

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 William 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	David V. James	Scott Smyers
Bob Bellino	James Kuo	Jeff Stai
Charles Brill	Lawrence J. Lamers	Roger Van Brunt
Andy Carter	Gene Milligan	Hans H. Wang
Greg Floryance	Jay Neer	Harvey Watersdorf
Giles Frazier	Florin Oprescu	Alan Wetzell
Bill Ham	Thomas J. Potyraj	Bob Whiteman
Norm Harris	Doug Riemer	Michael Wingard
David Hatch	Ron Roberts	

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	Dhiru Desai	Mark Hassel
John Atwood	Frank Duffy	Rick Heidick
James Baldwin	Sam Duncan	Dan Hillman
Peter Bartlett	W. P. Evertz	Mark Jander
Bryan Bell	Stephen Finch	Greg Kite
Phil Bolton	Mike Foster	Lawence Kopp
Paul L. Borrill	Richard Fryer	Ralph Lachenmaier
David Brief	Bob Gannon	Michael Lazar
Ed Cady	Edward A. Gardner	Wendell Lengefeld
Ramiro Calvo	Charles Grant	Fred Leung
Mike Chastain	Michael Griffin	John Lohmeyer
Kim Clohessy	David B. Gustavson	John Lopata
Forrest Crowell	Steffen Hagene	Ivy Lui
Wayne Davis	Emil Hahn	Don May
Tom Debiec	Ken Hallam	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 Bezirtoglou
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).

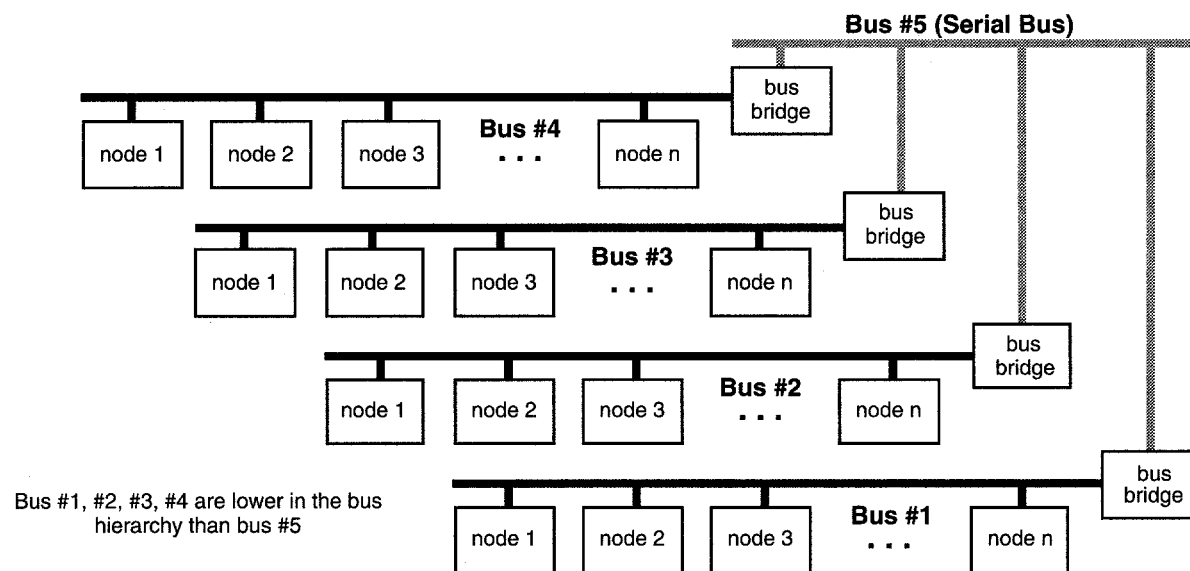


Figure 1.1—Example hierarchical bus topology

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:

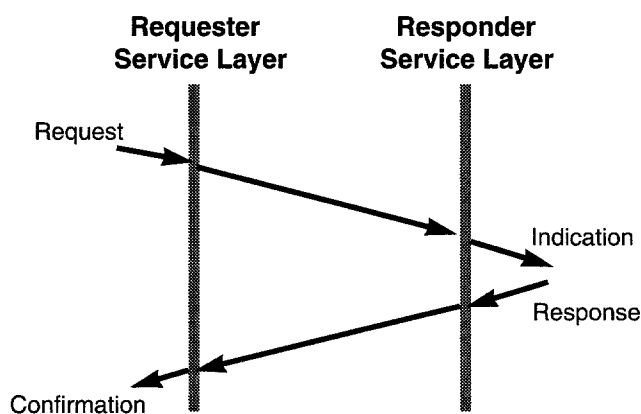


Figure 1.2—Service model

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.

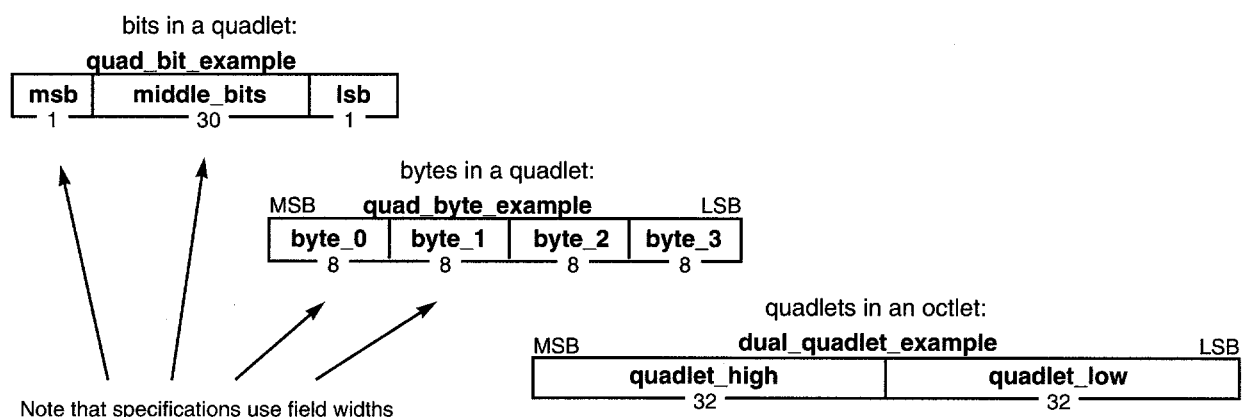


Figure 1.3—Bit and byte ordering

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.

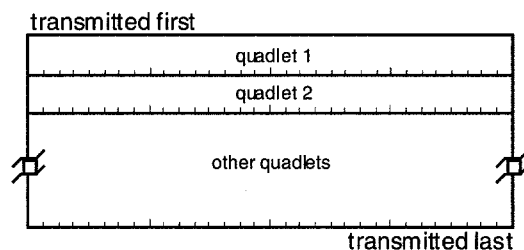


Figure 1.4—Example packet format

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.

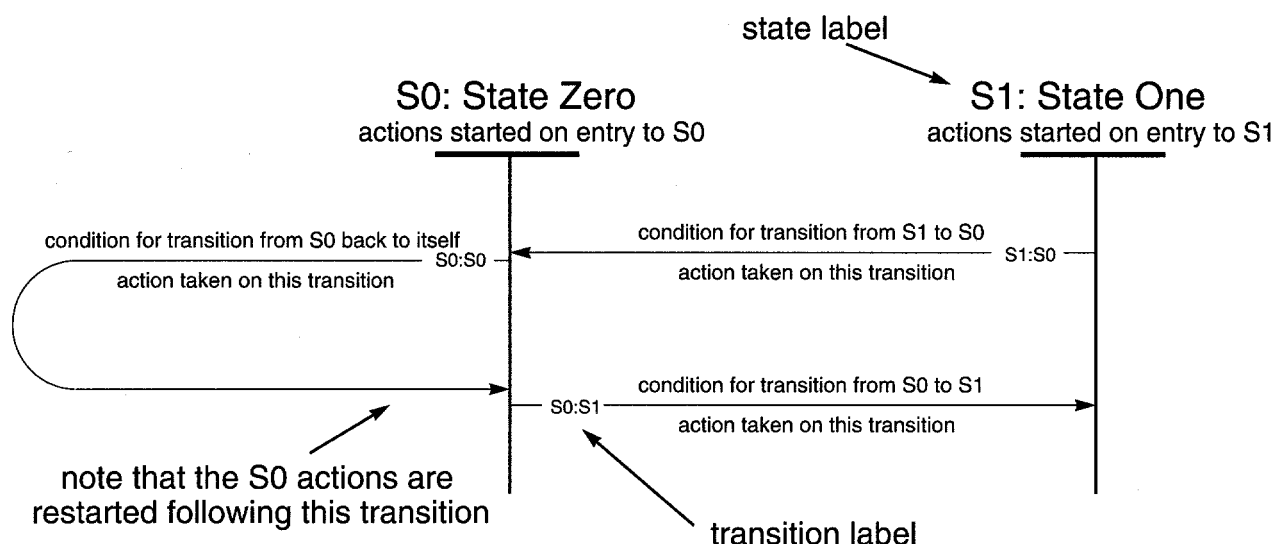


Figure 1.5—State machine example

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.

