities, other state machines may be executing concurrently. In this case, the Serial Bus management variables, described in 8.3.3, are the only means by which the state machines explicitly share information.

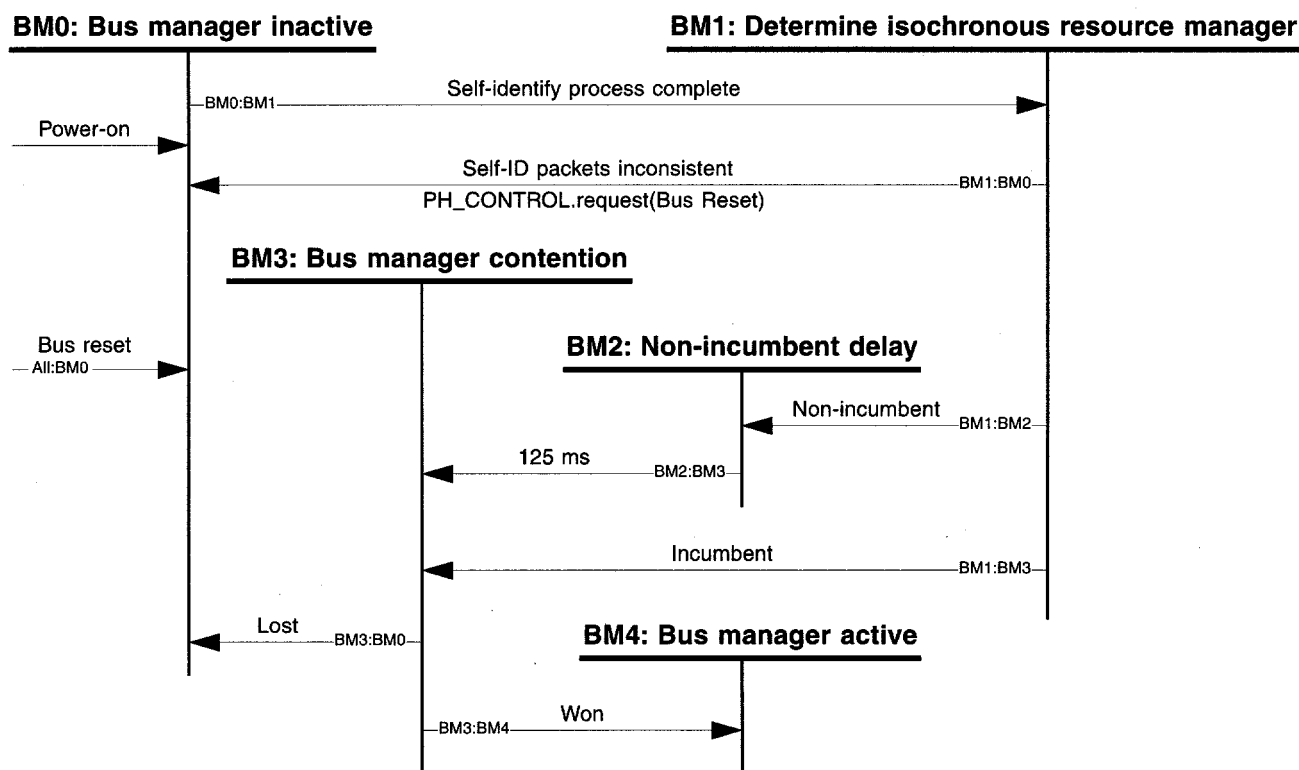


Figure 8.28—Candidate bus manager state machine

Transition All:BM0. A bus reset detected in any state shall cause the bus manager or candidate bus manager to transition to the inactive state.

State BM0: Bus manager inactive. The bus manager is inactive, either as a result of a power on reset or a bus reset. The candidate bus manager awaits the completion of the self-identify process that follows a bus reset.

Transition BM0:BM1. The candidate bus manager has received a PH_CONTROL.indication with an event of SELF ID COMPLETE. The candidate bus manager shall have stored all received self-ID packets, which at this time are available for inspection.

State BM1: Determine isochronous resource manager. The candidate bus manager examines the self-ID packets in order to determine the physical ID of the isochronous resource manager, as described in 8.4.2.3. The Serial Bus management variable Isochronous Resource Manager ID is set to the value determined by this process.

Transition BM1:BM0. The self-ID packets observed by this node are inconsistent, as described in 8.4.2.3. The candidate bus manager shall issue a PH_CONTROL.request with an action of Bus Reset and transition to the inactive state.

Transition BM1:BM2. The Serial Bus management variable Incumbent is FALSE.

State BM2: Non-incumbent delay. This node was not the bus manager prior to the bus reset or power reset. The candidate bus manager shall wait 125 ms and then transition to the bus manager contention state.

Transition BM1:BM3. The Serial Bus management variable Incumbent is TRUE.

Transition BM2:BM3. After 125 ms have elapsed, the candidate bus manager shall transition to the bus manager contention state.

State BM3: Bus manager contention. The candidate bus manager shall contend for the role of bus manager by means of the procedure described in 8.4.2.1. If the value of the BUS_MANAGER_ID register obtained from a lock (compare and swap) transaction to the isochronous resource manager is $3F_{16}$, the candidate bus manager has won and is confirmed as the bus manager. Otherwise, another candidate for the role of bus manager has won, and the bus manager at this node shall return to the inactive state.

Transition BM3:BM0. A value other than $3F_{16}$ has been returned by the lock (compare and swap) transaction to the BUS_MANAGER_ID register at the isochronous resource manager. The Serial Bus management variable Incumbent shall be set to FALSE, and the candidate bus manager shall transition to the inactive state.

Transition BM3:BM4. The lock (compare and swap) transaction to the BUS_MANAGER_ID register at the isochronous resource manager has returned a value of $3F_{16}$. The node variable Incumbent shall be set to TRUE, and the bus manager shall transition to the active state.

State BM4: Bus manager active. The candidate bus manager is now active and shall directly commence operations to ensure that, if any isochronous capable nodes are present, a cycle master is present and active; make the TOPOLOGY_MAP registers available; and make the SPEED_MAP registers available. In addition, the bus manager may optimize the arbitration gap characteristics of the Serial Bus. These procedures are described in 8.4.6.

Annex A Cable environment system properties

(Normative)

The cable environment has properties that are very important when creating real systems from multiple enclosures and when designing enclosures with external Serial Bus connectors. Attention to these properties is essential for user-friendly, safe, and high-quality operation of the Serial Bus.

This annex defines specifications in the following areas that together describe these properties and form the necessary rules:

- a) External shielded cable interconnections
- b) Internal unshielded interconnection
- c) Cable power sourcing and connections
- d) Powering PHY integrated circuits (ICs) and devices
- e) Electrical isolation requirements

A.1 External shielded cable interconnects

A.1.1 Definitions

A.1.1.1 backshell: That part of the cable assembly that goes between the connector and the cable wire. This includes the cable wire strain relief, the external surface, part of the shield circuit, and usually part of the positive retention scheme.

A.1.1.2 external: With regard to an enclosure box, outside the environment provided by the enclosure and therefore not protected from ambient radiation, mechanical stresses, and thermal stresses.

A.1.1.3 external connection: The entire system required to execute the attachment and de-attachment of an external cable to an enclosure. This includes the connectors, the backshells, the retention schemes, the cable wire near the backshell, the shield circuit and all its components, the electrical isolation schemes for the enclosure wall, the mounting of the enclosure side-mating connectors (especially the mechanical pieces required for strength), and any features involving interference between neighboring connections.

A.1.1.4 external connectors: That part of the cable assembly that directly supports the separable contacts, the bus power and ground contacts themselves, and the separable part of the shield.

A.1.1.5 passive retention: A retention means that allows separation of a fully mated connector by applying only axial force to the cable wire or the stationary parts of the connector body—no damage results from the separation.

A.1.1.6 positive retention: A retention means that requires a release act other than a simple axial force on the cable wire or stationary part of the connector body to produce damage-free separation of fully mated connector systems. Damage to cable assemblies may be expected if the separation is forced without performing the release act prior to executing the demating.

A.1.1.7 retention: Axial force required to produce separation of a fully mated connector.

A.1.1.8 shuttle latch: A positive retention scheme whose release act uses only axial force on a movable part of the external connector body. Continued application of axial force on the movable part after release will execute the demating.

A.1.2 Specifications

Clause 4.2.1.1 specifies the only external shielded connector permitted in Serial Bus implementations.

There are only two external cable retention schemes allowed: detent for passive retention as specified in 4.2.1.1 and shuttle latches for positive retention as specified in annex G.

Use of the shuttle latches on the external cables requires compatible receivers on the bulkheads. The shuttle latch compatible receivers do not interfere with the use of passive retention detent external cable connectors as specified in annex G.

External bulkhead connections that employ only the passive retention (detent) are not required to provide panel cutout space for positive retention latches on external cables.

Multiple external connections to the same enclosure are allowed (no limit is given in this standard). Minimum clearances should be observed.

A.2 Internal unshielded interconnects

A.2.1 Definitions

A.2.1.1 internal: Hardware located entirely within shielded enclosures.

A.2.1.2 unshielded: The lack of any shielding on the cable wires or connectors.

A.2.2 Specifications

Internal unshielded interconnections using either cables or printed wiring boards (backplanes) for the bus signals shall have the same electrical properties as the external IC system (except shielding requirements do not apply).

Power transmitted through internal ICs shall be distributed according to the power isolation and distribution architecture specified in this annex.

Connections to internal unshielded storage devices shall be made only with the standard serial unitized device connector as specified in annex C. For cable implementations, it is permissible to use additional bus ports providing the additional connectors are compatible subparts of the unitized connector.

Connections to internal controllers may use the internal unitized device connector, connectors derived from the individual bay subparts, or other connectors as required to meet the design goals of the enclosure. It is expected that the unitized connector will not be suitable for backplane pluggable controllers due to several individual connectors being required for multiple drives.

Although there are impedance control requirements on the cable wire and the cable assemblies, there are no specific impedance control requirements on the internal connectors because of the small total length occupied by the connector.

The length of the internal ICs has to be added when calculating the hop length if the internal interconnect directly connects to an external cable through the enclosure wall.

A.3 Cable power sourcing and connection

A.3.1 Definitions

A.3.1.1 cable power: Power transmitted through the cable power and ground conductors in external Serial Bus cables.

A.3.1.2 cable power connection: All of the hardware required to realize the sourcing or delivery of cable power to the external cable or to PHY ICs.

A.3.1.3 cable power source: Voltage supplies connected to the cable power lines that provide unregulated, unidirectional power.

A.3.1.4 system power distribution ground: The ground (not neutral) connection in US standard 120 V outlets and connected to earth ground at the local distribution panel. This is commonly called the “green wire” or “safety” ground within a utility power distribution system.

A.3.2 Specifications

Cable power shall be sourced from or delivered to the external cable only through the external cable connector defined in 4.2.1.1.

There shall be at least one source of cable power for every Serial Bus.

Any Serial Bus requires cable power sufficient to power the total number of PHYs not locally powered. While *not a requirement*, maximum availability will be achieved if every cable power source is capable of powering all the PHYs on the bus. Any loads on cable power beyond the minimum number of PHYs shall not reduce the power available to any PHY in the system below 8 V at 125 mA (1 W).

All sources of cable power shall not sink current.

Cable power sources are not required to be regulated and therefore shall not be directly connected to any device that requires regulated power. Cable power shall be delivered to end users (such as PHY ICs) only through local regulators.

The cable power ground shall not be connected to the system power distribution ground at any point in the system.

The cable power ground shall be continuous throughout the entire Serial Bus system. The cable power ground for multiple Serial Buses in the same system may be connected together within enclosures.

The cable power ground may be connected to local enclosure power domain grounds *if* the local power domain is isolated from the overall system power distribution ground.

The cable power ground shall be connected to the PHY ground.

A.4 Summary of electrical isolation requirements

The Serial Bus has three electrical isolation requirements for proper operation and safety:

- a) The cable shield shall be connected to external enclosure shielding surfaces and the system power distribution ground used to power the enclosure *only* through a special isolation circuit that provides high impedance at low frequencies and low impedance at high frequencies (see 4.2.1.4.8).
- b) The signal lines in the external cables shall be dc isolated from the system power distribution ground domain used to power the enclosure.
- c) The external cable power line and its ground shall be isolated from the system power distribution ground used to power the enclosure. In all cases, the isolation prevents any part of the external cable from becoming a low impedance electrical connection between different power domains.

The external cable is not designed to handle the power levels or noise of the type that can exist between different power domains. The problems that can occur range from melting and possible fire hazard to loss of integrity in the data transmissions to excessive radiation from the cable. The isolation features of the Serial Bus are one of its most important (yet largely invisible) user-friendly attributes.

A formal statement of the isolation requirement is given in 4.2.2.7. It states that the algebraic sum of all dc currents in the six external cable wires at every external connector shall be less than 50 μ A at all times.

The performance characteristics of the shield isolation circuit are specified in 4.2.1.4.8. The isolation of the signal lines may be accomplished in any manner that avoids a dc connection while allowing the ac coupling necessary for communication. Typical methods used for the signal isolation include transformer coupling and capacitive coupling.

It is not specified where the signal isolation must occur. The places that are usually the most convenient are

- Between PHY and link ICs or
- Between the external ac power source and the power supply

External cable power isolation is accomplished by isolating the external cable power sources from the system power distribution “green wire” ground.

Notice that for a given enclosure one can pick at most two of the following at the same time:

- a) Integrated PHY/link IC(s)
- b) Sourcing of cable power
- c) Connecting other directly coupled buses (e.g., SCSI) to the enclosure ground domain

This does not apply if the PHY/PHY* isolation strategy is used (see A.6).

The following clauses describe typical configurations that meet the architectural requirements. The signal isolation requirement shall be met if integrated PHY and link functions are provided on the same IC. This will commonly be a desirable integration for drive implementations and other severely space-constrained applications.

Table A.1—Restrictions on enclosures in systems

Enclosure function	Restrictions
Cable power source	Cable power supply shall be isolated from system power distribution ground. No other non-isolated bus (for example SCSI) may be attached to the cable power supply domain. If there is a single cable power source for the entire system <i>and</i> the external cable power ground is isolated in all enclosures in the system, these restrictions do not apply.
Not a cable power source	Cable power ground shall be isolated from system power distribution ground. This does not apply if <i>all</i> other enclosures in the system have isolated cable power grounds.
External repeater	The power for the PHY acting as the external repeater has to be derived from the external cable power. There shall be at least one source of external cable power in the system other than that supplied from the external repeater enclosure.
Not an external repeater	The power for the PHY (a “leaf” enclosure) connecting to the external cable may be powered from either an internal source or be derived from the external cable power. This is because only the “leaf” itself will be affected by the internal power level.

A.5 Examples

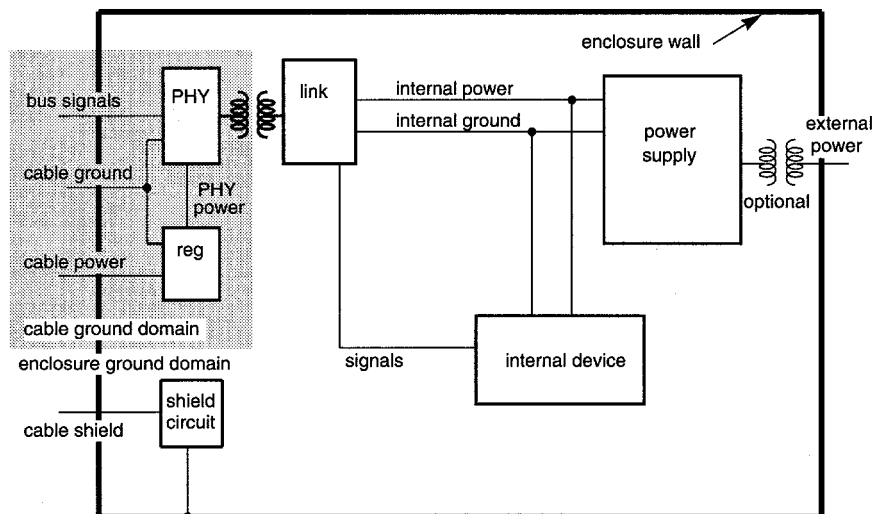


Figure A.1—Simple signal isolation with no cable power sourcing

Figure A.1 depicts the “classical” scheme where a separate PHY, cable power regulator, and link ICs are used. The signal isolation will occur between the PHY and link ICs. Note that the link function will likely be integrated into an IC with other non-PHY functions. It does not matter how the link function is physically implemented as long as it is electrically isolated from the PHY.

Figure A.1 also illustrates another important feature of the Serial Bus physical architecture: the use of external cable power for powering PHYs. The power available from the cable can be very highly unregulated. It must be between 8 V and 40 V and is therefore is virtually always unsuitable for direct use as PHY power. A regulator will typically be used between the external cable power and the PHYs.

The external cable ground in this case is connected *only* to the regulator and the PHY IC. It may not be connected anywhere else inside the enclosure.

Transformer isolation is shown for reference between the PHY and link ICs.

In this case, the power supply for the internal device(s) does not need to be isolated from the system power distribution ground, and other external (non-Serial Bus) connections may be made to the enclosure power domain.

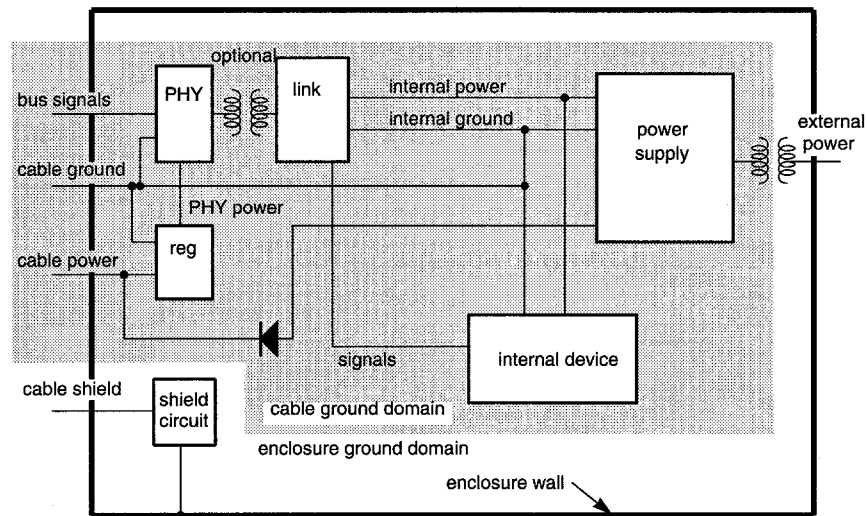


Figure A.2—Cable power sourcing from enclosure power domain

Figure A.2 depicts the case where the internal enclosure power supply is used to supply external cable power, internal device power, and link power. The internal power supply shall be isolated from the system power distribution ground. It shall also be “diode” isolated from the cable power connection to avoid sinking power if the external cable power voltage is higher than the cable power supply voltage in this enclosure (due to sources of external cable power elsewhere in the system) or if the internal power supply itself is turned off.

Notice that the cable ground shall be connected to the internal ground.

An integrated PHY/link IC could be used in this case, and no PHY/link isolation is required.

It is permissible to provide a separate cable power supply that is isolated from both the internal enclosure ground and the external power domain ground. In this case (not shown), the cable ground (also the PHY ground) shall be connected to the ground for the cable power source, and the PHY/link isolation is required.

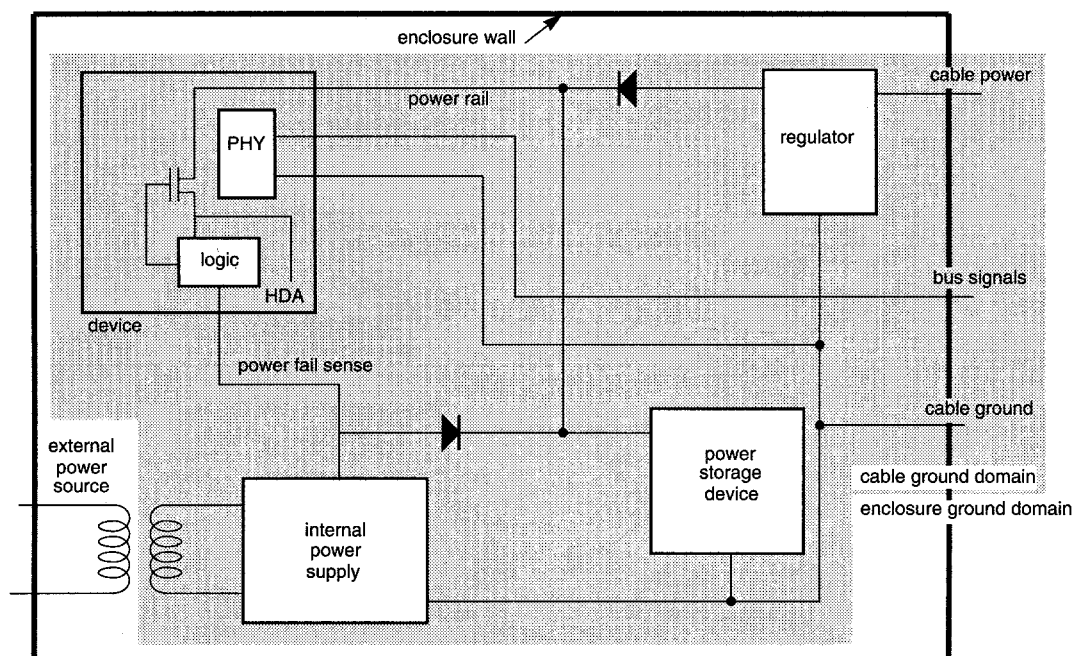


Figure A.3—Enclosure that uses cable power to trickle-charge a backup power storage device

The case depicted in figure A.3 uses a power storage device to provide power to the devices in the enclosure (labeled HDA) while these devices are gracefully shutting down after the internal power supply fails.

Normally the internal power supply provides power for the HDA, the PHY, and the power storage device. If the internal power supply fails, the power fail-sense line signals the logic that a failure has occurred and the load switches over to the power storage device.

After a period of time determined by the device load and the capacity of the power storage device, the logic disconnects the devices from the power (shown as a transistor). In this state, the cable power is supplying power to the PHY and is trickle-charging the power storage device so it will be ready immediately when the devices are powered back on.

In no case shall the external cable power voltage level be forced to less than 8 V by transient loads presented by the power storage device or the HDA.

This scheme requires that the internal power supply be isolated from the system power distribution ground and that the PHY ground be connected to the cable ground. This produces the same condition presented in figure A.2, where the PHY may or may not be isolated from the link.

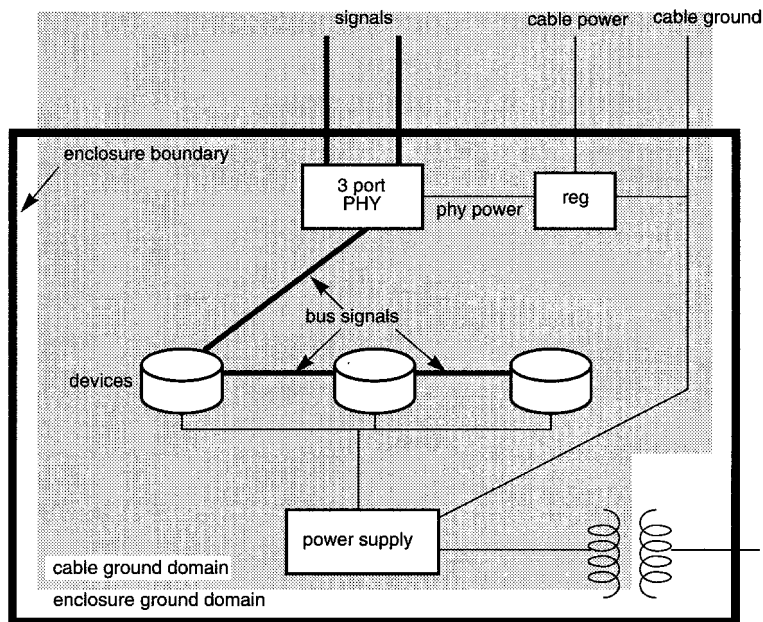


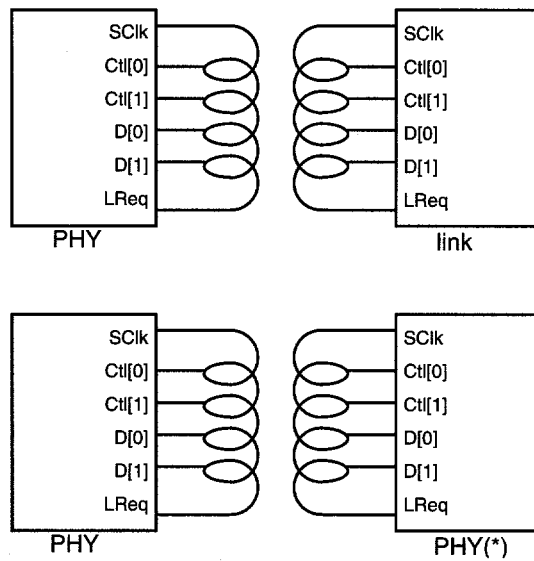
Figure A.4—Bus repeater enclosure with multiple internal devices

The example in A-4 shows a simple bus repeater PHY with an internal port to connect with PHYs on the devices. The only isolation required in this case is at the internal power supply.

A.6 Special isolation schemes using PHY/PHY* interface

In the case where it is desirable to keep the cable ground domain isolated from the enclosure ground domain and one wishes to use integrated PHY and link ICs on devices, it is necessary to introduce isolation between PHYs. This is similar to the isolation introduced between the PHY and link ICs but does not require a separate link IC or a full link function.

Figure A.5 shows the basic structure of the PHY* interface and its similarity to the link interface. The protocol used to transfer information across the PHY/PHY* interface is not defined in this standard. This concept is shown for reference only.



NOTES:

- 1—The number of data lines (D[n]) scales with speed (2 lines at S100, 4 at S200, and 8 at S400).
- 2—These figures are symbolic of an independent isolation circuit per signal. This is not a multitapped transformer.

Figure A.5—PHY* interfaces

Figure A.6 shows an example where the PHY/PHY* interface could be used in an array application with a wide PHY* IC.

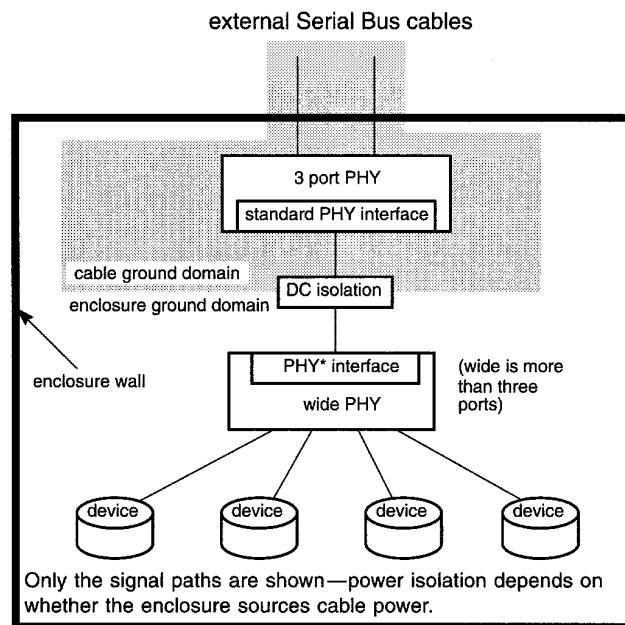


Figure A.6—Use of the PHY/PHY* interface in an array application

Annex B External connector positive retention

(Normative)

The external shielded Serial Bus connection may be made using a positive retention scheme in addition to the detent passive retention that is built into the external connector itself. Positive retention is used where the external cable has to be held securely in place unless released by an intentional unlocking action.

The standard external connector socket has features already built in that are intended as anchor points for a hook type latch on the external plug cable assembly. The use of positive retention does not require different connectors. The flanges on the narrow sides of the socket as detailed in figure 4.4 are used as the anchor points. Note that one of the flanges is considerably more narrow than the other.

In order to retain maximum flexibility in implementation, the detailed dimensioning of the actual latching mechanism on the plug side is not specified in this standard.

For implementations using positive retention, an exclusion zone on the socket side is defined that is reserved for positive retention mechanisms on the plug side (see figure B.1).

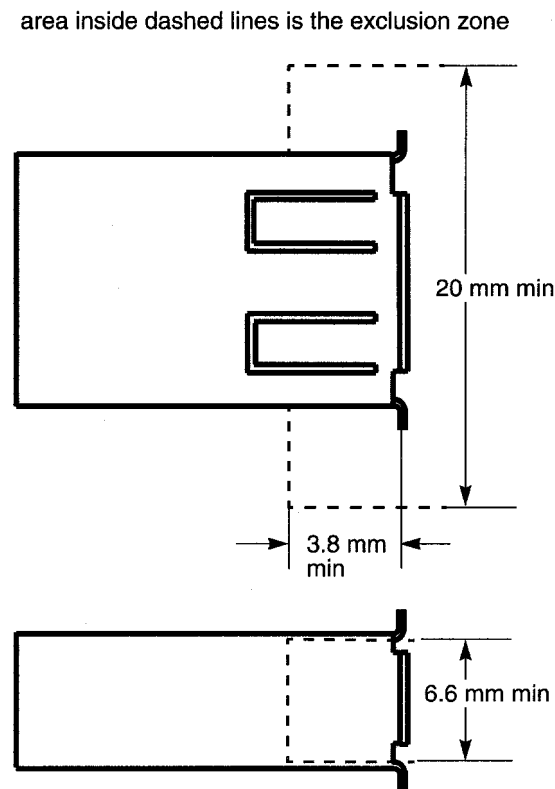


Figure B.1—Exclusion zone for positive retention mechanism

Figure B.2 shows a suggested shuttle latch mechanism that releases the hooks on the latch arms from the flanges on the socket connector by simply pulling on the outer housing of the external plug cable assembly. In this design, the arms of the shuttle latch will occupy part of the exclusion zone around the socket when the system is mated.

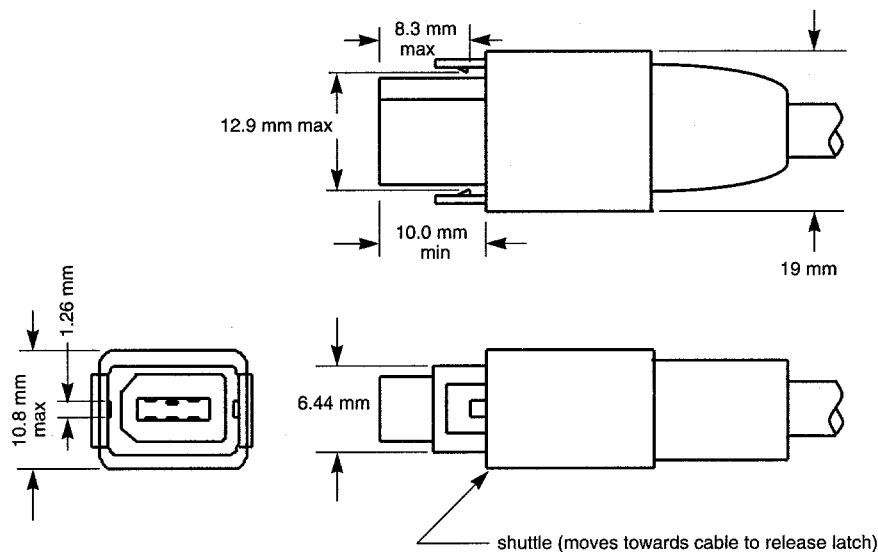


Figure B.2—Three views of plug with shuttle type latch

The retention forces on the socket side may be considerably higher when using positive retention than when using passive retention. It is recommended that a suitable mounting means be used to distribute the force onto the enclosure wall when using positive retention.

Any latching mechanism should withstand at least 44.5 N of axial force without demating and should nondestructively release at forces above 89 N.

The exclusion zones increase the bulkhead space required for an external connection. See figures I.2 through I.5 for minimum mounting intervals.

For example, if one wants to use positive retention on a common EISA/ISA/PCI external panel and the exclusion zone penetrates the panel, it requires horizontal cutouts to accommodate the positive retention. This is shown in figure B.3.

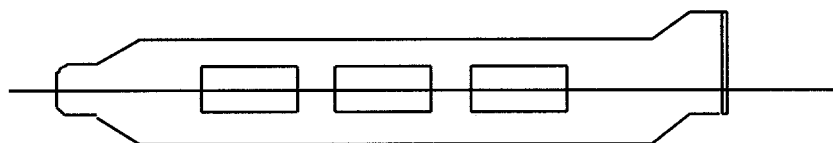


Figure B.3—Cutouts in a typical personal computer option panel

Annex C Internal device physical interface

(Normative)

C.1 Overview

The cable PHY layer specification described in clause 4. is designed for external box-to-box applications. (An example would be a CPU, printer, and video camera interconnected with a Serial Bus interface in which the CPU and printer are powered from different ac outlets while the camera takes power from the Serial Bus cable.) The external cable provides power to all PHYs on the bus so that they can maintain their bus repeater capability even if the node local power is off. To accommodate different power domains (different ac power sources), each node shall provide isolation between the node local power and the external cable power. The external environment requires mechanically strong shielded cables and connectors.

An internal device does not have the same design criteria as an external box-to-box application. Internal devices (such as DASD hard files) are optimized for low cost, low power, minimum components, and minimum package size. Internal devices share a common power domain with other devices packaged within a box and do not require mechanically strong shielded connectors and cables. Internal devices do require packaging options such as hot-plug, auto-dock, blind-mate, and various connector methodologies such as cable or board attachment with such connector systems as surface mount and card edge.

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

Because the physical interface and electrical requirements for internal devices differ from external box requirements, a different physical interface is required.

C.2 Electrical interface for internal devices

C.2.1 Power requirements

It is assumed that a single external node will support multiple internal devices. The external node will provide the power isolation for the box with respect to other external nodes. The internal devices will share a common power domain within the box, eliminating the need for power isolation for each internal device.

Internal device power is regulated +12 V, +5 V, and +3 V at 1.5 A. Connector pins are assigned for all three voltage levels, but it is a box option to select which voltage levels will be provided. (Note that most devices use only one or two of the specified voltage levels.) Providing regulated voltage to the devices eliminates the need for a voltage regulator as described for the external box interface.

It is assumed that fewer ground pins are needed than voltage pins. This is based upon the assumption that not all voltage pins will be utilized by any single device.

Since devices are not required to implement power isolation, separate power domains within the device are not required. This allows a higher level of device electronic integration.

Note that devices with different power requirements only need to implement a subset of these pins. See table C.2.

Table C.1—Internal device power connector pin allocation

Pin #s	Signal name
1	+12 V CHG (long)
2	+5 V CHG (long)
3, 7, 8, 12, 15, 16	GND (long)
4, 5, 6	+12 V
9, 10	+5 V
11	PWRFAIL*
13, 14	+3.3 V

*See table C.5.

Table C.2—Internal device power connector configurations

Configuration	Signal pins used	Usage
2×3	11 through 16	3.3 V device
2×4	9 through 16	3.3 V and 5 V device
2×5	7 through 16	3.3 V and 5 V device with extra grounding
2×6	5 through 16	3.3 V, 5 V, and medium-power 12 V device
2×7	3 through 16	3.3 V, 5 V, and high-power 12 V device
2×8	1 through 16	3.3 V, 5 V, high-power 12 V, and capacitive charge device

C.2.2 Bus signal requirements

Every Serial Bus device shall implement at least one port, referred to as the primary port. A device may optionally implement a second port, referred to as the secondary port. The secondary port may be configured as a bus repeater (as defined for the cable environment), or as a second independent Serial Bus (used for high-availability dual path support). The DUAL_BUS* signal provides the mode selection capability.

When the device is configured as a repeater, it shall maintain PHY functionality (signal repeat, etc.) while PWRFAIL is active and the appropriate power supply (either +3.3 V or +5 V depending on the implementation) is available.

Each port is made up of six signals, consisting of the pair of differential signals defined to the cable environment and two signal grounds.

To support hot-plugging of devices, the ground pin shall connect before the signal pins.

Table C.3—Internal device primary and secondary port pin allocation

Pin #s	Signal name
1	TPA
2	TPA*
3	VG (long)
4	VG (long)
5	TPB*
6	TPB

C.2.3 Miscellaneous signals

A set of option signals are defined for the option connector.

Table C.4—Internal device option connector pin allocation

Pin #s	Signal name
1	MTM*
2	AUTOSTART*
3	SYNC
4	DUAL_BUS*
5	GND
6	DEV_ACTIVE*
7, 8	RESERVED (OUT)
9, 10	RESERVED (I/O)

C.2.4 Signal descriptions

Table C.5—Internal device signal description

Name	Usage	Function
MTM*	input	Active low input places device in manufacturing test mode. The device will use a pullup resistor. When in this mode, all the options pins may be defined for custom use during manufacturing test.
AUTOSTART*	input	Active low input pin will start the motor when power is applied. This device will provide a pullup resistor.
SYNC	input/output	This I/O pin is used to synchronize spindles of multiple devices. This output should change to a high-impedance state when power is removed.
DUAL_BUS*	input	Active low input forces device to treat secondary port as a separate Serial Bus. If inactive, the secondary port and primary port are treated as the same Serial Bus, and signals on one port are repeated to the other. The device will provide a pullup resistor.
DEV_ACTIVE*	open drain output	This open-drain active low output pin is used to indicate device activity. It may be used to control an LED. This output should be high impedance with no power.
RESERVED		These are reserved for future standardization. Two types are used: Output and Input/Output.
PWRFAIL	input	Connected to a power supply output power fail signal that goes active a minimum of 4 ms prior to the power supply output falling below 2.5% of 3.3 V or 5 V. The signal shall stay active a minimum of 2 ms after reaching the threshold level. The output signal (from the power supply) should have the following properties: VOL = 0.5 V max VOH = 2.4 V min IOL = 48 mA min (open drain)

Table C.6—Electrical interface specifications

Signals	Specification
Input signals: MTM*, SYNC, AUTOSTART*, DUAL_BUS*, PWRFAIL	VIL = 0.8 V max VIH = 2.0 V min IIL = -0.6 mA max (includes 10 k pullup resistor for PWRFAIL only) IIH = 40 μA max
Output signals: SYNC, DEV_ACTIVE*	VOL = 0.6 V max IOL = 24 mA min (open drain)

General requirements:

- All ground pins shall be longer than other pins.
- All option pins not tied to ground shall go to a high-impedance state when device power is removed.
- All active low inputs require a pullup resistor on the device.
- All inputs shall be TTL compatible.

Functions for the options pins for Serial Storage Architecture (SSA) devices and Fibre Channel devices are shown in table C.7. The source standard for these non-Serial Bus devices should be consulted for the latest description of the details.

Table C.7—Options pin comparison

Pin number	Serial Bus	SSA	Fibre Channel
1	MTM*	MTM*	AUTOSTART2*
2	AUTOSTART*	AUTOSTART*	AUTOSTART1*
3	SYNC	SYNC	SYNC
4	DUAL_BUS*	EXT_FAULT*	RESERVED (IN)
5	GND (long)	GND (long)	GND (long)
6	DEV_ACTIVE*	DEV_ACTIVE*	DEV_ACTIVE*
7	RESERVED (OUT)	+5 V (OUT)	RESERVED (OUT)
8	RESERVED (OUT)	DEV_FAULT*	LRC
9	RESERVED (I/O)	RESERVED (I/O)	RESERVED (I/O)
10	RESERVED (I/O)	RESERVED (I/O)	RESERVED (I/O)

C.3 Internal unitized device connectors

The internal connectors include a 4-bay, unitized plug connector with a 16-contact power bay, a 10-contact options bay, and two 6-contact bus bays mating to the unitized 4-bay backpanel receptacle. This makes a total of 38 contacts per each half of the mating PCB connectors. Also, 3 cable connectors are specified for reference: a 16-contact power receptacle, a 10-contact options receptacle, and a 6-contact bus receptacle. These cable receptacles mate with the corresponding bays on the unitized plug connector. All connector contacts shall be rated to carry 1.5 A with a maximum temperature rise of 20 °C.

Within this annex, all dimensions and tolerances, and descriptions of those features that affect the intermateability of the plug and the receptacle, are given and are mandatory. No features are shown, however, for the termination side of any of the connectors, since the termination method and hardware are entirely optional. It is likely that there will be numerous variations of the unitized plug and receptacle to meet the design and production process requirements of different pieces of Serial Bus equipment. These requirements may require wave-soldered, through-hole mounted, or reflow-soldered surface mounted plugs and/or receptacles to a printed wiring board. Different hole and/or pad patterns may also be required on the board. Other variations may need a different mounting orientation of the connector relative to the board. Since these factors do not affect the intermateability of the connectors, they are not specified in this standard. However, a recommendation for the hole and/or pad pattern for the mounting of the unitized plug and receptacle to the PCB is suggested in figures C.1 and C.2.

