

1394.1™

IEEE Standard for High Performance Serial Bus Bridges

IEEE Computer Society

Sponsored by the
Microprocessor and Microcomputer Standards Committee



3 Park Avenue, New York, NY 10016-5997, USA

1 July 2005

Print: SH95311

PDF: 1394.1-2004



Documents

Document Sharing Hub

Recognized as an

IEEE Std 1394.1™-2004

American National Standard (ANSI)

IEEE Standard for High Performance Serial Bus Bridges

Sponsor

Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society

Approved 12 April 2005

American National Standards Institute

Approved 8 December 2004

IEEE-SA Standards Board

Abstract: The model, definition, and behaviors of High Performance Serial Bus bridges, which are devices that can be used to interconnect two separately enumerable buses, are specified.

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2005 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1 July 2005. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-4647-1 SH95311
PDF: ISBN 0-7381-4648-X SS95311

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 Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

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.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

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 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. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
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 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; +1 978 750 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 1394.1-2004, IEEE Standard for High Performance Serial Bus Bridges.

January 15 through 16, 1996, in Dallas, Texas, Gerald Marazas convened a study group authorized by the IEEE Microprocessor Standards Committee (MSC). The study group was chartered to investigate the industry desire for and the feasibility of enhancements to recently approved IEEE Std 1394™-1995; enhancements that would enable the geographic scope of a single Serial Bus to be extended to an interconnected net of multiple buses. Although the informative overview in IEEE Std 1394-1995 describes the use of bus bridges to connect up to 1023 Serial Buses into a single net, the standard is scant on normative details as to how bus bridges might operate.

The study group continued meeting through May 1996, during which time it considered presentations on possible designs for Serial Bus bridges and discussed the Scope and Purpose of a Project Authorization Request (PAR) to be submitted to the IEEE Standards Association. The basic objectives were agreed at the second study group meeting; drafting the PAR was delegated to a small group that completed its work and obtained study group ratification of the PAR in time to submit it to the IEEE-SA for consideration at its June meeting. The IEEE-SA authorized the project, IEEE P1394.1, High Performance Serial Bus Bridges, on June 20, 1996.

The IEEE P1394.1 working group held its first official meeting in July 1996, in San Jose, CA. At this meeting, Richard Scheel was elected Chair of the working group. Scheel shepherded the working group's deliberations until July 2000, when the focus of his work activity at Sony shifted away from IEEE 1394. During Scheel's tenure as Chair, the working group engaged in vigorous debate on topics central to Serial Bus bridges:

- Self-organizing behavior of bridge portals when net topology changes, which necessitates updates to each portal's routing information.
- Distribution and synchronization of `CYCLE_TIME.cycle_offset` from a single cycle master (dubbed the net cycle master) to all buses within the net.
- Knowledge of application-dependent isochronous data formats, such as those specified by the IEC 61883 family of standards, so that bridge portals can modify timestamps embedded in the data.
- Relatively stable 16-bit node IDs (dubbed global node IDs) used to reference remote nodes.
- Detection and elimination of routing loops introduced into the net topology by user actions.
- Connection management for isochronous streams.
- Minimum capabilities for "bridge-aware" devices that communicate with other, remote devices.
- Limited support for legacy devices that are not "bridge-aware."
- Congestion management strategies, *i.e.*, the persistence of bridge portals in attempts to forward request and response subactions to the next bridge portal.
- Assignment of unique bus IDs to distinct Serial Buses within the net.
- Device discovery protocols, both for local devices on the same bus and for remote devices connected to other parts of the net.
- Net management messages for inter-portal communication and, in some cases, communication between "bridge-aware" devices and bridge portals.

In April 2000, at the 1394 Trade Association meeting in Brussels, Dr. Judi Romijn of the Technische Universiteit Eindhoven in the Netherlands made a presentation on the use of formal methods in the discovery of a flaw in the IEEE 1394 PHY state machines that govern bus configuration. This was a fortuitous circumstance, since key members of the IEEE P1394.1 working group were present and engaged her in conversation about the possible application of formal methods to IEEE P1394.1. There was mutual interest, and since then Judi Romijn has been an active and invaluable contributor—particularly with respect to formal proof that the self-organizing behavior of bridge portals (net update) terminates in a consistent state.