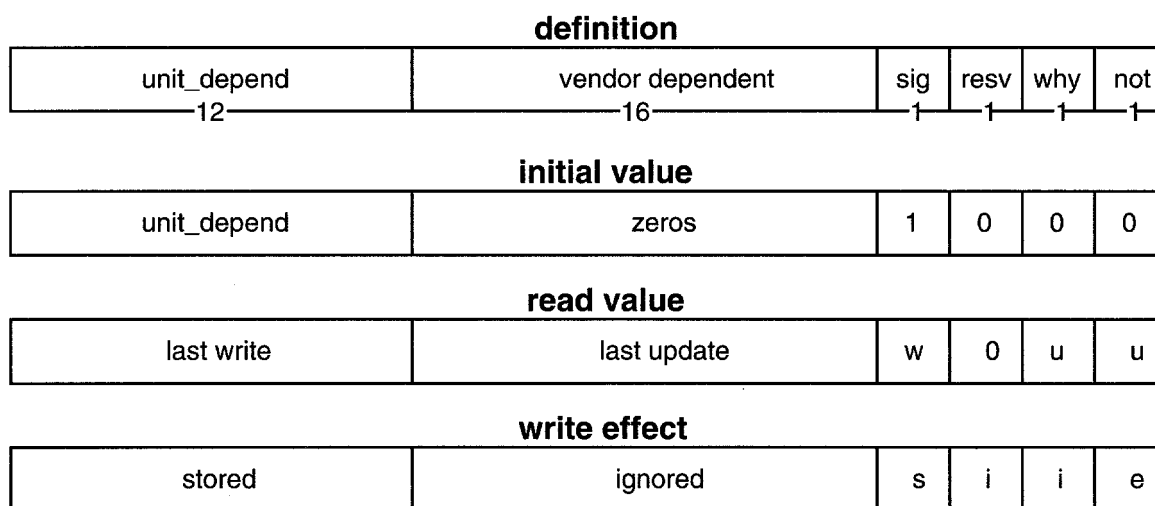


Figure 1.6—CSR format specification (example)

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_depend 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.

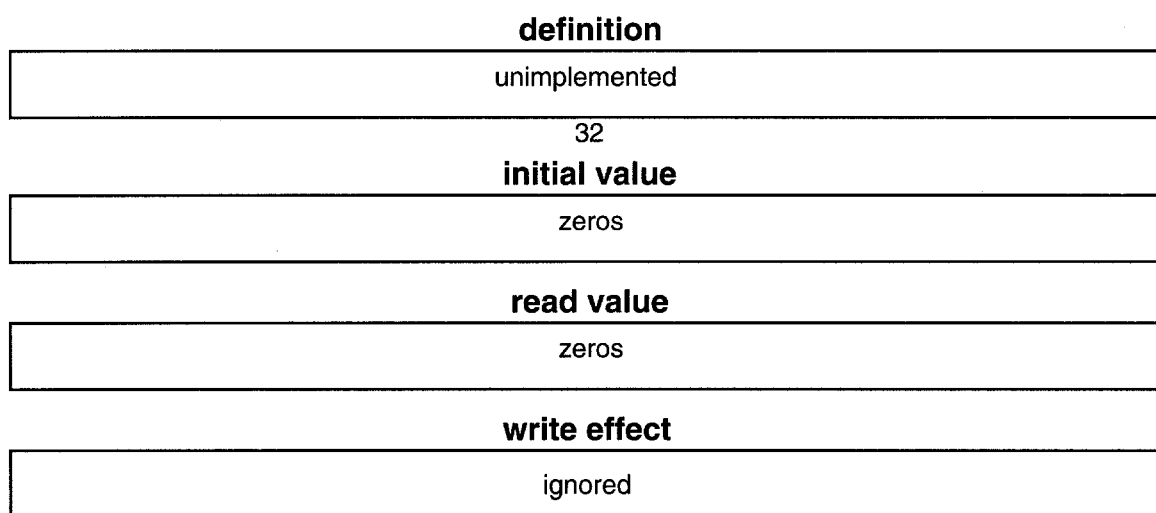
Table 1.7—Write value fields

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.

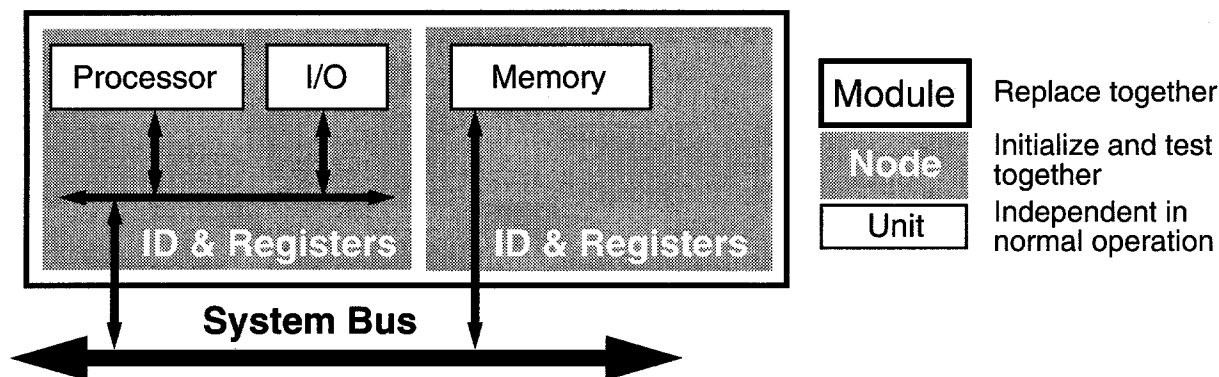


Figure 3.1—Module architecture

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.

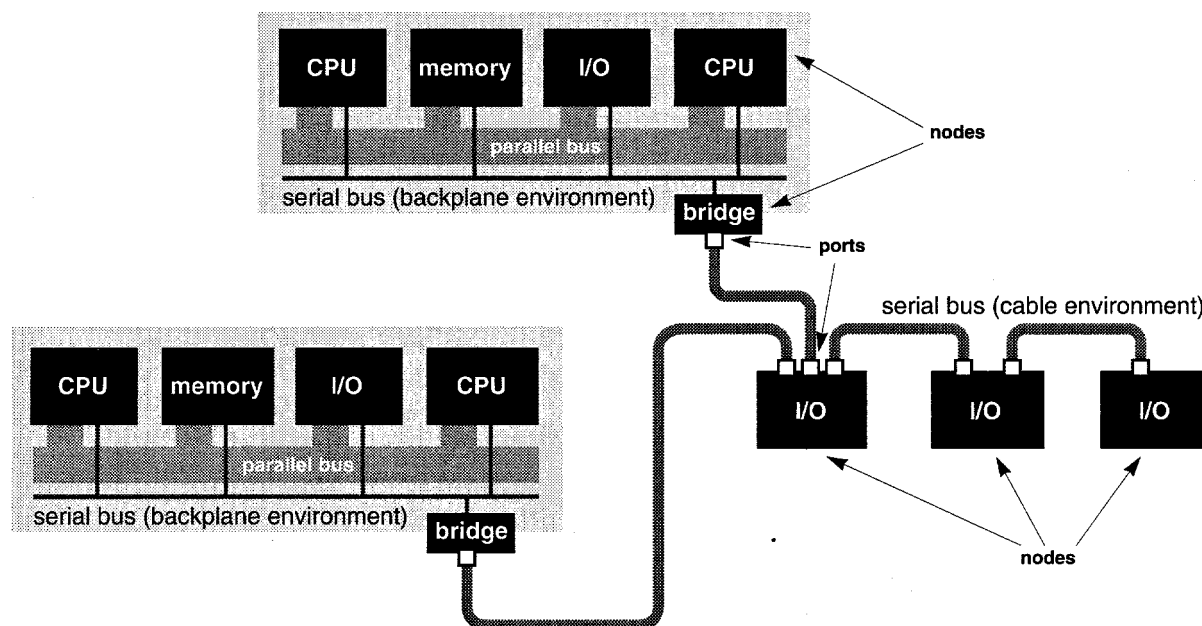


Figure 3.2—Serial Bus physical topology

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.