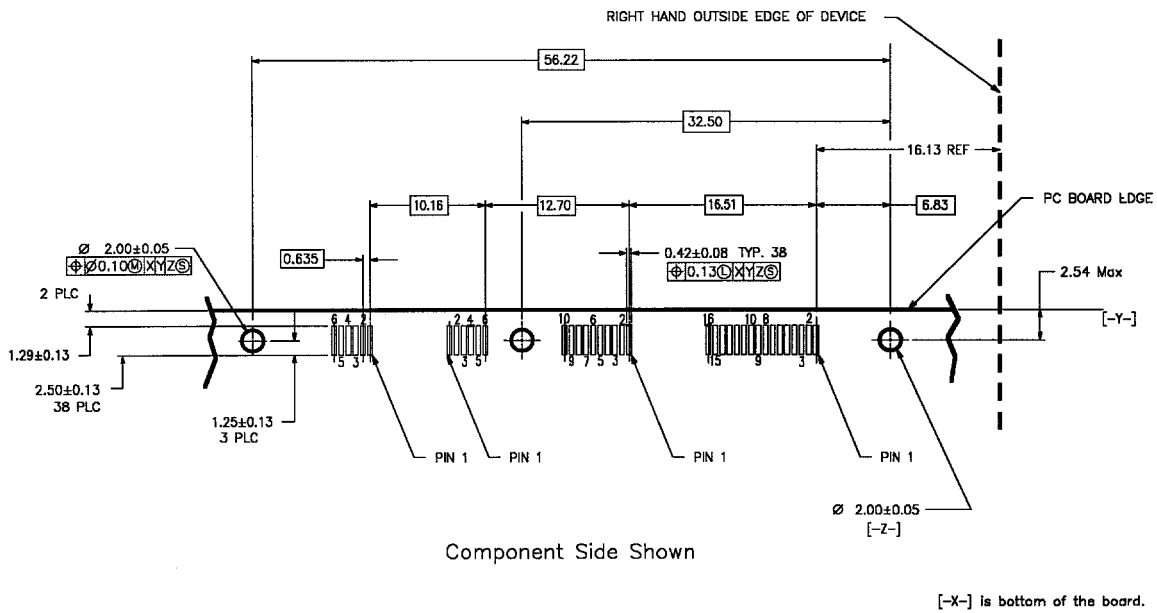


Figure C.1—Recommended SMT PCB layout (connector mounting to the PC board in the device)

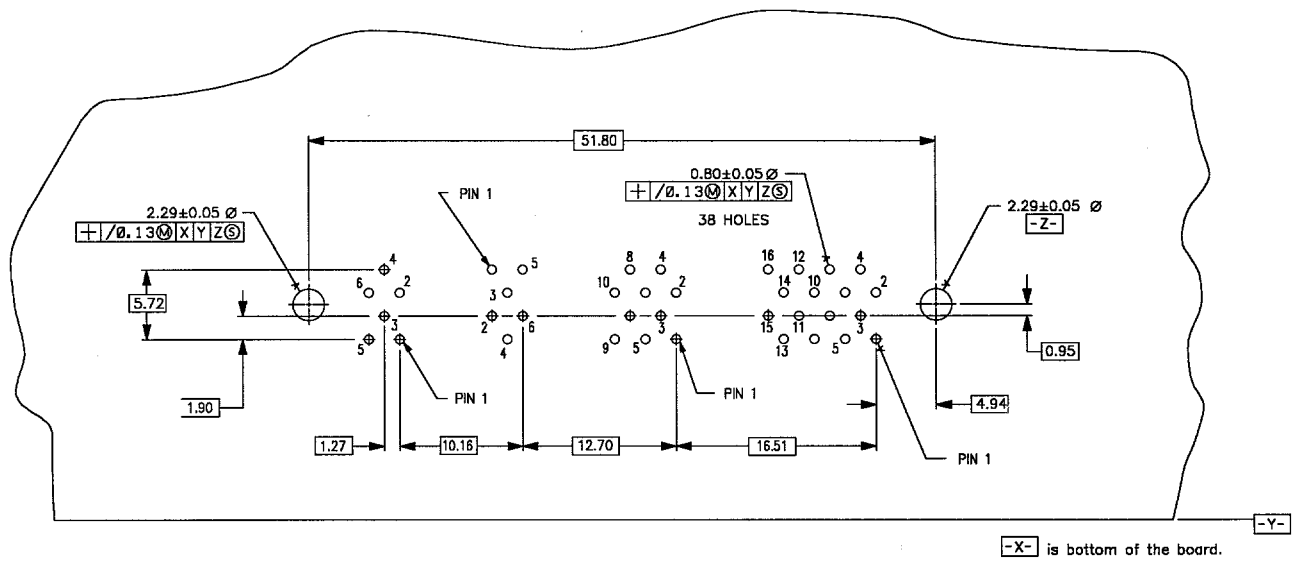


Figure C.2—Recommended PCB layout (connector mounting side)

All other features of the connectors that do not affect the intermateability are not described and may vary. They are controlled only by the performance requirements described in C.3.4.1.

C.3.1 Internal unitized plug

The mating features of the connector plug are illustrated in figure C.3. They will assure the intermateability of the plug with standardized receptacles.

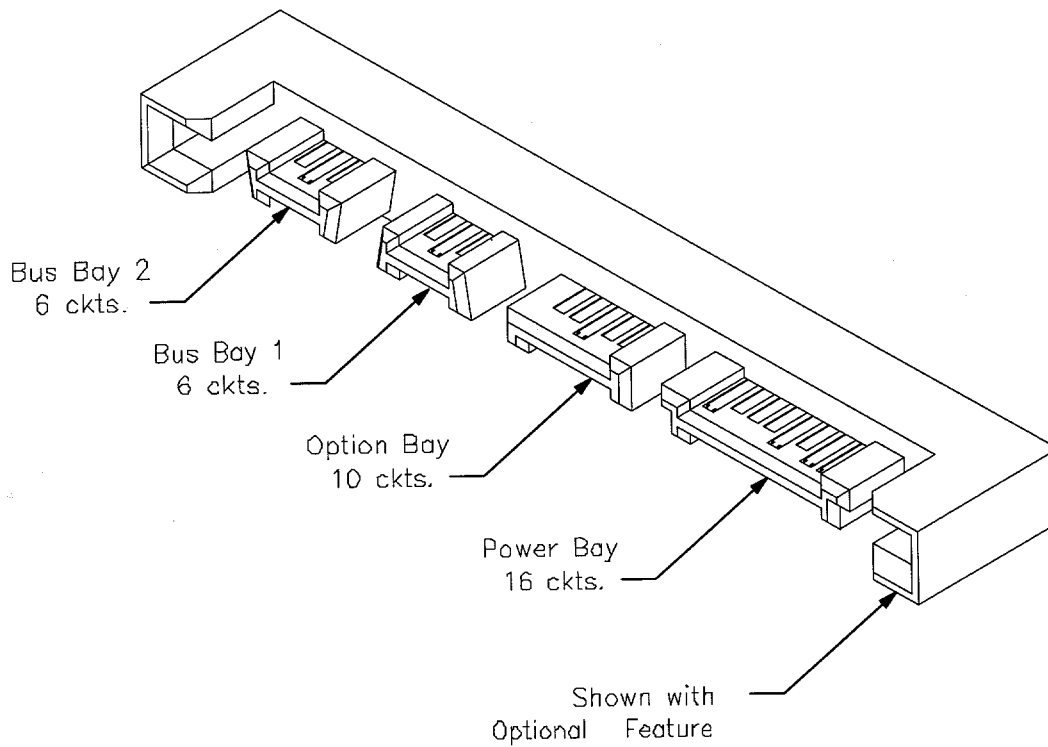
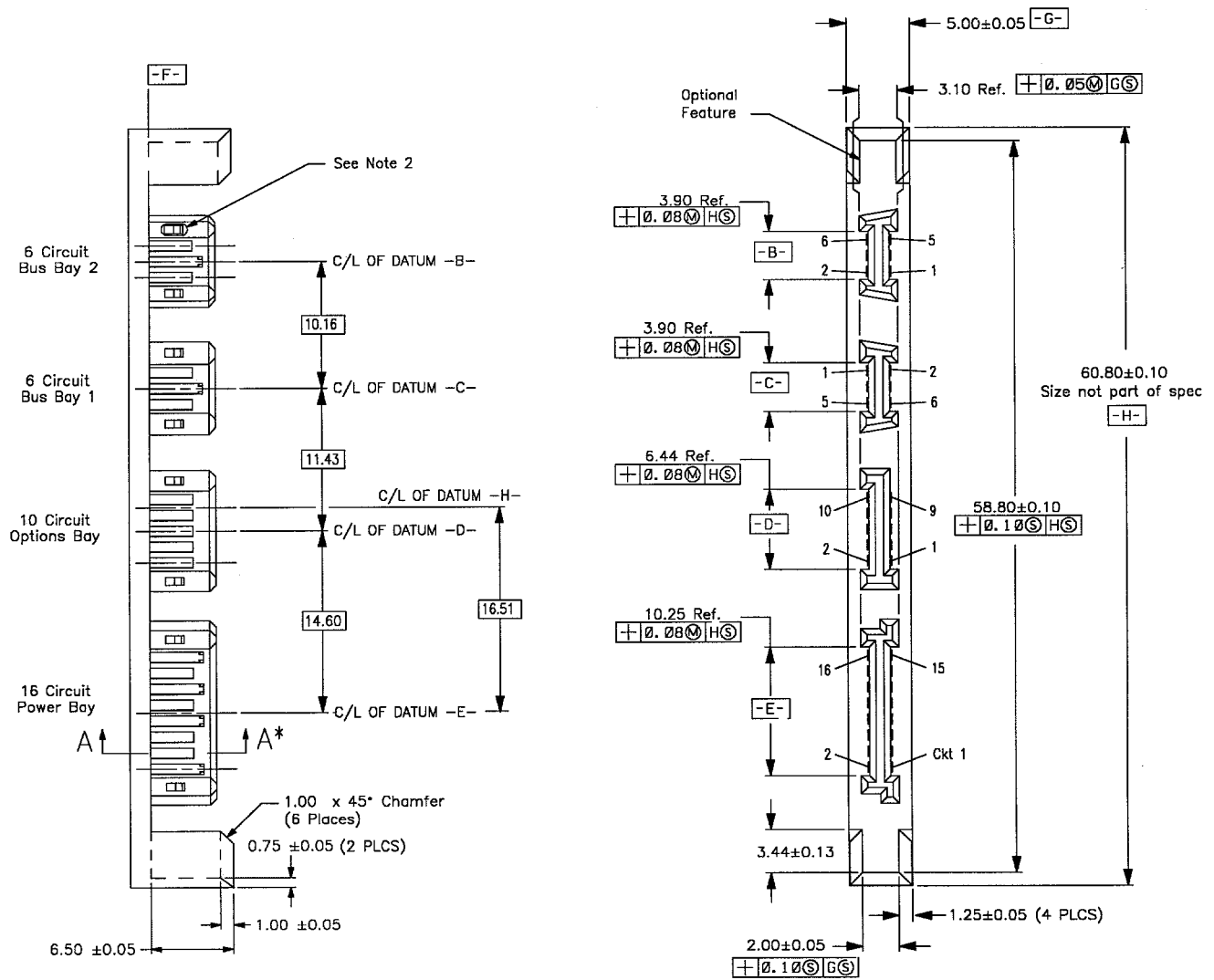


Figure C.3—Internal unitized plug features

Figures C.4 through C.9 describe a plug with a wafer that is segmented to form four separate bays. Each bay has solid, nonflexing contacts recessed within the wafer for protection against damage. Specified contacts in each bay are extended toward the mating face to provide “first-make, last-break” capability on designated power contacts. Each bay also contains a number of recessed detents to provide detent retention between the plug and receptacle connectors.



NOTES

- 1—For section A-A, see figure C-6.
- 2—See figure C-5 for details.

Figure C.4—Internal unitized plug

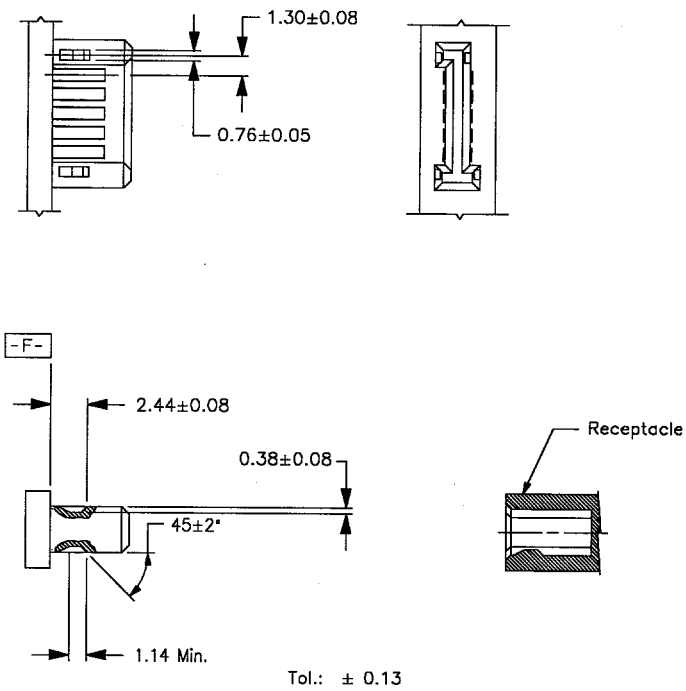


Figure C.5—Internal unitized plug retention (detent detail)

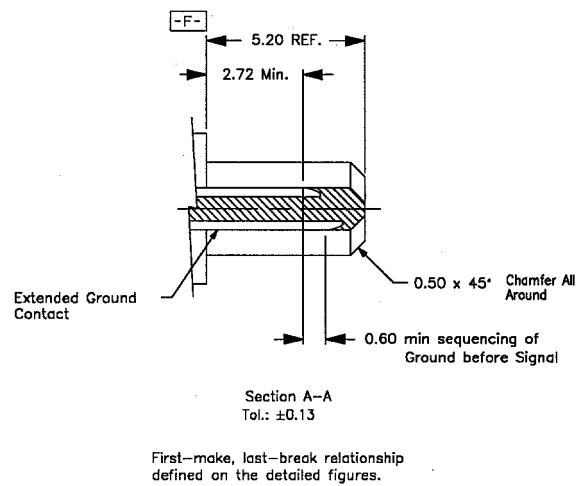


Figure C.6—Internal unitized plug header

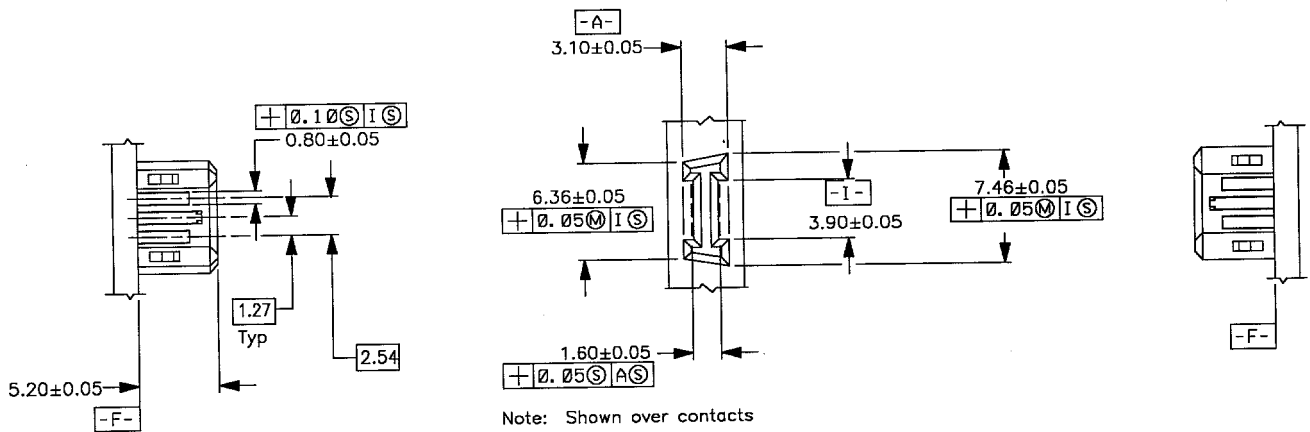


Figure C.7—Internal unitized plug bus (bay detail)

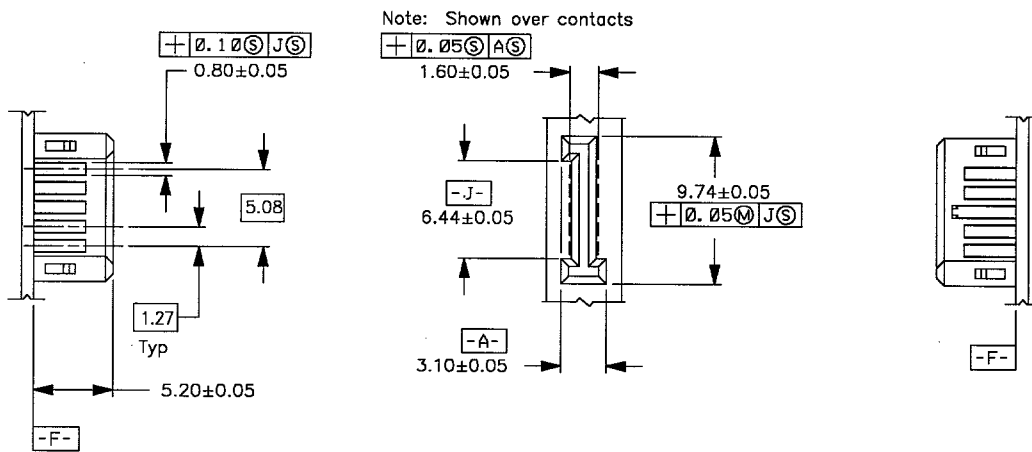


Figure C.8—Internal unitized plug options (bay detail)

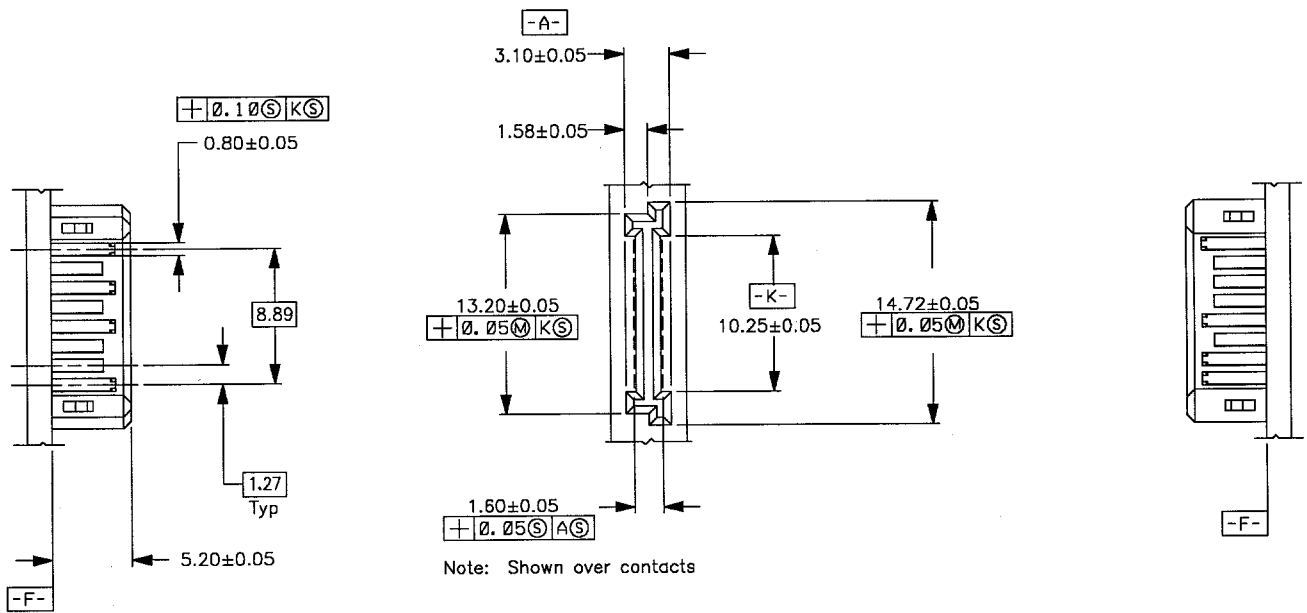
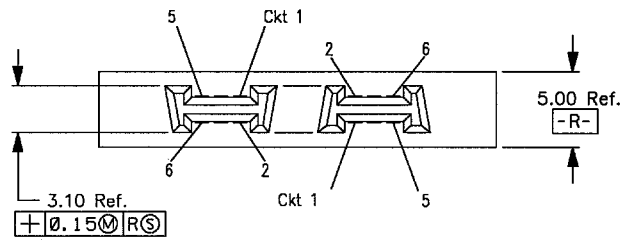
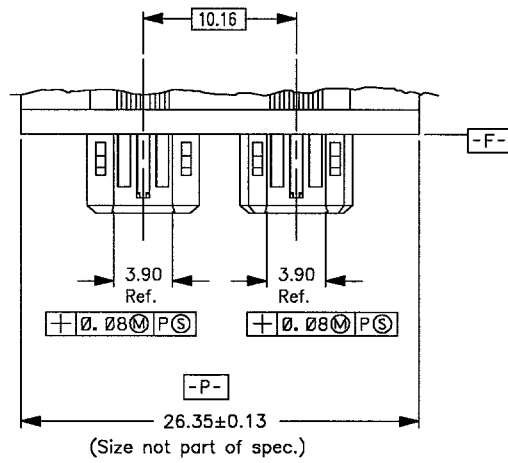


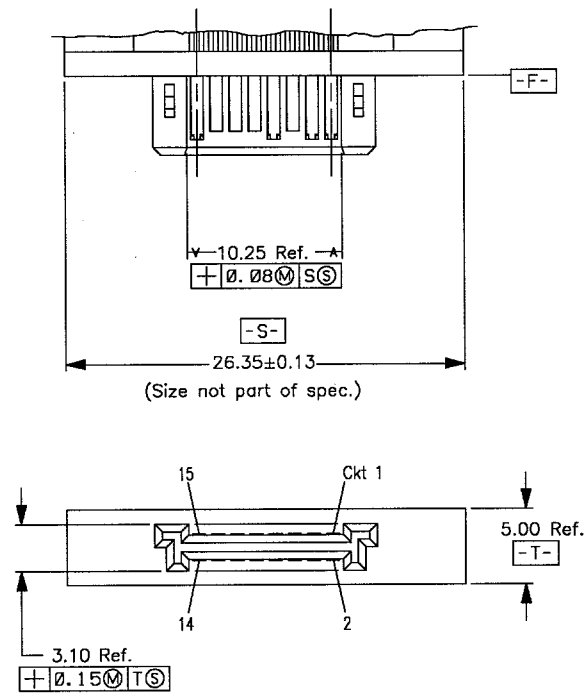
Figure C.9—Internal unitized plug power (bay detail)

Figures C.10 and C.11 describe separate plug headers used for controllers, power supplies, and other applications. These headers are individual versions of the internal unitized plug for bus and power. The bus is shown as a dual bay in figure C.10 and the power as a single in figure C.11.



NOTE—Refer to figures C-4 and C-7 for all other dimensions.

Figure C.10—Internal unitized plug dual bus (bay detail)



NOTE—Refer to figure C-9 for all other dimensions.

Figure C.11—Internal unitized plug discrete power (bay detail)

C.3.2 Internal unitized receptacles

The mating features of the connector backpanel receptacle are illustrated in figure C.12. They will assure the intermateability of the backpanel receptacle with a unitized plug.

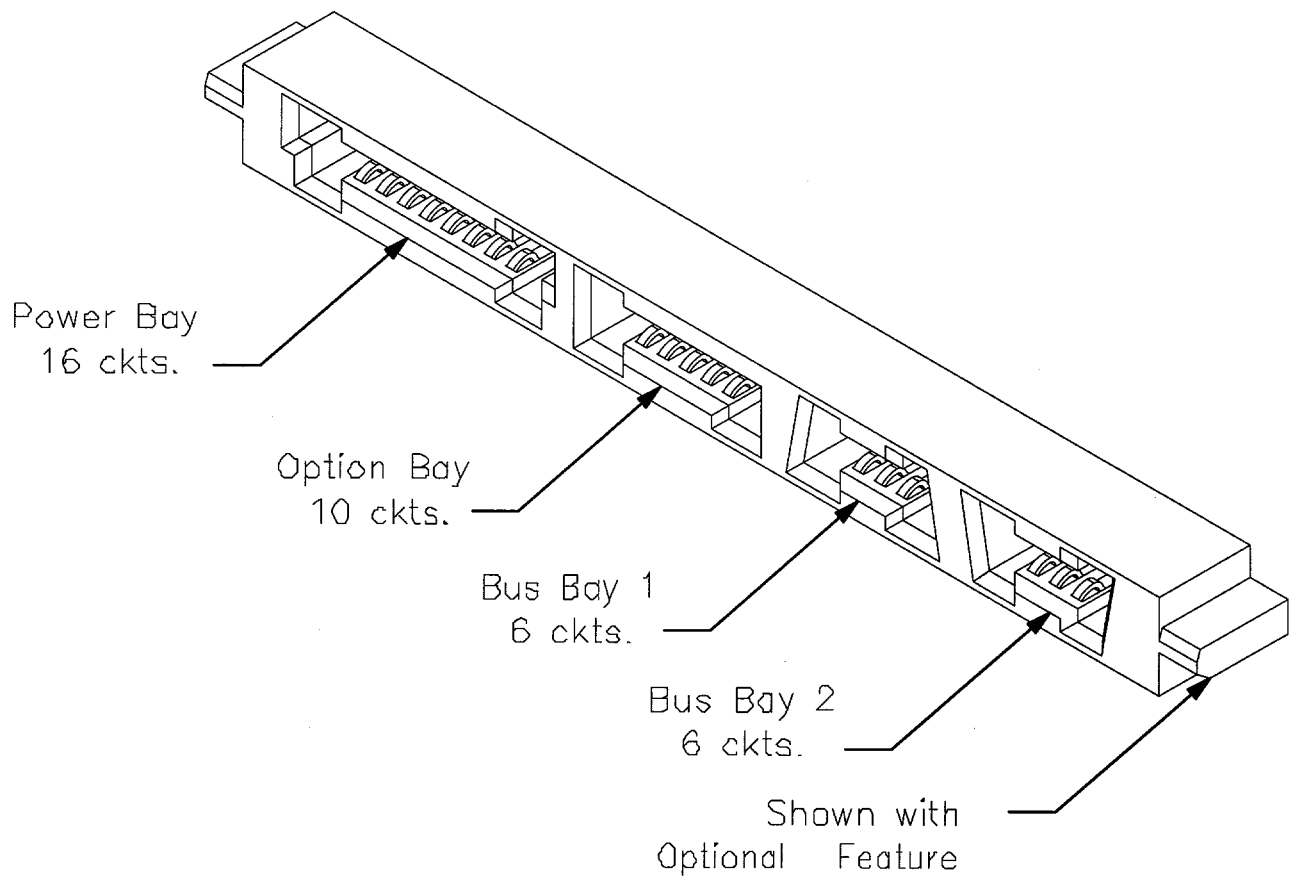
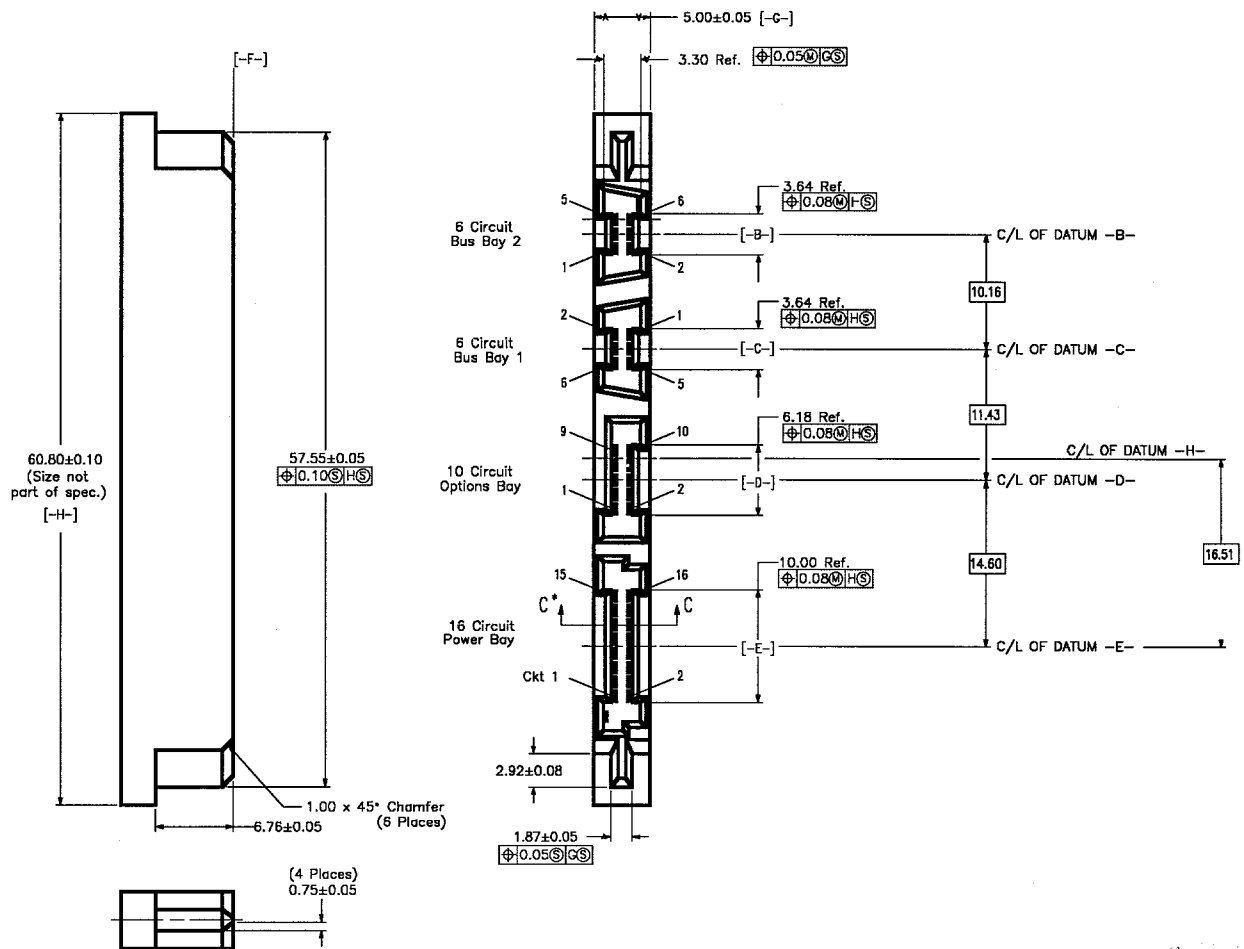


Figure C.12—Internal unitized receptacle features

Figures C.13 through C.17 describe the outer plastic profile, which prevents mechanical damage to the flexing contact members. This assures adequate lead-in for the plug wafers and provides proper contact registration with the contact blades in the plug. Figure C.14 illustrates the contact typically used in the receptacles. Each bay in the plastic shell also provides detents that mate with the recesses on each wafer.



NOTE—For Section C-C, see figure C-14.

Figure C.13—Internal unitized receptacle

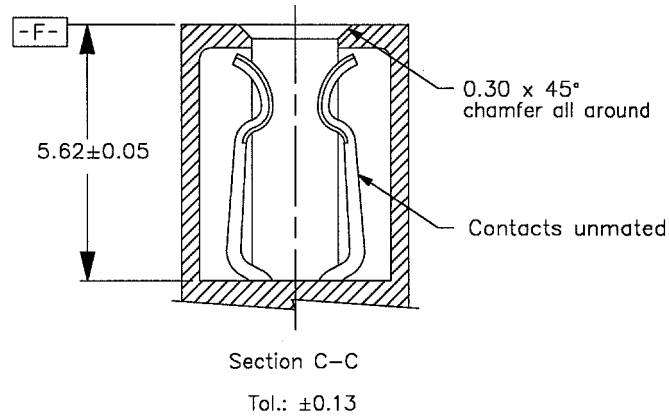


Figure C.14—Internal unitized receptacle contact detail

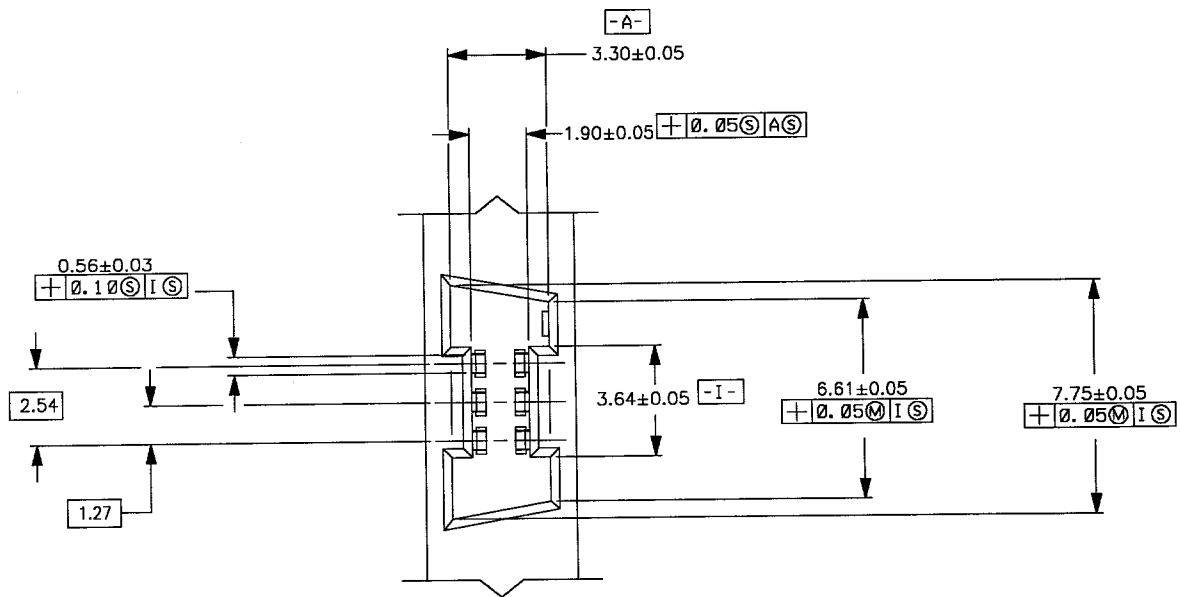


Figure C.15—Internal unitized receptacle bus (bay detail)

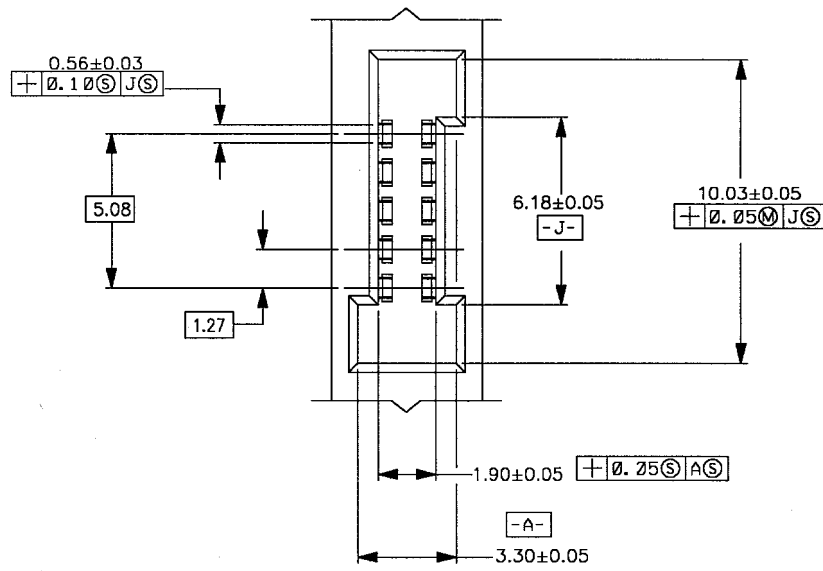


Figure C.16—Internal unitized receptacle option (bay detail)

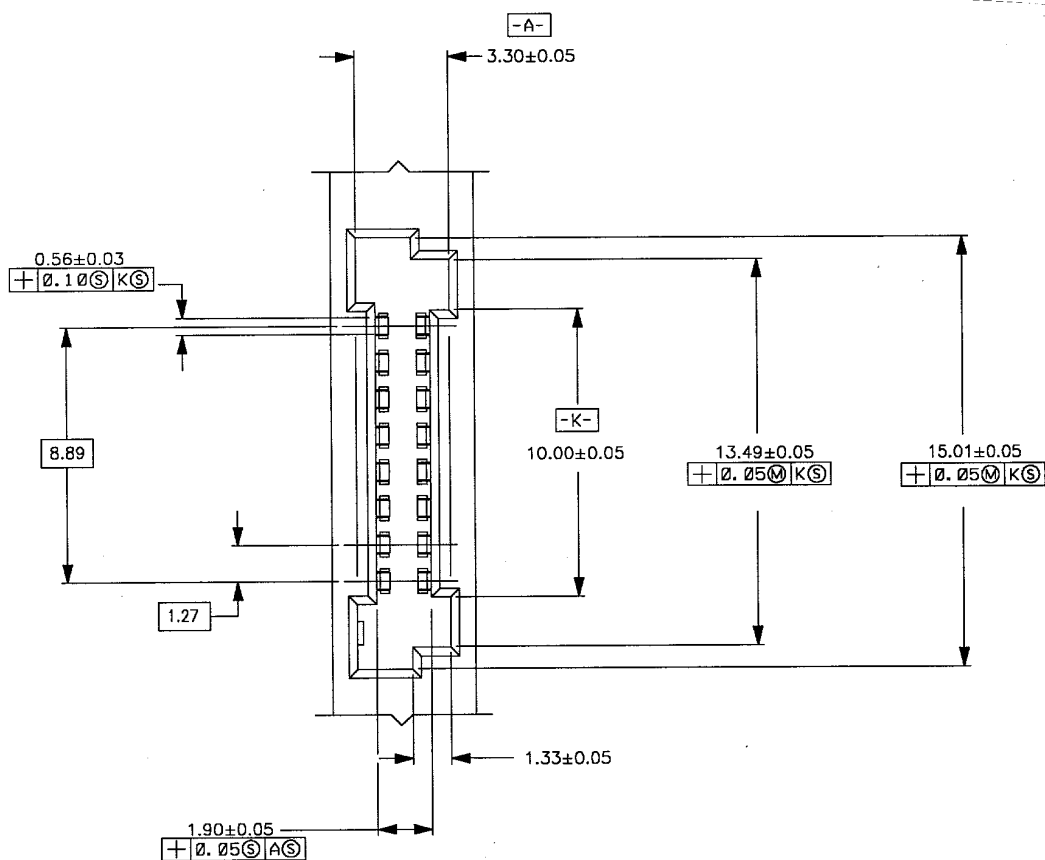


Figure C.17—Internal unitized receptacle power (bay detail)

Figure C.18 shows the necessary contact information to assure an adequate contact wipe distance. It also describes the force to be exerted by the plug contact at the point “Fn” as the “hertz normal force” and the plug contact as the “hertz mating surface.” Interface performance criteria are found in C.3.7.2 through C.3.7.8.

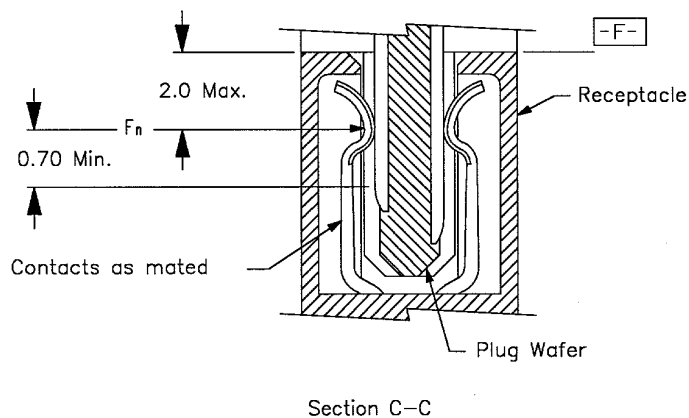
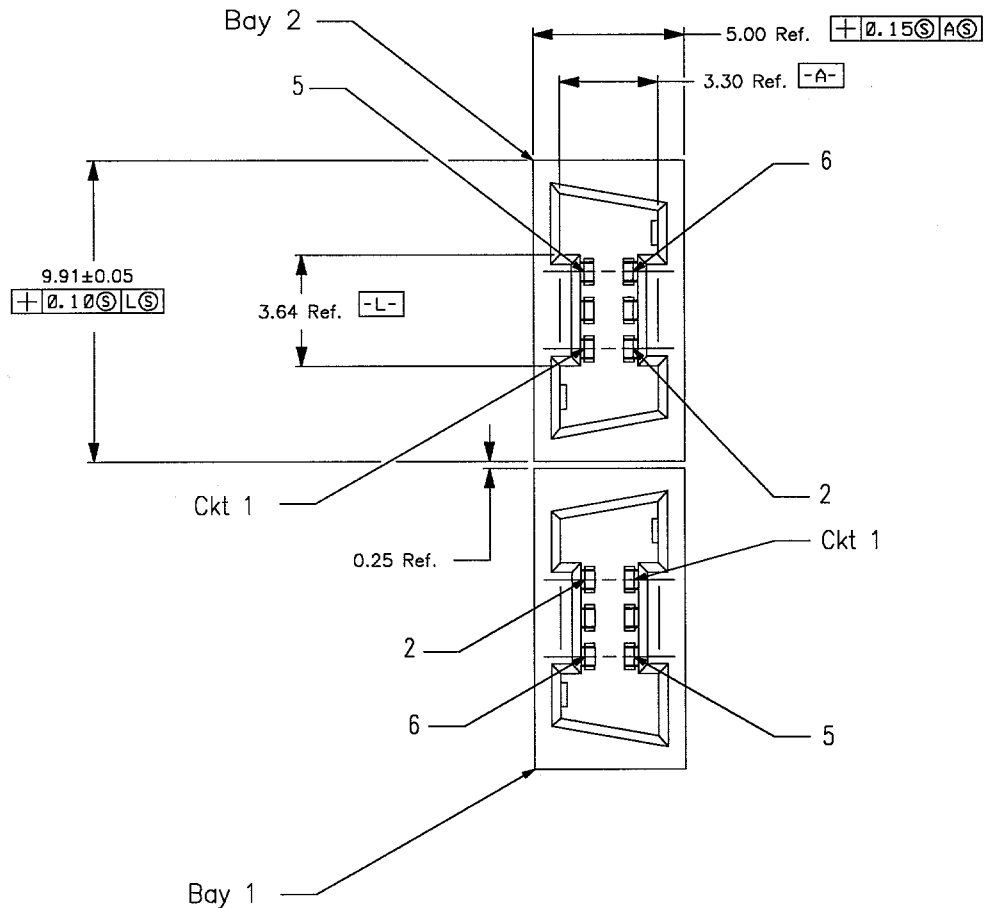


Figure C.18—Internal unitized device connector contact (wipe detail)

The mating interface is a leaf-style design. The receptacle contacts are compliant, with a coined contact area. The plug contact is fixed and mates with the compliant receptacle contact, thus forming the complete electrical interface.