Later that summer, at the July 2000 meeting, the working group elected Peter Johansson as Chair. One of the first orders of business was to take stock of the draft standard and determine what, if any, incomplete areas existed that required substantial “invention” before the draft could proceed to Sponsor Ballot. Consensus resulted that although significant effort remained to document the working group’s agreements, none of the unresolved issues were major. The working group anticipated a schedule that would yield a functionally complete and reviewed draft standard by the end of 2000, with Sponsor Ballot to follow in the first quarter of 2001. Work proceeded apace on the draft and by the end of March 2001, the penultimate draft before Sponsor Ballot was published for final review by the working group. A modest number of changes were made in the final draft, published in June, and Sponsor Ballot commenced the same month.

After collation of more than 500 ballot comments (the substance and volume of which revealed as overly optimistic the earlier working group consensus with respect to “no major issues”) and formation of the Ballot Response Committee (BRC), the Editor spent several months resolving the less controversial editorial and technical comments before publishing an interim draft in December 2001. The BRC held its first meeting in San Jose at the beginning of December 2001, and continued to meet the subsequent year until agreement in principle on the resolution of all comments was achieved in October 2002.

Because of a change in management support, as well as commitment to other projects, the Editor was unable to publish a candidate draft for Recirculation Ballot until November 2003. The BRC met in early December for final review and corrections to the draft before a Recirculation Ballot was initiated in March 2004.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

Patent notice

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. A patent holder has 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 licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Participants

The following is a list of active participants in the IEEE P1394.1 working group (those who attended three or more meetings from inception to the time of publication):

Peter Johansson, *Chair and Editor*
David Hunter, *Secretary*

Jean-Paul Accarie	Eric Hannah	Atsushi Nakamura	David Smith
Masa Akahane	Jerry Hauck	Yoshikatsu Niwa	Carlton Sparrell
Subrata Bannerjee	Dieter Haupt	Fritz Nordby	Gilles Straub
Steven Bard	Hisaki Hiraiwa	Takayuki Nyu	Thomas Thaler
Philippe Boucachard	Daisuke Hiraoka	Ozay Oktay	Kazonobu Toguchi
Mohamed Braneci	Du Hung Hou	Tomoki Saito	Satoru Toguchi
Richard Churchill	David James	Takashi Sato	Masatoshi Ueno
Beth Cooper	Mark Knecht	Tetsuya Sato	Colin Whitby-Stevens
Chris Dorsey	David LaFollette	Bradley Saunders	Calto Wong
Firooz Farhoomand	Yvon Legallais	Yoshi Sawada	David Wooten
Steve Finch	Jun-ichi Matsuda	Richard Scheel	Patrick Yu
Laurent Frouin	Daniel Meirsman	Hisato Shima	Dave Zalatimo
John Fuller	Neil Morrow	Michael Smith	Frank Zhao

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Jean-Paul Accarie	Roger D. Edwards	Mark Knecht	David Thompson
Stelios Akalestos	Firooz Farhoomand	Robert Mortonson	Kazunobu Toguchi
Eric Anderson	Michael Fischer	Chuck Rice	Robert Tripi
Larry Arnett	Gordon Force Sr	Gary Robinson	Colin Whitby-Stevens
Terry Arnold	Laurent Frouin	Bivabasu Sarkar	David Wooten
Mohamed Braneci	John Fuller	Takashi Sato	Paul Work
Keith Chow	Straub Gilles	Bradley Saunders	Don Wright
Elizabeth Cooper	Dieter Haupt	Thomas Schaal	Patrick Yu
Guru Dutt Dhingra	Neil Horman	Richard Scheel	Oren Yuen
Georg Dickmann	David Hunter	Akihiro Shimura	Janusz Zalewski
Sourav Dutta	David James	Michael Teener	
	Peter Johansson	Thomas Thaler	

The following people served on the ballot response committee:

Peter Johansson, *Chair*

Mohamed Braneci	David Hunter	Gilles Straub	Kazunobu Toguchi
Georg Dickmann	Judi Romijn	Eldad Teeni	Colin Whitby-Stevens
John Fuller	Takashi Sato	Thomas Thaler	David Wooten

This standard was approved by the IEEE-SA Standards Board on 8 December 2004 with the following membership:

Don Wright, *Chair*
Steve M. Mills, *Vice Chair*
Judith Gorman, *Secretary*

Chuck Adams
Stephen Berger
Mark D. Bowman
Joseph A. Bruder
Bob Davis
Roberto de Marca Boisson
Julian Forster*
Arnold M. Greenspan
Mark S. Halpin