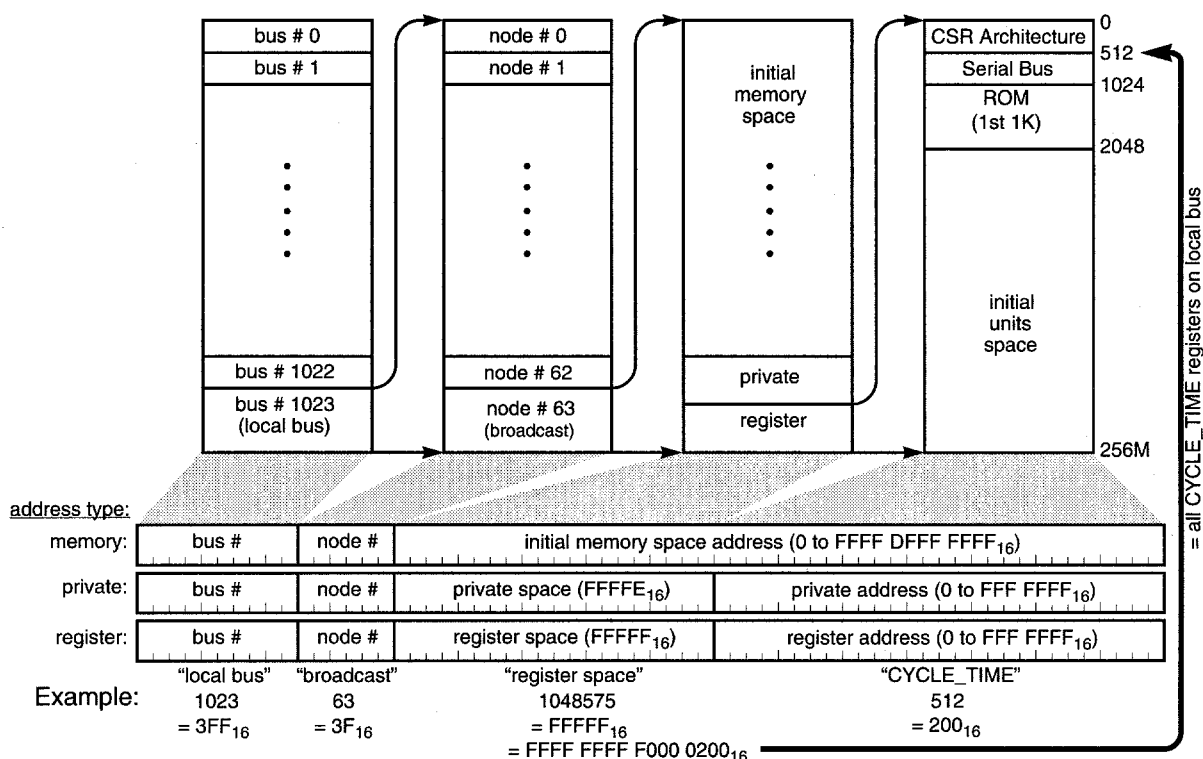
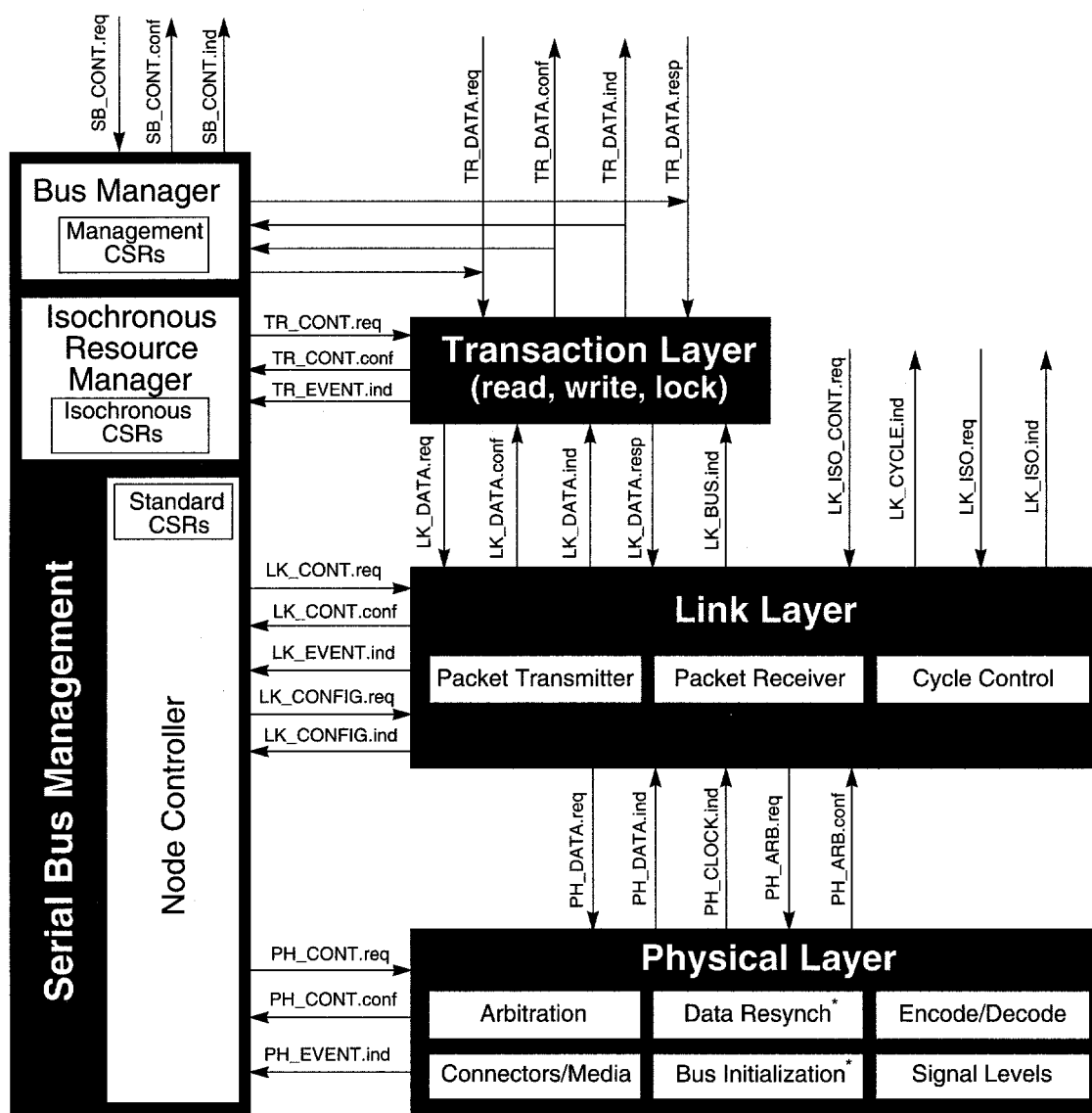


Figure 3.3—Serial Bus addressing

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

- a) 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.
- b) 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.”
- c) The physical layer has three major functions:
 - 1) 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.