C.3.3 Connector cable receptacles

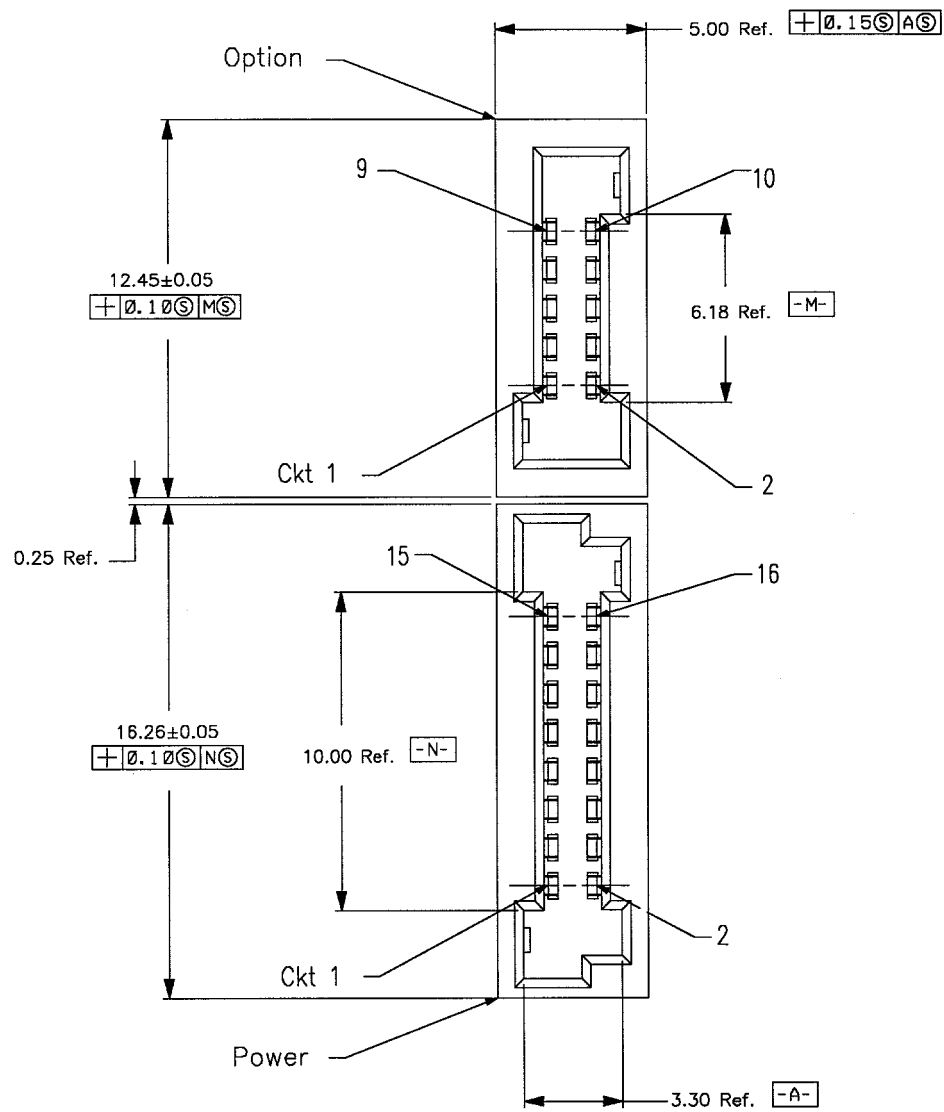
The mating features and the face-view dimensions of the cable receptacles are shown in figures C.19 and C.20 for reference. They will assure the side-to-side stacking and intermateability of the cable receptacles with the corresponding bays of a standardized unitized plug.



NOTES

- 1—The connectors are shown separate from one another.
- 2—Refer to figures C-14 and C-15 for all other dimensions.

Figure C.19—Internal cable receptacle bus (bay detail)



NOTES

- 1—The connectors are shown separate from one another.
 2—Refer to figures C-14, C-17, and C-18 for all other dimensions.

Figure C.20—Internal cable receptacle option and power (bay detail)

The overall terminated size of a power cable receptacle is shown in figure C.21 for reference only. Corresponding bus and optional cable receptacles will be proportionate. All other dimensions can be found in figures C.19 and C.20.

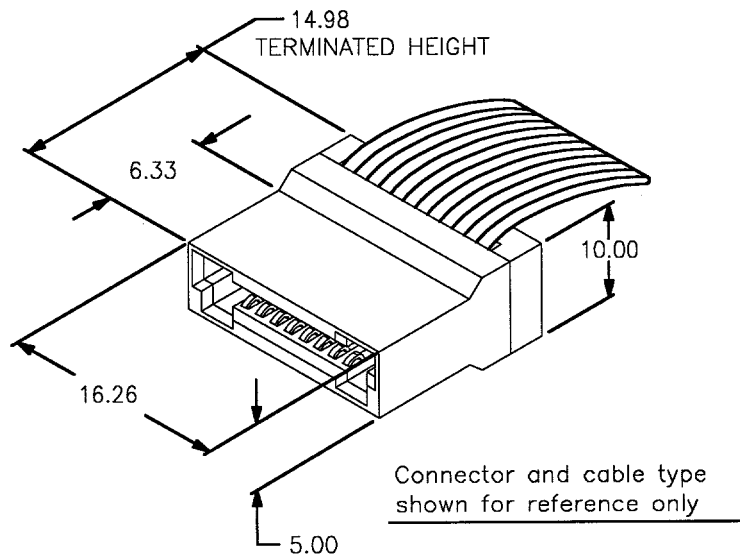


Figure C.21—Overall terminated size of a power cable receptacle

C.3.3.1 Cable receptacle termination

The termination of discrete stranded wire or ribbon cable to the cable receptacles may be varied to suit the manufacturing process needs of the cable assembler. Since the plug termination does not affect the intermateability of the connector/cable assembly, any termination technique may be used, provided it meets the performance requirements in C.3.4.1 and the overall dimensions given in this annex.

For reference, the following acceptable methods are listed:

- a) Crimp
- b) Insulation displacement (IDT)
- c) Insulation piercing
- d) Welding
- e) Soldering

C.3.4 Cable

It is recommended that one of the discrete wire or cables mentioned in the following subclauses be used in conjunction with the cable receptacles.

C.3.4.1 Flat ribbon cable

0.635mm pitch, #30 AWG solid or stranded copper conductor with PVC, TPE, or polypropylene insulation should be used.

C.3.4.2 Discrete wire

#28 or #30 stranded copper conductor with PVC, TPE, or polypropylene insulation should be used. Please note that the insulation diameter of discrete wire has to fit the cable cover profile.

C.3.5 Contact finish on mating surfaces of plug and receptacle contacts

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

- a) 0.76 μm (30 μin), minimum, gold, over 1.27 μm (50 μin), minimum, nickel.
- b) 0.05 μm (2 μin), minimum, 0.127 μm (5 μin), maximum, gold, over 0.76 μm (30 μin) minimum, palladium, over 1.27 μm (50 μin), minimum, nickel.
- c) 0.05 μm (2 μin), minimum, 0.127 μm (5 μin), maximum, 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, one of the electroplatings in the preceding list shall cover the complete area of contact, including the entire contact wipe area.
- 2 — A copper strike is acceptable under the nickel electroplate.
- 3 — A palladium strike is acceptable over the nickel electroplate.

C.3.6 Termination finish on plug and receptacle contact

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.

The recommended platings shall assure a minimum of durability of 500 cycles minimum.

C.3.7 Connector 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.

These Serial Bus internal unitized device connector 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 of ANSI/EIA 364-B-90 are: Temperature, +15 °C to +85 °C, Humidity, 95% maximum. Class 1.3 is further described as operating in a “harsh environmental” state, but with no marine atmosphere.

Accordingly, the performance groupings, sequences within each group, and the test procedures will follow the recommendations of ANSI/EIA 364-B-90, except where the unique requirements of the Serial Bus internal unitized device connector may call for tests that are not covered in ANSI/EIA 364-B-90 or where the requirements deviate substantially from those in that standard. In those cases, test procedures of other recognized authorities will be cited.

Unitized plugs and receptacles shall pass all the following tests in the groups and sequences shown.

C.3.7.1 Test sample preparation

The unitized plug and receptacle shall be tested unassembled for those tests that require it. They shall be tested assembled to randomly selected production printed wiring boards, typical of those used in Serial Bus equipment, for those tests that require assembled connectors.

The cable receptacles shall be tested unassembled for those tests that require it. They shall be tested with any of the cables specified in C.3.1 for those tests that require assembled connectors.

- a) All performance testing is to be done with cable material that conforms to this standard. In order to test to these performance groups, ANSI/EIA tests require that the cable construction used be specified.
- b) 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-B-90.
- c) The numbers 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.
- d) See figure C.22 for vibration and shock fixturing method.
- e) See figure C.23 for contact resistance measurement points.

C.3.7.2 Performance group A: Basic mechanical conformance and electrical functionality when subjected to mechanical shock and vibration

Number of samples:

- [2] 38 contact unitized plugs, unassembled to PCB used for Phase 1, A1, and A2 (one each)
- [4] 38 contact unitized plugs, assembled to PCB
- [2] 38 contact unitized receptacles, assembled to PCB
- [2 each] 16-contact, 10-contact, and 6-contact cable receptacles, assembled to cable, 200 mm long

Table C.8—Performance group A

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
A1	Visual and dimensional inspection	ANSI/EIA 364-18A-84	Unmated connectors	Dimensional inspection	Per Figs. C.4 thru C-11, C- 13 thru C-21	No defects that would impair normal operations. No deviation from dimensional tolerances.
A2	Plating thickness measurement					Record thickness; see C.3.1 and C.3.3.1.
A3	None		See figure C.23	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated contact.
A4	Vibration	ANSI/EIA 364-28A-83	Condition III (See note and figure C.22)	Continuity	ANSI/EIA 364-46-84	No discontinuity at 1 μs or longer. (Each contact
A5	None		Same as A3	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
A6	Mechanical shock (specified pulse)	ANSI/EIA 364-27A-83	Condition G (See note and figure C.22)	Continuity	ANSI/EIA 364-46-84	No discontinuity at 1 μs or longer. (Each contact)
A7	None		Same as A3	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
<p>NOTE — Connectors are to be mounted on a fixture that simulates typical usage. The PCB shall also be permanently affixed to the fixture. The PCB layout used will be in accordance with ones found in this annex. The cable receptacle shall be mated with the plug, and the other end of the cable shall be permanently clamped to the fixture. See figure C.22 for details.</p>						

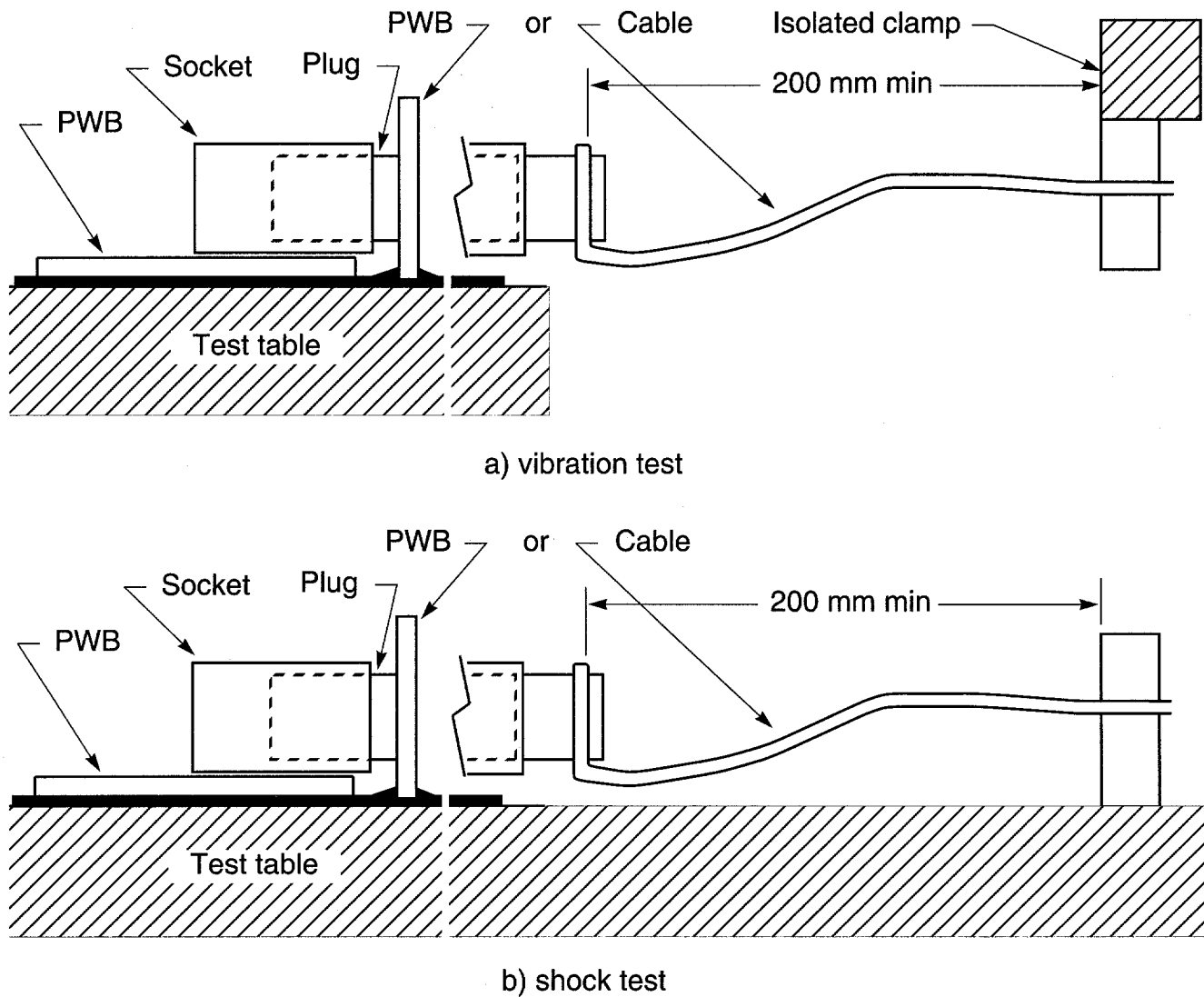


Figure C.22—Shock and vibration fixturing diagram

C.3.7.3 Performance group B: Low-level contact resistance when subjected to thermal shock and humidity stress

Number of samples:

- [4] 38 contact unitized plugs, assembled to PCB
- [2] 38 contact unitized receptacles, assembled to PCB
- [2 each] 16-contact, 10-contact, and 6-contact cable receptacles, assembled to cable, 200 mm long

Table C.9—Performance group C

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
B1	None		See figure C.23	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated contact.
B2	Thermal Shock	ANSI/EIA 364-32B-92	Condition 1 10 cycles (mated). See figure C.23.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
B3	Humidity	ANSI/EIA 364-31A-83	Condition C (504 h). Method III (cycling) nonenergized. Omit steps 7a and 7b (mated).	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

C.3.7.4 Performance group C: Insulator integrity when subjected to thermal shock and humidity stress

Number of samples:

- [4] 38 contact unitized plugs, assembled to PCB
- [2] 38 contact unitized receptacles, assembled to PCB
- [2 each] 16-contact, 10-contact, and 6-contact cable receptacles, unassembled

Table C.10—Performance group C

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
C1	Withstanding voltage	ANSI/EIA 364-20A-83	Test voltage 500 Vdc \pm 50 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	ANSI/EIA 364-32B-92	Condition 1 10 cycles (unmated).	Withstanding voltage (same condition 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 500 Vdc \pm 50 Vdc (unmated and unmounted).	Insulation resistance	ANSI/EIA 364-21A-83	100 M Ω , minimum, between adjacent contacts.
C4	Humidity cyclical	ANSI/EIA 364-31A-83	Condition A (96 h). Method III onenergized. Omit steps 7a and 7b.	Insulation resistance (same conditions as C3)	ANSI/EIA 364-31A-83	100 M Ω , minimum, between adjacent contacts.

C.3.7.5 Performance group D: Contact life and durability when subjected to mechanical cycling and corrosive gas exposure

Number of samples:

[4] 38 contact unitized plugs, assembled to PCB

[2] 38 contact unitized receptacles, assembled to PCB

[2 each] 16-contact, 10-contact, and 6-contact cable receptacles, assembled to cable, 200 mm long

Table C.11—Performance group D

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
D1	None		See figure C.23.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated contact
D2	Durability	ANSI/EIA 364-09B-91	a) 2 mated pairs, 5 cycles. b) 2 mated pairs, automatic cycles to 250 cycles, rate 500 cycles/h 50 cycles.			
D3	None		See figure C.23.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
D4	Mixed flowing gas	ANSI/EIA 364-65-92	See figure C.23. a) 2 mated pairs—unmated for 1 day. b) 2 mated pairs—mated for 10 days.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.

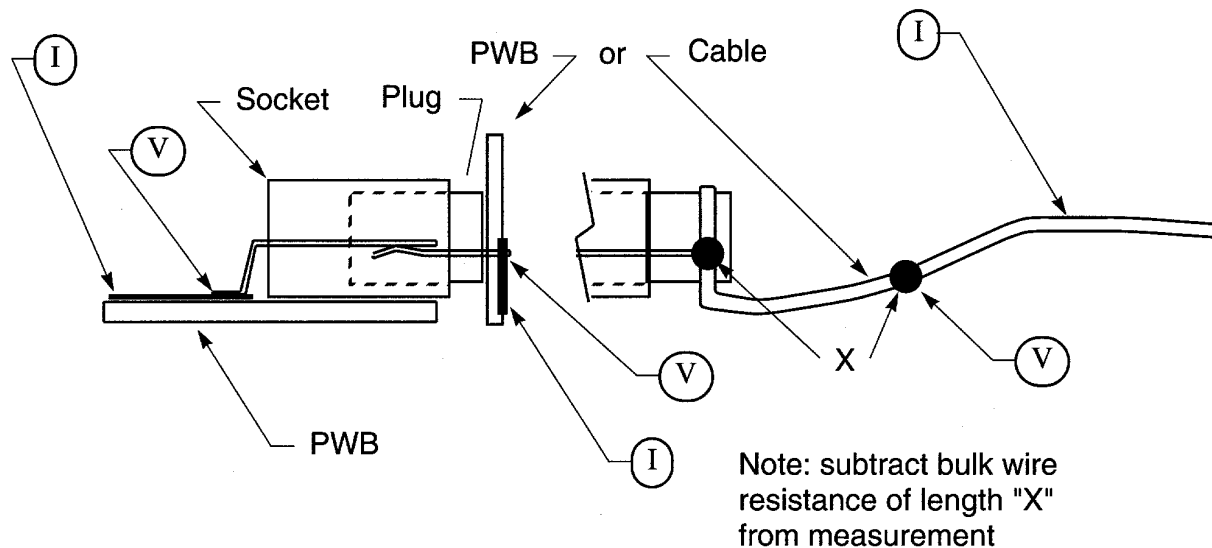


Figure C.23—Contact resistance measuring points

C.3.7.6 Performance group E: Contact resistance and mating and unmating force when subjected to temperature life stress

Number of samples:

- [4] 38 contact unitized plugs, assembled to PCB
- [2] 38 contact unitized receptacles, assembled to PCB
- [2 each] 16-contact, 10-contact, and 6-contact cable receptacles, assembled to cable, 200 mm long

Table C.12—Performance group E

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
E1	Mating and unmating	ANSI/EIA 364-13A-83	Mount plug rigidly. Insert receptacle by hand.	Mating only		
			Auto rate: 25 mm/min	Unmating only	ANSI/EIA 364-13A-83	Unmating force: 9.8 N minimum 39.2 N maximum
E2	None		See figure C.23.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ, maximum, initial per mated contact.
E3	Temperature life	ANSI/EIA 364-17A-87	Condition 4 (105 °C) 250 h. Method A (mated). See figure C.23.	Low-level contact resistance	ANSI/EIA 364-23A-85	30 mΩ maximum change from initial per mated contact.
E4	Mating and unmating forces	ANSI/EIA 364-13A-83	Same as E1.	Mating and unmating forces	ANSI/EIA 364-13A-83	Same as E1.

C.3.7.7 Performance group F: Mechanical retention and durability

Number of samples:

[4] 38 contact unitized plugs, assembled to PCB

[2] 38 contact unitized receptacles, assembled to PCB

[2 of each] 16-contact, 10-contact, and 6-contact cable receptacles, assembled to cable 254 mm long minimum

Table C.13—Performance group F

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
F1	Mating and unmating forces	ANSI/EIA 364-13A-83	Mount plug rigidly by hand.			
F2	Mating and unmating forces	ANSI/EIA 364-13A-83	Auto cycle: 100 cycles, 25 mm/min	Unmating forces	ANSI/EIA 364-13A-83	Unmating force: 9.8 N minimum 39.2 N maximum
F3	Durability	ANSI/EIA 364-09B-91	Automatic cycling to 500 cycles	Unmating only	ANSI/EIA 364-13A-83	Unmating force at end of durability cycling: 9.8 N minimum 39.2 N maximum

C.3.7.8 Performance group G: General tests

Suggested procedures to test miscellaneous, but important aspects of the connectors.

NOTE — Since the tests listed below may be destructive, separate samples have to be used for each test. The number of samples to be used is listed under the test title.

Table C.14—Performance group G

Phase	Test			Measurements to be performed		Requirements
	Title	ID number	Severity or conditions	Title	ID number	Performance level
G2	Cable axial pull test [2 plugs]		Fix plug housing and apply a 98 N load for 1 min on cable axis.	Continuity, Visual	ANSI/EIA 364-46-84	No discontinuity greater than 1 s. 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 = $3.7 \times$ cable diameter or thickness; 100 cycles, in each of two planes.	a) Withstanding voltage	Per C1.	Per C1.
				b) Insulation resistance	Per C3.	Per C3.
				c) Continuity of all contacts	Per A4.	Per A4.

Annex D Backplane PHY timing formulas

(Informative)

This annex contains the formulas used to derive the Serial Bus timing requirements for the backplane PHY. The information in the annex is not required for the design of the Serial Bus PHY. It is intended to serve as a guide for understanding the timing requirements contained within the backplane PHY specifications defined in clause 5. This annex may be useful for the configuration of nonstandard topologies of the Serial Bus as well as for future enhancements to the Serial Bus standard.

D.1 Backplane propagation delay

The propagation delay of Serial Bus signals on a “host” parallel backplane limits the speed of the arbitration process and packet transmission. In order to specify requirements for arbitration timing, gap timing, and skew tolerances, it is necessary to define a maximum propagation delay for Serial Bus signals.

The one-way propagation delay (T_{pd}) of a backplane can be calculated with the following equations:

$$T_{pd} = T_{pl} \cdot L \quad (D-1)$$

$$T_{pd} = T_{po} \cdot \sqrt{1 + (Cl \cdot N/C)} \cdot L \quad (D-2)$$

$$T_{pd} = T_{po} \cdot (\sqrt{1 + (Cl \cdot M)/(L \cdot C)}) \cdot L \quad (D-3)$$

where

M	is the number of cards on backplane
L	is the length of the backplane
N	is the electrical pitch of cards on the backplane
Cl	is the capacitive loading per card
C	is the capacitive loading per unit length on unloaded backplane
T_{po}	is the propagation delay per unit length on unloaded backplane
T_{pl}	is the propagation delay per unit length on loaded backplane
T_{pd}	is the one-way propagation delay on loaded backplane

Table D.1 indicates the “worst-case” one-way propagation delay (T_{pd}) and the round-trip propagation delay (T_{rt}) for a number of standard backplanes. Although this table addresses IEEE 896 (Futurebus+), ANSI VME64, and IEEE 1296 (MULTIBUS II), the application of Serial Bus is not limited to these backplanes.

Table D.1—Backplane propagation delay

Bus Name	M	L	Cl	C	T _{po}	T _{pl}	T _{pd}	T _{rt}
Futurebus+	14 cards	0.421 m (1.38 ft)	10 pF/card	66 pF/m (20 pF/ft)	5.68 ns/m (1.73 ns/ft)	13.98 ns/m (4.26 ns/ft)	5.88 ns	11.77 ns
Folded Futurebus+	31 cards	1.259 m (4.13 ft)	10 pF/card	66 pF/m (20 pF/ft)	5.68 ns/m (1.73 ns/ft)	12.37 ns/m (3.77 ns/ft)	15.58 ns	31.16 ns
VME64	21 cards	0.500 m (1.64 ft)	20 pF/card	66 pF/m (20 pF/ft)	5.81 ns/m (1.77 ns/ft)	21.59 ns/m (6.58 ns/ft)	10.79 ns	21.57 ns
MULTIBUS II	21 cards	0.427 m (1.40 ft)	20 pF/card	66 pF/m (20 pF/ft)	5.81 ns/m (1.77 ns/ft)	23.23 ns/m (7.08 ns/ft)	9.91 ns	19.82 ns
NOTE — “Folded Futurebus+” indicates a 31-slot Futurebus+ backplane that uses a special signal distribution to minimize capacitance between adjacent modules. This results in an electrical length (L) that is twice the physical length of the backplane. Although the length of VME64 backplanes is the same as MULTIBUS II backplanes, VME64 backplanes allow for the use of offboard terminators, which can increase the electrical length. This results in a slight increase in the maximum round-trip propagation delay for VME64 backplanes.								

In order to accommodate all of these buses with a margin of error, the one-way propagation is specified (in 5.2.4.2) to be a maximum of 18 ns.

D.2 Backplane arbitration timing

Because the arbitration process does not use any form of acknowledgment or handshaking, proper operation of the arbitration process can only occur if all arbitrating nodes adhere to the same arbitration timing. Because the configuration of a given backplane is unknown (e.g., propagation delay of the backplane), it is necessary to specify for an absolute worst case. The following subclauses use the physical parameters of a worst-case scenario in order to calculate the arbitration timing.

NOTE — It is possible to determine automatically the physical parameters of a given backplane. This would allow for dynamic configuration of “optimal” values for arbitration timing. This, however, would require a level of complexity that is beyond the scope of the backplane environment.

D.2.1 Synchronization timing

The arbitration process requires participating nodes to be synchronized with each other, rather than with a distributed clock. This assumes that the arbitration state machines of any two nodes on the Serial Bus must be “out of sync” within a maximum “sync delay” in order for the arbitration process to occur properly. This “sync delay” takes into account the time required for a signal to be asserted on the bus, the time required for the bus to settle, and the amount of time required to detect a change on the bus.

The time required for a signal to be asserted on the bus is equal to the clock-to-output delay of the decision logic (FF_{co}), plus the propagation delay through the bus driver (TX_{pd}).

It is assumed that a round-trip bus propagation delay (T_{rt}) is required for the bus to settle during arbitration. This is assumed because TTL backplanes require reflections to operate properly and because BTL backplanes require time for glitches to settle.

The time required for a signal to be detected on the bus is equal to the propagation delay through the bus receiver (RX_{pd}), plus the setup time for the sample circuitry (FF_{su}). To greatly reduce the possibility for metastability, the incoming arbitration sequence is synchronized to an internal clock before it is sampled. This process adds up to one arbitration clock time ($T_{clk} = 20.345$ ns) to the “sync delay.”

Consequently, the “sync delay” is equal to

$$T_{\text{sync}} = FF_{\text{co}} + TX_{\text{pd}} + T_{\text{rt}} + RX_{\text{pd}} + FF_{\text{su}} + T_{\text{clk}} \quad (\text{D-4})$$

where the following values were assumed:

FF_{co}	is the clock to output delay of decision circuitry = 15 ns
TX_{pd}	is the propagation delay through bus driver = 8 ns
T_{rt}	is the round trip bus delay = 36 ns
RX_{pd}	is the propagation delay through bus receiver = 8 ns
FF_{su}	is the setup time of sample circuitry = 15 ns
T_{clk}	is the period of clock used to sample STRB and DATA (20.345 ns)

so that

$$\begin{aligned} T_{\text{sync}} &= FF_{\text{co}} + TX_{\text{pd}} + T_{\text{rt}} + RX_{\text{pd}} + FF_{\text{su}} + T_{\text{clk}} \\ &= 15 \text{ ns} + 8 \text{ ns} + 36 \text{ ns} + 8 \text{ ns} + 15 \text{ ns} + 20.345 \text{ ns} \\ &= 102.345 \text{ ns} \end{aligned}$$

D.2.2 Arbitration sample timing

Before a node can begin arbitration, it has to sample the bus to determine if the bus is busy. If the bus is not busy, the node may begin the arbitration process by asserting the bus. It is possible, however, for one node to begin asserting the bus just as another node begins sampling the bus in order to begin the arbitration process. This second node will not detect activity on the bus because a finite amount of time is required for the asserted signal from the first node to reach the second node. Consequently, the second node will begin arbitrating as the first node continues arbitrating. In order to ensure that the bus has settled, the first node has to wait a certain amount of time (a “sample delay”) before it begins sampling the arbitration data on the bus.

The “sample delay,” T_{smp} , is an integral number of arbitration clock times, which is greater than the sum of the “sync delay” and the round-trip bus propagation delay, as well as the logic delay required by the PHY sampling/decision circuitry.

$$\begin{aligned} T_{\text{smp}} &= \text{int} (T_{\text{sync}} + FF_{\text{co}} + TX_{\text{pd}} + T_{\text{rt}} + RX_{\text{pd}} + FF_{\text{su}} / T_{\text{clk}} + 1) \\ &= \text{int} (102.345 \text{ ns} + 15 \text{ ns} + 8 \text{ ns} + 36 \text{ ns} + 8 \text{ ns} + 15 \text{ ns} / 20.345 \text{ ns} + 1) \\ &= \text{int} 184.345 \text{ ns} / 20.345 \text{ ns} + 1 \\ &= 9 + 1 \\ &= 10 \text{ bits} \end{aligned} \quad (\text{D-5})$$

Arbitration hold timing

In order to ensure that all nodes have had a chance to sample the arbitration data on the bus, an arbitrating node shall not change the state of the bus until it has waited a “hold delay” after sampling the bus.

The “hold delay” is equal to the integral number of arbitration clock times, which is greater than the sync delay.

$$\begin{aligned}
 T_{\text{hold}} &= \text{int}(T_{\text{sync}}/T_{\text{clk}}) + 1 \\
 &= \text{int}(102.345 \text{ ns}/20.345 \text{ ns}) + 1 \\
 &= 5 + 1 \\
 &= 6 \text{ bits}
 \end{aligned}
 \tag{D6}$$

D.2.3 Arbitration bit timing

The timing required (T_{arb}) to complete one bit of the arbitration sequence can be calculated with the following equation:

$$\begin{aligned}
 T_{\text{arb}} &= T_{\text{smp}} + T_{\text{hold}} \\
 &= 10 \text{ bits} + 6 \text{ bits} \\
 &= 16 \text{ bits}
 \end{aligned}
 \tag{D7}$$

This arbitration bit timing is specified in 5.2.4.3.

D.3 Backplane gap timing

An interpacket gap occurs when the bus signals (both STRB and DATA) are unasserted for a certain duration. The bus signals are sampled to determine the length of a gap (at 49.152 MHz). Depending upon the duration of the gap, a backplane PHY may or may not be allowed to perform certain actions (either on the bus or internal to the PHY).

The bus becomes idle once two data bit periods have occurred during which neither DATA nor STRB are unasserted. Because an incoming transmission may be at 49.152 Mbit/s or 24.576 Mbit/s, an idle bus is detected after four arbitration clock times (at 49.152 MHz, or approximately 20.345 ns) have occurred without a transition on either STRB or DATA.

Upon detecting an idle bus, a node may begin the arbitration process. It has to wait a certain number of clock periods (depending upon the type of access being used), while sampling the bus to determine that it has remained idle during this time, before it can enter the arbitration contest. As is indicated in table D.2, the type of access requested by a node determines the type of gap that has to be detected by the node, as well as the resulting action taken by the node.

Table D.2—Gap types

Gap type	Access type	Action
Acknowledge gap	Acknowledge	(send acknowledge)
	Isochronous	(begin arbitrating)
Subaction gap	Fair or Urgent	(begin arbitrating)
Arbitration reset gap	Fair or Urgent	(reset flags, begin arbitrating)

In order for a node to be able to distinguish between different types of gaps, it is necessary that each gap be of a different duration.

D.3.1 Acknowledge gap

D.3.1.1 Occurrence of acknowledge gap

Acknowledge Transfers—During the time a node sends an acknowledge, it is assured ownership of the bus. There can be no other node transmitting on the bus or contending for the bus (assuming that the acknowledge is transmitted in a timely manner). Theoretically, it is only necessary for the acknowledging node to begin transmission of the acknowledge once it determines that the bus is idle (after four arbitration clock times have occurred without the bus being asserted). However, in order to simplify the circuitry required for this process, access for the bus for an acknowledge transmission is treated in the same manner as that for an isochronous access.

Isochronous Transfers—When a node is attempting an isochronous access, it is not assured ownership of the bus. Just as with fair and urgent transfers, isochronous transfers first require a node to arbitrate for the bus (acknowledge transfers do not require arbitration for the bus).

D.3.1.2 Acknowledge gap timing

Since arbitration is a synchronous process, it is necessary for arbitrating nodes to obey certain timing rules. This allows two nodes to arbitrate simultaneously for the bus. If a node wishing to obtain isochronous access to the bus were to begin arbitrating as soon as it detected an idle bus, it is possible that other nodes would not be able to detect the idle bus (because it has already been asserted by the first node) and consequently not get a chance to join the arbitration contest.

If a node wishes to obtain access for an acknowledge or an isochronous transfer, it has to wait for an idle bus to occur. This occurs after a “sample time” equal to four arbitration clock times. Once the node detects that four arbitration clock times have occurred without an assertion on the bus, it is allowed to begin the arbitration contest. In order to allow other nodes to detect the idle bus, this node has to then wait a “hold time” before asserting the bus. This “hold time” is the integral number of clock periods greater than the maximum synchronization delay between nodes.

The “sample time” is equal to the value in table D.3.

Table D.3—Acknowledge gap sample time

Parameter	Value	Notes
$4 T_{\text{clk}}$	81.38 ns	Equal to two TTL bit periods (four clock periods)

The “hold time” is the sum of the values in table D.4.

Table D.4—Acknowledge gap hold time

Parameter	Value	Notes
T_{rt}	36 ns	Some node has just released the bus signals, and the transition has just reached the bus. It is assumed that node A detects this transition immediately, but that node B requires a reflected wave.
$B\text{-}RX_{\text{pd}}$	8 ns	Even though this signal has to go through the RX and meet the setup time in both nodes, it has to be assumed that these values are minimum (0) for node A and maximum for node B (worst case)
$B\text{-}FF_{\text{su}}$	15 ns	
$B\text{-}T_{\text{clk}}$	20.345 ns	
TOTAL	79.345 ns	3.90 clock periods
NOTE — When considering bus delay, T_{rt} (round trip bus delay) has to be used when one or more nodes are releasing the bus. This is because reflections may be required on a TTL backplane and because glitches may occur on a BTL backplane (if two nodes release the bus at the same time). T_{pd} (one-way bus delay) can only be used if it is certain that one or more nodes are asserting the bus.		

Since the sample time needs to be four clock periods and the hold time needs to be four clock periods, the acknowledge gap must be $4 + 4 \text{ clks} = 8 \text{ clks}$.

Consequently, if a node is to transmit an acknowledge on the bus, it first has to detect that the bus is unasserted for four clock periods and then wait four more clock periods before it can begin transmitting the acknowledge.

If a node is to arbitrate for isochronous access, it first has to detect that the bus is unasserted for four clock periods and then wait four more clock periods before it can begin arbitrating for the bus.

D.3.2 Subaction gap and arbitration reset gap

D.3.2.1 Occurrence of subaction and arbitration reset gaps

Fair or Urgent Transfers—When a node is attempting a fair or urgent access, it is not assured ownership of the bus. Consequently, these transfers first require a node to arbitrate for the bus. Depending upon the state of the `urgent_count` or `arbitration_enable_flag` for a node, arbitration can occur after a subaction gap or an arbitration reset gap.

D.3.2.2 Subaction gap and arbitration reset gap timing

In order for a node to be able to distinguish between an acknowledge gap and a subaction gap, as well as between a subaction gap and an arbitration reset gap, it is necessary that each gap be of a different duration.

Assume node B is one sync time behind node A, that node A is waiting to begin arbitrating after a subaction gap (e.g., for a fair transfer), and that node B is waiting to begin arbitrating after an acknowledge gap (e.g., for an isochronous transfer). Node A has to wait long enough so that it can detect that node B has begun arbitrating.

Table D.5—Difference in gap times

Parameter	Value	Notes
T_{rt}	36 ns	Some node has just released the bus signals, and the transition has just reached the bus. It is assumed that node A detects this transition immediately, but that node B requires a reflected wave.
B- RX_{pd}	8 ns	Even though this signal has to go through the RX and meet the setup time in both nodes, it has to be assumed that these values are minimum (0) for node A and maximum for node B (worst case).
B- FF_{su}	15 ns	
B- T_{clk}	20.345 ns	For synchronization (B is one clock period behind node A)
B- FF_{co}	15 ns	B has just asserted the bus to begin arbitration
B- TX_{pd}	8 ns	
T_{pd}	18 ns	Use T_{pd} because node B is asserting the bus. It has to be assumed that these values are now at their maximum for node A (worst case).
A- RX_{pd}	8 ns	
A- FF_{su}	15 ns	
TOTAL	143.345 ns	7.05 clock periods

Consequently, the sample time for a subaction gap has to be eight clock periods longer than an acknowledge gap. Since other nodes may also be sampling for the occurrence of a subaction gap (and they may be “lagging” by a maximum of four clock periods), a node has to wait four more clock periods after detecting a subaction gap before it can begin arbitrating for the bus.

The same calculations can be made to determine the length of an arbitration reset gap. The sample time for an arbitration reset gap has to be eight clock periods longer than a subaction gap. Since other nodes may also be sampling

for the occurrence of an arbitration reset gap, a node has to wait four more clock periods after detecting an arbitration reset gap, before it can begin arbitrating for the bus.

Therefore, a node has to sample the bus at the time indicated in table D.6 and cannot assert the bus until the time indicated in table D.6.

Table D.6—Gap timing

Gap type	Sample bus after	Do not assert until
Acknowledge gap	4 clocks	8 clocks
Subaction gap	16 clocks	20 clocks
Arbitration reset gap	28 clocks	32 clocks

This arbitration gap timing is reflected in 5.3.3.

D.3.3 Arbitration gap scenarios

The following scenarios are the “worst-case” examples used to determine the gap times.

D.3.3.1 Scenario 1: acknowledge gap

There are three nodes on the end of a bus.

Node 1 (in slot 1) has just finished the transmission of a packet and releases the bus.

Node 2 (in slot 2) is waiting to begin arbitration for isochronous access and is waiting for an ack gap to occur. It detects that Node 1 has released the bus at the incident wave; moreover, it has very fast RX and sample circuitry and has a clock that happens to be “in phase” with Node 1. Therefore, this detection occurs almost immediately.

Node 3 (in slot 3) is also waiting to begin arbitration for isochronous access and is waiting for an ack gap to occur. It detects that Node 1 has released the bus at the reflected wave; moreover, it has very slow RX and sample circuitry and has a clock that is out of phase with Node 1. Therefore, this detection occurs at a time equal to the maximum sync delay, which is equal to $T_{rt} + RX_{pa} + FF_{su} + T_{clk}$. Substituting for these values yields a delay of $36 \text{ ns} + 8 \text{ ns} + 15 \text{ ns} + 20.345 \text{ ns} = 79.345 \text{ ns}$, or four clock periods.

Node 2 now has counted four unasserted clock periods and wants to begin arbitration. It cannot do so, however, because if it asserts the bus, this action can and will be immediately detected by Node 3 (assume that Node 3 is able to detect the assertion at the incident wave for Node 2, and that the detection occurs immediately). In order to ensure that Node 3 has had enough time to detect that Node 1 has released the bus, Node 2 has to wait one max sync time ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods) before it can assert the bus.

Therefore, in order for both Node 2 and Node 3 to have the opportunity to arbitrate, both nodes have to wait one sample time (four clocks) before sampling the bus, and then one hold time (four clocks) before asserting the bus. This is a total of $4 + 4 = 8$ clock periods.

D.3.3.2 Scenario 2: subaction gap

There are three nodes at the end of a bus.

Node 1 (in slot 1) has just finished the transmission of a packet and releases the bus.

Node 2 (in slot 2) is waiting to begin arbitration for fair access and is waiting for a subaction gap to occur. It detects that Node 1 has released the bus at the incident wave; moreover, it has very fast RX and sample circuitry and has a clock that happens to be “in phase” with Node 1. Therefore, this detection occurs almost immediately.

Node 3 (in slot 3) is also waiting to begin arbitration for fair access and is waiting for a subaction gap to occur. It detects that Node 1 has released the bus at the reflected wave; moreover, it has very slow RX and sample circuitry and has a clock that is out of phase with Node 1. Therefore, this detection occurs at a time equal to the maximum sync delay ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods).

Node 2 now has counted four unasserted clock periods, but it does not know if there are other nodes that are waiting for an ack gap to obtain access (for an acknowledge or isochronous access). It knows that such a node would assert the bus after detecting a total of eight unasserted clock periods. Moreover, this node could be one max sync delay out of phase ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$), and if it were to assert the bus, Node 2 might not detect this until some time later ($FF_{co} + TX_{pd} + T_{pd} + RX_{pd} + FF_{su}$). The sum of these two delays is $79.345 \text{ ns} + 64 \text{ ns} = 143.345 \text{ ns}$, or eight clock periods.

Therefore, Node 2 has to count the first four clock periods, plus four additional clocks (to complete the ack gap), plus eight more clocks (to all another node to detect the ack gap and assert the bus) before Node 2 can sample the bus. This is a total of $4 + 4 + 8 = 16$ clocks.

Node 2 now has counted 16 unasserted clocks and wants to begin arbitration.

Node 3, however, is one max sync time ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods) behind Node 2 and has only counted 12 unasserted clock periods. If it is to join in the arbitration process, it must be allowed to detect four more clock periods.

Consequently, Node 2 cannot immediately assert the bus after sampling 16 clock periods. If it asserts the bus, this action can and will be immediately detected by Node 3 (assuming that Node 3 is able to detect the assertion by Node 2 at the incident wave, and that this detection occurs immediately). In order to ensure that Node 3 has had enough time to detect that Node 1 has released the bus, Node 2 has to wait one max sync time ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods) before it can assert the bus.

Therefore, in order for both Node 2 and Node 3 to have the opportunity to arbitrate, both nodes have to wait one sample time (16 clock periods) before sampling the bus, and then one hold time (four clock periods) before asserting the bus. This is a total of $16 + 4 = 20$ clock periods.

D.3.3.3 Scenario 3: arbitration reset gap

There are three nodes at the end of a bus.

Node 1 (in slot 1) has just finished the transmission of a packet and releases the bus.

Node 2 (in slot 2) is waiting to begin arbitration for fair access and is waiting for an arbitration reset gap to occur. It detects that Node 1 has released the bus at the incident wave; moreover, it has very fast RX and sample circuitry and has a clock that happens to be “in phase” with Node 1. Therefore, this detection occurs almost immediately.

Node 3 (in slot 3) is also waiting to begin arbitration for fair access and is waiting for an arbitration reset gap to occur. It detects that Node 1 has released the bus at the reflected wave; moreover, it has very slow RX and sample circuitry and has a clock that is out of phase with Node 1. Therefore, this detection occurs at a time equal to the maximum sync delay ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods).