Raymond Hapeman
Richard J. Holleman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Hermann Koch
Thomas J. McGean

Daleep C. Mohla
Paul Nikolich
T. W. Olsen
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Doug Topping
Joe D. Watson

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Michelle Turner
IEEE Standards Project Editor

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
2. Normative references	3
3. Definitions and notation	5
3.1 Conformance	5
3.2 Technical.....	5
3.3 Document notation.....	9
4. Bridge model (informative).....	15
4.1 Global node IDs.....	16
4.2 Remote time-out	17
4.3 Clan affinity and net update	18
4.4 Cycle time distribution and synchronization.....	20
4.5 Universal time.....	22
4.6 Stream connection management	24
5. Bridge portal and bridge-aware node facilities.....	31
5.1 Configuration ROM.....	31
5.2 Control and status registers.....	33
6. Packet formats.....	41
6.1 Self-ID packet zero	41
6.2 Cycle master adjustment packet.....	41
6.3 Response packet	42
6.4 Global asynchronous stream packets (GASP).....	44
6.5 Net management message interception	45
6.6 Net management messages	46
6.7 UPDATE ROUTES message	55
7. Transaction routing and operations	57
7.1 Source bus (initial entry portal)	57
7.2 Intermediate buses	58
7.3 Destination bus (terminal exit portal)	60
7.4 Maximum forward time	61
7.5 Congestion management.....	62
8. Stream operations and routing.....	65
8.1 Cycle timer synchronization	65
8.2 Net time	68
8.3 GASP routing and operations.....	69
8.4 Listening portal operations (isochronous streams).....	70
8.5 Talking portal operations (isochronous streams).....	70
8.6 Isochronous stream connection management.....	70

8.7 Common Isochronous Packet (CIP) format headers	86
9. Operations in a bridged environment	89
9.1 CSR architecture assumptions	89
9.2 Bridge-aware devices.....	89
9.3 Legacy devices	91
9.4 TIMEOUT message operations.....	91
9.5 Modifications to the BUS_TIME and CYCLE_TIME registers	93
9.6 Remote access to core and bus-dependent CSRs	93
10. Net update	95
10.1 Power reset initialization	95
10.2 Bus reset operations.....	95
10.3 Coherency during net update	101
10.4 Mute bridge portals.....	102
10.5 Route map updates.....	103
10.6 Net panic.....	107
11. Global node ID management.....	109
11.1 Virtual ID management.....	109
11.2 Bus ID management.....	111
Annex A (normative) Net correctness properties	115
Annex B (normative) Minimum Serial Bus capabilities for bridge portals.....	117
Annex C (normative) Pseudocode data structures and constants	119
Annex D (normative) Transaction routing.....	127
Annex E (nomative) Discovery and enumeration protocol (DEP)	135
Annex F (normative) Plug control registers.....	143
Annex G (informative) Bus topology analysis	149
Annex H (informative) Sample configuration ROM	159
Annex I (informative) Bibliography	161

IEEE Standard for High Performance Serial Bus Bridges

1. Overview

1.1 Scope

This is a full-use standard whose scope is to extend the already defined asynchronous and isochronous services of High Performance Serial Bus beyond the local bus by means of a device, the bridge, which consists of two nodes, each connected to a separate bus and both interconnected by implementation-dependent means.

The project is intended to standardize the model, definition, and behaviors of High Performance Serial Bus bridges, which are devices that may be used to interconnect two separately enumerable buses. This project extends IEEE Std 1394TM-1995¹, as amended by IEEE Std 1394aTM-2000, and IEEE Std 1394bTM-2002, and is based upon those documents as well as upon IEEE Std 1212TM-2001, Control and Status Registers (CSR) Architecture for microcomputer buses.

NOTE—The set of IEEE 1394 standards specifies the interfaces, functions, and operations necessary to ensure interoperability between conforming implementations. These standards are functional descriptions. An implementation may employ any design whose behavior is compliant with the pertinent standards.²

1.2 Purpose

IEEE Std 1394-1995, as amended by IEEE Std 1394a-2000 and IEEE Std 1394b-2002, is a cost-effective desktop interconnect for both computer peripherals and consumer electronics. However, the use of High Performance Serial Bus in other environments, *e.g.*, an interconnect to carry high-speed digital video data between rooms of a house, is hampered by the incomplete architectural and protocol specifications for bridges in the existing standards. This project proposes to adequately specify bridge requirements in order to enable a larger consumer and computer market for High Performance Serial Bus products.