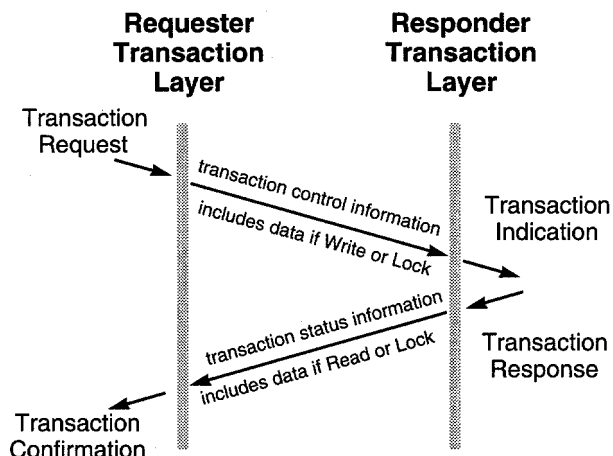


Figure 3.5—Transaction services

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.

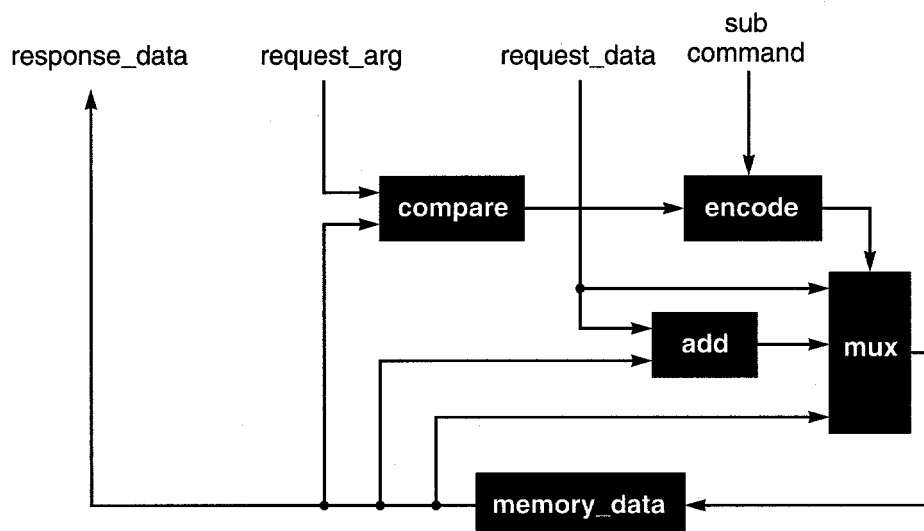


Figure 3.6—Simplified lock model

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).

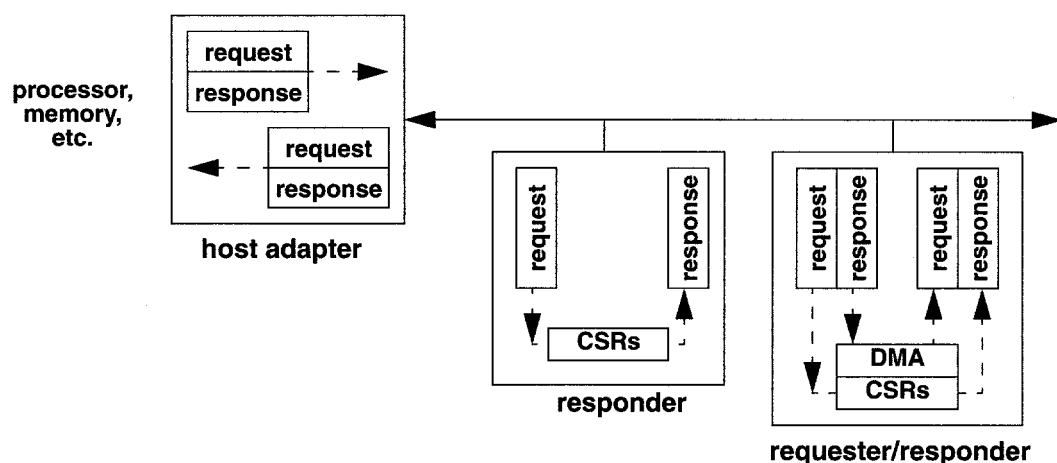


Figure 3.7—Transaction request and response subaction queues

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:

- 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.
- 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.

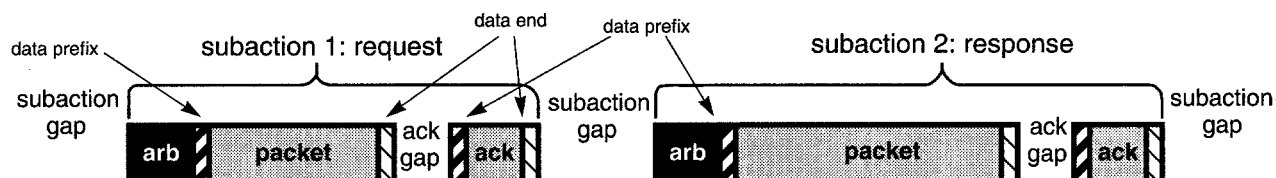


Figure 3.8—Example asynchronous subactions

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

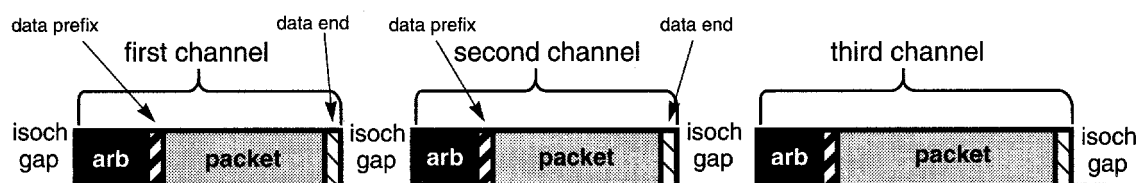


Figure 3.9—Example isochronous subactions

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.

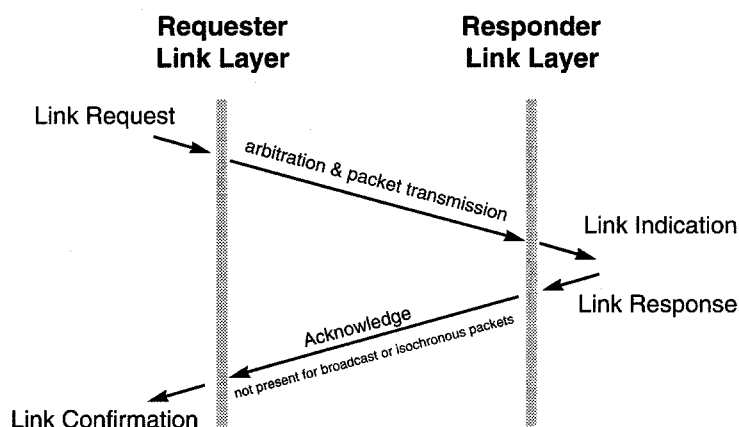


Figure 3.10—Link layer services

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.

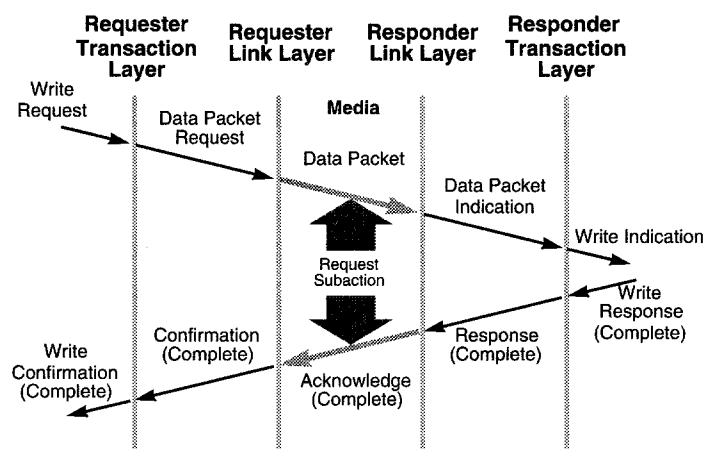


Figure 3.11—Unified transaction example

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.