Node 2 now has counted four unasserted clock periods, but it does not know if there are other nodes that are waiting for a subaction gap to get access (for fair or urgent access). It knows that such a node would assert the bus after detecting a total of 20 unasserted clock periods. Moreover, this node could be one max sync delay out of phase ($T_{rt} +$

$RX_{pd} + FF_{su} + T_{clk}$), and that if it were to assert the bus, Node 2 might not detect this until some time later ($FF_{co} + TX_{pd} + T_{pd} + RX_{pd} + FF_{su}$). The sum of these two delays is: $79.345 \text{ ns} + 64 \text{ ns} = 143.345 \text{ ns}$, or eight clock periods.

Therefore, Node 2 has to count the first four clock periods, plus 16 additional clock periods (to complete the subaction gap), plus eight more clocks (to allow another node to detect the subaction gap and assert the bus) before Node 2 can sample the bus. This is a total of $4 + 16 + 8 = 28$ clock periods.

Node 2 now has counted 28 unasserted clock periods and wants to begin arbitration.

Node 3, however, is one max sync time ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods) behind Node 2 and has only counted 24 unasserted clock periods. If it is to join in the arbitration process, it has to be allowed to detect four more clock periods.

Consequently, Node 2 cannot immediately assert the bus after sampling 28 clock periods. If it asserts the bus, this action can and will be immediately detected by Node 3 (assuming that Node 3 is able to detect the assertion by Node 2 at the incident wave, and that this detection occurs immediately). In order to ensure that Node 3 has had enough time to detect that Node 1 has released the bus, Node 2 has to wait one max sync time ($T_{rt} + RX_{pd} + FF_{su} + T_{clk}$, or four clock periods) before it can assert the bus.

Therefore, in order for both Node 2 and Node 3 to have the opportunity to arbitrate, both nodes have to wait one sample time (28 clock periods) before sampling the bus, and then one hold time (four clock periods) before asserting the bus. This is a total of $28 + 4 = 32$ clock periods.

D.4 Backplane environment skew

Table D.7 gives the maximum allowable skew for various backplane technologies.

Table D.7—Backplane environment skew analysis

	TTL	BTL	ECL
Bit cell period	$1/(24.576 \text{ MHz} \pm 100 \text{ ppm})$ $\approx 40.690 \text{ ns}$	$1/(49.152 \text{ MHz} \pm 100 \text{ ppm})$ $\approx 20.345 \text{ ns}$	$1/(49.152 \text{ MHz} \pm 100 \text{ ppm})$ $\approx 20.345 \text{ ns}$
TX package skew	5.0 ns	3.0 ns	3.0 ns
Spatial skew	1.0 ns	0.5 ns	0.5 ns
Logic skew	1.690 ns	1.845 ns	1.845 ns
Total TX skew	7.690 ns	5.345 ns	5.345 ns
Backplane skew	7.0 ns	6.0 ns	6.0 ns
RX package skew	5.0 ns	3.0 ns	3.0 ns
Spatial skew	1.0 ns	0.5 ns	0.5 ns
RX setup/hold	5.0 ns	3.0 ns	3.0 ns
Total RX skew	11.0 ns	6.5 ns	6.5 ns
Margin	15.0 ns	2.5 ns	2.5 ns

This table can be used to compute the TX edge separation, the RX edge separation, and the skew margin for each technology using the following formulas:

$$\text{TX edge separation} = \text{bit cell period} - \text{total TX skew}$$

$\text{RX edge separation} = \text{TX edge separation} - \text{backplane skew}$

$\text{margin} = \text{RX edge separation} - \text{total RX skew}$

More simply:

$\text{margin} = (\text{bit cell period} - (\text{total TX skew} + \text{backplane skew} + \text{total RX skew}))$

The results of this analysis are reflected in table 5.4.

Annex E Cable operation and implementation examples

(Informative)

E.1 Timing formulas for cable environment gap control

When configuring the cable topology, there are three primary ways to increase Serial Bus performance. All methods are dependent on the topology implementor and the ability of the Bus Manager to optimize for that topology.

- a) Reduce the number of hops
- b) Put nodes of the same speed capability next to one another
- c) Optimize the gap_count for the Bus Manager for the current system configuration

Items a) and b) are only dependent on the topology implementor and are beyond the control of the bus management software. The third item is controlled by the bus management software. Setting the gap_count to its optimum value should be done to increase the efficiency of the Serial Bus. In the following paragraphs the purpose and value for the gap_count are explained.

The gap_count ensures that all nodes on the bus see the appropriate gap times. For example, in a cable topology where there are 16 hops from one end of the Serial Bus to the other, the signal propagation delay is significant. With a gap_count of 1, a subaction gap time is just $0.44 \mu\text{s}$ and an arb reset gap is $0.84 \mu\text{s}$. If the propagation delay is greater than the arb reset gap time minus the subaction gap time ($0.40 \mu\text{s}$), then the node that released the bus (beginning its gap timer) will see an arb reset gap before the node that is 16 hops away sees a subaction gap. This can cause the asynchronous bus band-width to be allocated unfairly. The following example demonstrates the importance of optimizing the gap_count and reducing the number of hops to increase the Serial Bus performance.

This example assumes the following:

- All timing constants and formulas are from tables 4.32, 4.33, and 4.34.
- S100 transfer rate
- Asynchronous transfer
- 512 bytes data, 24 bytes overhead
- Cable velocity of propagation of 5.05 ns/m as specified in 4.2.1.4.3
- Cable assemblies of 4.5 m as specified in 4.2.1.2.2

The last two assumptions result in a single-hop cable delay of

$$\text{Cable Delay} = 4.5 \text{ m} \cdot 0.00505 \mu\text{s} = 0.022725 \mu\text{s}$$

Sending of a packet can be divided into four phases:

- 1) Arbitration (Arb Phase)
- 2) Data transfer (Data Phase)
- 3) Acknowledgment (Ack Phase)
- 4) Between packet gap times (Gap Phase)

E.1.1 Arbitration phase

Arbitration delay is the delay that all nodes have to wait before starting arbitration. This value varies with the gap_count.

$$\text{arb_delay} = \text{gap_count} \cdot 4/\text{BASE_RATE}$$

Table E.1—Arbitration delays

gap_count	Minimum arb_delay (μ s)	Maximum arb_delay (μ s)
1	0.04068	0.04094
10	0.40686	0.40942
20	0.81371	0.81388
30	1.22057	1.22082
40	1.62743	1.62777
50	2.03429	2.03471
63	2.56321	2.56373

Speed signaling and the data prefix are done in parallel on the bus. ARB_SPEED_SIGNAL_START is the time that speed signaling leads the data prefix. In the minimum case, speed signaling is more time consuming than the data prefix time. From table 4.32:

$$-0.02 \mu\text{s} < \text{ARB_SPEED_SIGNAL_START}$$

$$0.10 \mu\text{s} < \text{SPEED_SIGNAL_LENGTH} < 0.12 \mu\text{s}$$

$$0.04 \mu\text{s} < \text{DATA_PREFIX_TIME} < 0.16 \mu\text{s}$$

MAX_DATA_PREFIX_DELAY is the maximum delay between the data prefix arriving at a node and the same data prefix being sent by that node. This delay is multiplied by the number of hops in the topology to find the worst-case system delay. The total arbitration phase time varies with the gap_count and the number of hops (N) in the bus topology. The following formulas were used to calculate this time:

Minimum Case:

$$\text{Arb Phase Time} = (\text{Cable Delay} \cdot N) + \text{arb_delay} + \text{SPEED_SIGNAL_LENGTH}$$

Maximum Case:

$$\text{Arb Phase Time} = (\text{Cable Delay} \cdot N) + \text{arb_delay} - (-\text{ARB_SPEED_SIGNAL_START}) + \text{DATA_PREFIX_TIME} + (N \cdot \text{MAX_DATA_PREFIX_DELAY})$$

Table E.2—Total arbitration phase time (μ s)

gap_count	Minimum	Maximum 1 hop	Maximum 8 hops	Maximum 16 hops
1	0.1434	1.8558	3.0119	4.3332
10	0.5096	2.2221	3.3782	4.6994
20	0.9164	2.6290	3.7851	5.1064
30	1.3233	3.0360	4.1921	5.5133
40	1.7302	3.4429	4.5990	5.9202
50	2.1370	3.8499	5.0060	6.3272
63	2.6659	4.3789	5.5350	6.8562

E.1.2 Data transfer phase

The packet transmission time is based on an asynchronous block write request with a data packet size of 512 bytes and 24 bytes of overhead.

Packet Transmission Time =

$$(512 + 24) \cdot 8 / \text{Base Rate}$$

giving

$$43.6153 \mu\text{s} < \text{Packet Transmission Time} < 43.6242 \mu\text{s}$$

MAX_PHY_DATA_DELAY is the maximum delay between the data arriving at a node and the same data being sent by that node. This delay is multiplied by the number of hops in the topology to find the worst-case system delay.

Data Transfer Phase Time =

$$(\text{Cable Delay} \cdot N) + \text{Pkt Transmission Time} + \text{MAX_PHY_DATA_DELAY} \cdot N$$

The results from this formula can be found in table E.3.

Table E.3—Total data transfer phase time

N = number of hops	Minimum (μs)	Maximum (μs)
1	43.638	43.789
8	43.797	44.942
16	43.979	46.260

E.1.3 Acknowledgment phase

The Acknowledgment phase consists of the acknowledge gap time, ACK_RESPONSE_TIME, and the acknowledge transmission time. The acknowledge gap time is the time between the end of a packet and the beginning of the acknowledge:

$$0.04 \mu\text{s} < \text{acknowledge_gap_time} < 0.05 \mu\text{s}$$

The ACK_RESPONSE_TIME is the time between reporting the end of a packet (DATA_END) and the acknowledging link layer requesting arbitration to send the acknowledgment.

$$0.05 \mu\text{s} < \text{ACK_RESPONSE_TIME} < 0.17 \mu\text{s}$$

There are 8 bits in an acknowledge, so the ack transmission time is 8 / BASE_RATE:

$$0.04 \mu\text{s} < \text{Ack Transmission Time} < 0.05 \mu\text{s}$$

The total acknowledge phase time is

Ack Phase Time =

$$\text{ack gap} + (\text{ACK_RESPONSE_TIME} - \text{acknowledge_gap_time}) + (\text{Cable Delay} \cdot N) + \text{Ack Transmission Time} + \text{MAX_PHY_DATA_DELAY} \cdot N$$

The results from this formula can be found in table E.4.

Table E.4—Total acknowledge phase time

N = number of hops	Minimum (μs)	Maximum (μs)
1	0.1541	0.4161
8	0.3132	1.5692
16	0.4950	2.8870

E.1.4 Between packet gap times

In this phase, the time required for the subaction and arb reset gaps is determined for various value of gap_count.

$$(27 + \text{gap_count} \cdot 16)/\text{BASE_RATE} \leq \text{subaction gap} \leq (29 + \text{gap_count} \cdot 16)/\text{BASE_RATE}$$

$$(51 + \text{gap_count} \cdot 32)/\text{BASE_RATE} \leq \text{arb reset gap} \leq (53 + \text{gap_count} \cdot 32)/\text{BASE_RATE}$$

The total time from the subaction gap through the arb reset gap for an asynchronous write request with 512 bytes of data and 24 bytes of header and the corresponding acknowledge is listed in table E.5. The total time was calculated using the following formula:

$$\text{Total time} = \text{Arb Phase} + \text{Data Transfer Phase} + \text{Ack Phase} + \text{subaction gap} + \text{arb reset gap}$$

Table E.5—Total time from subaction gap through arb reset gap

gap_count	Minimum (μs)	Maximum for 1 hop (μs)	Maximum for 8 hops (μs)	Maximum for 16 hops (μs)
1	45.21720	47.40620	51.02751	55.16615
10	50.18198	52.16743	55.78874	59.92738
20	55.26663	57.45768	61.07899	65.21763
30	60.55580	63.24211	66.36924	70.50788
40	65.84498	68.69709	71.65950	75.79813
50	71.13416	74.15206	76.94975	81.08839
63	78.01008	81.19412	83.82707	87.96571

Table E.5 illustrates the importance of setting the gap_count to an optimum value. To do this, it is important that the worst-case topology be realized for the number of hops in the system.

For example: A system has 16 hops, Node_1 is 16 hops away from Node_16, and Node_1 is root and cannot send another packet during this fairness interval. When Node_1 releases the bus (starting its gap timer), the gap propagates through the topology, reaching Node_16 some propagation delay time later (Prop1). Node_16 can arbitrate for the bus after a subaction gap plus arb_delay (note that the gap timer for Node 1 has advanced to subaction gap + arb delay + Prop1). The arbitration signal for Node 16 now propagates through the topology, reaching Node_1 some propagation delay time later (Prop2). The total delay time seen by Node_1 is (subaction gap + arb_delay + Prop1 + Prop2). The total delay time has to be less than the arbitration reset gap; otherwise, Node_1 can arbitrate for the bus and win. Therefore, the gap_count needs to be set to value where the (arb reset gap – subaction gap > Prop1 + Prop2).

Total propagation delay = Prop1 + Prop2 = $N \cdot 2 \cdot (\text{Cable Delay} + \text{MAX_PHY_DATA_DELAY})$

With the total propagation delay known, it is now possible to select the appropriate gap_count for the number of hops by using the following formula:

arb reset gap – subaction gap > total propagation delay

The resulting table of gap_count values is given in table E.6.

Table E.6—Calculated gap counts

Maximum number of hops	Total propagation delay (μs)	Gap_count
1	0.3295	1
2	0.6589	4
3	0.9884	6
4	1.3178	9
5	1.6473	12
6	1.9767	14
7	2.3062	17
8	2.6356	20
9	2.9651	23
10	3.2945	25
11	3.6240	28
12	3.9534	31
13	4.2829	33
14	4.6123	36
15	4.9418	39
16	5.2712	42

E.2 Cable environment jitter budget

Tables E.7 through E.9 give the jitter budget for the three cable PHY data rates. These can be used to compute the jitter margin for each data rate using the following formula:

(Bit cell time – (Data jitter + Strobe jitter + Skew)) = Margin

Table E.7—S100 jitter budget (ns)

	Data jitter	Strobe jitter	Skew
Transmitter skew			0.4
Transmitter jitter	0.80	0.80	
Cable reflections	0.13	0.13	
Cable intersymbol	0.1	0.1	
Cable delay mismatch			0.4
Channel margin	0.05	0.05	
Jitter at receive pins	1.08	1.08	0.8
Receiver offset	0.5	0.5	0.2
Receiver intersymbol and power supply rejection	0.5	0.5	
Flip flop setup and hold	1.0	1.0	
Total	3.08	3.08	1.0

The margin for the S100 rate is equal to $(10.17 - (3.08 + 3.08 + 1.0)) = 3.01$ ns.

Table E.8—S200 jitter budget (ns)

	Data jitter	Strobe jitter	Skew
Transmitter skew			0.15
Transmitter jitter	0.25	0.25	
Cable reflections	0.08	0.08	
Cable intersymbol	0.12	0.12	
Cable delay mismatch			0.4
Channel margin	0.05	0.05	
Jitter at receive pins	0.50	0.50	0.55
Receiver offset	0.25	0.25	0.1
Receiver intersymbol and power supply rejection	0.25	0.25	
Flip flop setup and hold	0.5	0.5	
Total	1.50	1.50	0.65

The margin for the S200 rate is equal to $(5.08 - (1.50 + 1.50 + 0.65)) = 1.48$ ns.

Table E.9—S400 jitter budget (ns)

	Data jitter	Strobe jitter	Skew
Transmitter skew			0.1
Transmitter jitter	0.15	0.15	
Cable reflections	0.035	0.035	
Cable intersymbol	0.13	0.13	
Cable delay mismatch			0.4
Channel margin	0	0	
Jitter at receive pins	0.315	0.315	0.50
Receiver offset	0.14	0.14	0.05
Receiver intersymbol and power supply rejection	0.1	0.1	
Flip flop setup and hold	0.2	0.2	
Total	0.755	0.755	0.55

The margin for the S400 rate is equal to $(2.54 - (0.755 + 0.755 + 0.565)) = 0.48$ ns.

E.3 Cable PHY configuration example

This subclause gives a detailed example of the three phases of configuring a cable environment: bus initialization, tree identify, and self-identify.

E.3.1 Bus initialize

Whenever a node joins the bus, a signal forces all nodes into a special state that clears all topology information and starts the next phase.

During this phase, the connection status of each port is latched internally by the PHY. If the connection status of any port changes after the bus initialization phase (i.e., an adjacent device is removed or added), the PHY will automatically drop back to the bus initialization phase. Nodes with no connected ports are isolated; nodes with a single connected port are “leaves”; nodes with two or more connected ports are “branches.” The eventual root (next section) may be either a branch or a leaf.

An example configuration after the bus initialization is complete is shown in figure E.1.

NOTE — In figures E.1 through E.18, the two arrows representing the physical connection between the nodes are just an abstract representation of line state signaling. They do not imply that the signaling uses different wire pairs for the two directions; in fact, both directions use both pairs, and the received signal is the result of the dominance of “1” over “0” over “Z.”

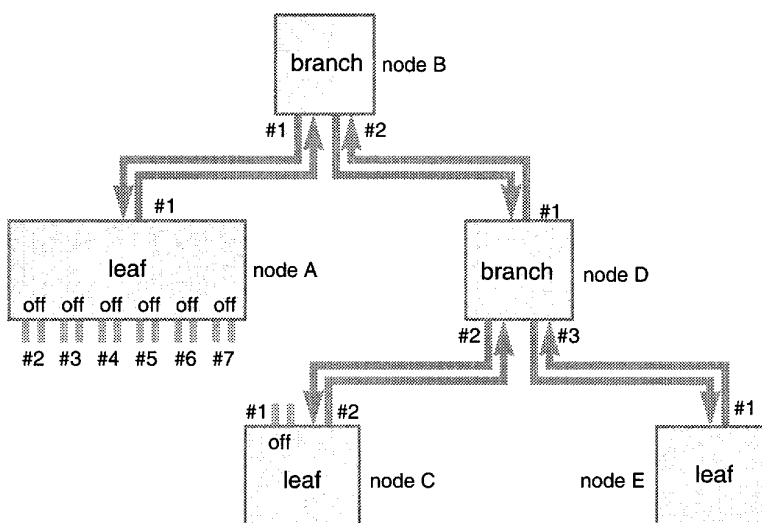


Figure E.1—Example cable state after bus initialize

Note that each port of the node is individually numbered. There is no particular order to the numbering. It is just a way to give each port a unique label.

E.3.2 Tree identify

After a bus initialize, the tree ID process translates the general network topology into a tree, where one node is designated a root and all of the physical connections have a direction associated with them pointing towards the root node. The direction is set by labeling each connected port as a “parent” (connected to a node closer to the root) or “child” port (connected to a node further from the root). Any unconnected ports are labeled “off” and do not participate in further arbitration processes. Any loop in the topology is detected by a timeout in the tree-ID process.

- a) The first step is for all leaf nodes (those with a single connected port) to notify their probable parents. In this example, nodes A, C, and E send a parent_notify to their single connected port. This is the start of the parent-child handshake process.

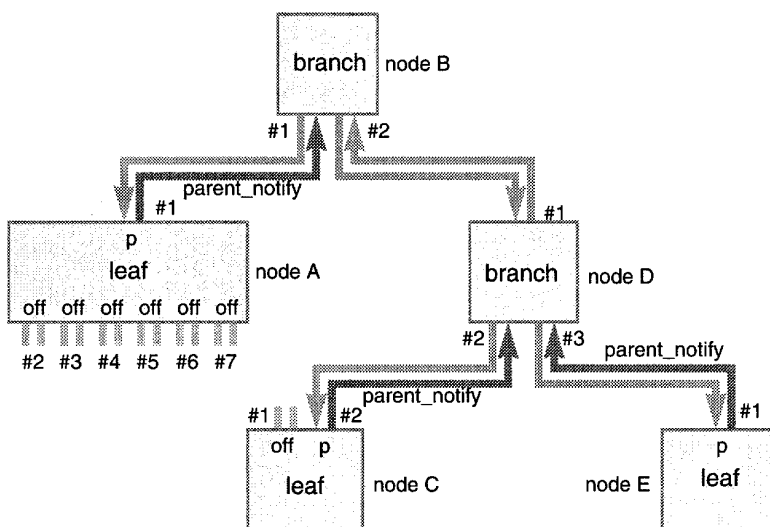


Figure E.2—Tree-ID start, beginning of child handshake

- b) At this point, nodes B and D internally recognize the parent_notify signals and mark the ports receiving parent_notify as child ports. Thus, B and D now each have one port remaining that is connected but not yet identified as child or parent. They each now send parent_notify up to their probable parents, and at the same time they send down child_notify signals to their child ports.
- If a node has been selected by software to be the root, this is the point of intervention. The selected node (i.e., the node with force_root set) will wait before sending the parent_notify signal for a 160 μs timeout period. Unless another node in the network also has its force_root set, this insures that the selected node will receive parent_notify on all its connected ports and thus mark them all child ports.

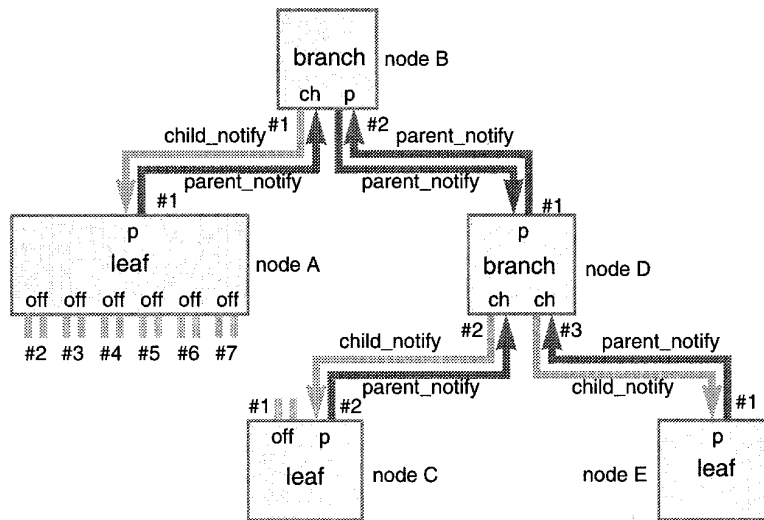


Figure E.3—End of child handshake

- c) When the leaf nodes receive the child_notify, they know that port is truly their parent port and that their part of the tree-ID process is complete. They then withdraw their parent_notify as a parent handshake. At the same time, both nodes B and D discover that they are receiving the parent_notify port from their proposed parent. Since one of the two nodes has to become the parent of the other, this collision of intentions starts a process called “root contention.” This starts with both nodes withdrawing their parent_notify signals, as shown in figure E.4.

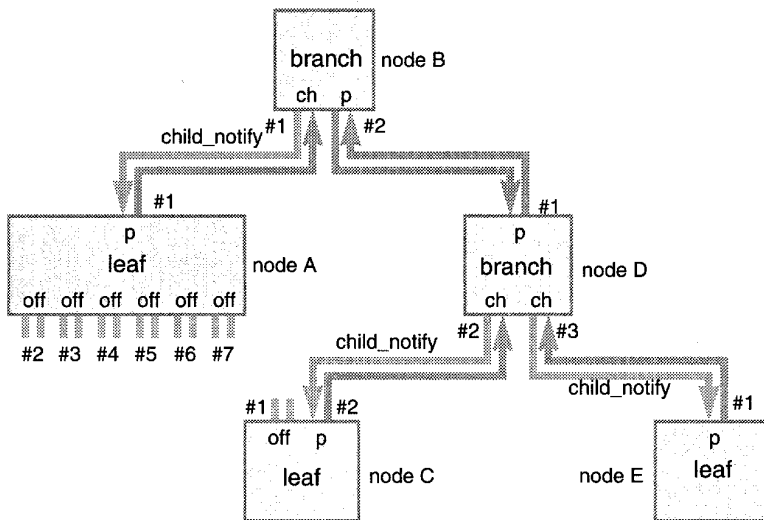


Figure E.4—Parent handshake and start of root contention

- d) Each competing node then starts a timer, with the time being one of two values chosen randomly. When the timer expires, each node checks the status of its contending port; the contending port will now either be idle or receiving a parent_notify signal. If the port is idle, it then sends the parent_notify signal to that port again. If the port is already receiving parent_notify, then the node will respond with child_notify.

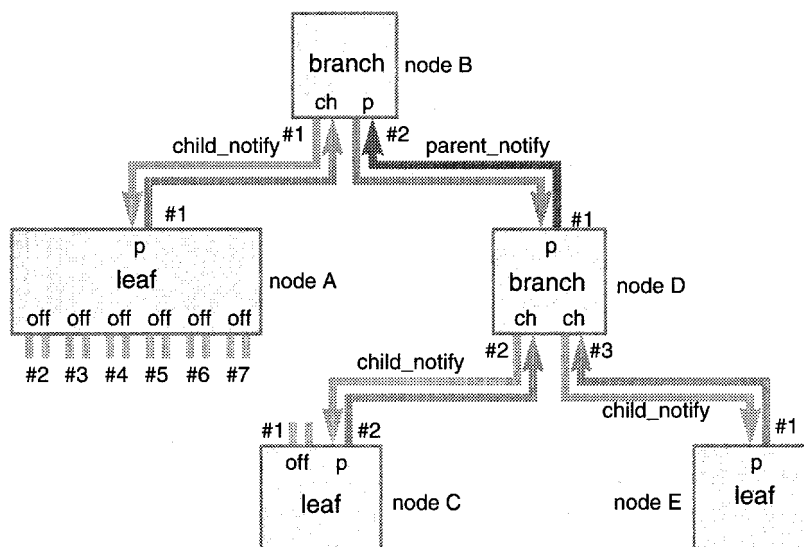


Figure E.5—End of root contention, and new child handshake start

If both nodes happen to pick the same timeout value (both long or both short), then the outcome may be another contention cycle; the two nodes can both see idle, and both again send the parent_notify signal to the unresolved port. The two dueling nodes may continue cycling through the contention state until eventually one times out sooner than the other.

In this example, Node D uses the shorter timeout while node B uses the longer timeout, so that Node D sends parent_notify while Node B is still waiting for its timer to expire.

- e) When the timer for node B expires, node B samples its proposed parent port and finds that it is receiving the parent_notify, so it labels that port as a child and ends the child handshake process (sending down the child_notify). In addition, since node B has labeled all ports as children, it takes the root function for itself.

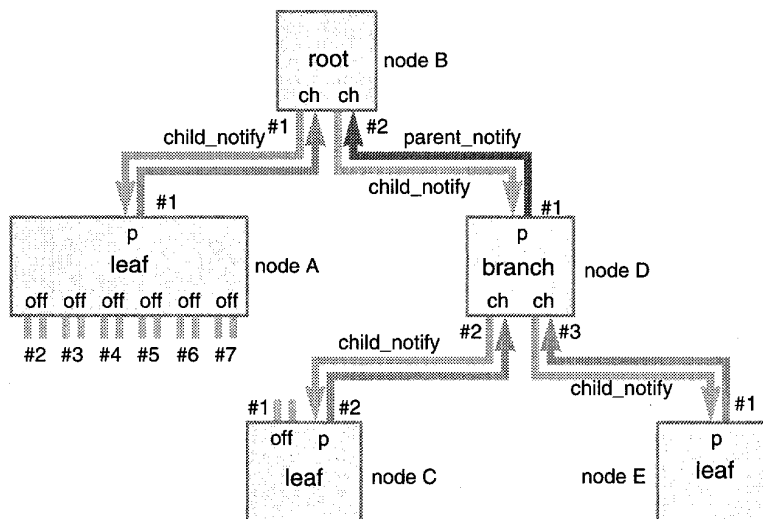


Figure E.6—End of final child handshake and root selection

- f) When node D receives the child_notify on its parent port, it drops the child_notify being sent to its children and the parent_notify to node B, ending its parent handshake.

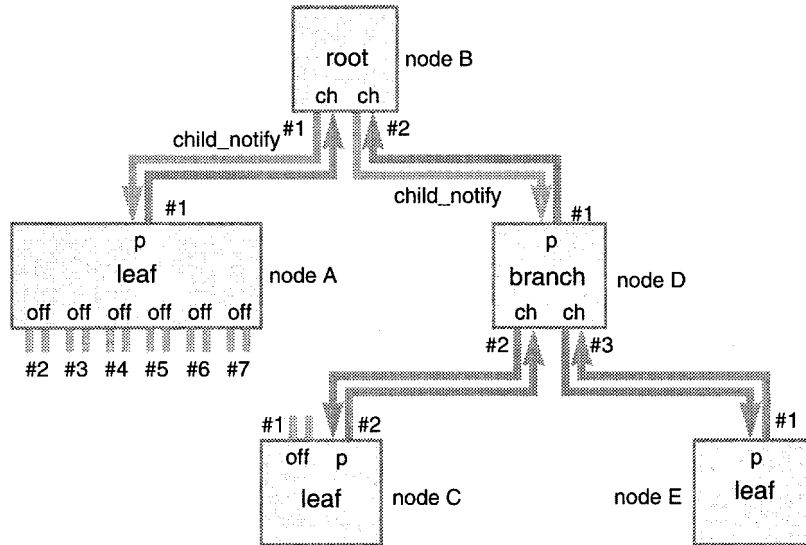


Figure E.7—Final parent handshake

- g) When node B, the root, stops receiving the parent_notify, it finishes the tree ID process by withdrawing the child notify signals sent to its children.

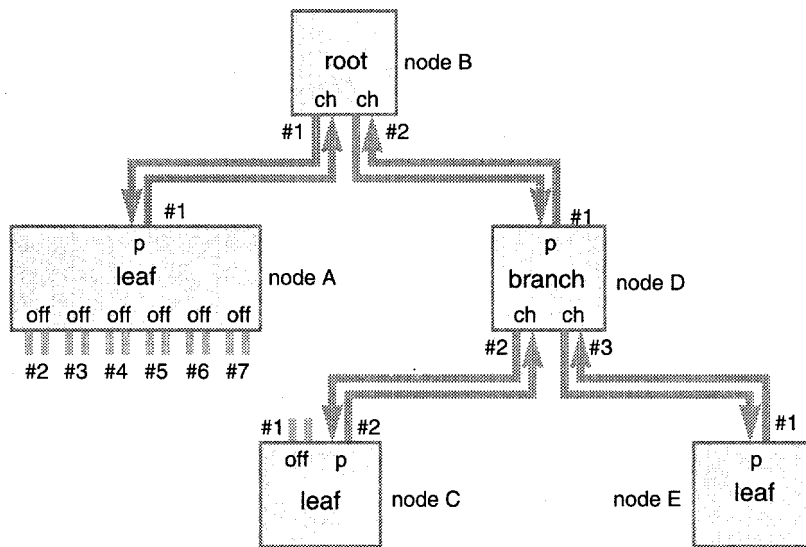


Figure E.8—Tree identify complete

Note that the selection of the root node is not topology dependent. It is completely acceptable that the root node also be a leaf. The only requirement is that the cycle master (the isochronous cycle master described in 3.6.4) also has to be the root, since the root has the highest natural priority.

The node that waits the longest after the bus reset to start participating in the tree identify process becomes the root. If, for example, node A in the previous example waited a long enough time to start the tree ID process, node B would end up sending the parent_notify to node A after step b). This would cause node A to become the root.

A particular node can be forced to wait a longer time by setting its force_root bit (see 4.3.8 and 4.3.4.3).

E.3.3 Self identify

The next step is to give each node an opportunity to select a unique physical_ID and identify itself to any management entity attached to the bus. This is needed to allow low-level power management and the building of a system topology map. Clause 7.3.5.2.1 discusses this process.

Sending the self-ID information is done by transmitting one to four 8-byte packets onto the cable that includes the physical ID and other management information as described in 4.3.4.1. The physical ID is simply the count of the number of times a node passes through the state of receiving self-ID information before having its own opportunity to do so, i.e., the first node sending self-ID packet(s) chooses zero as its physical ID, the second chooses one, and so on. Note that a node is not required to decode the self-ID packet; it merely has to count the number of self-identify sequences since the bus reset.

The management information included in the self-ID packet includes codes for the power needed to turn on the attached link layer, the state of the various ports (unconnected, connected to child, connected to parent), and data rate limitations.

The self-ID process uses a deterministic selection process, where the root node passes control of the media to the node attached to its lowest numbered connected port and waits for that node to signal that it and all of its children have identified themselves. The root then passes control to its next highest port and waits for that node to finish. When the nodes attached to all the ports of the root are finished, the root itself does a self identify. The child nodes use the same process in a recursive manner, as is illustrated in the following example.

- a) In this example, there are five nodes; one with a single port, two with two ports, one with three ports, and one with seven. In figure E.9, the system has just finished the tree-ID process. At this point, the “self_ID_count” values of all nodes are 0. The root starts the process by sending a grant to its lowest numbered unidentified port and a data_prefix to all other ports. Note that an unconnected port is automatically flagged as self-identified.

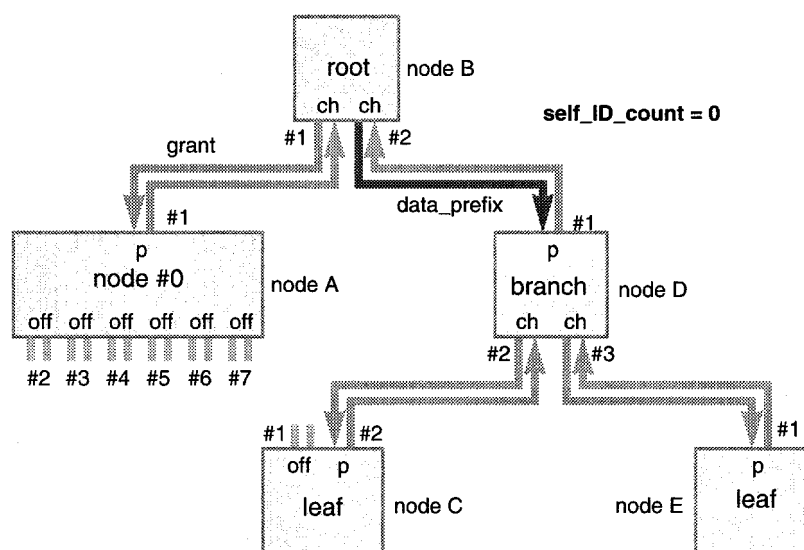


Figure E.9—Start of self-ID process

- b) When a node receives the grant, it propagates it to its lowest numbered unidentified child port or, if there is no requesting child, takes the current value of the self_ID_count as its node_ID and starts sending its self-identification information (node A in figure E.10). The start of this information is indicated by a data_prefix.

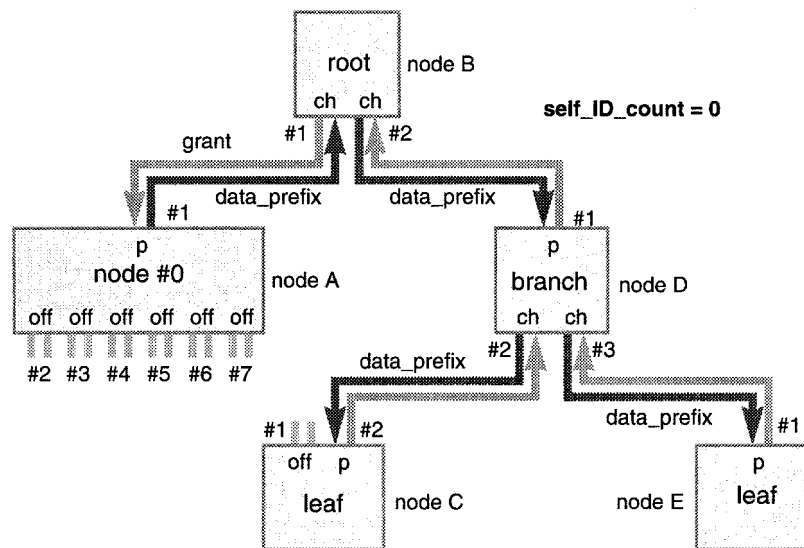


Figure E.10—First node sends self-ID information

When node B sees the data_prefix, it withdraws its grant. Meanwhile, node D has seen the data_prefix sent by node B, so it propagates the data_prefix to its children, nodes C and E. At this point the data prefixes are all pointed away from node A (figure E.10), so it can start sending the self-ID information. This is encoded in small 32-bit packets, with the first packet containing power requirements, performance capabilities, and the status of the first four ports. Since in this example node A has 7 ports, it needs to send a second self-ID packet to identify itself fully. The first packet is terminated with the data_prefix, which holds the bus until the second packet is sent. The second packet terminates normally with a data_end. All other nodes see the normal packet termination and use that event to increment their self_ID_count. Note that the bus-holding termination of the first ID packet does not cause the self_ID_count to be incremented. Note also that all self-ID packets are sent at the base rate (98.304 Mbit/s).

- c) When node A finishes sending its self-ID packets, it sends an ident_done to its parent. The parent (node B) flags that port as identified, sends a data_prefix to that port, and continues to send idle to its other ports. When node A sees data_prefix, it leaves self-ID mode and could start normal arbitration, but as long as the self-ID process continues, there will never be an idle period long enough for the PHY to request the bus. Nodes C, D, and E respond to the idle by incrementing self_ID_count.

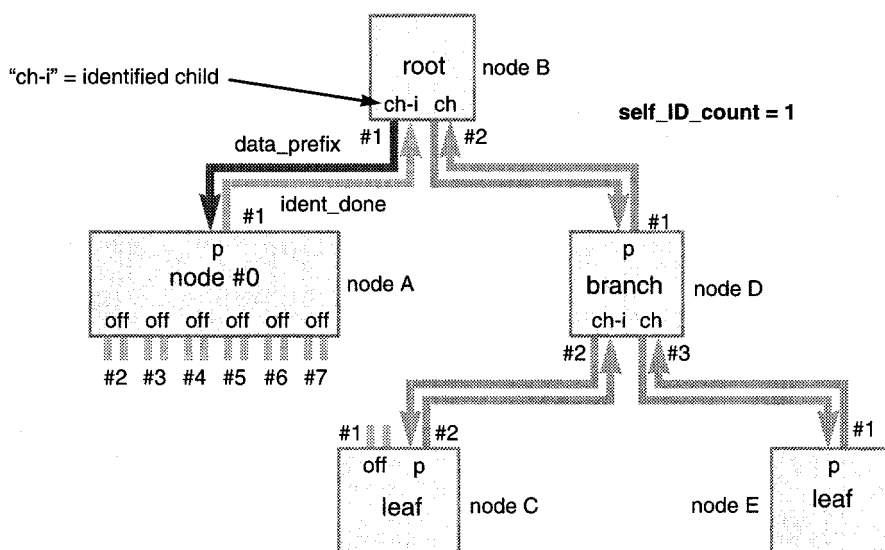


Figure E.11—First node finishes self-identify

- d) Node B, the root, then sends a grant to its lowest numbered unidentified child port, the one connected to node D. It also sends a data_prefix to its other ports (the one connected to node A).

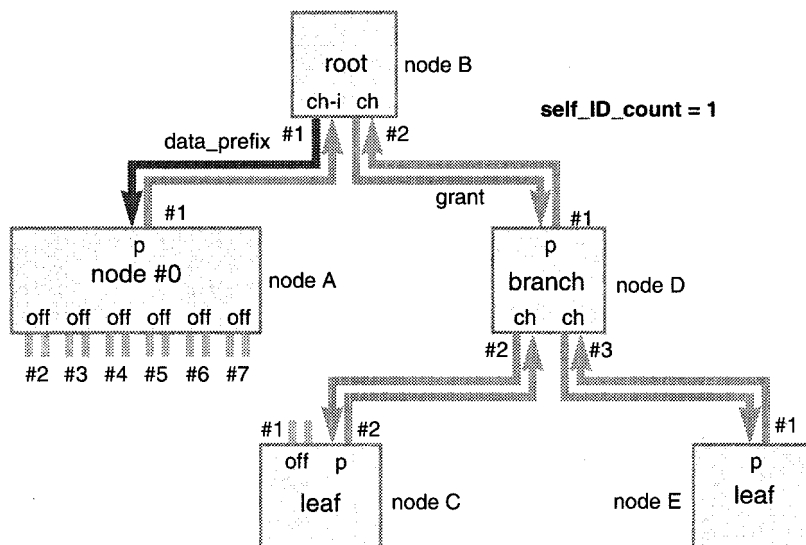


Figure E.12—Start of bus grant for second node self-identify

- e) Node D has unidentified child ports, so it passes the grant to its lowest numbered one (node C) and sends a data_prefix to the other child ports.

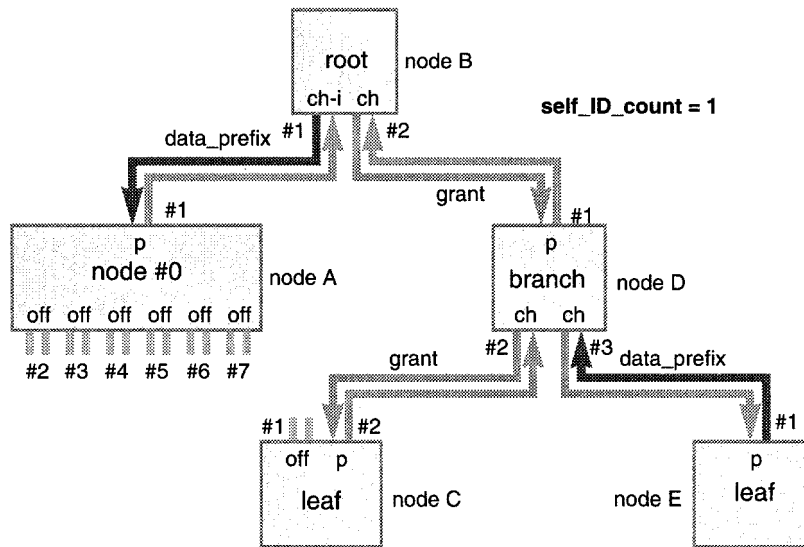


Figure E.13—Finish of bus grant for second node self-identify

- f) Node C does not have any unidentified children, so it responds by taking the value of self_ID_count as its node_ID, sending a data_prefix and a single self-ID packet as shown in figure E.14. Only a single self-ID packet is needed since node C only has a pair of ports. As the other participating nodes see the normal termination line state at the end of the self-ID packet, they all increment their self-id counters. Node A has already left the self-ID process; it sees all successive self-ID packets as normal receive packets.