¹Information on references can be found in Clause 2.

²Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

2. Normative references

The standards named in this section contain provisions that, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision; parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid international standards.

IEC 61883-1 (2003-01), Consumer audio/video equipment—Digital interface—Part 1: General.²

IEEE Std 1212TM-2001, IEEE Standard for Control and Status Registers (CSR) Architecture for microcomputer buses.^{3, 4}

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

IEEE Std 1394a-2000, IEEE Standard for a High Performance Serial Bus—Amendment 1.

IEEE Std 1394b-2002, IEEE Standard for a High Performance Serial Bus—Amendment 2.

Throughout this document, the term “IEEE 1394” shall be understood as a reference to IEEE Std 1394-1995 as amended by IEEE Std 1394a-2000 and IEEE Std 1394b-2002.

² IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iec.ch/>). IEC 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. (<http://www.ansi.org/>)

³ The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

⁴ IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (<http://standards.ieee.org/>).

3. Definitions and notation

For the purposes of this standard, the following definitions, terms and notational conventions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B2]⁵, should be consulted for terms not defined in this clause.

3.1 Conformance

The following keywords are used to differentiate between different levels of requirements and optionality in this standard:

3.1.1 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.

3.1.2 ignored: A keyword that describes bits, bytes, quadlets, octlets, or fields whose values are not checked by the recipient.

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

3.1.4 reserved: A keyword used to describe objects—bits, bytes, quadlets, octlets and fields—or the code values assigned to these objects; the object or the code value is set aside for future standardization by the IEEE. A reserved object shall be zeroed by its originator or, upon development of a future IEEE standard, set to a value specified by such a standard. The recipient of a reserved object shall ignore its value. The recipient of an object whose code values are defined by this standard shall inspect its value and reject reserved code values.

3.1.5 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products conforming to this standard.

3.1.6 should: A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “is recommended.”

3.2 Technical

The following are terms that are used within this standard:

3.2.1 adjustable cycle master: A cycle master that recognizes cycle master adjustment packets and, according to the instruction received, lengthens or shortens the interval to the next cycle start event by a single cycle timer tick.

3.2.2 allocation map: A data structure that specifies for each of the 1023 possible bus IDs whether it is in use (VALID), not in use (CLEAN) or in a transitional state from in use to not in use (DIRTY). An allocation map is similar to a bridge portal's route map except that it contains less information: the route map states FORWARD and VALID are collapsed into a single state, VALID.

3.2.3 alpha portal: A portal on a bus that snoops and forwards transaction subactions addressed to the prime portal. Except during net update, there is only one alpha portal on a bus. On the bus to which the prime portal is connected, the alpha portal and the prime portal are one and the same.

3.2.4 bridge: A device capable of connecting two buses to form a net. A bridge implements two portals, forwards asynchronous subactions and might forward isochronous subactions, both as determined by the route information it maintains. A distributed algorithm executed by bridge portals initializes asynchronous routing information; applications initialize, on a stream-by-stream basis, isochronous routing information.

3.2.5 bridge-aware: A device capable of originating and timing transaction requests addressed to remote nodes. Bridge-aware devices also interact with bridges by means of messages or other signals.

⁵The numbers in brackets correspond to those of the bibliography in Annex I.

3.2.6 bridge-bound: A description applied to a request, response, or stream subaction received by a bridge portal with the intent to forward the subaction to its co-portal. The subaction is bridge-bound in that its next (intermediate) destination is the internal fabric of the bridge that connects the two portals. A portal that snoops a bridge-bound subaction is called an entry portal.

3.2.7 bus: A group of Serial Bus nodes interconnected within the same arbitration domain and mutually addressable by packets with a *destination_bus_ID* field of $3FF_{16}$.

3.2.8 bus ID: A 10-bit identifier that, unless net update is in progress, is unique for each bus within a Serial Bus net. The bus ID assigned to a particular bus is not visible in the most significant ten bits of the *NODE_IDS* register, which always exhibits a *bus_ID* value of $3FF_{16}$, the local bus. Bridge portals maintain the assigned bus ID internally and use it to transform local node IDs into global node IDs and vice-versa.