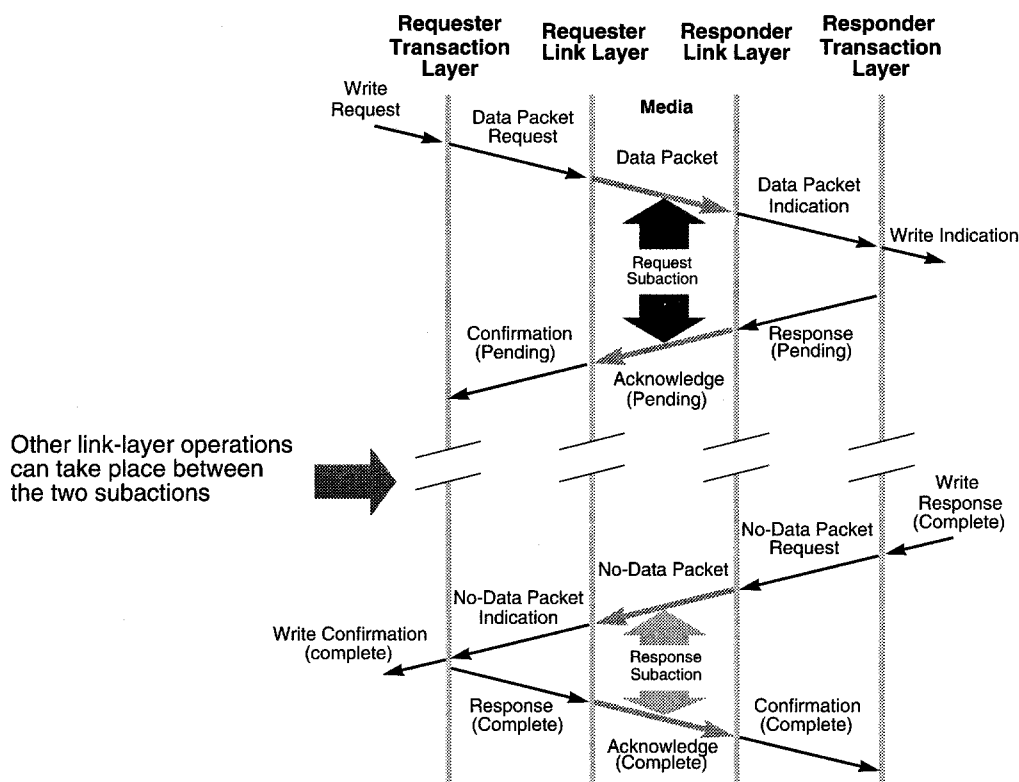


Figure 3.12—Split transaction example

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.

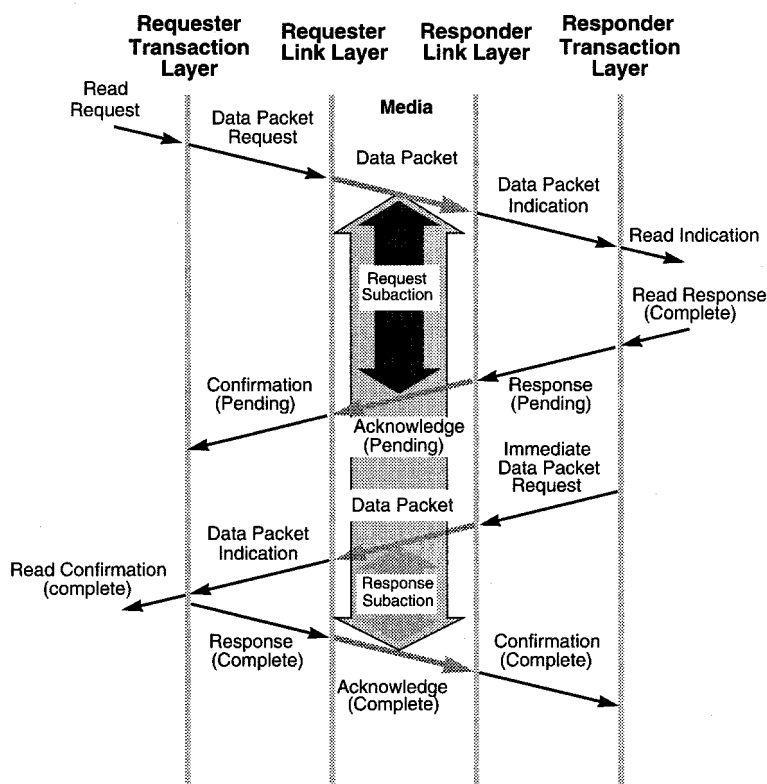


Figure 3.13—Split transaction using concatenated subactions

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

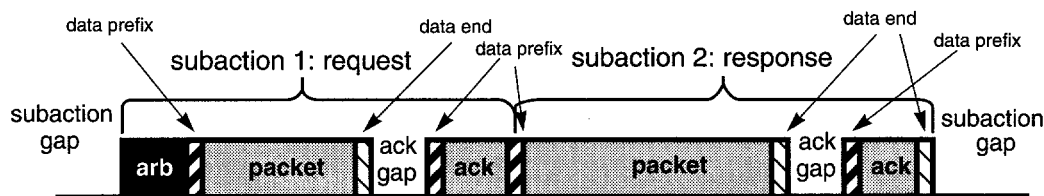


Figure 3.14—Example of concatenated asynchronous subactions

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

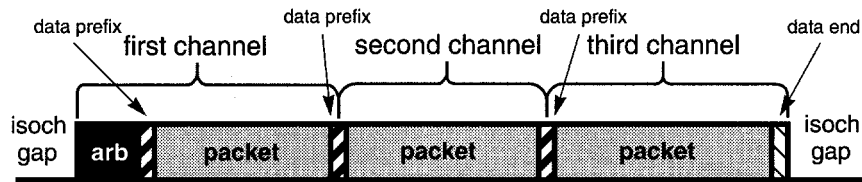


Figure 3.15—Example of concatenated isochronous subactions

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).

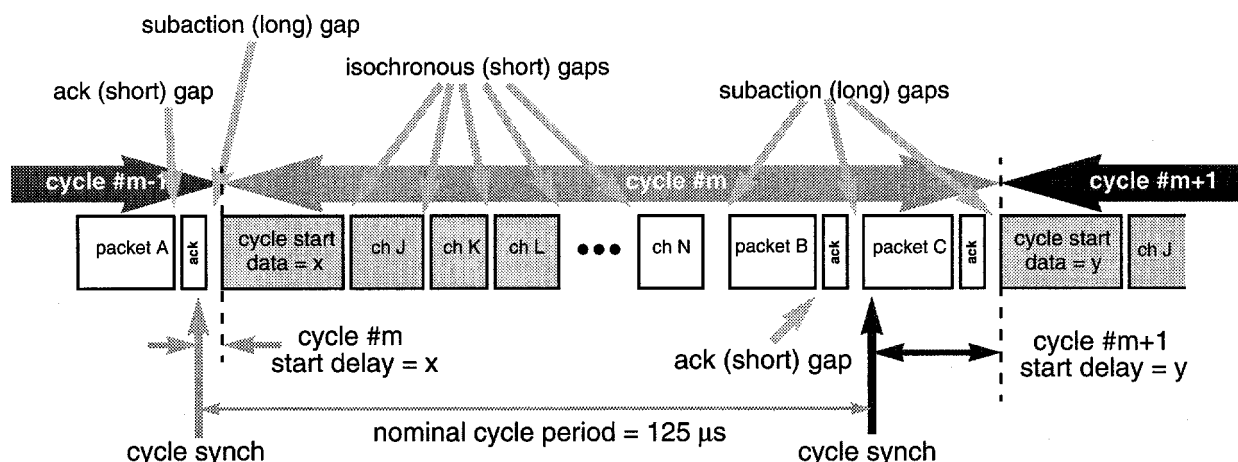


Figure 3.16—Cycle structure

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.