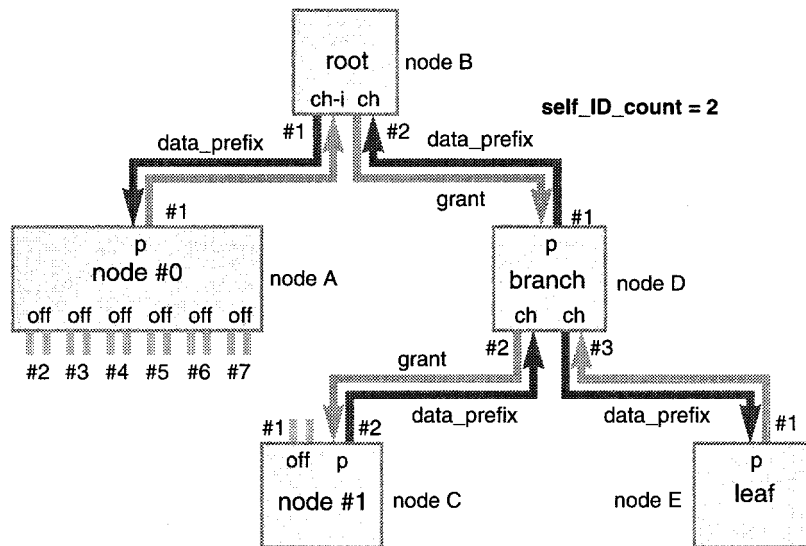


Figure E.14—Second node self-identify

- g) Node C then sends an ident_done to its parent (node D). Node D responds by labelling that port as identified, sending data_prefix on the newly identified port, and continuing to send idle on each of its other ports.

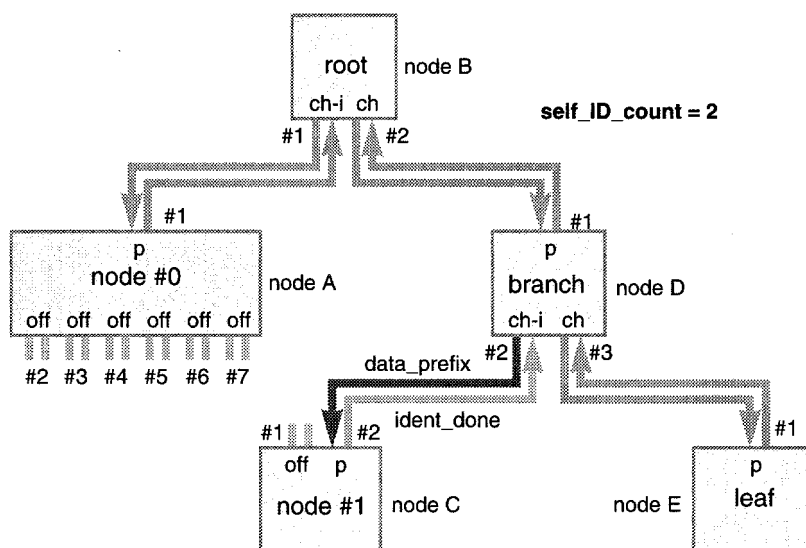


Figure E.15—Second node finishes self-identify

- h) When node B (the root) receives the idle, it sends a grant to its lowest numbered unidentified child port, the one connected to node D. It also sends a data_prefix to its other ports. This is the same process as step d) since node D has not yet signaled self-ID completion.

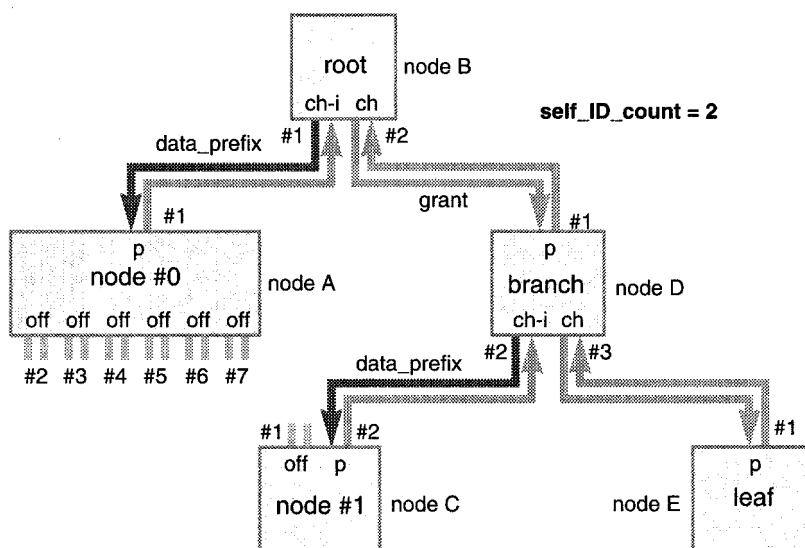


Figure E.16—Start of grant for third self-identify

- i) When node D gets the grant, it propagates it to its lowest numbered unidentified child port (#3), which is connected to node E (figure E.17). Node E then gets its opportunity to send self-ID information. When it is finished, it will signal node D, which will label its port #3 as identified. Node B will send a grant down its port #2 a third time, which will finally allow node D to send its self-identify information, since all its child ports are now labeled as identified. When finished, node D will send the ident_done to node B, the root.

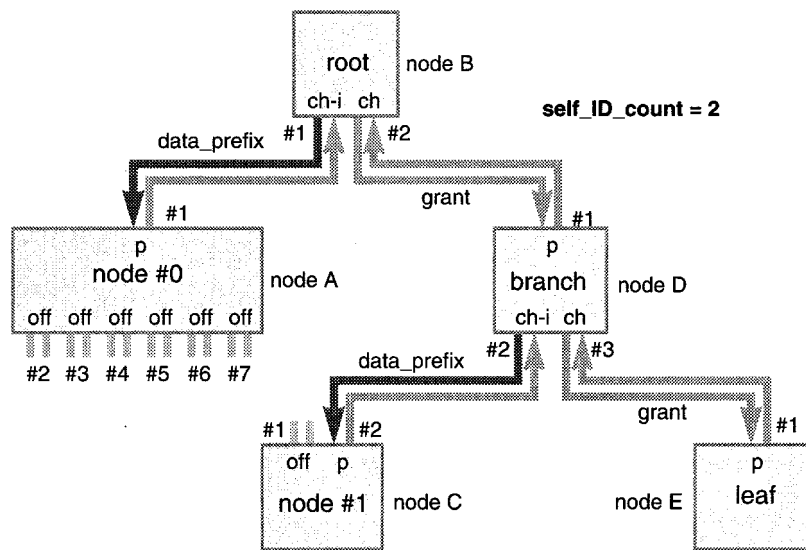


Figure E.17—Completion of grant for third self-identify

- j) The root will be the last node to send its self-ID information, and when it is finished, it leaves the self_ID process itself and sends idle out to all its child ports. All nodes exit the self_ID process with their fair bits cleared, so the bus will generally stay idle for the duration of an arbitration reset gap, unless an overeager node initiates a higher priority bus request. At this point (figure E.18), each node has a unique node number and the self-ID information has been broadcast.

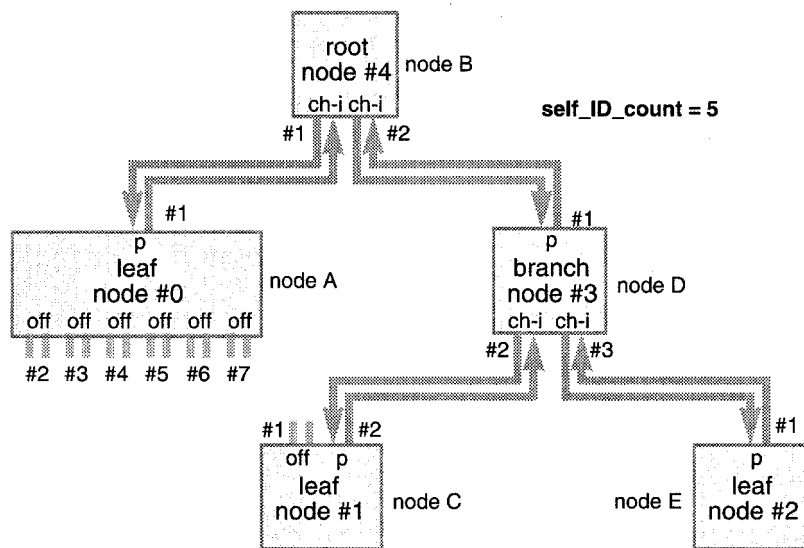


Figure E.18—Example cable state after self-identify phase

E.3.4 Topology construction

A higher level software entity can deduce the network topology from the self-ID packets. Each port is assigned two bits of the self-ID packet; the significance of those bits is given in table E.10.

Table E.10—Port status encoding

Port status	Meaning
11	child port
10	parent port
01	unconnected port (unconn)
00	unimplemented port - i.e., port 4 of a 3 port device (noport)

Note that whether a port is “implemented” or not is a function of the PHY. For example, if a card designer provides only two connectors for a PHY IC that has three ports, then three ports will show up in the topology map, even though the user has no way of using the connectorless port.

A node with more than four ports has to append a second self-ID packet; a node with more than 13 ports has to append a third; and so on.

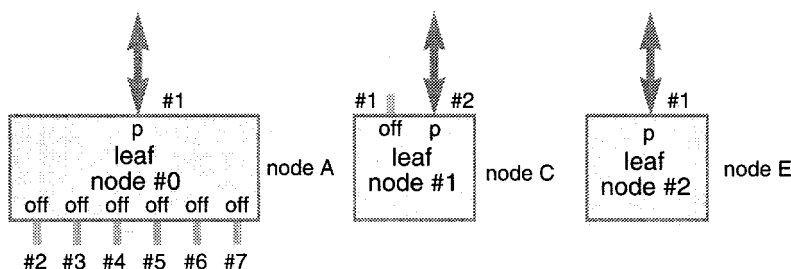
Reflecting back on the example network in the previous subclause, the pertinent data from the self-ID packets is given in table E.11.

Table E.11—Example self-ID packet port status values

phy_ID	port 1	port 2	port 3	port 4	port 5	port 6	port 7	ports 8:16
00	parent	unconn	unconn	unconn	unconn	unconn	unconn	noport
01	unconn	parent	noport	-	-	-	-	-
02	parent	noport	noport	-	-	-	-	-
03	parent	child	child	-	-	-	-	-
04	child	child	noport	-	-	-	-	-

- The node A (phy_ID 0) packet represents a seven-port PHY, with a parent connection on port 1 and no other connections. Thus, it is a leaf node.
- The node C (phy_ID 1) packet represents a two-port PHY, with a parent connection on port 2 and no other connections. This is also a leaf node.
- The node E (phy_ID 2) packet represents a single-port PHY, which by definition must be a leaf node. Its sole connection is a parent connection.

At this point, any node receiving the self-ID packets will have complete information on the three leaves, nodes A, D, and E, as shown in figure E.19.

**Figure E.19—Topology information after identification of leaf nodes**

- d) The node C (phy_ID 3) packet represents a three-port node with all three ports connected. Port 1 is the parent port; ports 2 and 3 are child ports. Which of the previous branches and/or leaves connect to these child ports? Remember that during the self-id process, a node receiving a bus grant passes the grant down to its highest numbered child port last. So port 3 must connect to node E (phy_ID 2), and port 2 must connect to node D (phy_ID 1). This is shown in figure E.20.

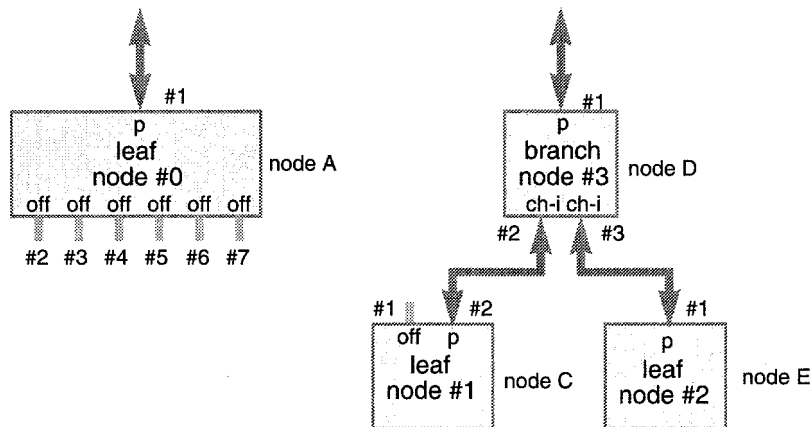


Figure E.20—Topology information after identification of first branch node

In general, as the reconstruction process proceeds, there will always be a supply of branches and/or leaves. These pieces can be ranked according to the node-IDs of the node having the “open” parent port (“open” meaning not yet reconstructed by the topology mapping software). The highest ranking branch and/or leaf, i.e., the one with the highest node-ID numbered parent port, is the one that must be connected to the highest numbered next-available child port.

- e) The node B (phy_ID 4) packet represents a two-port PHY with both ports being child ports. So this must be the root, and it must be the last self-ID packet. Again, the higher numbered port must be connected to higher ranked branch and/or leaf; port 2 must connect to node C (phy_ID 3), and port 1 must connect to node A (phy_ID 0). The final topology would look like figure E.21.

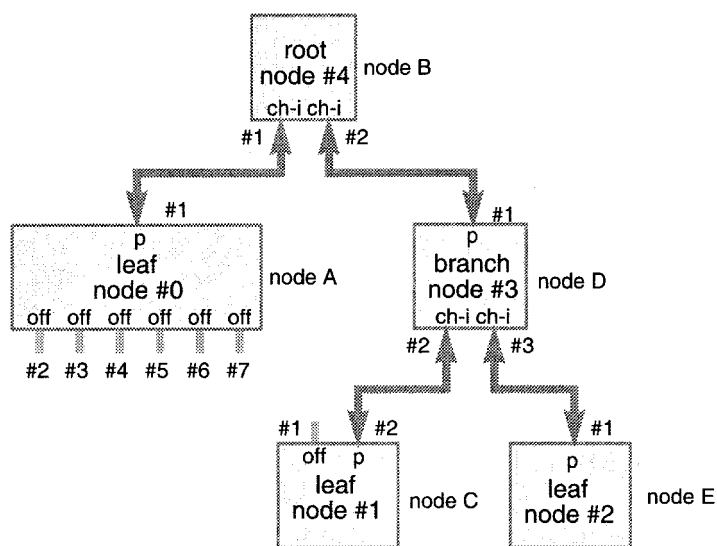


Figure E.21—Topology information after identification of root node

Annex F Backplane physical implementation example

(Informative)

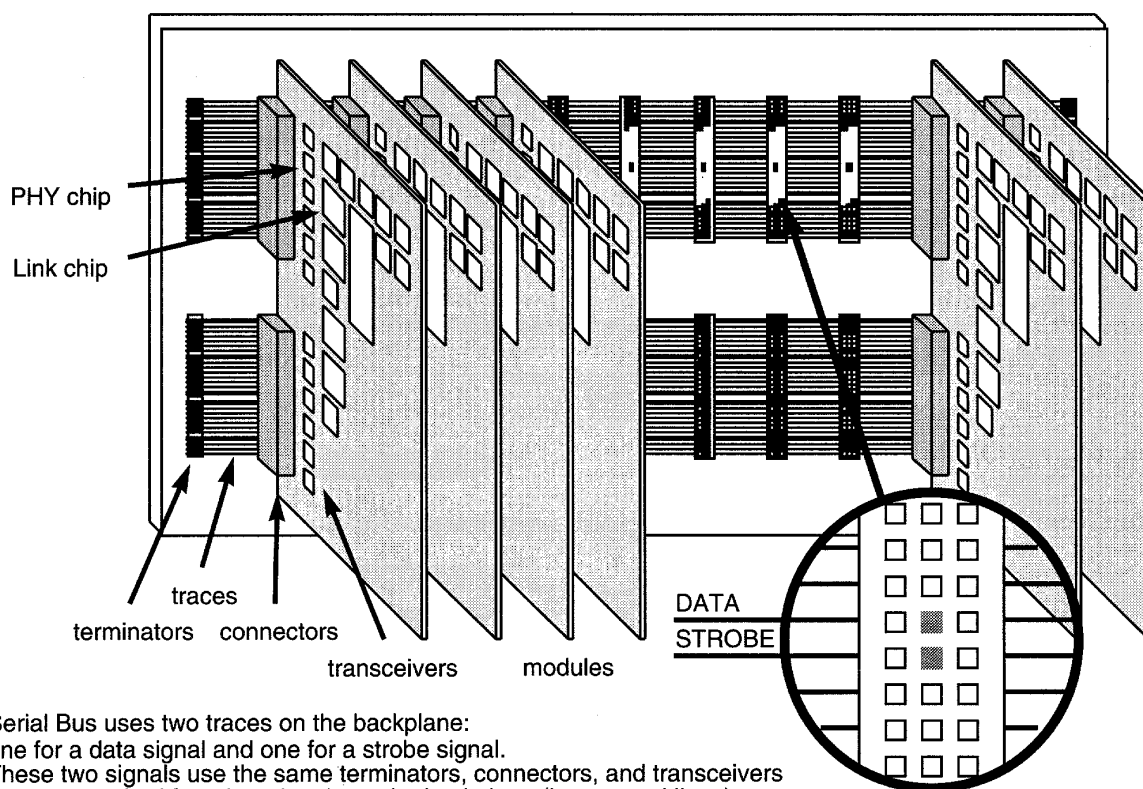
This annex describes how the physical layer of the Serial Bus may be implemented in the backplane environment. These descriptions illustrate *possible* implementations of the Serial Bus within given application environments, and are intended to be informative.

Clause F.1 addresses the media signal interface between the parallel backplane and Serial Bus. This description includes signal and pin assignments that may be used to implement Serial Bus on certain standard backplanes.

Clause F.2 describes an interface between the Serial Bus PHY and link layers, as well as the logical components within the PHY layer. It is intended as a guide for the design of the logic required to implement the PHY layer.

F.1 Standardized parallel bus implementations

An example of a Serial Bus implementation within the backplane environment is shown in figure F.1. In this example, the two signals used for Serial Bus accompany a number of signals that make up a standardized parallel backplane bus.



Serial Bus uses two traces on the backplane: one for a data signal and one for a strobe signal. These two signals use the same terminators, connectors, and transceivers that are required for other signals on the backplane (i.e., control lines).

Figure F.1—Typical application of Serial Bus in the backplane environment

The media signal interface of Serial Bus is similar to that of the parallel bus. Transceivers, terminations, connectors, and other components of the media signal interface are the same as those used for the control lines of the host parallel

bus. The host bus provides the requirements for signal transmission, mechanical arrangement of modules, and the environmental conditions over which operation of the bus is guaranteed.

In order to minimize the single points of failure between the two buses in this example, the Serial Bus is kept functionally separate (and logically distinct) from the host bus. In such an implementation, a Serial Bus node may have a different address than a node on the host bus even though they exist on the same physical module. Also, nodes on the Serial Bus may not respond to the system reset or bus initialize signals of the host. Nonetheless, design requirements may make it advantageous to have the Serial Bus node directly access the CSRs of the host bus, and vice versa.

Serial Bus may be used to interface to an IEEE 1149.1 (JTAG) test access port or other test control logic on a module. This implementation of Serial Bus as a test and maintenance (TM) bus provides a path for failure data to be captured via a module stress/error history log. It can also be used to initiate extended diagnostics, either internally or from a remote source, and can be used to download the stress/error log to facilitate repair. As a result, faults on the host bus can be isolated to the module or the bus level.

Table F.1 describes how Serial Bus *may* be implemented on certain backplanes. Although this table only addresses IEEE 896 (Futurebus+) and ANSI VME64, other potential applications include (but are not limited to) IEEE 960 (Fastbus), IEEE 1196 (NuBus[®]), and IEEE 1296 (MULTIBUS II). Each of these parallel buses reserves two backplane signals for the implementation of a serial bus. Table F.1 assigns these lines to the two signals used for Serial Bus: DATA and STRB.

Table F.1—Serial Bus signal mapping and pin assignment

Serial Bus	Futurebus+ Profiles A, B, F, M, and T		VME64	
	Signal	Pin assignment	Signal	Pin assignment
DATA	SB0	B a 15	SERB	P1/J1 b22
STRB	SB1	B c 15	SERA	P1/J1 b21
NOTE — Futurebus+ Profile M uses this pin assignment for MIL-12SU modules. MIL-10SU and MIL-Format E modules assign SB0 to “C a 39” and SB 1 to “C d 39.”				

It should be noted that at the time this standard was approved, Futurebus+ profiles M and T require that SB0 and SB1 be reserved for the implementation of IEEE 1394 Serial Bus. Futurebus+ profiles A, B, and F reserve these Dines for an optional serial bus, but do not require IEEE 1394 Serial Bus specifically (i.e., they merely indicate a preference for IEEE 1394 Serial Bus). VME64 reserves the SERA and SERB lines (formerly referred to as SERCLK and SERDAT*, respectively, in the IEEE 1014 VME Specification) for a serial bus, but it does not require IEEE 1394 Serial Bus specifically.

F.2 Physical layer implementation

F.2.1 PHY layer overview

The Serial Bus backplane PHY layer has three primary functions: transmission and reception of packets, arbitration, and provision for an electrical/mechanical interface. Figure F.2 describes the interaction between the PHY layer, the link layer, and other components within a node implementing Serial Bus. These components are described in the following subclauses. Functions of the link layer are performed by “link logic.” Functions of the PHY layer (except for the clock and transceivers) are performed by “PHY logic.”

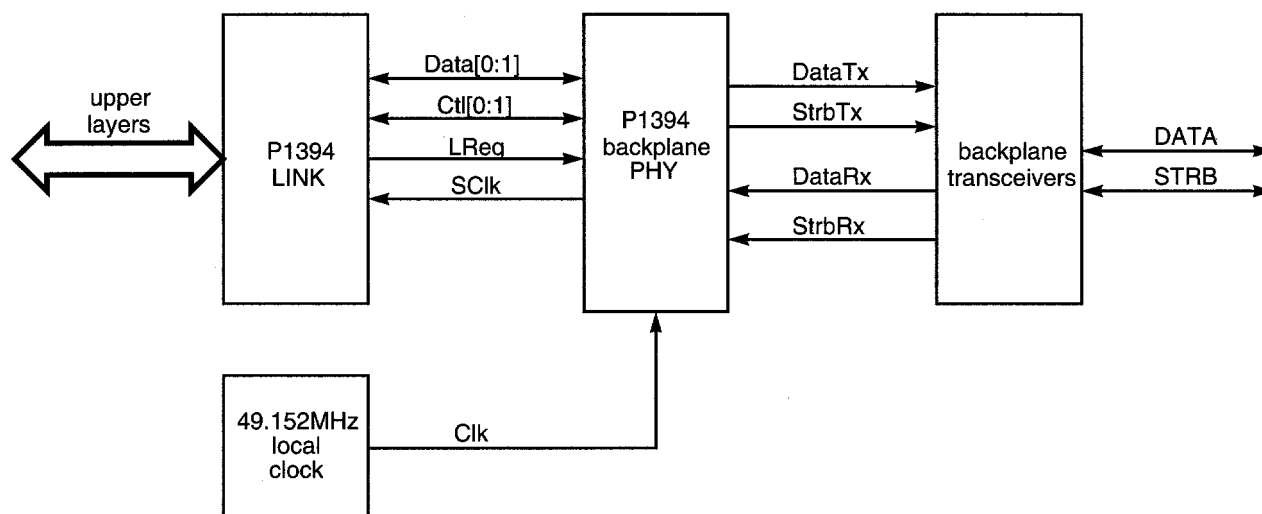


Figure F.2—Link/PHY interface

F.2.1.1 Serial Bus PHY logic

The transceivers and the bus to which they are connected make up the electrical/mechanical interface of the Serial Bus. Other functions of the PHY layer (e.g., arbitration and packet transmission/reception) are performed within the backplane PHY logic and are described in F.2.2.

F.2.1.2 Serial Bus link logic

The Serial Bus link layer provides acknowledged one-way data transfer services. It responds to read/write/lock requests from the upper layers of a Serial Bus node, and it prepares packets for transmission through the PHY layer and onto the Serial Bus. The link layer also responds to changes in the state of the Serial Bus (i.e., received data) as indicated by the PHY logic. The functions of the link logic include addressing, error checking, and data framing (i.e., within a packet).

F.2.1.3 Backplane transceivers

Transceivers may be packaged separately from the PHY logic so that transceivers implementing different technologies can be used. Bidirectional signals, DATA and STRB, go to and from the transceivers and into the PHY logic as unidirectional signals.

The type of transceiver that is used determines the packet transmission rate on the serial bus. Transceivers using BTL or ECL transmit and receive at 49.152 MB/s. Transceivers using Enhanced Transceiver Logic (ETL) operate at 24.576 MB/s.

F.2.1.4 Local clock

A local clock (Clk) is used for the synchronization of state machines within the PHY logic. The frequency of this clock is 49.152 MHz (± 100 ppm), regardless if data is transmitted at 49.152 MB/s or 24.576 MB/s.

F.2.2 High-level PHY logic description

Figure F.3 provides a high-level description of the backplane PHY logic. The components within this diagram are described in the following subclauses.

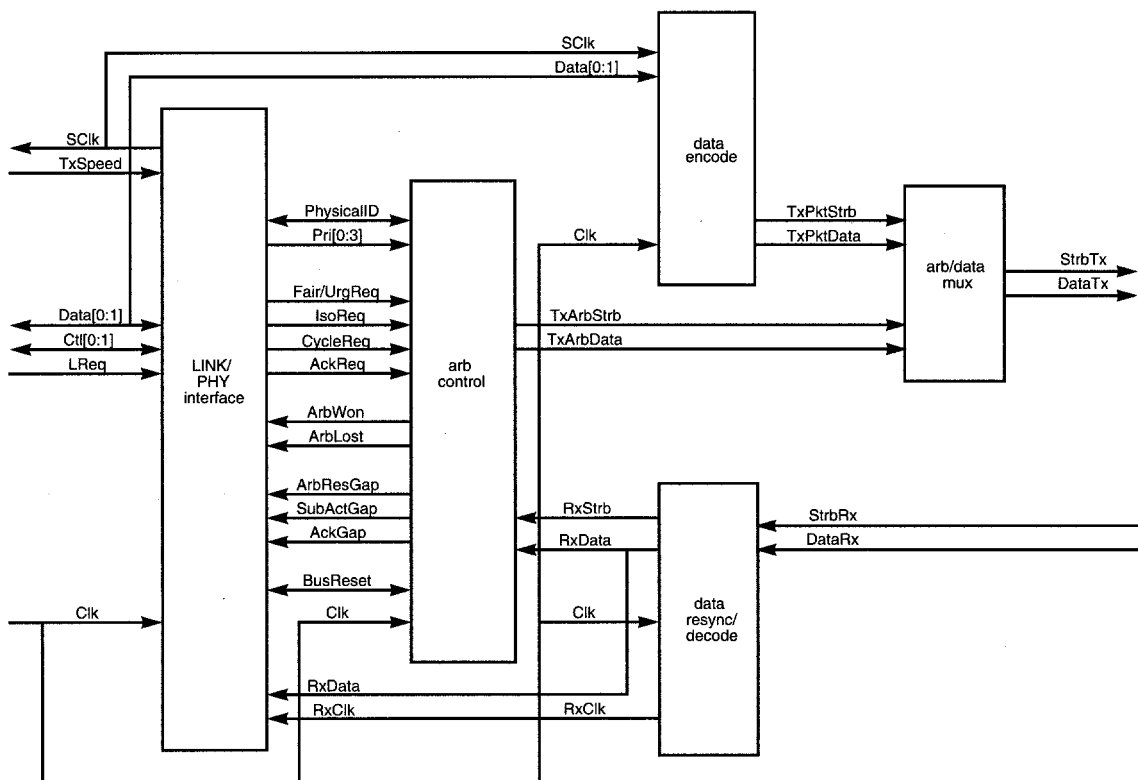


Figure F.3—PHY logic block diagram

F.2.2.1 LINK/PHY interface controller

This module provides much on the interface between the backplane PHY and the upper layers within the node, particularly the link layer. The protocol used in this interface is defined within annex J. Information is transmitted bidirectionally between the link and the PHY on Data[0:1] and Ctl[0:1]. In addition, LReq is used to transmit requests unidirectionally from the link to the PHY.

There are four basic operations that can occur between the link and the PHY: request, status, transmit, and receive. All but requests are initiated by the PHY. The link uses the request operation to read or write an internal PHY register or to request the PHY to initiate a transmit action. The PHY initiates a receive action whenever a packet is received from the serial bus. The PHY initiates a status action when the status of the serial bus changes.

To request the bus or access a PHY register, the link sends a short serial stream to the PHY on the LReq line. The information sent includes the type of request (read/write register or transmit packet). If the request is to read or write a register within the PHY, this serial stream also includes the address of the register and the data to be written to the register (for a write action). If the request is to transmit a packet, this serial stream also includes the type (Fair, Urgent, Isochronous, Cycle Master, or Acknowledge) and priority of the request (a 4-bit field used for Urgent requests). The format of this serial bit stream is described within annex J.

The data and control buses are both 2-bit bidirectional buses. The control bus (Ctl[0:1]) indicates the status of the data bus, as it is defined in annex J. The data bus (Data[0: 1]) is used to transmit packet data between the link and the PHY, or to transmit register information from the PHY to the link (allowing access to registers that are used within the PHY layer). The interface module regulates the transfer rate of these buses using Serial Clock (SCLK) indications.

For packet data transfers, the rate at which SCLK indications are given to the link layer determines the transmission rate on the serial bus. Since PHY-link transfers occur two bits at a time, SCLK is used to clock the transfer rate at one-half of the bus data rate. Therefore, for backplanes with a bus data rate of 49.152 MHz, SCLK would be used to clock PHY-link transfers at 24.756 MHz. For backplanes with a bus data rate of 24.756 MHz, SCLK would be used to clock PHY-link transfers at 12.288 MHz. The state of TxSpeed could be used to determine the rate of SCLK, allowing the PHY layer to be configured for a particular transmission rate.)

F.2.2.2 Arbitration controller

This module controls access onto the serial bus. It performs the arbitration process and enables packet transmission onto the serial bus.

Requests for a transmit action on the serial bus are received from the PHY-link interface. These requests are received on one of the following lines: Fair/UrgReq, IsoReq, CycleReq, or AckReq. When a request is received on Fair/UrgReq, the type of request, as well as the priority, is indicated by the status of the priority lines, Pri[0:3]. If the status of the priority lines is all zeros, then the request is fair. If the priority lines are not all zero, then these lines indicate the priority level of an urgent transfer. The priority level of all ones (the highest priority) is reserved for cycle start requests, which are requested with CycleReq. Requests on IsoReq and AckReq indicate requests for isochronous transfers or acknowledges, respectively.

When one of these requests occur, arbitration begins after an appropriate “gap time.” Different requests allow arbitration to begin after different “gap times” have occurred (e.g., an ArbResGap, a SubactGap, or an AckGap), allowing nodes with particular requests to begin arbitrating for the bus before other nodes. When a request for a transmit action is received, the controller samples the bus (RxStrb and RxData) in order to determine if the bus is idle and when the appropriate time to begin the arbitration sequence will be. The arbitration controller then asserts ArbStrb while transmitting the arbitration sequence on ArbData. The arbitration sequence incorporates the physical_ID and, if appropriate, the priority level indicated by Pri[0:3].

Timing for transmission and sampling of the arbitration sequence during the arbitration process is also controlled by the arbitration controller. Once a bit of the arbitration sequence has been put on the bus for a certain amount of time, the controller samples the bus to determine if another node with a greater arbitration sequence is also arbitrating for the bus. If so, the controller backs out of the arbitration contest until the next opportunity. If the arbitration controller is able to complete its entire arbitration sequence successfully, it controls the bus. The controller asserts ArbWon or ArbLost to indicate the outcome of the contest.

Once the arbitration controller successfully completes an arbitration contest, it immediately asserts the data_prefix symbol on DataTx and StrbTx. After data_prefix is transmitted for a certain amount of time (see 5.4.2.1), the arbitration controller asserts ArbWon. The data_prefix symbol is transmitted long enough so that the data resync/decode modules on other nodes can distinguish the arbitration sequence from the beginning of a data packet.

F.2.2.3 Data encode

The data encode module encodes the bits to be transmitted, one at a time, using a Data/Strobe encoding mechanism. Data propagates from the data encode module as DataTx. It is accompanied by a strobe signal, StrbTx. The data encode module causes the strobe signal to change state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on one of the two signals.

At the end of a packet, the data encode module receives a data_end symbol from the PHY-link interface controller. This causes the data encode module to assert the data_end symbol on DataTx and StrbTx for a certain amount of time

(see 5.4.2.1) before releasing the bus. The transitions used to release the bus are described in 5.4.2.1. The receipt of a data end symbol also indicates that SCLK shall no longer be generated.

F.2.2.4 Arb/data multiplexer

This module receives the DataTx and StrbTx signals from the data encode module and the ArbData and ArbStrb signals from the arbitration controller module. The arb/data mux module essentially combines the output from these other modules, allowing either module to assert the backplane.

F.2.2.5 Data resync/decode

Once DataRx and StrbRx signals are received from the bus, they are synchronized to the local clock and decoded by the data resync/decode module. As a packet is received, a clock that transitions every bit period can be extracted by performing an exclusive OR of DataRx and StrbRx. The rationale for using this “Data-Strobe” bit-level encoding mechanism is to improve the transmission characteristics of data packets across the serial bus. In particular, the code ensures that transitions occurring on DataRx and StrbRx are approximately one bit period apart. This results in an increase in skew tolerance that could not be obtained with a clocked NRZ format.

Annex G Backplane isochronous resource manager selection

(Informative)

G.1 Backplane configuration management

In a basic backplane application, an isochronous resource manager (IRM) is not required. Upon power-up, all nodes on the bus are ready to arbitrate for asynchronous transfers. If isochronous services are to be supported, it is necessary (in most cases) that a node on the bus perform the function of IRM. It is also necessary that some node perform the function of cycle master.

G.2 Isochronous resource manager selection process

Although an example of an IRM selection process is described in G.3, a particular selection process is not specified within this document. The IRM selection process is to be defined by the application environment.

NOTE — One example for this reasoning is as follows: a particular parallel bus that supports Serial Bus as a test/maintenance bus may require all of the Serial Bus management functions to exist on the module that is responsible for the monarch functions on the parallel bus. Yet another parallel bus that supports Serial Bus as an alternate access path may require, for survivability reasons, that a module shall never perform management functions for both the Serial Bus and the host bus.

It is recommended that the selection mechanism support the following two criteria: that there can only be one IRM and that the selection process should be deterministic (i.e., repeatable, for debugging purposes).

G.3 Example of a isochronous resource manager selection process

This subclause describes examples of processes that *may* be used to select the IRM on a Serial Bus in the backplane environment. These examples are intended to be informative. The first assumes that an IRM is to be selected from one or more nodes on a bus. The second assumes that an IRM is not to be designated at all.

G.3.1 IRM-capable node environment

This example designates one node to be the IRM and establishes this node as the “well-known” address of the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers. It requires that such a node be capable of functioning as cycle master.

- a) After a reset event, all nodes that have the *irmc* (isochronous resource manager capable) and the *cmc* (cycle master capable) bits set within their Bus_info_Block shall set their BUS_MANAGER_ID.*bus_mgr_id* field to $3F_{16}$.
- b) Each of these nodes shall examine the *irmc* and *cmc* bits within the Bus_info_Block of other nodes on the bus, beginning at the location with the largest physical_ID (this may be 62 if the largest “occupied” location is not known).
- c) If a node is present at a location that has a physical_ID larger than that of the requesting node, and if the *irmc* and *cmc* (cycle master capable) bits are set within the Bus_Info_Block of this node, then the requesting node shall recognize this node to be the IRM and exit from the IRM selection process.
- d) If the physical_ID of the addressed node is equal to that of the requesting node, then the node recognizes itself to be the IRM and sets the BUS_MANAGER_ID.*bus_mgr_id* field to its own physical_ID.

Note that in a fault-tolerant environment, a mechanism has to exist to exclude a failed IRM-capable node from the selection process.

G.3.2 Non-IRM environment

It is possible that an environment may allow for isochronous resources to be allocated without the use of an IRM. In such a case, each node on the bus may have “a priori” knowledge of its own isochronous bandwidth and channel allocations.

NOTE — Such a scenario would still require the presence of a cycle master (whose identity could be “pre-ordained” as well).

Annex H Serial Bus configuration in the cable environment

(Informative)

This annex provides examples of typical Serial Bus configuration procedures. The operations described in this annex are intended to assist the reader in understanding the formal definitions of Serial Bus management present in clause 8..

H.1 Bus configuration timeline

Clause 8.4.2 gives a step-by-step narration of the events necessary to reconfigure a Serial Bus in the cable environment after a bus reset. The description that follows attempts to show the time relationships between these phases of bus configuration, many of which can proceed concurrently with other phases. In figure H.1, time commences at the left at the time the bus reset is initiated.

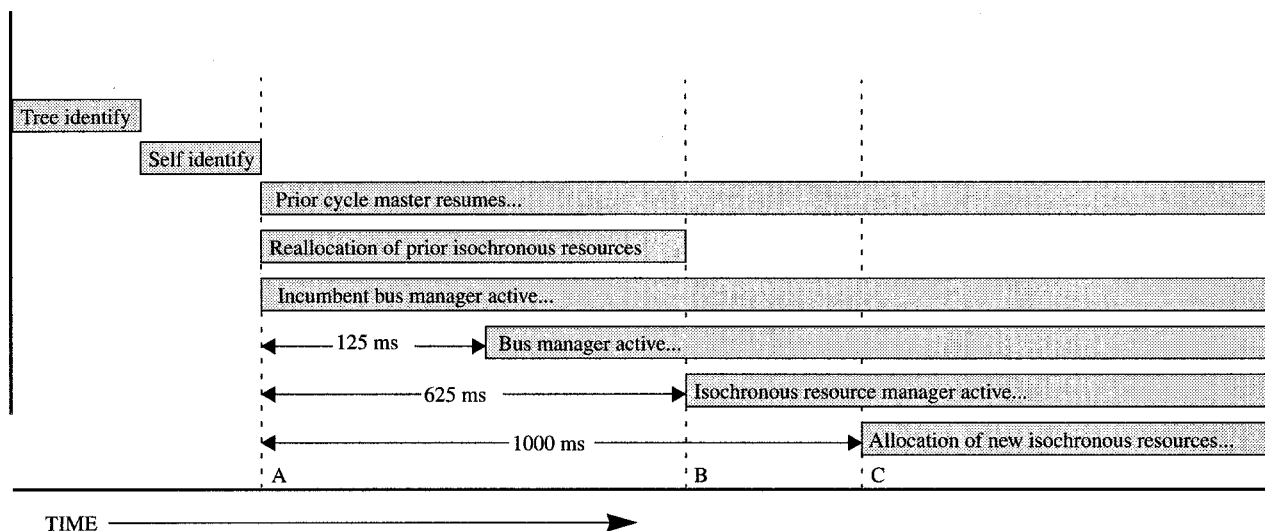


Figure H.1—Bus configuration timeline

In figure H.1, both the tree identify and the self-identify processes must complete before any of the other reconfiguration steps. At point A, all the self-ID packets have been generated and observed, and the following statements are true:

- a) Asynchronous transactions may commence
- b) The identity of the isochronous resource manager (if any) is known
- c) The previous cycle master, if still the root, may resume broadcast of cycle start packets
- d) Isochronous data streams may resume as soon as a cycle start packet is observed
- e) The incumbent bus manager (if any) may reestablish itself as the bus manager
- f) All owners of bus resources prior to the bus reset are required to reallocate the resources

125 ms after the self-identify process has completed, candidates for bus manager that were not incumbent may attempt to establish themselves as the bus manager. Very shortly after this point in time, a bus manager is active, whether or not it was the incumbent.

At point B, the isochronous resource manager may examine its own `BUS_MANAGER_ID` register to determine whether or not a bus manager is active. At this point in time, the following are true:

- All prior isochronous resources have been reallocated
- The bus manager has made the SPEED_MAP registers available
- The bus manager has made the TOPOLOGY_MAP registers available

If the isochronous resource manager discovered, at point B, that no bus manager is active, it is permitted to engage in some forms of bus management, e.g., the activation of a cycle master, transmission of link-on packets to unpowered nodes and, optionally, a rudimentary form of gap count optimization.

Lastly, at point C a second has elapsed, and nodes that wish to allocate new isochronous resources are permitted to do so.

H.2 Bus configuration scenarios

Clause 8.5 describes state machines that shall be implemented at bus-manager-capable, cycle-master-capable, and isochronous-resource-manager-capable nodes. These state machines are sufficient to govern the process of selection of a bus manager, cycle master or isochronous resource manager after a Serial Bus reset in the cable environment. However, the overall picture of bus configuration is difficult to perceive from the limited vantage point of the state machines of a single node.

This subclause remedies the problem through scenarios that illustrate typical Serial Bus configuration procedures. The salient points are

- Selection of a root
- Selection of an isochronous resource manager
- Selection of a bus manager
- Designation of a new root (in order to activate a cycle master)
- Reconfiguration after a second bus reset (in order to confirm the new root)
- Allocation of isochronous resources and commencement of isochronous operations
- Insertion of a new node (and the bus reconfiguration it forces)
- Resumption of isochronous operations

H.2.1 Bus configuration with a bus manager and an isochronous resource manager

Figure H.2 shows the configuration determined after a bus reset for an arbitrary Serial Bus configuration. The physical Id assigned to each node is shown at the top of the box that represents the node. The node capabilities are shown abbreviated to the right, e.g., *irmc* represents isochronous resource manager capable. Each of the PHY ports of the node are arbitrarily numbered. In addition, each node has a description, such as DVCR, printer, or SBP Disk 1, which is not essential to the example but helps anchor the scenario as a possible real-life example.

After a bus reset, the root is first determined by the tree identify process described in E.1.4. Note that the resolution of node five versus node six as the root is arbitrary. Both nodes would have attempted to send a parent_notify signal at the same point in time. After detection of the collision, the node that prevails as the root is dependent upon the random back-off timers used to resolve the impasse.

The node selected as the root in this example has its link layer powered off. This does not prevent its function as the root, but it does make any other capabilities that might reside in the inactive link and transaction layers (bus manager, cycle master, or isochronous resource manager) unavailable. Because a cycle master is needed for isochronous operations, the fact that the link layer of the root is inactive eventually precipitates another bus reset, as described in the following paragraphs.

The self-identify process that immediately follows the tree identify process assigns physical IDs to all nodes and determines which node becomes the isochronous resource manager. Nodes 2 and 5 transmit self-ID packets with both the *l* and the *c* bits set. This indicates that both are contenders for the role of isochronous resource manager. Node 5 has the highest physical ID and is confirmed as the isochronous resource manager.