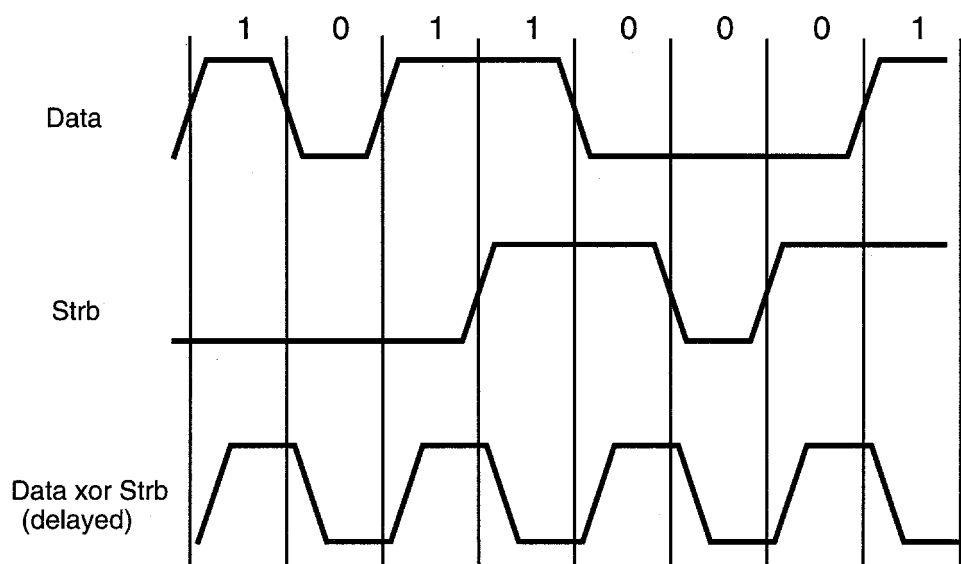


Figure 3.17—Data-strobe encoding

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.

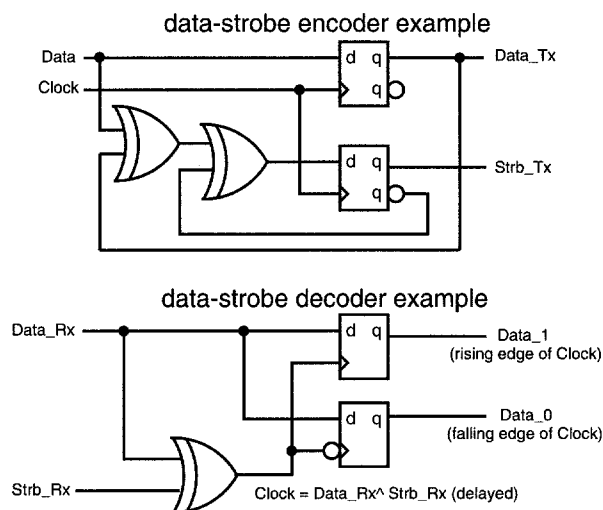


Figure 3.18—Data-strobe encoder and decoder example

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.

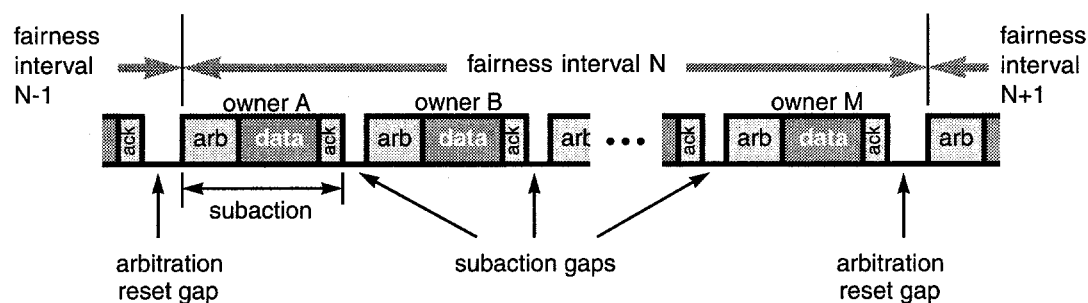


Figure 3.19—Fairness interval

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.

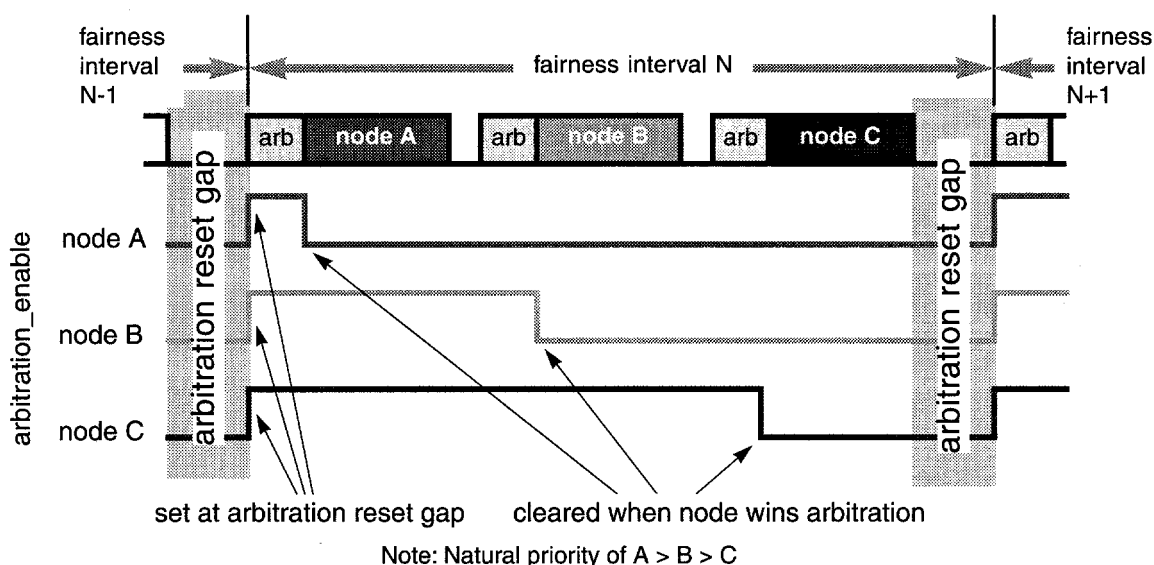


Figure 3.20—Fair arbitration

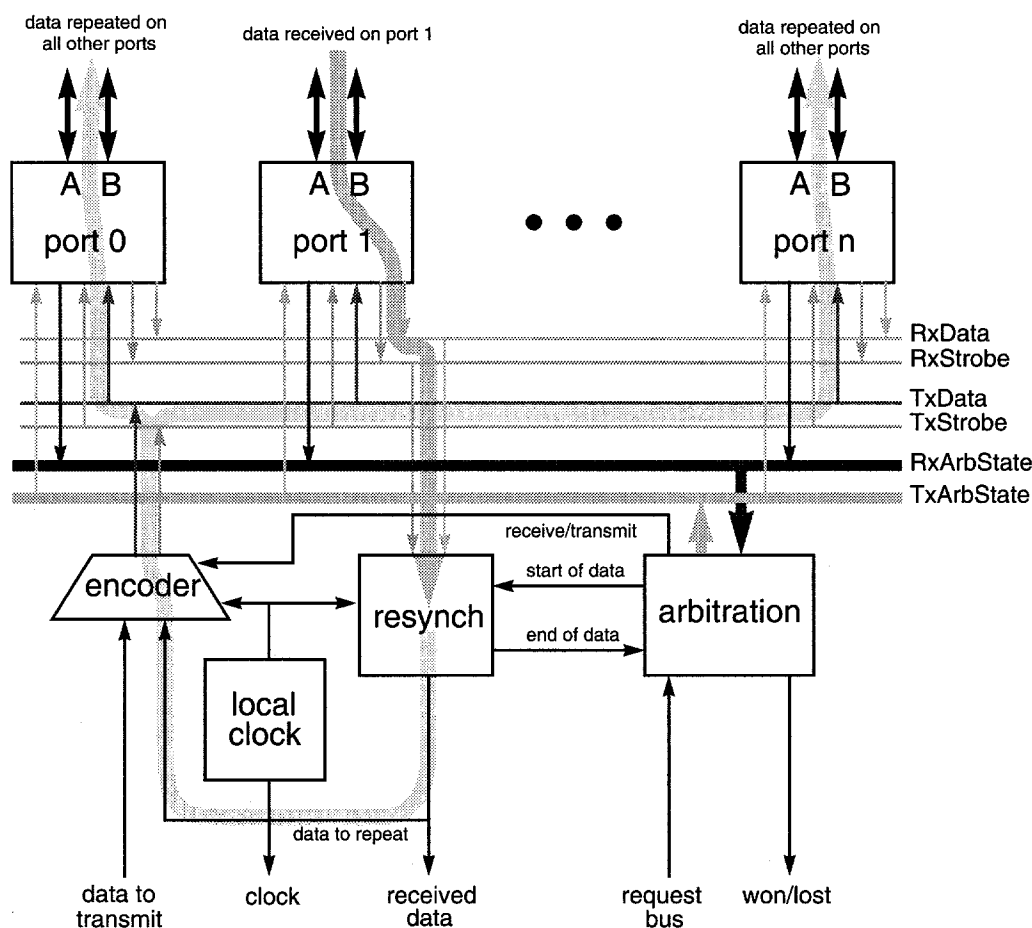
Note that a node sending a concatenated subaction (see 3.6.2.3) does not reset its arb_enable bit.

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:

- Ports, which provide the cable media interface.
- Arbitration logic, which provides access to the bus.
- The resynchronizer, which takes received data-strobe encoded data bits and generates data bits synchronized to a local clock.
- 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.”

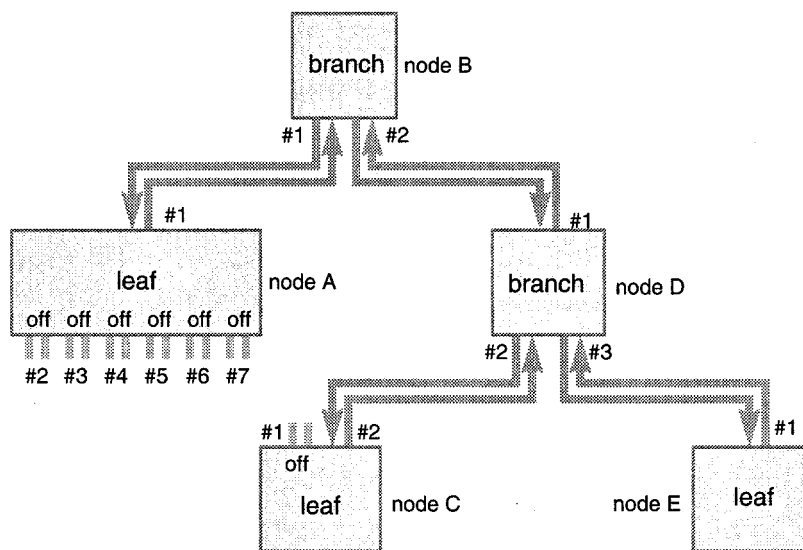


Figure 3.22—Example cable state after bus initialize

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.

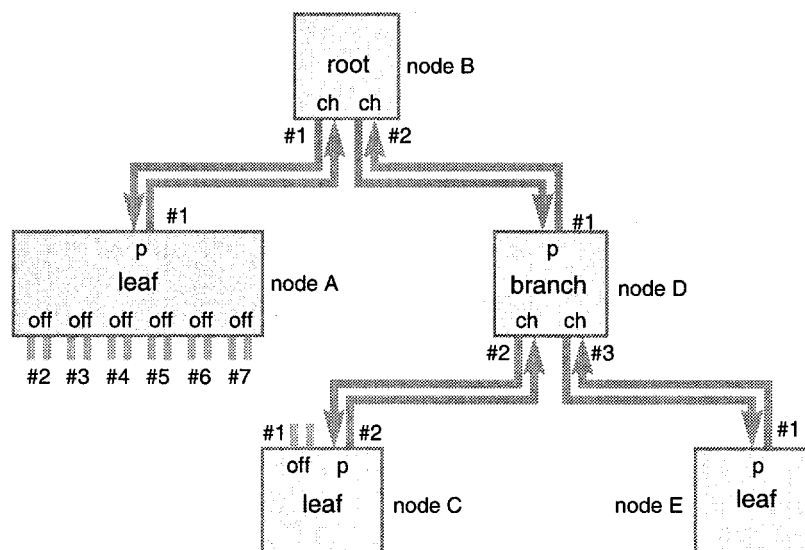


Figure 3.23—Tree identify complete

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.

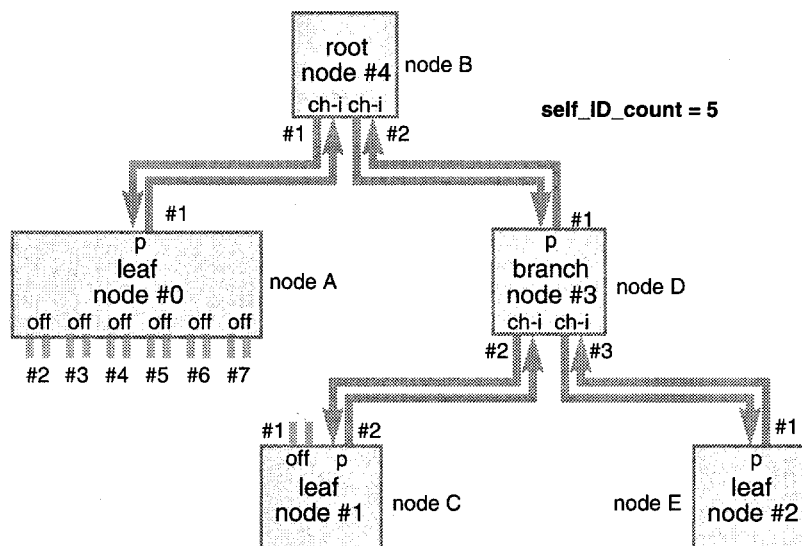


Figure 3.24—Example cable state after self-identify phase

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.

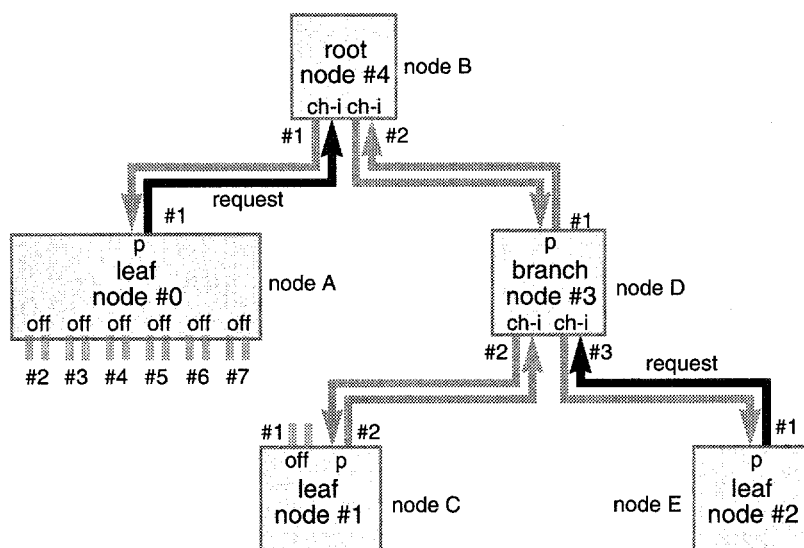


Figure 3.25—Arbitration request

- 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.

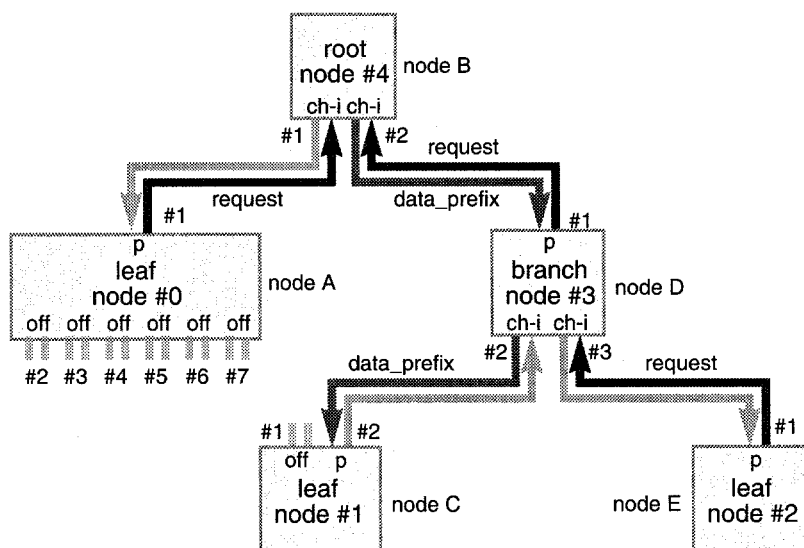


Figure 3.26—Arbitration request (continued)

- 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.