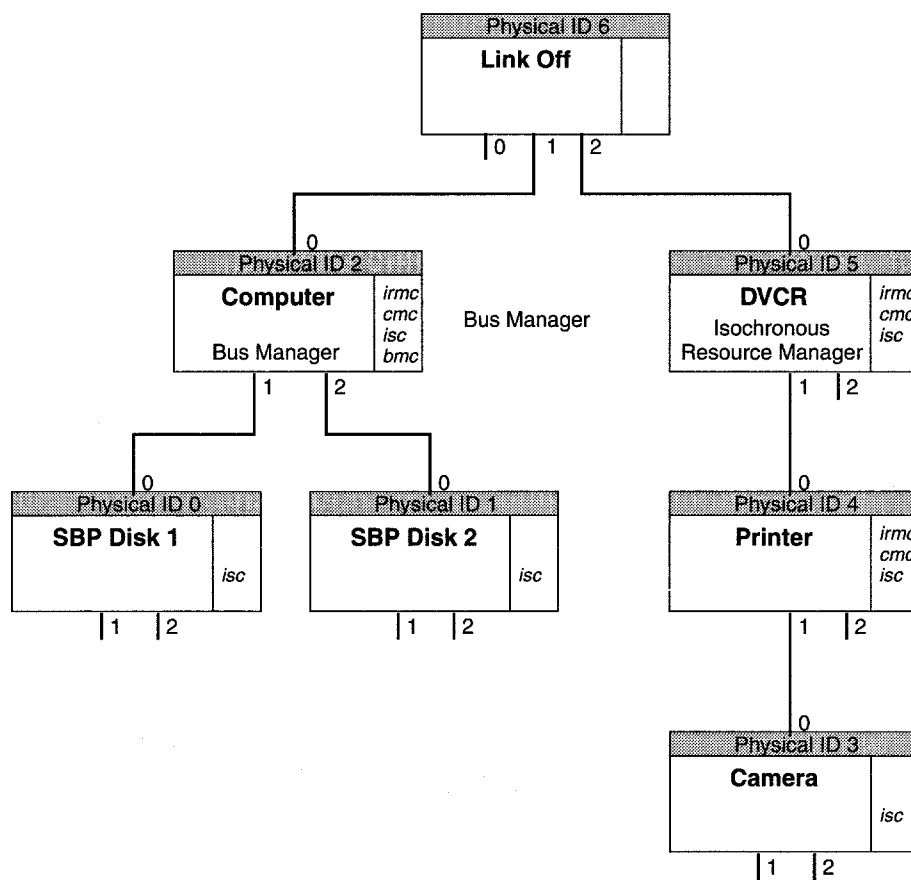


Figure H.2—Bus configuration example (after power-on)

Assume that this is the first bus reset after the Serial Bus nodes have been powered, i.e., there is no incumbent bus manager. As soon as the self-identify process is complete, bus-manager-capable nodes whose Incumbent variable is TRUE are eligible to contend for that role. Since none are incumbent, 125 ms shall elapse before node 2 attempts a compare and swap transaction to place its own physical ID into the BUS_MANAGER_ID register at node 5, the isochronous resource manager. Node 2 succeeds and obtains the role of bus manager.

One of the first obligations of a bus manager is to activate a cycle master if Serial Bus is configured for isochronous operations. The Serial Bus standard places no explicit requirements on the bus manager as to what circumstances require a cycle master, but it is expected that the bus manager shall activate a cycle master if it detects at least two isochronous-capable nodes. The bus manager is expected to examine the *isc* bit in the Bus_Info_Block of all nodes to determine how many isochronous-capable nodes are present. In this case, nodes 2, 3, 4, and 5 are isochronous capable.

The bus manager cannot activate the root as the cycle master since the link layer is inactive. The bus manager might transmit a link-on packet to activate the link layer of the root and then examine the Bus_Info_Block, but the simple, expected solution is to transmit a PHY configuration packet to cause node 5 to set its Force Root variable to TRUE. The bus manager resets the bus after the PHY configuration packet is broadcast.

NOTE — The bus manager may optionally set a new gap count in the PHY configuration packet in order to optimize performance on the bus. This shortcut may be taken since no topology change is expected in connection with this bus reset.

Figure H.3 shows the resultant Serial Bus configuration after the reset.

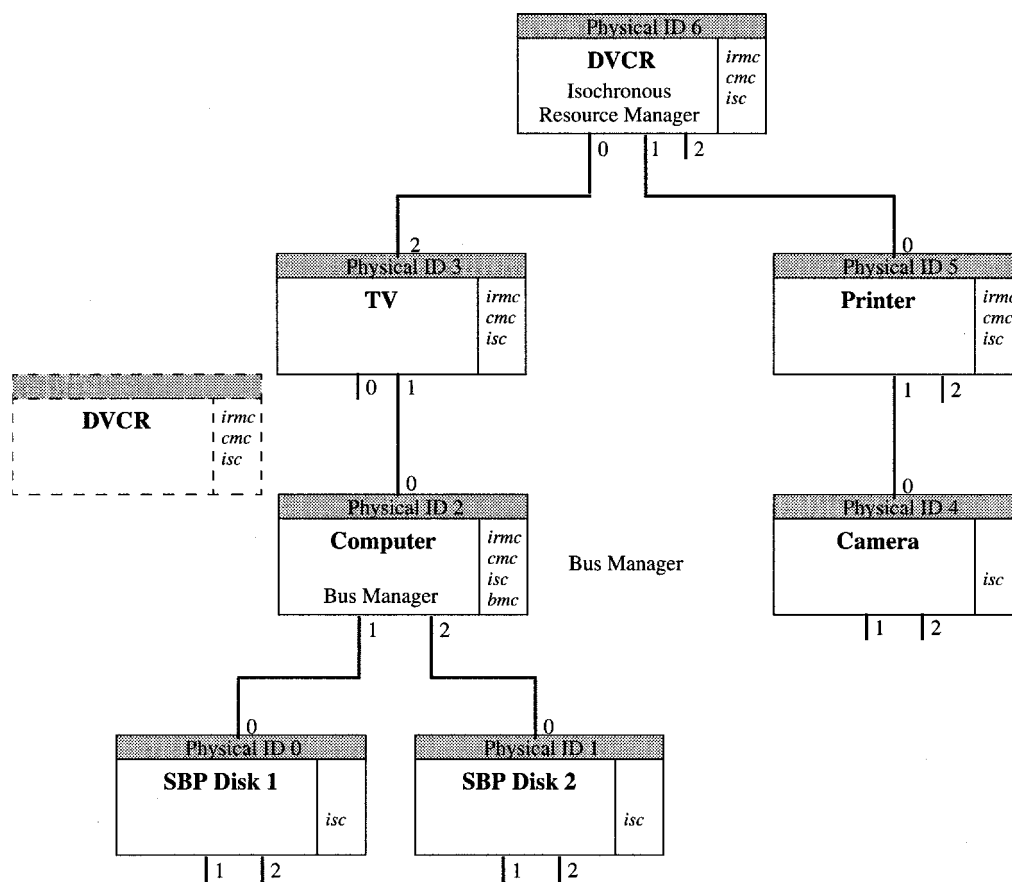


Figure H.3—Bus configuration example (after force root and reset)

Because its Force Root variable was TRUE, node 6 has become the root. The same node that was the isochronous resource manager before the bus reset remains the isochronous resource manager, although its physical ID has changed. Similarly, node 2, the incumbent bus manager, remains the bus manager. However, because node 2 is incumbent before the bus reset it need not wait 125 ms after the self-identify process before it attempts a compare and swap transaction to the BUS_MANAGER_ID register.

Assume that an application at the computer wishes to record video data from the camera on the digital VCR. First, the application allocates the necessary bandwidth and a channel from the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers at the isochronous resource manager. Although the application is not to be either the talker or the listener, it is now the owner of these isochronous resources. At this point, the application may write the appropriate series of commands to the camera CSRs to cause it to start talking on the selected channel and another series of commands to the digital VCR CSRs to cause it to listen. Isochronous operations continue from this point without further intervention by the application at the computer.

As soon as the bus manager is selected, it sets the *cmstr* bit in the STATE_CLEAR register of the root, node 6. This permits isochronous operations to commence with the first broadcast of a cycle start packet by the cycle master.

Subsequent to the activation of the cycle master, the bus manager analyzes the self-ID packets received during the self-identify process and makes the SPEED_MAP and TOPOLOGY_MAP registers available. The bus manager may also perform gap count optimization (not described for this example) or power management. In this example, the power

requirements of node 6 are less than the power available on Serial Bus, so the bus manager transmits a link-on packet to node 6. This reveals node 3 to be a television monitor, another isochronous-capable device.

The last part of this example shows the effects of a node insertion on the bus. The preceding figure shows a not-yet-connected device, a second digital DVCR, available to be added to the unused PHY port of the digital TV. The new Serial Bus configuration that results is shown in figure H.4.

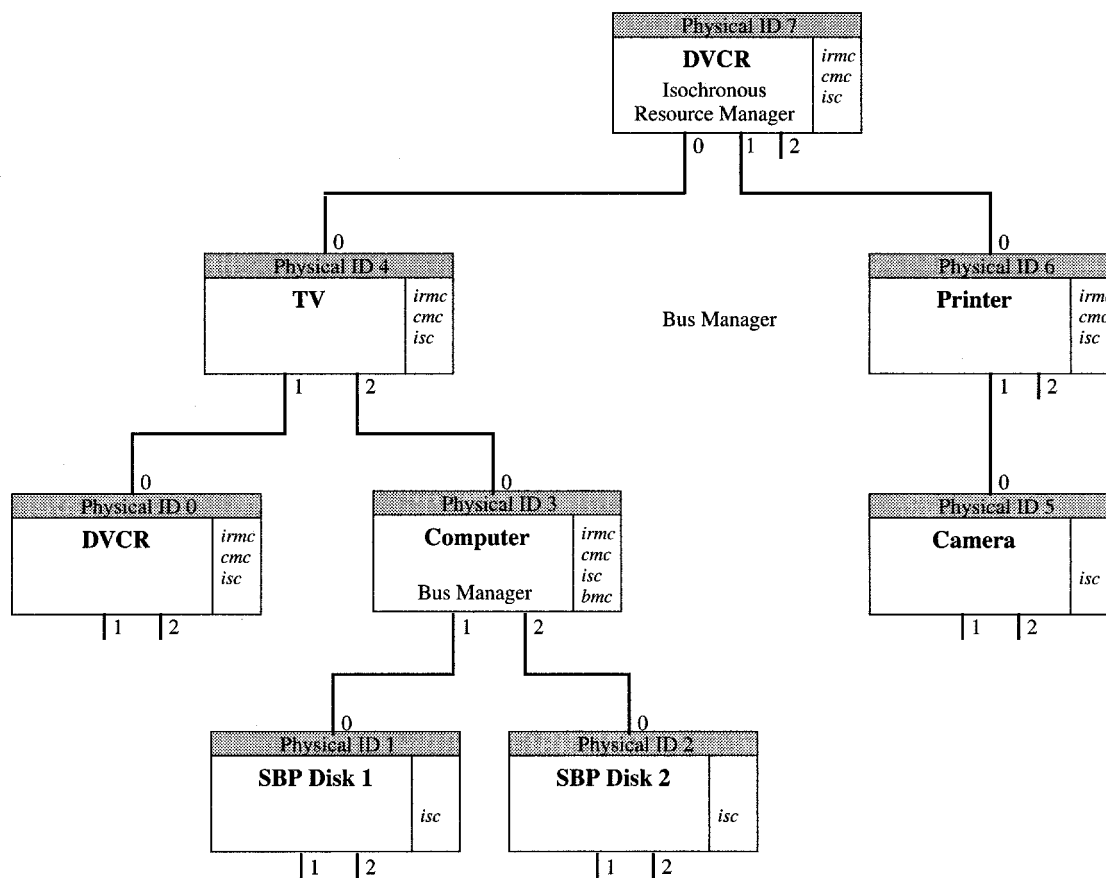


Figure H.4—Bus configuration example (after new node insertion and reset)

Very little of significance changes after the bus reset caused by the insertion of the new node. Physical IDs are reassigned, of course, but the same devices that were the bus manager, cycle master, and isochronous resource manager prior to the bus reset retain their functions. The cycle master resumes broadcast of cycle start packets immediately upon completion of the self-identify process. This permits the resumption of isochronous data flow from the digital camera (the talker) to the digital VCR (the listener) as soon as possible. The owner of the isochronous resources (the computer) reallocates both bandwidth and channel number at the isochronous resource manager BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers as soon as the self-identify process completes. If any new isochronous operations were to commence, e.g., playback from the second digital VCR to the TV monitor, 1 s would have to elapse before these new isochronous resources could be allocated.

This completes the example of Serial Bus configuration when both a bus manager and isochronous resource manager are present. Clause H.2.2 gives an example of another topology without any bus-manager-capable nodes.

H.2.2 Bus configuration with only an isochronous resource manager

Figure H.5 shows the configuration determined after a bus reset for an arbitrary Serial Bus configuration without any bus-manager-capable nodes. The physical ID assigned to each node is shown at the top of the box that represents the node. The node capabilities are shown abbreviated to the right, e.g., *irmc* represents isochronous resource manager capable. Each of the PHY ports of the node are arbitrarily numbered. In addition, each node has a description, such as DVCR, printer, or camcorder, which is not essential to the example but helps anchor the scenario as a possible real-life example.

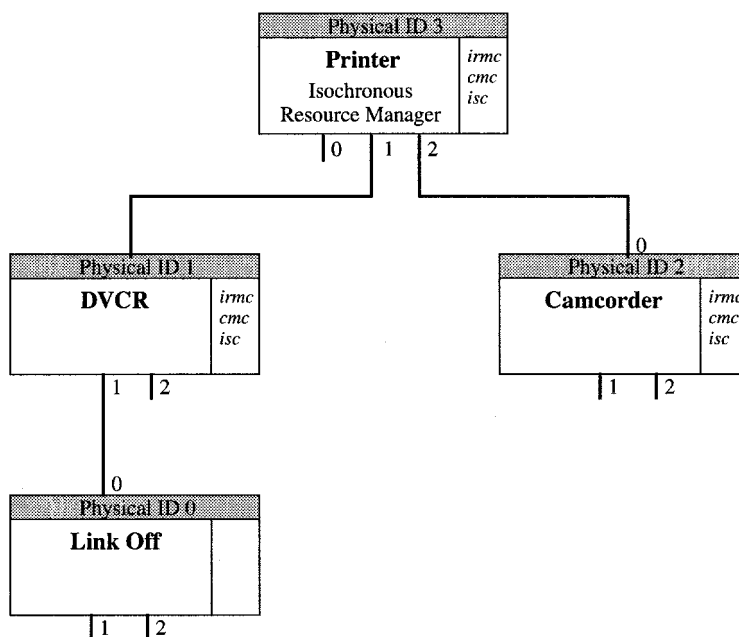


Figure H.5—IRM-only bus configuration example (after power-on)

After a bus reset, the root is first determined by the tree identify process described in E.1.4. Note that the resolution of node 1 versus node 3 as the root is arbitrary. Both nodes would have attempted to send a *parent_notify* signal at the same point in time. After detection of the collision, the node that prevails as the root is dependent upon the random back-off timers used to resolve the impasse.

The self-identify process that immediately follows the tree identify process assigns physical IDs to all nodes and determines which node becomes the isochronous resource manager. Nodes 1 and 3 transmit self-ID packets with both the *l* and the *c* bits set. This indicates that both are contenders for the role of isochronous resource manager. Node 3 has the highest physical ID and is confirmed as the isochronous resource manager.

Because no bus-manager-capable nodes are present, the isochronous resource assumes some of the functions that would otherwise be the province of the bus manager. The isochronous resource manager shall wait 625 ms before it examines its own *BUS_MANAGER_ID* register to determine that no bus manager is present. At this point, the isochronous resource manager shall activate a cycle master. The expected, simple implementation is that the isochronous resource manager activates itself as the cycle master if it is the root.

The isochronous resource manager may also transmit link-on packets to any Serial Bus nodes with unpowered link layers.

Figure H.6 shows the resultant Serial Bus configuration after the isochronous resource manager has transmitted a link-on packet to node 0. This reveals the node to be another isochronous capable node, a digital TV.

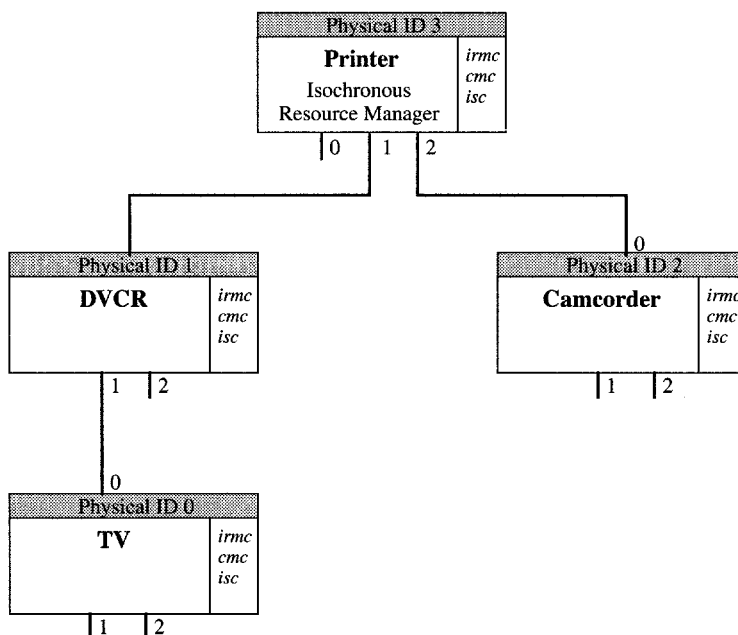


Figure H.6—IRM-only bus configuration example (after link-on to node 0)

Assume that a user wishes to play video data from the camcorder on the digital TV by means of external controls on both devices. In simple consumer applications such as this, the devices may assume that a fixed isochronous channel is always used. If this is the case, there might be no allocation of isochronous resources, the camcorder starts talking in response to external controls and the TV starts listening in the same fashion.

This completes the example of Serial Bus configuration when no bus manager is present.

H.3 Combined bus manager and isochronous resource manager

Although the examples of H.2.1 and H.2.2 show the bus manager and isochronous resource manager as distinct nodes on Serial Bus, there is no prohibition of a node assuming both roles.

The functions of bus manager and isochronous resource manager are distinct, and any node that is both bus manager capable and isochronous resource manager capable shall implement both sets of state machines described in 8.5.2 and 8.5.3. The state machines shall operate independently but shall share information by means of the node variables described in 8.3.3.

In practice, it is a simple matter for a single node to become both the bus manager and the isochronous resource manager. If a bus-manager-capable node becomes the bus manager but it is not the isochronous resource manager, it need only set the Force Root variable to TRUE and issue a bus reset. Since the bus manager is both the incumbent and is destined to become the root (the node with the highest physical ID), it therefore will assume the roles of both the bus manager and the isochronous resource manager in the configuration process that follows the bus reset.

NOTE — It is not required but it is likely that many bus manager implementations choose, for the sake of their own simplicity, to assume the role of isochronous resource manager as well as bus manager.

H.4 Abdication by the bus manager

In a Serial Bus configuration that includes more than one bus-manager-capable node, it may be the case that the node that is confirmed as the bus manager is, for some reason, not the most suitable choice. Whatever the means are that are used to determine the “most suitable choice” for bus manager are beyond the scope of this standard. The existence of some higher level protocol that bus-manager-capable nodes or bus management applications use to determine the most suitable bus manager is simply assumed to exist.

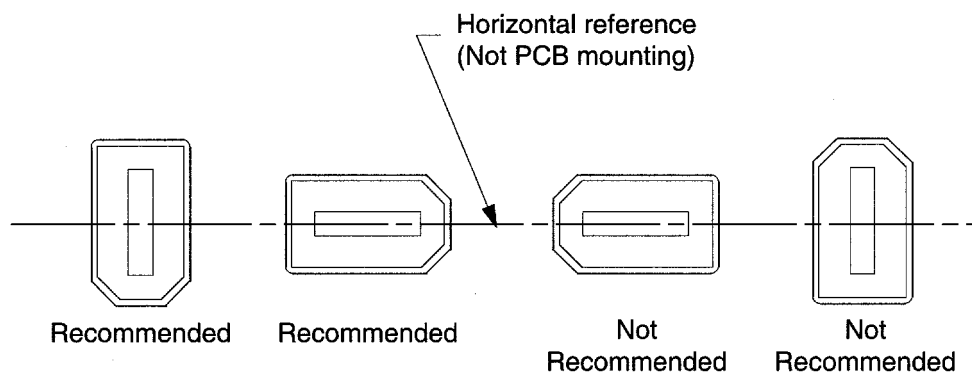
However, once it is determined, by whatever means, that the role of bus manager shall be shifted from the current bus manager to some other bus-manager-capable node, there shall be an orderly way to transfer bus management responsibility. One method is for the current bus manager to communicate to the favored bus manager candidate that it should “cheat” in the next bus configuration process. That is, the favored candidate is informed that it may set its Incumbent variable to TRUE even though it is not the incumbent. The bus manager that is to abdicate shall, of course, set its own Incumbent variable to FALSE even though it is the incumbent. If the bus manager then initiates a bus reset, the favored candidate shall become the new bus manager, since it will be the first to attempt a compare and swap transaction to the BUS_MANAGER_ID register.

Annex I Socket PCB terminal patterns and mounting

(Informative)

I.1 Socket orientation

A socket may be mounted in a variety of ways to suit the application. It is recommended that the socket orientation relative to the normal positioning of the unit be standardized according to figure I.1.



NOTES

1—Not to scale.

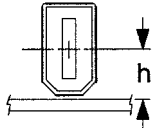
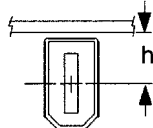
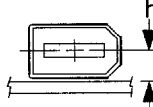
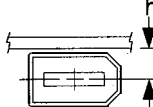
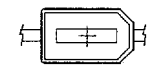
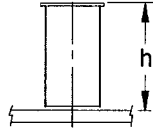
2—This figure illustrates the preference of orientations of the socket connector *as presented to the user by the equipment* in its normal mounting orientation.

Figure I.1—Socket orientations

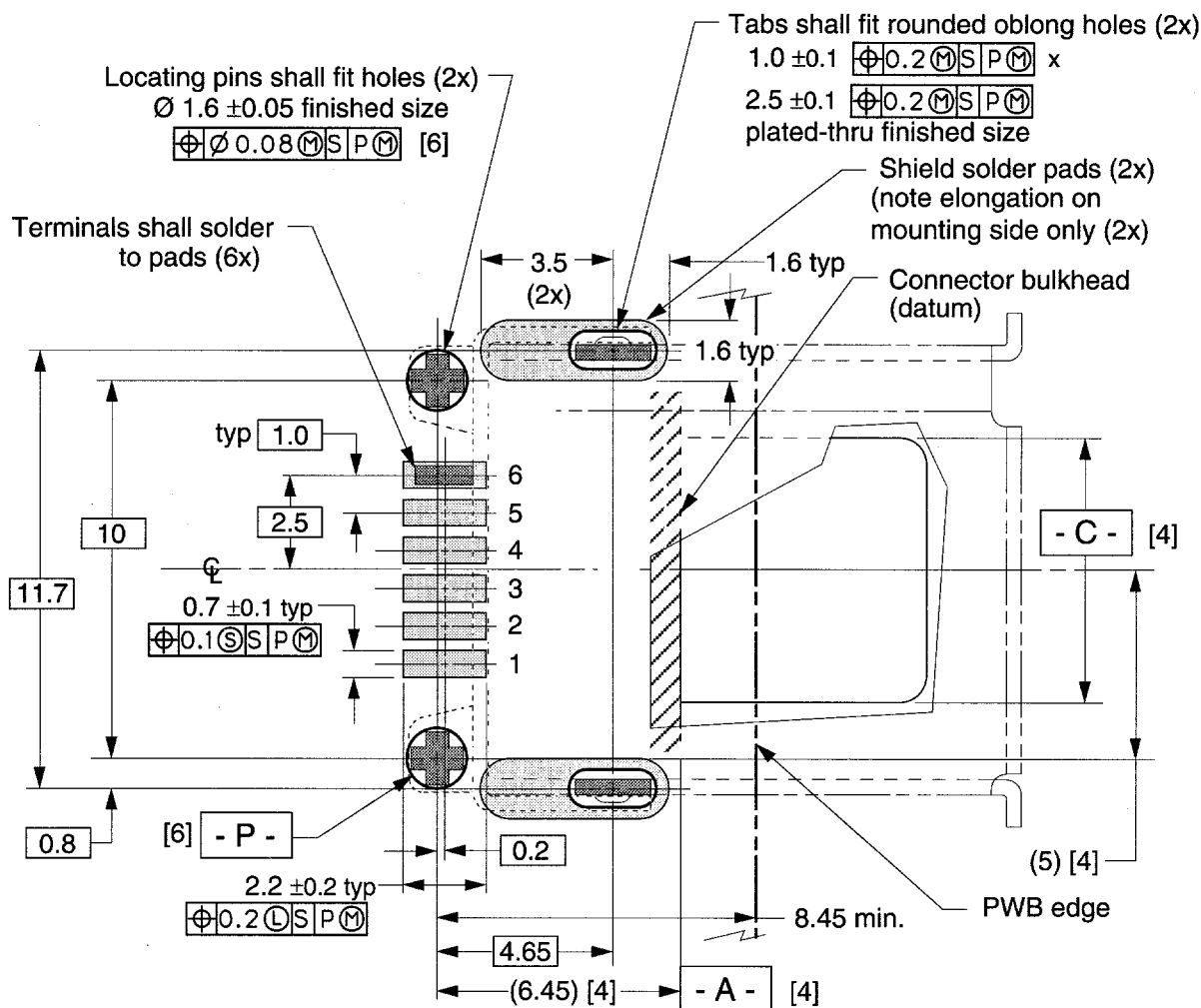
I.2 PCB mounting 0

A socket may be attached to a PCB in a variety of ways to suit the application. Figure I.1 illustrates a number of possibilities.

Table I.1—Table of socket PCB mounting styles and footprint figures

Mounting orientation	Mounting designation	Through-hole mounting	Surface mounting
	Right-angle upright	Figure I.2 h = 6.25 mm	
	Right-angle upright inverted	Figure I.3 h = 6.25 mm	
	Right-angle flat		Figure I.4 h = 4.50 mm or h = 6.35 mm
	Right-angle flat inverted		Figure I.5 h = 4.50 mm or h = 6.35 mm
	Right-angle flat straddle		
	Straight (perpendicular)		

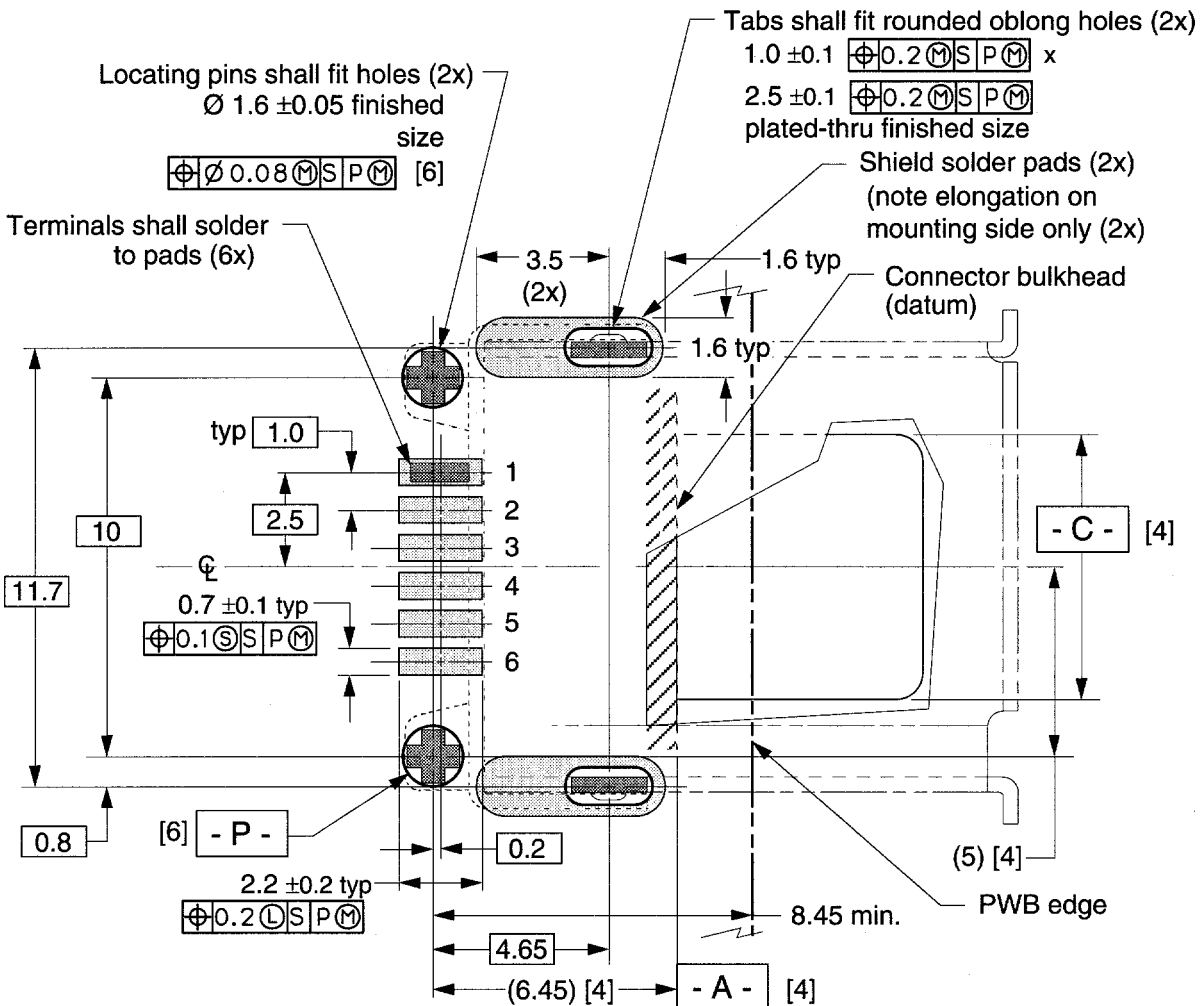
The first column shows how the socket may be mounted on either side of, or straddling, the PCB. The second column designates the style of mounting. Column 3 designates the figure that describes, in detail, the hole and pad pattern for each mounting designation for use with through-hole soldering to the PCB. Column 4 designates the figure that describes, in detail, the hole and pad pattern for each mounting designation for use with surface-mounted, reflow-soldered socket terminals. The dimension “h” has a nominal tolerance of ± 0.20 mm, and the PC board thickness is presumed to be 1.6 mm nominal unless otherwise specified in the specific pattern figure.



Notes:

1. Dimensions are in mm
2. Interpret dimensions & tolerances per ANSI Y14.5M - 1982
3. Untoleranced dimensions are ± 0.12
- [4] Figure is composite of PWB and connector in assembly; connector datums A, C shown in true position relative to PWB datum P (refer to Clause 4 for connector datums and dimensions); connector side of PWB is shown.
5. PWB mounting surface is datum S; thickness is not specified
- [6] Datum P hole excluded from positional tolerance
7. Minimum mounting intervals:
 16 mm for detent-only plugs
 22 mm for positive-retention plugs

Figure I.4—Right angle flat surface mount



Notes:

1. Dimensions are in mm
2. Interpret dimensions & tolerances per ANSI Y14.5M - 1982
3. Untoleranced dimensions are ± 0.12
- [4] Figure is composite of PWB and connector in assembly; connector datums A, C shown in true position relative to PWB datum P (refer to Clause 4 for connector datums and dimensions); connector side of PWB is shown
5. PWB mounting surface is datum S; thickness is not specified
- [6] Datum P hole excluded from positional tolerance
7. Minimum mounting intervals:
 16 mm for detent-only plugs
 22 mm for positive-retention plugs

Figure I.5—Right angle flat surface mount (inverted)

Annex J PHY-link interface specification

(Informative)

J.1 (Scope)

This annex describes an example PHY-link interface for this standard. It covers the protocol, signal timing, and galvanic isolation circuitry. It does not cover specific operation of the PHY except for behavior with respect to this interface.

In this annex, bit 0 of any multiple-bit bus is the most significant and transmitted first on the serial bus. The proposed data rates that are referred to in multiples of 100 Mbit/s are actually multiples of 98.304 Mbit/s. The interface described in this annex supports the following data rates for the cable environment: 100 Mbit/s, 200 Mbit/s, and 400 Mbit/s. (Data rates beyond 400 Mbit/s can be supported if the need arises.) The interface can also support the following data rates for the backplane environment: 25 Mbit/s and 50 Mbit/s. These rates are the actual “bit” rates, independent of the encoding scheme. 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 annex, each bit cell represents one clock sample time. The specific clock-to-data timing relationships are described in J.7.

J.2 Overview

The interface described in this annex 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 pins per 100 Mbit/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 pins. The link only drives these pins when control is transferred to it by the PHY. The link performs all unsolicited activity through a dedicated request pin. The possible actions that may occur on the interface are categorized as transmit, receive, status, and request. These actions are described in detail later in this annex.

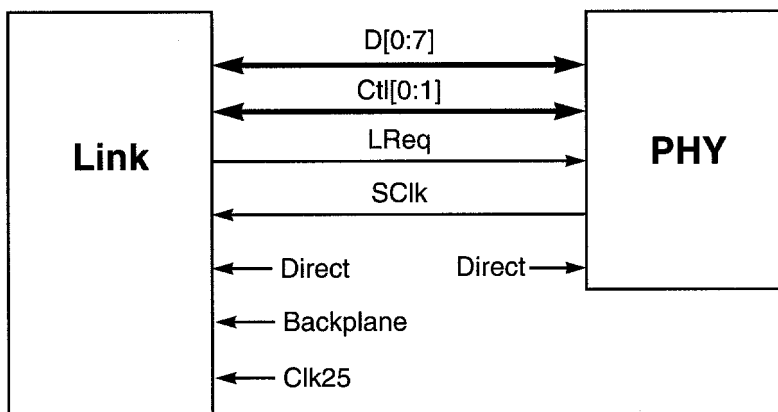


Figure J.1—PHY-link interface

Table J.1—Pin 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
Direct	Neither	Controls differentiator for interface pins
Backplane	Neither	Set high if backplane PHY
Clk25	Neither	Set high to force SCLK to 24.576 MHz

Data is carried between the two chips on the D bus. The width of the D bus depends on the maximum speed of the connected PHY chip, 2 bits per 100 Mbit/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 100 Mbit/s transfers use D[0:1], 200 Mbit/s transfers use D[0:3], and 400 Mbit/s 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 pin is used by the link to request access to the serial bus and to read or write PHY registers. The Direct pin is used to disable the digital differentiator on the D, Ctl, and LReq pins, 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 pins to logic 0 levels for one clock before tristating 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 operated properly. This procedure is built into the operational descriptions and timing diagrams that follow.

J.3 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 the serial bus.

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

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

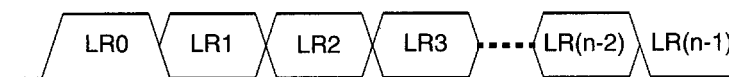
Ctl[0:1]	Name	Meaning
00	Idle	Transmission complete, release bus.
01	Hold	The link is holding the bus while preparing data or indicating that it wishes to reacquire the bus without arbitrating to send another packet.
10	Transmit	The link is sending a packet to the PHY.
11	unused	Unused.

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

J.3.1 Request

To request the bus or to access a PHY register, the link sends a short serial stream to the PHY on the LReq pin. The information sent includes the type of request or the speed at which the packet is to be sent, or a read or write command. The transfer can be either 7 bits, 9 bits, or 17 bits, depending on whether it is a bus request, a read access, or a write access, respectively. A stop bit of 0 is required after each type of request transfer before another transfer may begin.

The timing for this pin and the definition of the bits in the transfer are shown in figure J.2.

**Figure J.2—LReq timing**

If the LReq transfer is a bus request in the cable environment, it is 7 bits long and has the format given in table J.4.

Table J.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 J.8 for the encoding of this field.
4–5	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 J.9 for the encoding of this field. This field can be expanded to support data rates higher than 400 Mbit/s in the future.
6	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 J.5.

Table J.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 J.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 J.6.

Table J.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 J.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 J.7.

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

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

Table J.8—Request type field

LR[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 gap, ignore fair protocol. Used for cycle master request.
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.
110–111	Reserved	Reserved.

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

Table J.9—Request speed field

LR[4:5]	Data rate
00	100 Mbit/s
01	200 Mbit/s
10	400 Mbit/s
11	>400 Mbit/s

NOTE — LR[4:5] is always set to 00 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 pins 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. 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 Mbit/s acknowledge packet. The minimum request packet is eight clock cycles. It is important that the link and PHY agree to interpret a lost request the same way.

The cycle master node uses a priority request (PriReq) to send the cycle start message. To request the bus to send isochronous data, the link can issue the request at any time after receiving the cycle start. 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 because the delay from end of packet to acknowledge request adds directly to the minimum delay every PHY has to wait after every packet to allow an acknowledge to occur. 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, iso, fair, or priority), it cannot issue another request until the PHY indicates “lost” (incoming packet) or “won” (transmit). The PHY ignores new requests while a previous request is pending.

J.3.2 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 pins, 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 pins. This should 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 request transfer that contains a read request, or when the PHY has pertinent information to send to the link or transaction layers. The only defined condition where the PHY automatically sends a register to the link is after self-identification, where it sends the “physical_ID” register that contains the new node address.

The timing for the transfer is shown in figure J.3.

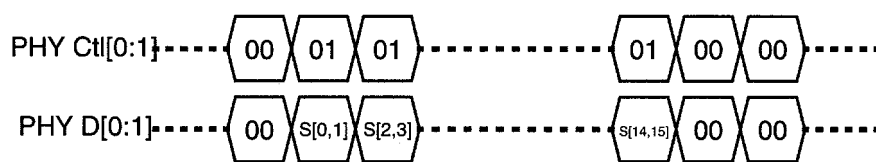


Figure J.3—Status timing

Table J.10—Status bits

Bit(s)	Name	Description
0	Arbitration Reset Gap	The PHY has detected that the serial bus has been idle for an arbitration reset gap time. This bit is used by the link in the busy/retry state machine. The arbitration reset gap time is defined in 4.3.6.
1	Subaction Gap	The PHY has detected that the 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. The subaction gap time is defined in 4.3.6.
2	Bus Reset	The PHY has entered bus reset state.
3	State Time-out	The PHY stayed in a particular state for too long a period. This is usually the effect of a loop in the cable topology.
4–7	Address	When transferring the contents of a register to the link, such as when responding to a read through the LReq pin, this field holds the address of the register being read.
8–15	Data	This field holds the data corresponding to the register being transferred.

J.3.3 Transmit

When the link requests access to the serial bus through the LReq pin, the PHY arbitrates for access to the serial bus. If the PHY wins the arbitration, it grants the bus to the link by asserting *transmit* on the Ctl pins for one SClk cycle, followed by *idle* for one cycle. After sampling the *transmit* state from the PHY, the link takes over control of the interface by asserting either *hold* or *transmit* on the Ctl pins. The link asserts *hold* to keep ownership of the bus while preparing data. The PHY asserts the data-on state on the serial bus during this time. When it is ready to begin transmitting a packet, the link asserts *transmit* on the Ctl pins 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 pins for one cycle, and then it asserts *idle* for one additional cycle before tristating those pins.

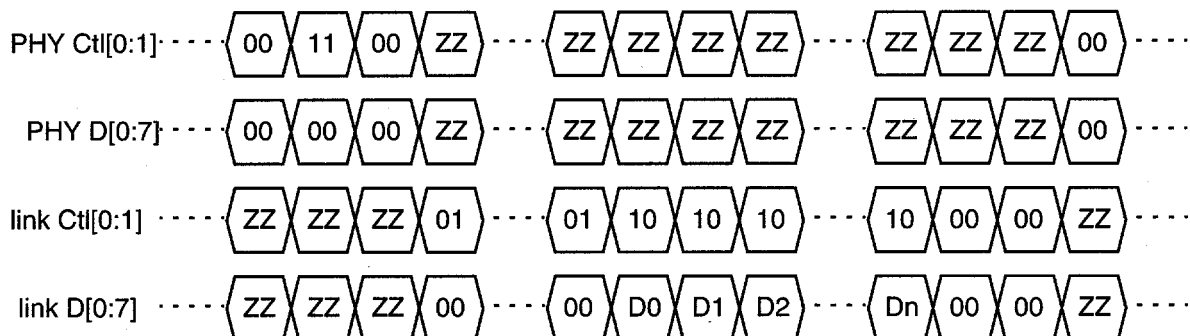
The *hold* state here indicates to the PHY that the link needs to send another packet without releasing the bus. The PHY responds to this *hold* state by waiting the required minimum time and then asserting *transmit* as before. This function would be used after sending an acknowledge if the link intends to send a unified response, or to send consecutive isochronous packets during a single cycle. The only requirement when sending multiple packets during a single bus ownership is that all shall be transmitted at the same speed, since the speed of the packet transmission is set before the first 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 pins for two SClk cycles. The PHY begins asserting *idle* on the Ctl pins 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 J.4. In the diagram, DO through Dn are the data symbols of the packet, ZZ represents high impedance state.

Single Packet



Continued Packet

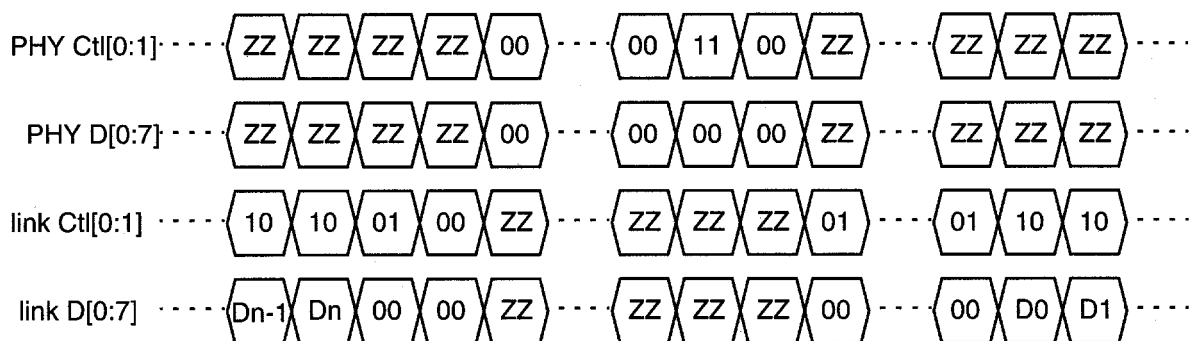


Figure J.4—Transmit timing

J.3.4 Receive

Whenever the PHY sees the “data-on” state on the serial bus, it initiates a receive operation by asserting *receive* on the Ctl pins and “1” on each of the D pins. The PHY indicates the start of a packet by placing the speed code (encoding shown in table J.11) on the D pins, followed by the contents of the packet. The PHY holds the Ctl pins 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 pins. 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 the 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 J.5. In the diagram, SP refers to the speed code, and D0 through Dn are the data symbols of the packet.