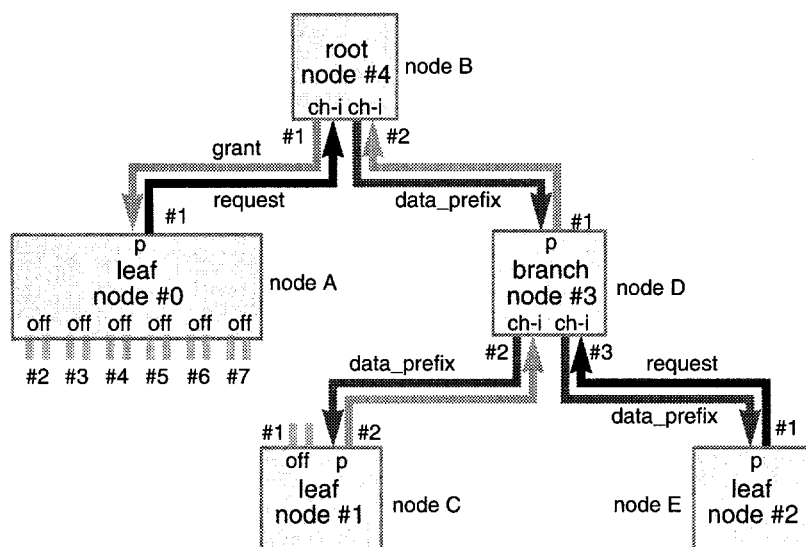


Figure 3.27—Arbitration grant

- 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.

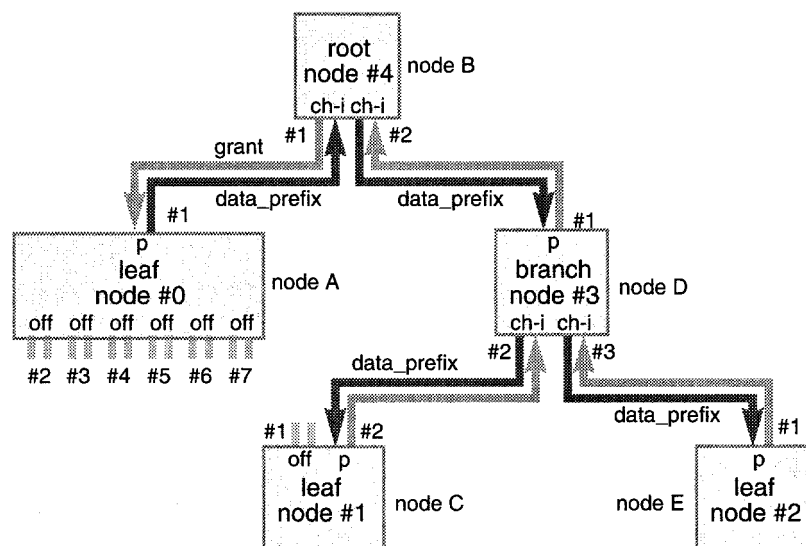


Figure 3.28—Data prefix

- 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.

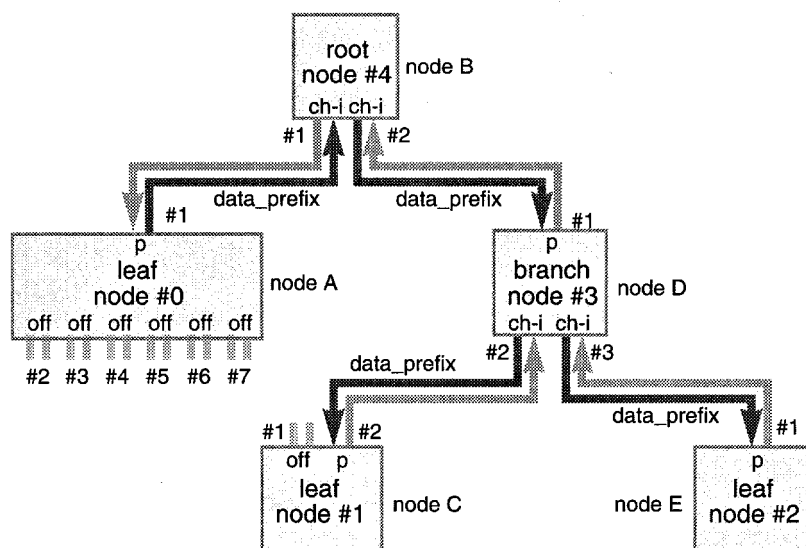


Figure 3.29—Start of data transmission

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.

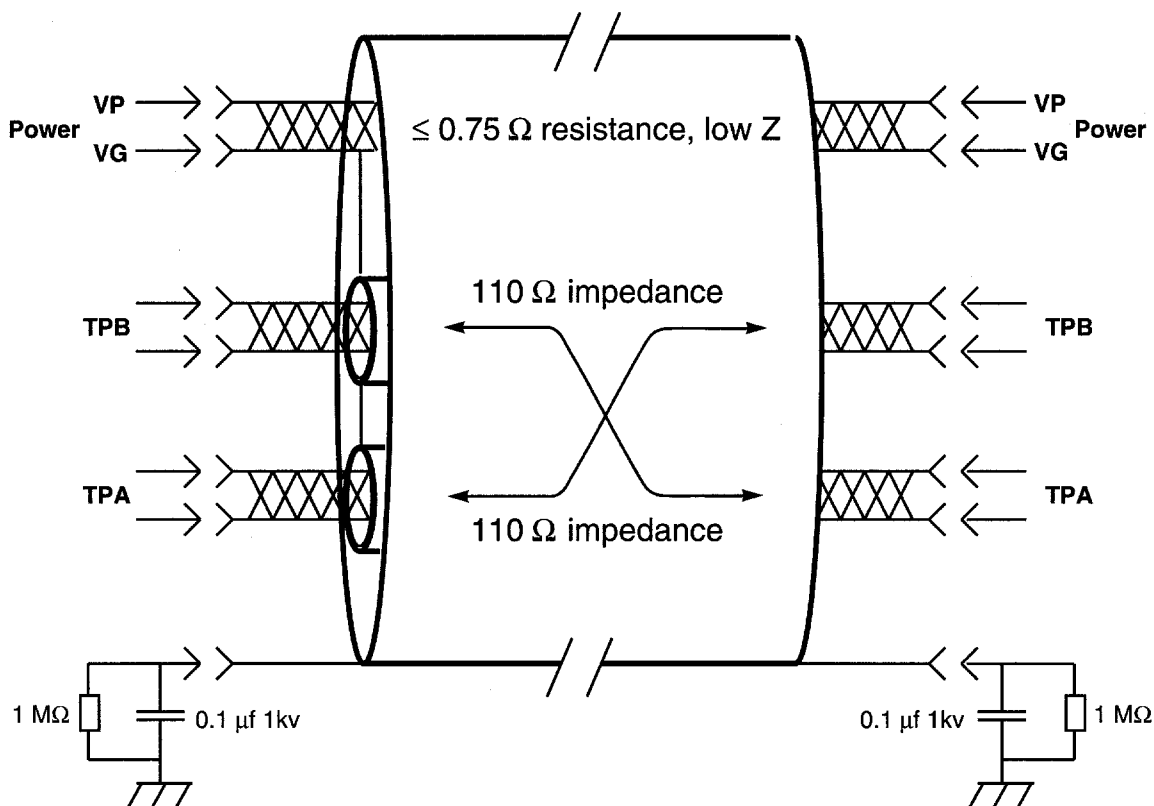


Figure 3.30—Cable interface

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.