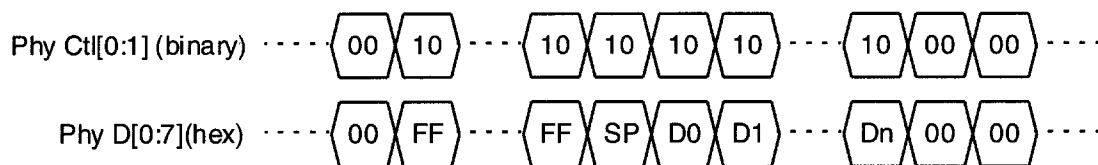


Figure J.5—Receive timing

The speed code for the receive operation is defined as shown in table J.11.

Table J.11—Receive speed code

D[0:7]	Data rate
00xxxxxx *	100 Mbit/s
0100xxxx	200 Mbit/s
01010000	400 Mbit/s
11111111	“data-on” indication

*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.

J.4 PHY register map

J.4.1 PHY register map (cable environment)

The accessible registers in the cable PHY are shown in the following diagram. All registers beyond the last port status register (AstatN...), or beyond the last register indicated by Reg_count (if an enhanced PHY register map is used), are implementation dependent. Note that the ordering convention for the bits (i.e., most significant to least significant) is as indicated in 1.6.3.

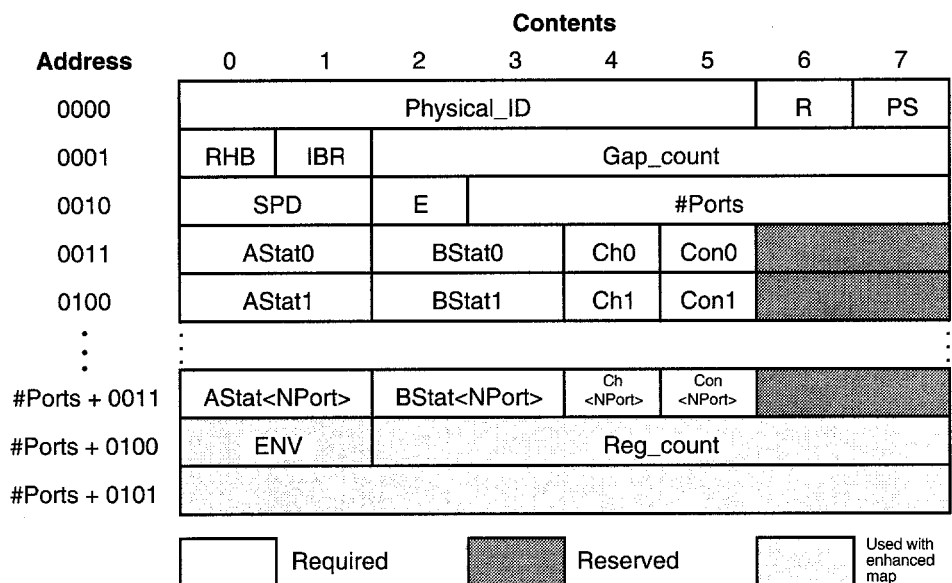


Figure J.6—PHY register map for the cable environment

Table J.12—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	Indicates that this node is the root.
PS	1	r	Cable power status.
RHB	1	rw	Root hold-off bit. Instructs the PHY to attempt to become the root during the next bus reset.
IBR	1	rw	Initiate bus reset. Instructs the PHY to initiate a bus reset at the next opportunity.
Gap_count	6	rw	Arbitration timer setting. Used to optimize gap times based on the size of the network. See 4.3.6 for the encoding of this field.
SPD	2	r	Indicates the top speed this PHY can handle. Same encoding as for speed code in the request packet.
E	1	r	If = 1, enhanced register map is used (i.e., registers #Ports+0100 and beyond are defined).
#Ports	5	r	The number of ports on this PHY. This also indicates how port status registers will follow.
AStat<n>	2	r	TPA line state on port <n> where 0 ≤ n < #Ports: 11 = ZZ 01 = 1 10 = 0 00 = invalid
BStat<n>	2	r	TPB line state on port <n> (same encoding as AStat<n>).
Ch<n>	1	r	If = 1, port <n> is a child, else parent.

Table J.12—PHY register fields for the cable environment (Continued)

Field	Size	Type	Description
Con⟨n⟩	1	r	If = 1, port ⟨n⟩ is connected, else disconnected.
ENV	2	r	Used with enhanced register map. Indicates the type of environment:
			00 = backplane 01 = cable 10, 11 reserved
Reg_count	6	r	Used with enhanced register map. Indicates the number of additional registers that follow.

J.4.2 PHY register map (backplane environment)

The backplane environment has a PHY register map similar to that of the cable environment, except that certain fields are not used and that other fields may be always set to a particular value. In addition, the backplane environment may make use of the “enhanced” register map to allow Priority and Transceiver Disable fields to be defined.

If the enhanced PHY register map is not used, the backplane environment requires only the following fields: Physical_ID, IBR, #Ports, AStat0, and BStat0. These fields are used in the same manner as in the cable environment (see table J.12), with the following exceptions:

- The Physical_ID field is read/write.
- Since there can only be 1 port per PHY layer in the backplane environment, the #Ports field is always set to 1.
- The AStat0 and BStat0 fields use the same encoding as the cable environment, except that the AStat0 field contains the Data line state and the BStat0 contains the Strobe line state.

The following fields are not used in the backplane environment: R, PS, RHB, Gap_count, SPD, AStat(1..n), BStat(1..n), Ch⟨n⟩, and Con⟨n⟩. The contents of these fields shall be set to zero (and are reserved for future use).

J.4.3 Enhanced PHY register map

The optional capability of an enhanced register map allows for additional fields to be defined for use by both the cable and backplane environments. Such fields are located at addresses #Ports+0100 and beyond. If this capability is used, the E field shall be set to 1 (otherwise, it is set to 0) and Reg_count shall be set equal to the number of additional registers that follow the Reg-count field.

For the backplane environment, the enhanced register map may be used to define priority and transceiver disable fields. If these fields are to be defined, the following steps shall be taken:

- E shall be set to 1 (indicating that an enhanced register map is used).
- ENV shall be set to 00 (indicating the backplane environment).
- Reg_count shall be set to 1 (indicating that one additional register is to be defined).
- The transceiver disable (TD) field shall be located within bit 0 of the additional register (i.e., the leftmost bit in figure J.6). When this bit is set, the backplane PHY layer shall set all bus outputs to a high-impedance state. This bus output state shall be maintained until this bit is cleared. Link layer service actions that would require a change in bus output state shall not be performed. The contents of this field may be changed using the Disable Transmit and the Enable Transmit services defined in 5.1.1.1.
- The priority field shall be located within bits 4 through 7 of the additional register. This field shall contain the four-bit parameter used during the urgent arbitration process and may be changed using the priority service defined in 5.1.1.1. The contents of this field shall be consistent with the “Pri” field that is communicated within the header of the packet to be transmitted (see 6.2.4.6).
- Bits 1 through 3 of the additional register shall be reserved.

NOTE — Because there is a maximum of one port per backplane PHY, #Ports is always set to 1 and the register located at address 0011 is the only register used to contain port status fields. Consequently, the register containing the ENV and Reg_count fields is located at address 0100. The register containing the TD and priority fields is located at address 0101.

J.5 State diagrams

The following state diagrams describe the actions required of the interface in the link and the PHY to support the functions described in this annex.

J.5.1 Link request

Figure J.7 describes the how the link should behave for the various types of requests.

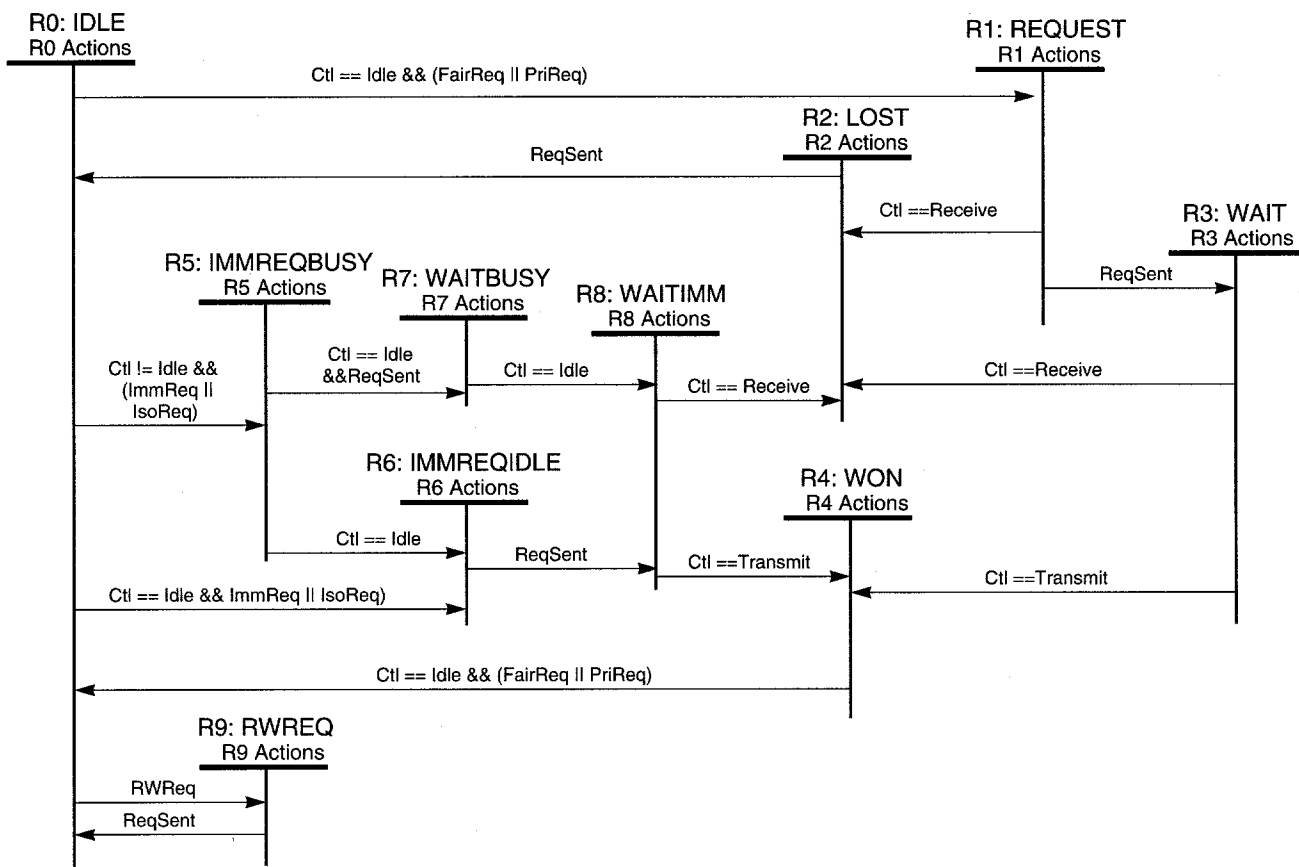


Figure J.7—Link request state machine

Table J.13—Link request state machine inputs

Name	Description
FairReq	Fair asynchronous request
PriReq	Priority asynchronous request
ImmReq	Request to send an acknowledge
IsoReq	Isochronous data request
RWReq	Read or write register request
ReqSent	Entire request has been shifted out
Ctl	Interface control pins

Table J.14—Link request state machine outputs

Name	Description
Won	Indicates bus ownership to transmitter
Lost	Indicates lost request to transmitter
ShiftEn	Enables LReq shift register

Table J.15—Link request state machine state definitions

State	Name	Actions
R0	IDLE	None
R1	REQUEST	ShiftEn=true (send request stream to PHY)
R2	LOST	Lost=true
R3	WAIT	None
R4	WON	Won=true, go to IDLE
R5	IMMREQBUSY	ShiftEn=true (send request stream to PHY)
R6	IMMREQIDLE	ShiftEn=true (send request stream to PHY)
R7	WAITBUSY	None
R8	WAITIMM	None
R9	RWREQ	ShiftEn=true (send read or write request to PHY)

J.5.2 Link general

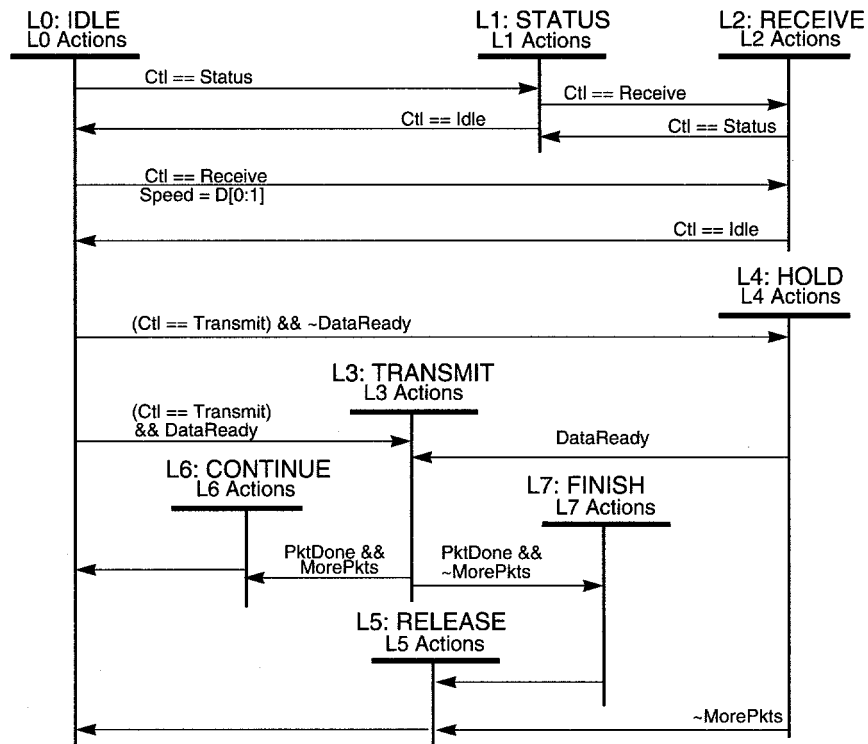


Figure J.8—Link general state diagram

Table J.16—Link general state machine inputs

Name	Description
DataReady	Transmitter can send on next cycle
PktDone	Transmitter is sending last bits of packet
MorePkts	Transmitter wants to send consecutive packets
Ctl	Interface control pins

Table J.17—Link general state machine outputs

Name	Description
Tristate	Tristates D, Ctl pins
StatusShiftEn	Enables status shift register
DInShiftEn	Enables receive shift register
DOutShiftEn	Enables transmit shift register
Speed	Speed of packet to be received
Ctl	Interface control pins

Table J.18—Link general state definitions

State	Name	Actions
L0	IDLE	tristate = true
L1	STATUS	tristate = true, statusShiftEn = true
L2	RECEIVE	tristate = true, dInShiftEn = true
L3	TRANSMIT	Ctl = transmit, DOutShiftEn = true
L4	HOLD	Ctl = hold
L5	RELEASE	Ctl = idle, go to IDLE
L6	CONTINUE	Ctl = hold, go to IDLE
L7	FINISH	Ctl = idle, go to RELEASE

J.5.3 PHY general

Figure J.9 describes how the PHY handles the various events that happen in this interface. It purposely does not describe how the PHY requests the bus or observes the required gaps.

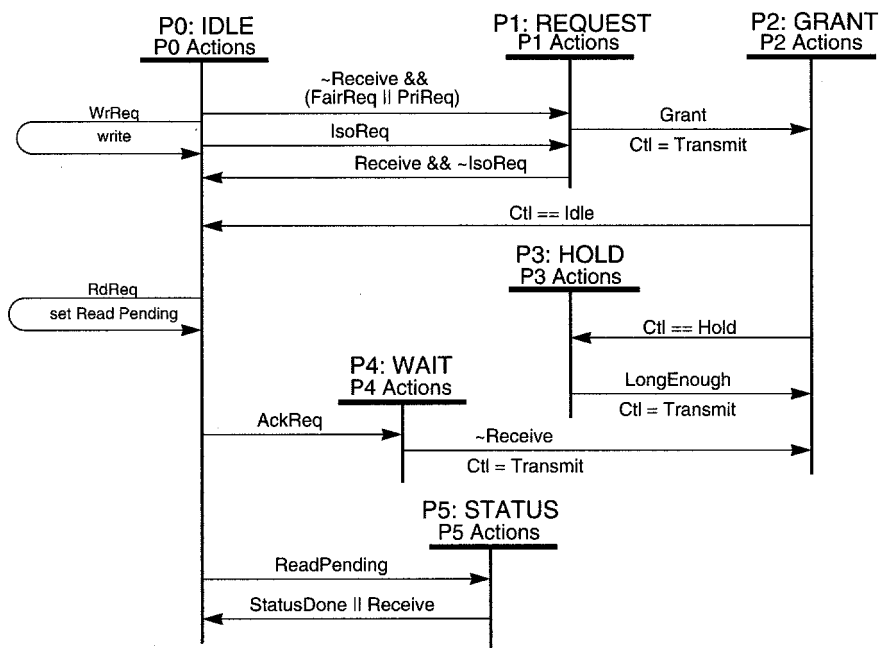


Figure J.9—PHY general state diagram

Table J.19—PHY general state machine inputs

Name	Description
FairReq	Fair asynchronous request
PriReq	Priority asynchronous request
AckReq	Request to send an acknowledge
IsoReq	Isochronous data request
RdReq	Read register request
WrReq	Write register request
Receive	A packet is incoming
Grant	Arbitration won
ReadPending	A register read has been received
Ctl	Interface control pins

Table J.20—PHY general state machine outputs

Name	Description
Tristate	Tristates D, Ctl pins
Request	Request bus on next idle (to rest of PHY)
Ctl	Interface control pins

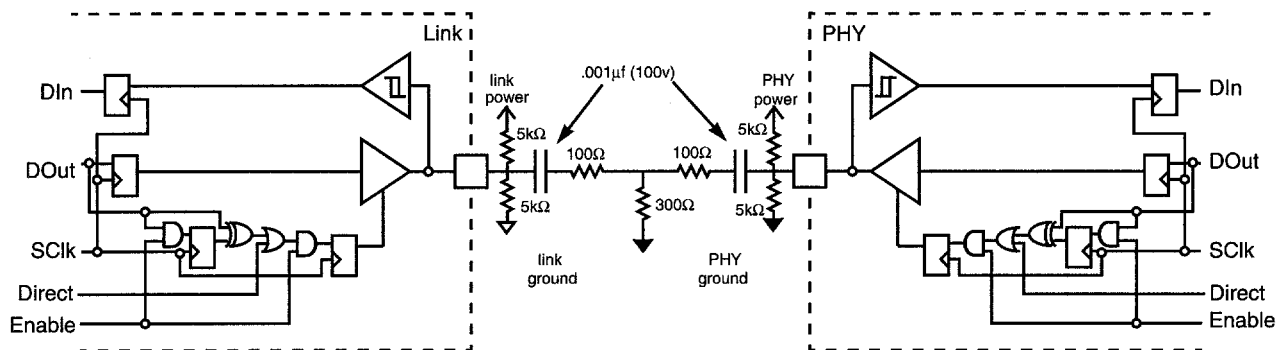


Figure J.11—Capacitive isolation barrier circuit example

J.7 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 J.22 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 assume a maximum delay through an isolation barrier.

Table J.22—AC timing

Parameter	Unit	Minimum	Maximum
D, Ctl, LReq setup to SClk rise	ns	0	
D, Ctl, LReq hold to SClk rise	ns	10	
SClk rise to D, Ctl, LReq out	ns		10
Delay through isolation barrier	ns		3
Hysteresis input rising threshold	V	$V_{cc}/2 + 0.2$	$V_{cc}/2 + 1.3$
Hysteresis input falling threshold	V	$V_{cc}/2 - 1.3$	$V_{cc}/2 - 0.2$

Annex K Serial Bus cable test procedures

(Informative)

K.1 Scope

This annex describes a set of test procedures that attempts to characterize completely the electrical performance of the Serial Bus cable assembly. The tests are intended to provide results of maximum relevance to the system implementor.

Toward this goal, the procedures presented in this annex provide an “end to end” characterization of the Serial Bus cable and connector system. This includes the cable itself, two cable assembled plugs, two PCB assembled sockets, and a length of controlled impedance PCB traces relevant for practical applications. While the device under test is only the cable assembly itself, the sockets and the PCB traces are included in the test fixtures. The limit specifications and the test procedures described in this annex apply to complete cable assemblies of any length.

The measuring equipment listed in the following text or shown in the figures is shown for completeness and to guarantee maximum repeatability of measurements. Equipment of equal capabilities may be used, although the procedures described in this annex may have to be modified accordingly.

K.2 Test fixture

The test procedures proposed utilize a test fixture that provides the transition between a Serial Bus board-mounted socket (which can receive the cable assembly under test) and six 50 Ω SMA connectors (which can be connected to standard 50 Ω coaxial test equipment).

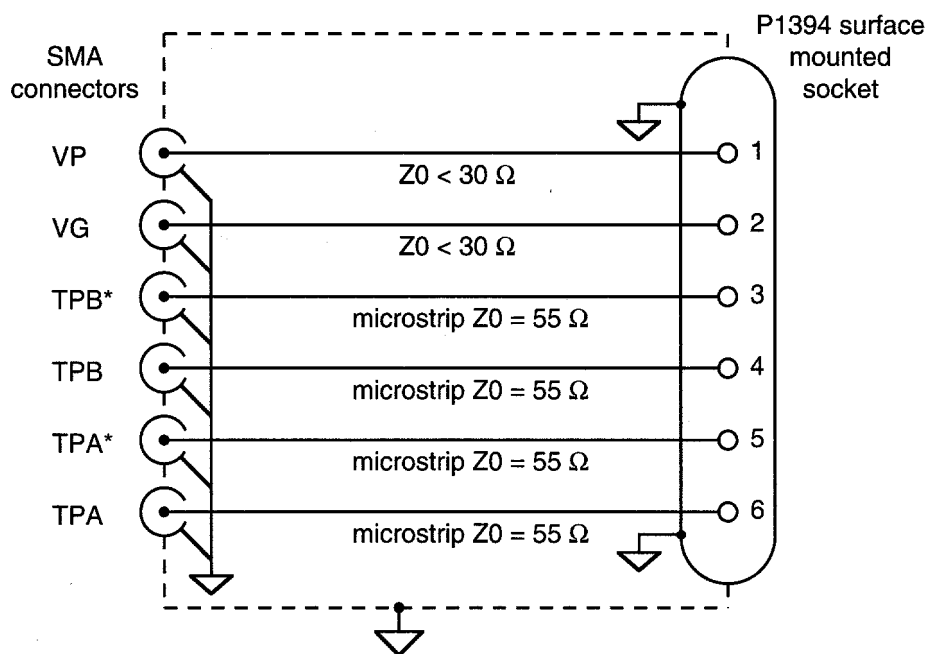


Figure K.1—Cable test fixture schematic

The electrical diagram of such a test fixture is shown in figure K.1. The fixture shall be constructed using a multilayer board enclosed in a metal case. The case is electrically connected to the shield of the Serial Bus socket and to the shield of the six SMA connectors.

The Serial Bus socket shall be surface mounted to the board. The four signal pins of the socket (TPA, TPA*, TPB, and TPB*) are connected to the four SMA connectors using microstrip lines with a characteristic impedance of $55\ \Omega \pm 3\ \Omega$. The length of the connections shall be less than 50 mm. The length mismatch between any two of the four connections shall be less than 2 mm. It is important to minimize crosstalk within the fixture by using the ground plane to isolate the connections corresponding to different signal pairs.

The two power pins on the Serial Bus socket (VP and VG) are connected to the two SMA connectors using traces with a characteristic impedance of less than $30\ \Omega$. These traces shall be designed such as to minimize their dc resistance. The uniformity of their characteristic impedance is of a lesser significance so via holes can be used along the connection traces.

Two test fixtures are used for every cable assembly test, and their electrical performance becomes an integral part of the test results. The effect of the test fixtures upon the test results is not calibrated out during the test setup calibration. Thus, the test fixtures should be maintained in conditions representative for reasonable practical system usage. The socket shall be replaced at least every 1000 connections.

The schematic diagram of the test fixture used in all the following test configuration diagrams is shown in figure K.2.

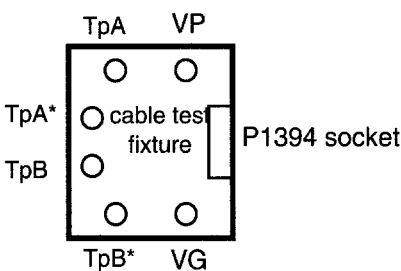


Figure K.2—Test fixture graphic symbol

The construction of the test fixture raises the issue of impedance matching between a pair of single-ended $50\ \Omega$ coaxial connectors and the differential mode $110\ \Omega$ Serial Bus signal lines. The connection traces for the signal lines located inside the test fixture are single-ended $55\ \Omega$ matched electrical length; thus, it can be assumed that there is a reasonable differential mode matching between them and the Serial Bus socket and cable assembly. The impedance matching problem is therefore shifted to the level of the SMA connectors, where matching circuits can be added.

In order to improve accuracy, some of the tests use a minimum loss-resistive matching pad. The schematic diagram of such a pad is shown in figure K.3.

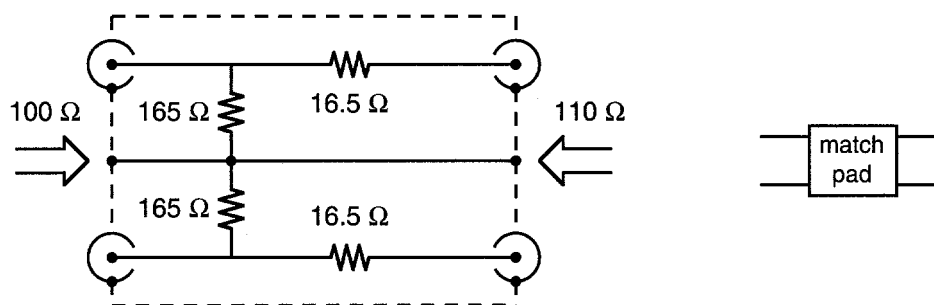


Figure K.3—100 Ω to 110 Ω matching pad

The test configurations contain also precision 20 dB attenuators, which are used to isolate the test equipment from the cable/connector mismatch. Due to the relative low level of mismatch and the isolation of the attenuator, the matching pads may be omitted at the expense of a slight increase in the frequency domain ripple. This effect can be removed by the data filtering algorithms provided by the suggested test equipment.

The impedance matching pads, if utilized, are always included in the test calibration setup to eliminate their effect upon the final test result.

K.3 Signal pairs characteristic impedance

The differential mode characteristic impedance shall be measured in the time domain using a single-ended time domain reflectometer (TDR) with an edge rate of less than 0.2 ns. The differential mode impedance value is calculated from the single-ended measurements described in the following subclauses. The result of each single-ended measurement is calculated as the average of the impedance measured at two points along the cable. These points are selected at 1 ns and at 2.5 ns along the cable from the plug closest to the launching connector. It should be noticed that, because the TDR displays the round-trip propagation delay, the measurements shall be made at 2 ns and 5 ns from the plug closest to the launching connector as measured by the TDR instrument. This test shall be repeated for both signal pairs.

K.3.1 Signal pairs impedance setup calibration—short and load

This calibration should be performed as shown in figure K.4 using the calibration algorithms built into the TDR equipment suggested (HP 54120B and HP 54121A or equivalent).

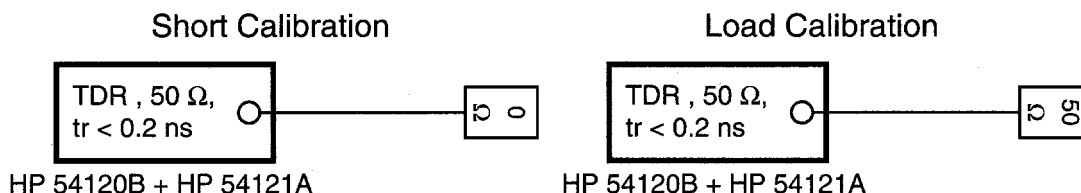


Figure K.4—Signal pairs impedance setup calibration

K.3.2 Signal pairs impedance test procedure

Using the test configuration described in figure K.5 and the connection matrix shown in table K.2, the various characteristic impedances of the signal wires shall be measured in two points and the results averaged.

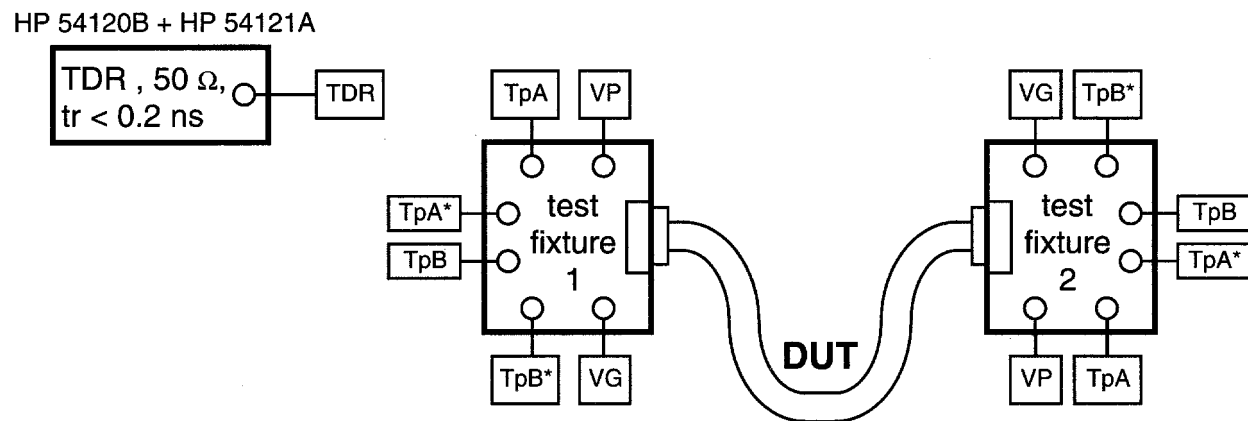


Figure K.5—Signal pairs impedance measurement configuration

Table K.1—Connection matrix for signal pairs impedance tests

Measured value	Fixture 1						Fixture 2					
	VP	TPA	TPA*	TPB	TPB*	VG	VP	TPA	TPA*	TPB	TPB*	VG
Single-ended TPA (Z_{TPA1})	0 Ω	TDR	0 Ω	50 Ω	50 Ω	0 Ω	0 Ω	50 Ω	50 Ω	∞	0 Ω	0 Ω
Single-ended TPA (Z_{TPA2})	0 Ω	0 Ω	TDR	50 Ω	50 Ω	0 Ω	0 Ω	50 Ω	50 Ω	0 Ω	∞	0 Ω
Single-ended TPB (Z_{TPB1})	0 Ω	50 Ω	50 Ω	TDR	0 Ω	0 Ω	0 Ω	∞	0 Ω	50 Ω	50 Ω	0 Ω
Single-ended TPB (Z_{TPB2})	0 Ω	50 Ω	50 Ω	0 Ω	TDR	0 Ω	0 Ω	0 Ω	∞	50 Ω	50 Ω	0 Ω
Common-mode TPA (Z_{TPA3})	0 Ω	TDR	TDR	50 Ω	50 Ω	0 Ω	0 Ω	50 Ω	50 Ω	∞	∞	0 Ω
Common-mode TPB (Z_{TPB3})	0 Ω	50 Ω	50 Ω	TDR	TDR	0 Ω	0 Ω	∞	∞	50 Ω	50 Ω	0 Ω

$Z_{TPn}(2)$ is measured along the cable 2 ns from the plug connected to test fixture 1.

$Z_{TPn}(5)$ is measured along the cable 5 ns from the plug connected to test fixture 1.

Z_{TPn} is calculated as

$$Z_{TPn} = (Z_{TPn}(2) + Z_{TPn}(5))/2 \tag{K-1}$$

“TPn” is TPA1, TPA2, TPA3, TPB1, TPB2, or TPB3.

The differential mode characteristic impedance of signal twisted pair TPA is calculated as

$$Z_{\text{TPA}} = 4 \cdot Z_{\text{TPA3}} \cdot (Z_{\text{TPA1}} + Z_{\text{TPA2}}) / (8 \cdot Z_{\text{TPA3}} - Z_{\text{TPA1}} - Z_{\text{TPA2}}) \quad (\text{K-2})$$

The differential mode characteristic impedance of signal twisted pair TPB is calculated as

$$Z_{\text{TPB}} = 4 \cdot Z_{\text{TPB3}} \cdot (Z_{\text{TPB1}} + Z_{\text{TPB2}}) / (8 \cdot Z_{\text{TPB3}} - Z_{\text{TPB1}} - Z_{\text{TPB2}}) \quad (\text{K-3})$$

The common mode characteristic impedance of signal twisted pair TPA is calculated as

$$Z_{\text{TPACM}} = Z_{\text{TPA3}} \quad (\text{K-4})$$

The common mode characteristic impedance of signal twisted pair TPB is calculated as

$$Z_{\text{TPBCM}} = Z_{\text{TPB3}} \quad (\text{K-5})$$

K.3.3 Signal pairs impedance limits

The test limits are

$$\begin{aligned} Z_{\text{TPA}} &= 110 \, \Omega \pm 6 \, \Omega \\ Z_{\text{TPB}} &= 110 \, \Omega \pm 6 \, \Omega \\ Z_{\text{TPA1}} &= Z_{\text{TPA2}} \pm 4\% \\ Z_{\text{TPB1}} &= Z_{\text{TPB2}} \pm 4\% \\ Z_{\text{TPACM}} &= 33 \, \Omega \pm 4 \, \Omega \\ Z_{\text{TPBCM}} &= 33 \, \Omega \pm 4 \, \Omega \end{aligned}$$

K.4 Signal pairs attenuation

The differential mode attenuation of the signal pairs shall be measured in the frequency domain using a network analyzer in the frequency range of 1 MHz to 500 MHz.

The three frequency points (100 MHz, 200 MHz, and 400 MHz) at which the cable is measured are only coincidentally the same approximate numbers at the three bit rates. These numbers were chosen because

- a) In fact, only one point is adequate in characterizing the cable. For reasonable accuracy, this test frequency point should be higher than the maximum fundamental frequency component that will be transmitted through the cable which in this case is about 200 MHz for an S400 (393.216 Mbit/s) transmission. For relatively simple and repeatable cable measurements, this test frequency should not be too high. 400 MHz was selected because it is about double the maximum frequency and the multiplication factor becomes $\sqrt{2}$. A higher number will make the measurements more difficult (many RF test fixtures have a 500 MHz maximum frequency limit), while a lower number will make computations more difficult (for example, if 300 MHz is chosen, calculations will need to be made using $\sqrt{1.5}$).
- b) The selection of the attenuation value at 400 MHz is such that the proposed cable construction will be adequate at 4.5 m under worst-case circumstances.
- c) The additional two data points (at 100 MHz and 200 MHz) are just a means of assistance. If pure skin effect cable attenuation is assumed, the attenuation numbers would be higher at 200 MHz (4.1 dB) and 100 MHz (2.9 dB), which will make the transceiver design harder. In reality, the cable attenuation is specified for the *cable assembly*. It contains not only the attenuation of the cable itself (the skin effect) but also the attenuation

of the connectors, the effect of the wire termination, etc. These numbers change faster than the square root of frequency, and this is the reason the attenuation values are more optimistic at 100 MHz and 200 MHz.

In order to perform this measurement using a single-ended test instrument, a differential excitation is generated using a 180° splitter. The output is reconverted to single ended using a 180° combiner. It is very important for the accuracy of this measurement to maintain identical electrical length for the two branches of the differential signal path (in figure K.6 the electrical length of the segments a, b, c, d, and e shall match the electrical length of the corresponding segments a', b', c, d', and e'; in figure K.7 the electrical length of the segments a, b, d, e, f, and g shall match the electrical length of the corresponding segments a', b, d', e', f', and g').

This test shall be repeated for both signal pairs. Because of the cross-connection of the two signal pairs at the two plugs, a simple way of accomplishing this goal is to maintain the test setup and to reverse the two ends of the cable. In order to represent this procedure, one plug of the assembly under test is arbitrarily labeled as “A” while the other plug is labeled as “B” in the following diagrams.

The test consists of four distinct steps to be performed in the order described in the following subclauses.

K.4.1 Signal pairs attenuation setup calibration

The “through” calibration shall be performed as shown in figure K.6. The electrical characteristics of the two interconnects (labeled c and c') between the two match pads are essential for the accuracy of this calibration. Their length shall be kept to a minimum, and the difference between their length shall be less than 1 mm. Practically, they can be implemented using two identical SMA female-to-female adaptors.

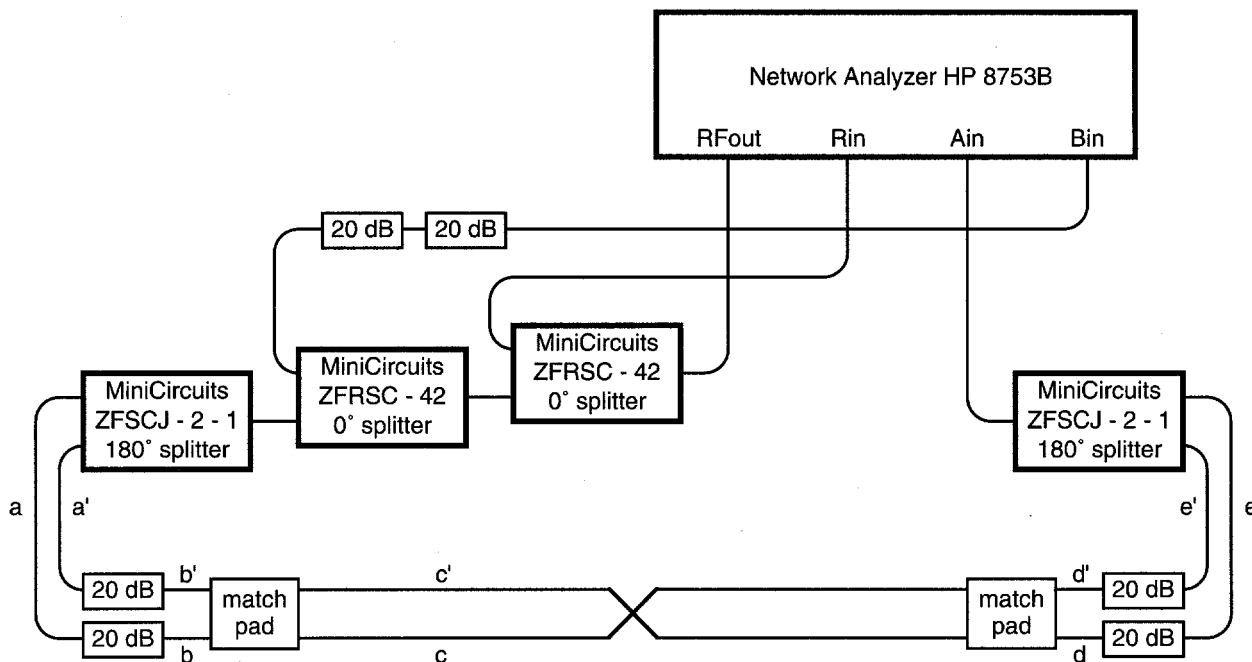


Figure K.6—Signal pairs attenuation and velocity setup calibration

The network analyzer shall use the following setup:

Format	Log Mag
Sweep	Log Freq
Averaging	16
Power	6 dBm
Start	1 MHz
Stop	500 MHz
Measure	A/B
Display	Data
Move data into memory	

K.4.2 ATPA

The signal pair A attenuation variation with frequency shall be tested as shown in figure K.7. Three data points shall be measured as follows: ATPA(100) is measured at 100 MHz, ATPA(200) is measured at 200 MHz, and ATPA(400) is measured at 400 MHz.

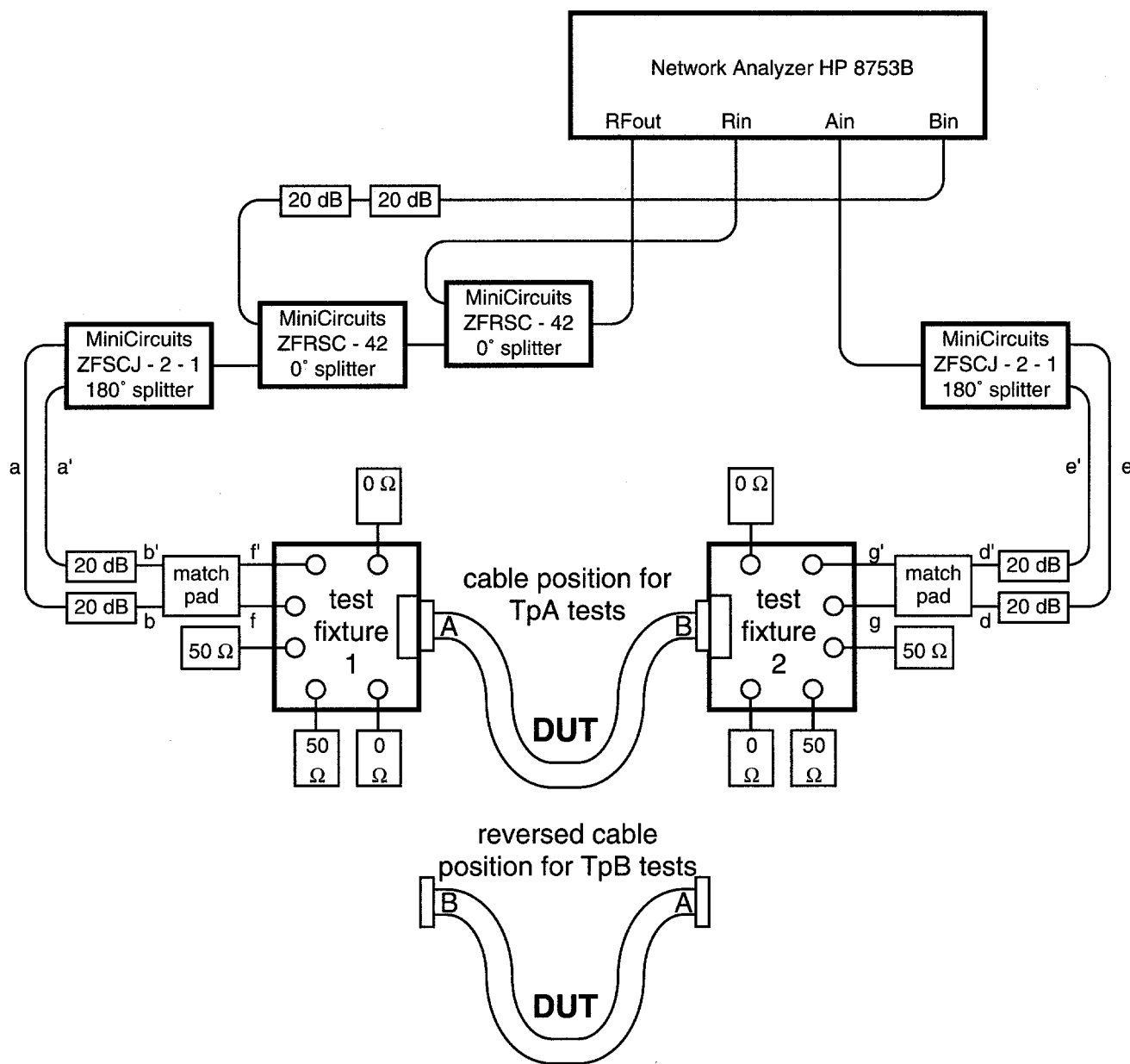


Figure K.7—Signal pair attenuation and velocity measurement

K.4.3 ATPB

The signal pair B attenuation variation with frequency shall be tested as shown in figure K.7 using the reversed cable position (“TPB tests”). Three data points shall be measured as follows: ATPB(100) is measured at 100 MHz, ATPB(200) is measured at 200 MHz, and ATPB(400) is measured at 400 MHz.

K.4.4 Signal pairs attenuation limits

The absolute maximum attenuation limit for the two signal twisted pairs is verified by the following relations:

$$\text{ATPA}(100) \leq 2.3 \text{ dB}$$

$$\text{ATPA}(200) \leq 3.2 \text{ dB}$$

$$\text{ATPA}(400) \leq 5.8 \text{ dB}$$

$$\text{ATPB}(100) \leq 2.3 \text{ dB}$$

$$\text{ATPB}(200) \leq 3.2 \text{ dB}$$

$$\text{ATPB}(400) \leq 5.8 \text{ dB}$$

K.5 Signal pairs velocity of propagation

The differential mode velocity of propagation of the signal pairs shall be measured in the frequency domain using a vector network analyzer in the frequency range of 1 MHz to 500 MHz. The calibration and measurement setup is identical to that used for signal pairs attenuation, with exception of the network analyzer setup.

The result of this test is directly dependent upon the length of the cable assembly under test. This length in meters is represented by “L” in the following test descriptions.

This test shall be repeated for both signal pairs. Because of the cross-connection of the two signal pairs at the two plugs, a simple way of accomplishing this goal is to maintain the test setup and to reverse the two ends of the cable. In order to represent this procedure, one plug of the assembly under test is arbitrarily labeled as “A” while the other plug is labeled as “B” in the following diagrams.

The test consists of four distinct steps to be performed in the order described in the following subclauses.

K.5.1 Signal pairs velocity of propagation setup calibration

The “through” calibration shall be performed as shown in figure K.6. The electrical characteristics of the two interconnects (labeled c and c') between the two match pads are essential for the accuracy of this calibration. Their length shall be kept to a minimum, and the difference between their electrical length shall be less than 1 mm. The DUT interconnect segments labeled f, f', g, and g' in figures K.6 and K.7 are included in the calibration interconnects c and c'. Practically, they can be implemented using identical SMA adaptors.

The network analyzer shall use the following setup (items different from the attenuation test are shown in bold type):

Format	Delay
Sweep	Log Freq
Averaging	32
Smoothing aperture	5%
Power	6 dBm
Start	1 MHz
Stop	500 MHz
Measure	A/B
Display	Data
Move data into memory	

K.5.2 VTPA

The signal pair A velocity of propagation variation with frequency shall be tested as shown in figure K.7. Three data points shall be measured as follows: VTPA(50) is measured at 50 MHz, VTPA(100) is measured at 100 MHz, and VTPA(200) is measured at 200 MHz.

The average velocity of propagation for the signal pair A is calculated as

$$VTPA = (VTPA(50) + VTPA(100) + VTPA(200)) / (3 \cdot L) \quad (\text{K-6})$$

K.5.3 VTPB

The signal pair B velocity of propagation variation with frequency shall be tested as shown in figure K.7 using the reversed cable position (“TPB tests”). Three data points shall be measured as follows: ATPB(50) is measured at 50 MHz, ATPB(100) is measured at 100 MHz, and ATPB(200) is measured at 200 MHz.

The average velocity of propagation for the signal pair B is calculated as

$$VTPB = (VTPB(50) + VTPB(100) + VTPB(200)) / (3 \cdot L) \quad (\text{K-7})$$

K.5.4 Signal pairs velocity of propagation limits

The absolute minimum velocity of propagation limit for the two signal twisted pairs is verified by the following relations:

$$\begin{aligned} VTPA &\leq 5.05 \text{ ns/m} \\ VTPB &\leq 5.05 \text{ ns/m} \end{aligned}$$

K.6 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 a vector network analyzer in the frequency range of 1 MHz to 500 MHz.

While the signal pairs relative skew can be calculated from the velocity of propagation measurement described in K.5, the very high accuracy required by the intended application of this cable assembly demands a separate, high-resolution test.

In order to perform this measurement using a single-ended test instrument, a differential excitation is generated using a 180° splitter. The output is reconverted to single ended using a 180° combiner. It is very important for the accuracy of this measurement to maintain identical electrical length for the two branches of the differential signal path (in figure K.8, the electrical length of the segments b1, b3, c1, c3, f1, f3, g1, and g3 shall match the electrical length of the corresponding segments b2, b4, c2, c4, f2, f4, g2, and g4; the electrical length of the segments b1, b3, c1, c3, d1, d3, e1, e3, f1, f3, g1, and g3 shall match the electrical length of the corresponding segments b2, b4, c2, c4, d2, d4, e2, e4, f2, f4, g2, and g4).

The test consists of three distinct steps to be performed in the order described in the following subclauses.

K.6.1 Signal pairs skew setup calibration

The “through” calibration shall be performed as shown in figure K.8. The electrical characteristics of the four calibration interconnects (labeled i1, i2, i3, and i4) between the match pads are essential for the accuracy of this

procedure. Their length shall be kept to a minimum, and the difference between their electrical length shall be less than 1 mm. The DUT interconnect segments labeled d1, d2, d3, d4, e1, e2, e3, and e4 in figure K.9 are included in the calibration interconnects i1, i2, i3, and i4. Practically, they can be implemented using identical SMA adaptors.

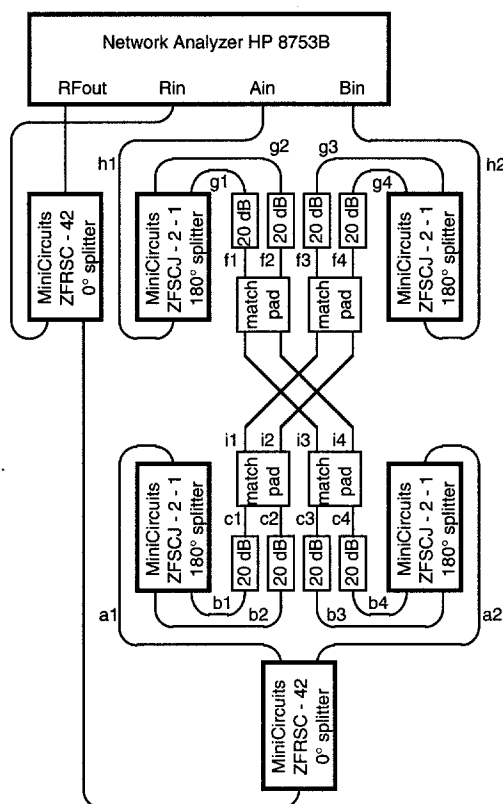


Figure K.8—Skew setup calibration

The skew is measured by comparing the signal propagation delay on the two twisted pairs. The calibration process attempts to remove the difference in electrical length between the two measurement paths. A close matching of these two paths by construction will help the setup accuracy. Accordingly, in figure K.8 the electrical length of the connection segments marked a1, b1, b2, c1, c2, f1, f2, g1, g2, and h1 shall match the electrical length of the corresponding segments a2, b3, b4, c3, c4, f3, f4, g3, g4, and h2. Similarly, in figure K.9 the electrical length of the connection segments marked a1, b1, b2, c1, c2, d1, d2, e1, e2, f1, f2, g1, g2, and h1 shall match the electrical length of the corresponding segments a2, b3, b4, c3, c4, d3, d4, e3, e4, f3, f4, g3, g4, and h2.

The network analyzer shall use the following setup:

Format	Delay
Sweep	Log Freq
Averaging	32
Smoothing aperture	10%
Power	6 dBm
Start	1 MHz
Stop	500 MHz
Measure	A/B
Display	Data/Memory

K.6.2 Signal pairs skew test procedure

The signal pairs relative propagation skew shall be tested as shown in figure K.9. Three data points shall be measured as follows: S(50) is measured at 50 MHz, S(100) is measured at 100 MHz, and S(200) is measured at 200 MHz.

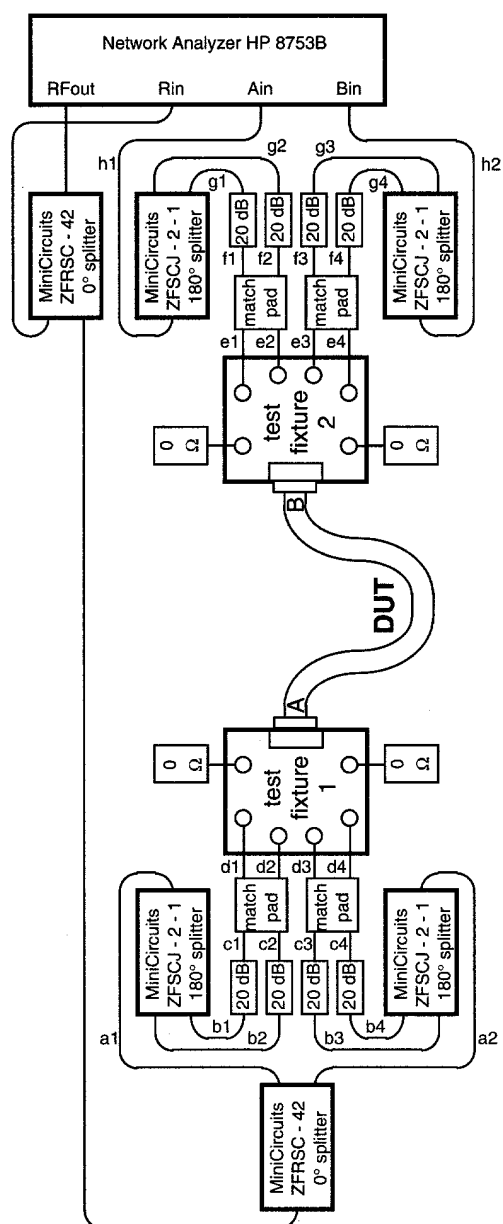


Figure K.9—Skew measurement

The average value for the signal pairs relative propagation skew is calculated as

$$S = (S(50) + S(100) + S(200))/3 \tag{K-8}$$

K.6.3 Signal pairs skew limits

The absolute maximum relative propagation skew of the two signal pairs is verified by the following relation:

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

K.7 Power pair characteristic impedance

The differential mode characteristic impedance shall be measured in the time domain using a single-ended TDR with an edge rate of less than 0.1 ns. The differential mode impedance value is calculated from the three single-ended measurements described in the following subclauses. The result of each single-ended measurement is calculated as the average of the impedance measured at two points along the cable. These points are selected at 1 ns and at 2.5 ns along the cable from the plug closest to the launching connector. It should be noted that, because the TDR displays the round-trip propagation delay, the measurements shall be made at 2 ns and 5 ns from the plug closest to the launching connector as measured by the TDR instrument.

This test procedure uses the same test setup described in K.3, only with a different connection matrix.

K.7.1 Power pair impedance setup calibration—short and load

This calibration should be performed as shown in figure K.4 using the calibration algorithms built into the TDR equipment suggested in J.7.

K.7.2 Power pair impedance test procedure

Using the test configuration described in figure K.5 and the connection matrix shown in table K.2, the various characteristic impedances of the power pair shall be measured in two points and the results averaged.

Table K.2—Connection matrix for power pair impedance tests

Measured value	Fixture 1						Fixture 2					
	VP	TPA	TPA*	TPB	TPB*	VG	VP	TPA	TPA*	TPB	TPB*	VG
Single-ended VP (Z_{VP})	TDR	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω		50 Ω	50	50 Ω	50	0 Ω
Single-ended VG (Z_{VG})	0	50 Ω	50	50 Ω	50	TDR	0 Ω	50	50 Ω	50	50 Ω	∞
Common-mode power pair (Z_{VPG})	TDR	50 Ω	50 Ω	50 Ω	50 Ω	TDR	∞	50 Ω	50 Ω	50 Ω	50 Ω	∞

$Z_{Vn}(2)$ is measured along the cable at 2 ns from the plug connected to test fixture 1.

$Z_{Vn}(5)$ is measured along the cable at 5 ns from the plug connected to test fixture 1.

Z_{Vn} is calculated as

$$Z_{Vn} = Z_{Vn(2)} + Z_{Vn(5)} / 2 \quad (\text{K-9})$$

“Vn” is VP, VG, or VPG.

The differential mode characteristic impedance of the power pair is calculated as

$$Z_V = 4 \cdot Z_{VPG} \cdot (Z_{VP} + Z_{VG}) / (8 \cdot Z_{VPG} - Z_{VP} - Z_{VG}) \quad (\text{K-10})$$

The test limit is

$$Z_{TPA} \leq 65 \Omega$$

K.7.3 Power pair dc resistance

The dc resistance of the power wires is measured with a milliohmmeter capable of “four-wire” resistance measurements.

The accuracy of this measurement depends upon the connection segments labeled “a” and “b” in figure K.10. Their length and their dc resistance shall be kept to a minimum. In a noisy environment, the accuracy of this measurement can be improved by connecting the Guard terminal of the test instrument to the shield of one of the cable test fixtures.

The test consists of four steps to be performed in the order described in the following subclauses.

K.7.4 DC resistance setup calibration

Previous to the start of the measurement, the test instrument shall be warmed up for at least 1 h followed by a resistance auto calibration procedure (ACAL OHM).

The setup calibration resistance (RCAL) shall be measured as shown in figure K.10. It is essential to maintain a very low-resistance contact between the connection segments a and b during this measurement.

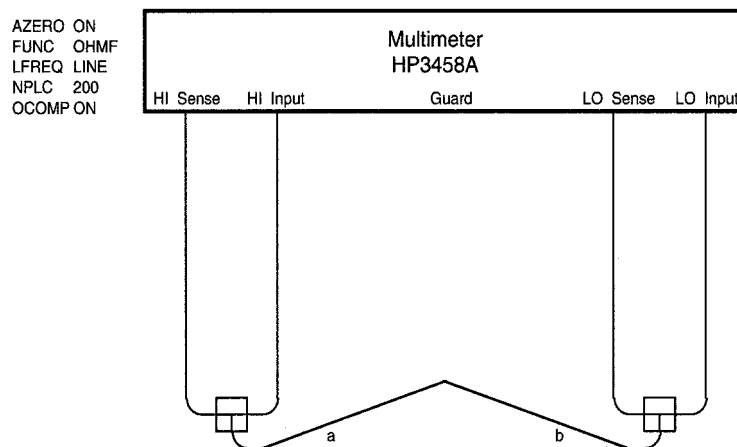


Figure K.10—Power pair dc resistance setup calibration

K.7.5 DC resistance test procedure

Using the test configuration described in figure K.11 and the connection matrix shown in table K.2, the DC resistance of the power pair shall be measured.

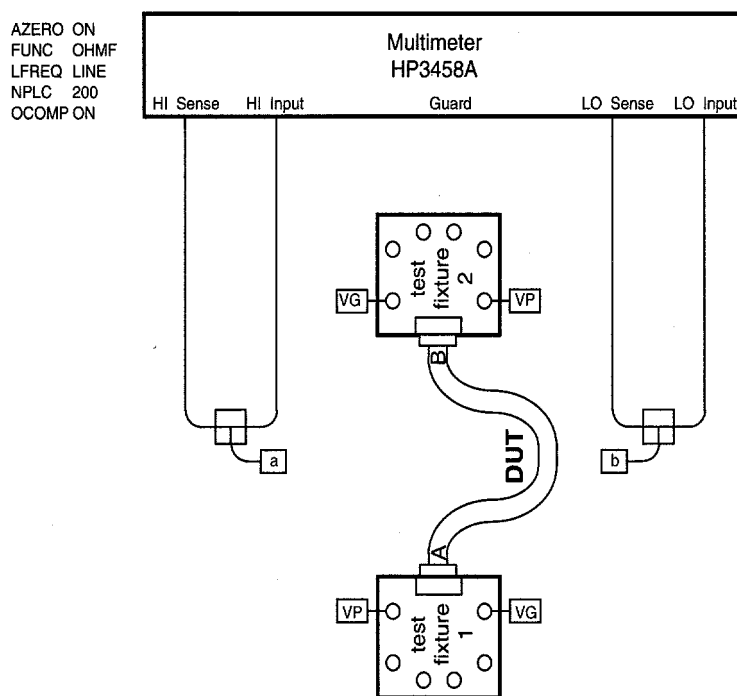


Figure K.11—Power pair resistance measurement

Table K.3—Connection matrix for power pair resistance tests

Measured value	Fixture 1						Fixture 2					
	VP	TPA	TPA*	TPB	TPB*	VG	VP	TPA	TPA*	TPB	TPB*	VG
Power wire resistance (R_{VP})	a		∞	∞		∞	a		∞	∞		∞
Ground wire resistance (R_{VG})	∞		∞	∞		b	∞	∞		∞	∞	b

K.7.6 DC resistance limits

The test limits are

$$RPV - RCAL \leq 0.333 \Omega$$

$$RPG - RCAL \leq 0.333 \Omega$$

K.8 Crosstalk

The pair-to-pair crosstalk shall be measured in the frequency domain using a network analyzer in the frequency range of 1 MHz to 500 MHz.

K.8.1 Crosstalk setup calibration

The through calibration can be performed as shown in figure K.12. The attenuation introduced by the calibration interconnect that replaces the DUT shall be kept to a minimum. Practically, it can be implemented using an SMA adaptor.

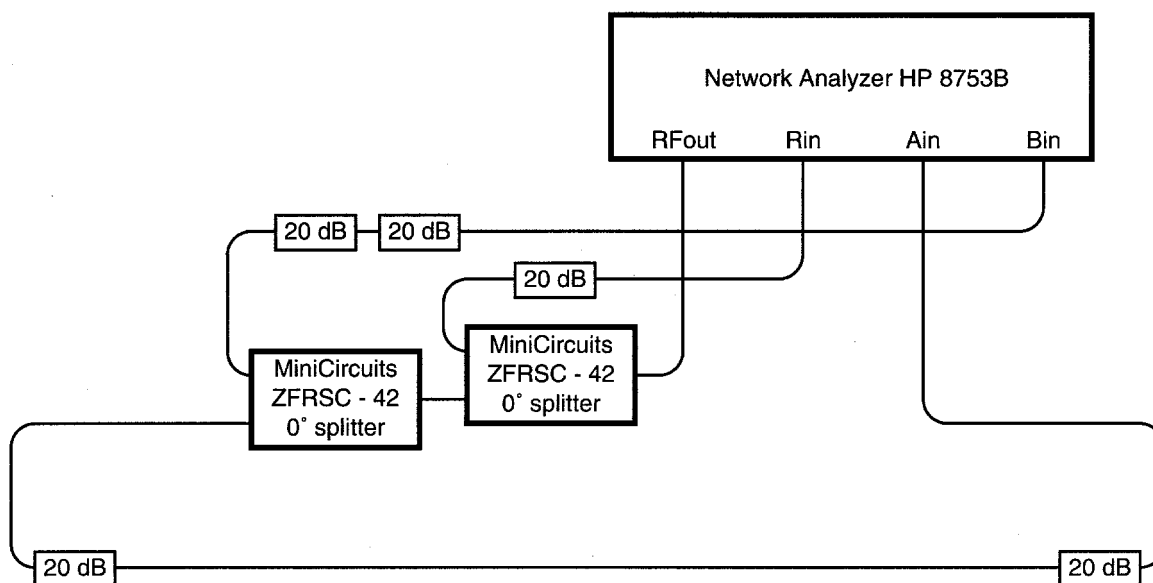


Figure K.12—Crosstalk setup calibration

The network analyzer shall use the following setup:

Format	Log Mag
Sweep	Log Freq
Averaging	16
Smoothing aperture	10%
Power	25 dBm
Start	1 MHz
Stop	500 MHz
Measure	A/B
Display	Data
Move data into memory	

K.8.2 Crosstalk test procedure

Using the test configuration described in figure K.11 and the connection matrix shown in table K.2, the crosstalk between signal pairs and between a signal pair and the power pair shall be measured.

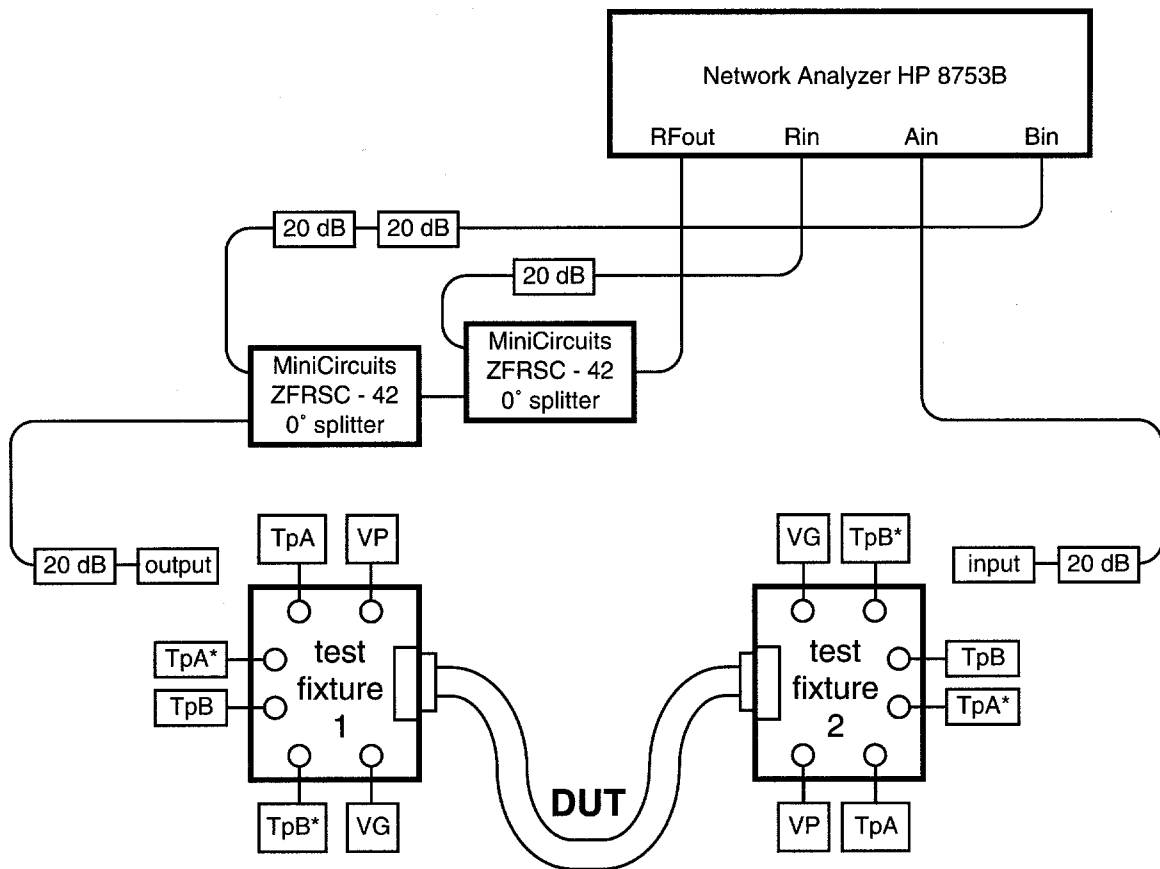


Figure K.13—Crosstalk measurement

Table K.4—Connection matrix for crosstalk tests

Measured value	Fixture 1						Fixture 2					
	VP	TPA	TPA*	TPB	TPB*	VG	VP	TPA	TPA*	TPB	TPB*	VG
Crosstalk between TPA and TPB (X_{AB})	0 Ω	Out	50 Ω	50 Ω	50 Ω	0 Ω	0 Ω	In	50	50 Ω	50	0 Ω
Crosstalk between TPA and TPB* (X_{AB^*})	0 Ω	Out	50 Ω	50 Ω	50 Ω	0 Ω	0 Ω	50 Ω	In	50	50 Ω	0
Crosstalk between TPA* and TPB (X_{A^*B})	0 Ω	50	Out	50 Ω	50	0 Ω	0	In	50 Ω	50	50 Ω	0
Crosstalk between TPA* and TPB* ($X_{A^*B^*}$)	0 Ω	50	Out	50 Ω	50	0 Ω	0	50 Ω	In	50 Ω	50 Ω	0 Ω
Crosstalk between VP and TPA (X_{PA})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	In	50	50 Ω	50	0 Ω
Crosstalk between VP and TPA* (X_{PA^*})	Out	50 Ω	50 Ω	50 Ω	50 Ω	0 Ω	50 Ω	50 Ω	In	50 Ω	50 Ω	0 Ω
Crosstalk between VP and TPB (X_{PB})	Out	50	50 Ω	50	50 Ω	0	50 Ω	50	50 Ω	In	50 Ω	0 Ω
Crosstalk between VP and TPB* (X_{PB^*})	Out	50	50 Ω	50	50 Ω	0	50 Ω	50	50 Ω	50	In	0 Ω
Crosstalk between VG and TPA (X_{PA})	0	50 Ω	50	50 Ω	50	Out	0 Ω	In	50 Ω	50 Ω	50 Ω	50 Ω
Crosstalk between VG and TPA* (X_{PA^*})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0	50 Ω	In	50 Ω	50 Ω	50 Ω
Crosstalk between VG and TPB (X_{PB})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	50 Ω	50 Ω	In	50 Ω	50 Ω
Crosstalk between VG and TPB* (X_{PB^*})	0 Ω	50 Ω	50 Ω	50 Ω	50 Ω	Out	0 Ω	50 Ω	50 Ω	50 Ω	In	50

K.8.3 Crosstalk limits

The test limits are

- $X_{AB} \leq -26$ dB
- $X_{AB^*} \leq -26$ dB
- $X_{A^*B} \leq -26$ dB
- $X_{A^*B^*} \leq -26$ dB
- $X_{PA} \leq -26$ dB
- $X_{PA^*} \leq -26$ dB
- $X_{PB} \leq -26$ dB
- $X_{PB^*} \leq -26$ dB
- $X_{GA} \leq -26$ dB
- $X_{GA^*} \leq -26$ dB
- $X_{GB} \leq -26$ dB
- $X_{GB^*} \leq -26$ dB

Annex L Shielding effectiveness and transfer impedance testing

(Informative)

L.1 Content

This annex allows the transfer impedance (Z_t) of electrical connectors to be determined from dc to 500 MHz. It measures the leakage of connector assemblies located between a bulkhead panel and a shielded cable.

L.2 Definitions

connector transfer impedance (Z_t): The ratio of the longitudinal voltage that appears on the outside of the connector under test to a known common-mode current impressed on the inside conductors.

L.3 Test equipment

A vector or scalar network analyzer shall be used. A spectrum analyzer and tracking generator may be used if provision has been made for dividing the magnitudes of two traces (subtracting in decibels). Equipment shall be capable of generating plots directly or from magnetic media, if used.

Measured results shall be reported in decibels from 1 Ω ($\text{dB}=20\log(Z_t/(1 \Omega))$). More negative values represent better performance. Zero decibels is 1 Ω . Positive results represent insignificant shielding. Each 20 dB represents a ten-fold change in transfer impedance.

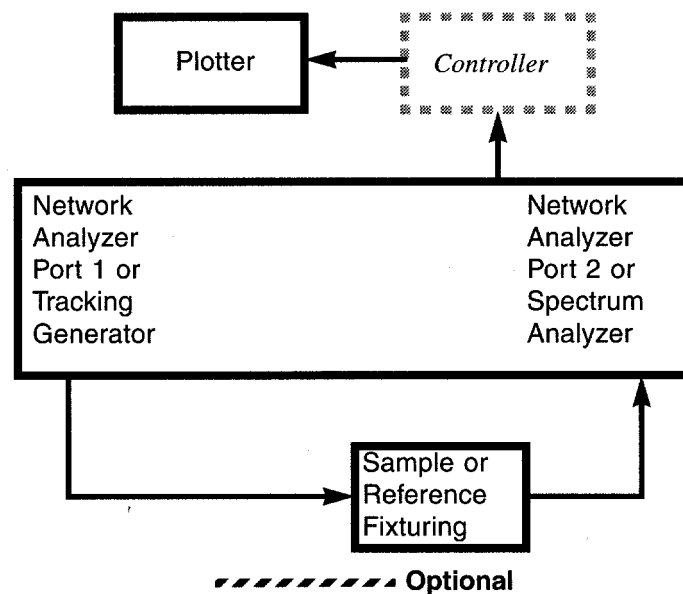


Figure L.1—Equipment block diagram

L.4 Theory

L.4.1 Reference measurement

This technique relies on comparing two measurements that differ only in where the drive energy is placed. In the reference measurement (see figure L-2), the radio frequency (RF) current flows through the $1\ \Omega$ standard. The resulting voltage drives the sample on the outside, and the resulting power is measured at port 2 [or this can be measured as S_{21} (reference)]. This voltage is equal to the voltage that would be generated if the sample had a $1\ \Omega$ transfer impedance at all frequencies.

This simple resistance is made to mimic a transfer impedance. Thus, the reference measurement only includes the fixturing response when the sample is mounted in it.

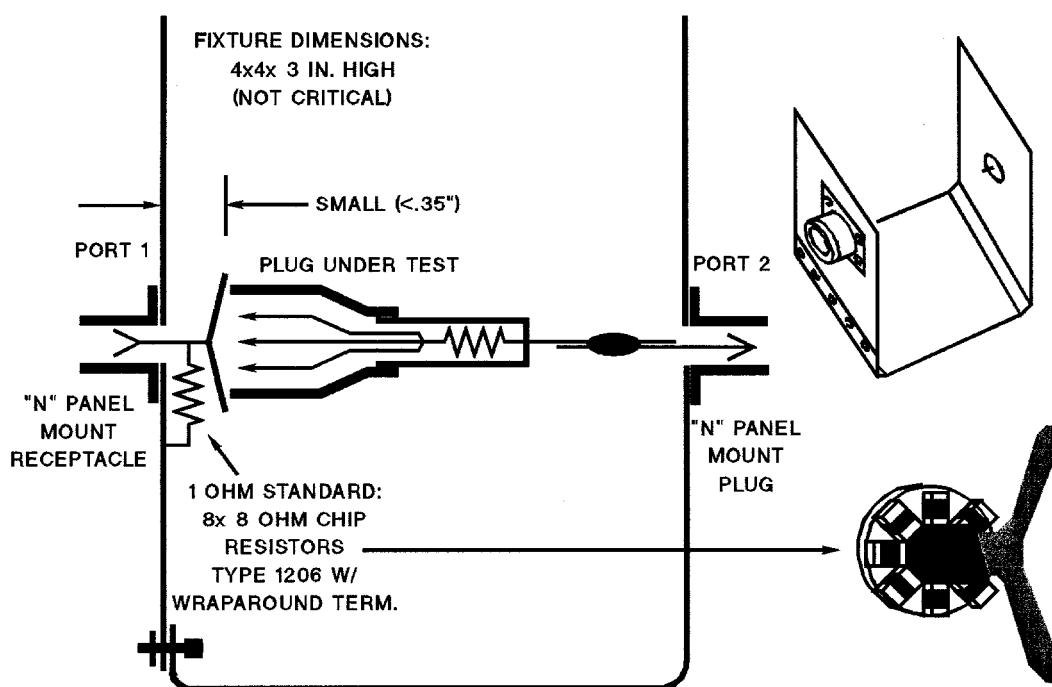


Figure L.2—Reference measurement fixturing

L.4.2 Sample measurement

The second measurement (sample measurement, figure L-2) applies the same amount of incident RF power to the inside of the sample. The fixturing has exactly the same dimensions. The current from this incident power passes through the connector under test transfer impedance, producing a voltage along the length of the sample. The resulting power is measured at port 2 of figure L-2 [or this can be measured as S_{21} (sample)]. This measurement includes the fixturing response previously measured times the transfer impedance of the sample.

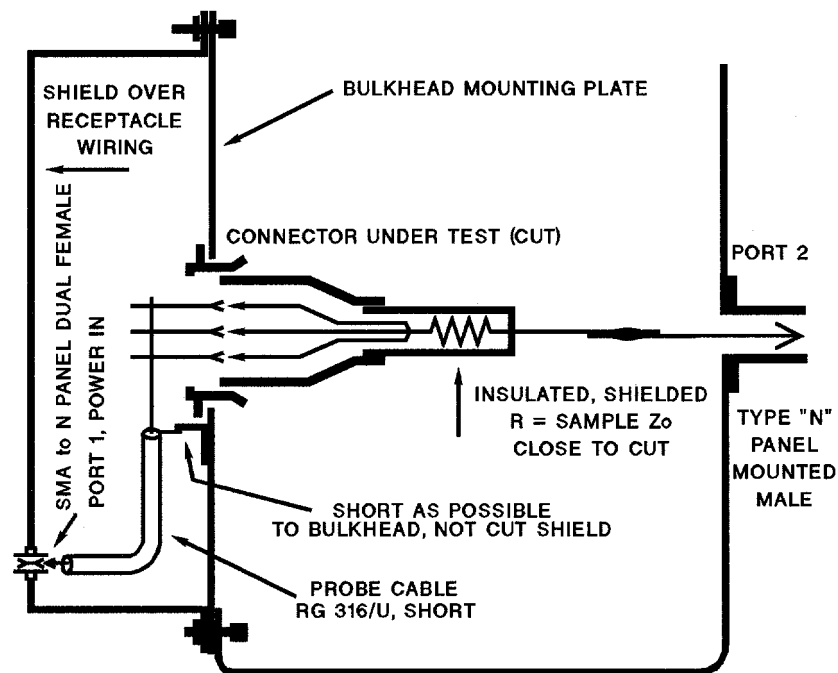


Figure L.3—Sample measurement fixturing

L.4.3 Calculations

The transfer impedance can be derived from these two measurements. The sample measurement results are divided by the reference results at each frequency (subtract decibels), leaving just the sample transfer impedance. Finally, the results are multiplied by a frequency independent correction factor (CF). Even with identical RF drive levels, the current through the 1 standard of the reference measurement will differ from the inside current of the sample measurement.

L.5 Sample preparation

L.5.1 Panel-mounted connector sample

The connector shall be mounted in the center of a bulkhead mounting plate (see figure L-2). Connect all contacts as close to the bulkhead as possible. A 1-in wide copper strap shall be soldered to the bulkhead mounting plate close to the CUT. The free end should be very close to the bused panel mount connector conductors. The shield of a short probe cable shall be soldered to the strap to form a low-inductance path to the bulkhead, not the panel-mount connector shield. This connection has to be as short as possible. The center conductor shall be soldered to the bused panel-mount connector conductors. This connection should be as short as possible. The fixturing of figure L-2 shall be assembled. Ensure that joined metal surfaces are clean for a good connection.

L.5.2 Measure sample Z_0 with TDR

A sample connector shall be mounted on several inches of shielded cable. All conductors in this connector shall terminate a conductor in the cable. Mate to panel mounted connector sample fixtures in clause L.5.1. Use time domain reflectometry (TDR) to determine the voltage reflection coefficient of the mated pair. Estimate an average value if necessary. This will be used to calculate a correction factor (CF).

L.5.3 Cable-mounted connector sample

The cable on each sample shall be cut about 1 in from the rear. The outer jacket shall be removed without nicking the braid, leaving 0.2 in of outer jacket. The back braid shall be folded over the remaining outer jacket. The second foil shield shall be removed, if present. Take care to avoid stressing the connector shield braid capture region. All internal conductors shall be stripped as close to the braid as practical. All internal conductors and solder shall be twisted together. Soldered wires shall be cut to 0.1 in. A noninductive (metal film) termination resistor shall be soldered to this. This joint shall be wrapped with high-temperature insulating tape. The termination resistor shall be chosen to give a voltage reflection coefficient close to that measured in L.5.2. The joint and termination resistor shall be wrapped with copper tape. The braid shall be dressed up over the copper tape. The braid and copper tape shall be wrapped up to the free resistor lead with small-gauge hook-up wire. These shall be soldered together to prevent any RF leakage.

L.6 Procedure

- a) Do a full two-port calibration if using a vector network analyzer for improved accuracy.
- b) Get a noise floor plot (L.7).
- c) Prepare panel-mounted connector sample(s) (L.5.1).
- d) Measure sample Z_o (L.5.2). Note the reflection coefficient (p) for use in step h).
- e) Prepare cable-mounted samples (L.5.3).
- f) Do a reference measurement (L.4.1). Solder the electrical connection at the right so that pressure is applied at the mechanical connection at the left (the fixture connection to the mating interface). Measure S_{21} (reference) expressed in decibels, or measure the power at the spectrum analyzer in decibels per minute. The frequency range shall include 30 MHz to 500 MHz, but may include more.
- g) Do the sample measurements (L.4.2). With the same cables, substitute the sample measurement fixturing of figure L-2 for the reference measurement fixturing of step f) (figure L-2). Mate the connectors under test and connect to the type “N” connector at the right. Measure S_{21} (sample) expressed in decibels, or measure the power at the spectrum analyzer in decibels per minute while driving the sample with the same forward power (into fixture port 1, figure L-2) as used in step f).
- h) Calculate the correction factor. In decibels it would be

$$CF \text{ (dB)} = 20 \log(2/(1 - p)) \quad (\text{L-1})$$

where

p is the voltage reflection coefficient of the termination resistor, determined in step e) (see L.5.3)

- i) The vertical axis of the trace can now be labeled (in units of decibels-ohm):

$$\begin{aligned} Z_t \text{ (dB-}\Omega\text{)} &= \text{dBm (sample)} - \text{dBm (reference)} + CF \text{ (dB)} \\ &= S_{21} \text{ (sample)} - S_{21} \text{ (reference)} + CF \text{ (dB)} \end{aligned}$$

where S_{21} is expressed in decibels

NOTE — The subtraction above can be done by many instruments real time. If so, the reference gradicule of the subtracted results can be assigned a value of $(-1)CF(\text{dB})$, which would make it equal to 0 dB Ω . A controller may do these calculations.

- j) Plot transfer impedance in log format, save plot file, and record single frequency results.
- k) Check that all points on the transfer impedance plot (trace) exceed the noise floor plot (see L.7) by at least 10 dB. Note invalid regions in the plot data.

L.7 “Noise floor” plot

If the samples have very good shielding, the fixturing shielding may be inadequate or the receiver may be picking up ambient signals. Before samples are measured, some idea of this level has to be available. This shall be obtained with the noise floor plot fixturing of figure L-4.

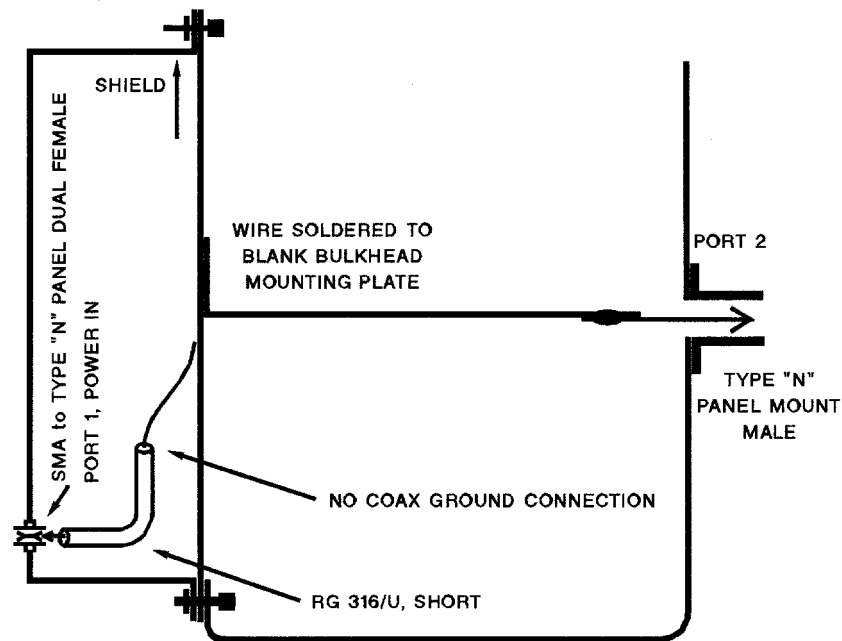


Figure L.4—Noise floor check fixturing

The noise floor plot is obtained just like a transfer impedance measurement but with a solid panel and no connector under test. A reference measurement is first made using a wire instead of a cable-mounted sample. The sample measurement is made as in figure L-4. The sample measurement is then divided by the reference measurement (subtract decibels). All transfer impedance data that does not exceed this ratio by at least 10 dB shall be reported as inaccurate data.

L.8 Documentation

L.8.1 Plots and magnetic files

- Company name
- Test number
- Date
- Sample identification and test conditions
- Correction factor in decibels, or termination resistance used

L.8.2 Test report

- Plots, if requested (log format, including 30–500 MHz data)
- Single-frequency results (30 MHz and 159 MHz recommended)
- Noise floor plot (note any data that fails the noise floor requirement)
- Equipment used
- Termination resistor values used

L.9 Performance

For the Serial Bus external connector, the transfer impedance values shown in table L-1 are needed.

Table L.1—Transfer impedance performance requirements

Frequency	Value
30 MHz	-25 dB Ω
159 MHz	-16 dB Ω
500 MHz	-10 dB Ω