3.2.9 bus-bound: A description applied to a request, response, or stream subaction received by a bridge portal from its co-portal with the intent to transmit the subaction on the portal's local bus. A portal that transmits a bus-bound subaction is called an exit portal.

3.2.10 channel: A relationship between a group of nodes: zero or one talker and zero or more listeners. A number between zero and 63, inclusive, identifies the group. Channel numbers are allocated cooperatively through isochronous resource management facilities.

3.2.11 clan: A group of affiliated bridge portals; all of the members of a clan exhibit allegiance to the same prime portal, which is identified by its EUI-64. When two or more nets are joined, it is possible for bridge portals on a bus to belong to different clans—but this is a temporary condition that is resolved by net update procedures.

3.2.12 clan allocation map: An allocation map derived from the route maps of bridge portals that belong to the same clan (*see: net allocation map*).

3.2.13 coordinator: A bridge portal selected after each bus reset; it manages net update, which might be required as a consequence of local or remote topology changes in the net. The coordinator is the bridge portal with the largest physical ID; it retains this role until a subsequent bus reset. The coordinator is also responsible to maintain virtual ID mappings and distribute them to all bridge portals on the bus.

3.2.14 co-portal: From the perspective of a particular bridge portal, the other portal. A portal is connected to its co-portal by the bridge's internal fabric.

3.2.15 CSR architecture: IEEE Std 1212-2001, Standard for a Control and Status Registers (CSR) Architecture for microcomputer buses.

3.2.16 cycle master: On a particular bus, the node that generates the periodic cycle start packet 8000 times a second. In a net of interconnected buses there is a cycle master for each bus.

3.2.17 downstream: An adjective used to describe the relationship of a node to a particular location within the net. When used in the context of cycle time synchronization with respect to the net cycle master, a downstream cycle master or a downstream portal has more bridge portals between itself and the net cycle master than the portal to which it is compared. Alternately, in the context of a particular isochronous stream, within a bridge the downstream portal is the one with more bridge portals between itself and the talker.

3.2.18 entry portal: A description applied to a bridge portal when it snoops and assumes responsibility for the disposition of a bridge-bound request, response or stream subaction. Commonplace actions are to forward the subaction to the co-portal (which acts as the exit portal) or to echo the subaction on the local bus, but an entry portal may also discard a subaction or create and transmit a response packet if errors are detected.

3.2.19 exit portal: A description applied to a bridge portal when it transmits bus-bound request, response or stream subactions forwarded by the entry portal. The same portal might be both the entry and exit portal for a subaction; this is the case when a transaction request or response is echoed to the local bus.

3.2.20 GASP: Global asynchronous stream packet, specified by IEEE Std 1394a-2000.

3.2.21 global node ID: A 16-bit address that can be used as the *destination_ID* in an asynchronous primary packet so long as the packet passes through at least one bridge portal en route to its destination. The value of the most significant ten bits, the bus ID, ranges between zero and $3FE_{16}$, inclusive. Within a net, a bus ID in this range uniquely identifies a single bus. The least significant six bits of a global node ID, the virtual ID, uniquely identify a node on a particular bus.

3.2.22 global subaction: Either a) an asynchronous subaction either of whose *source_ID* and *destination_ID* contains a global node ID or b) a GASP subaction whose *sy* value is greater than or equal to eight.

3.2.23 initial entry portal: The first bridge portal that snoops request, response or stream subaction and transforms the subaction's *source_ID*, if present, from a local node ID to a global node ID.

3.2.24 heterogeneous bridge: A bridge, one portal of which is an IEEE 1394 node but whose other portal implements a different transport protocol, *i.e.*, the portal is not an IEEE 1394 node. The operational details of heterogeneous bridges are beyond the scope of this standard.

3.2.25 isochronous bridge: A bridge that implements the facilities (buffers, CSRs, stream management messages, *etc.*) necessary to transfer isochronous subactions between its portals. These capabilities are in addition to the base requirements for a bridge to forward asynchronous subactions and to maintain cycle time synchronization.

3.2.26 isochronous period: A period that begins after a cycle start packet is sent and ends when a subaction gap is detected. During an isochronous period, only isochronous subactions may occur. An isochronous period nominally begins every 125 μ s.

3.2.27 isochronous resource manager: A node that implements the *BUS_MANAGER_ID*, *BANDWIDTH_AVAILABLE*, *CHANNELS_AVAILABLE* and *BROADCAST_CHANNEL* registers (some of which permit the cooperative allocation of isochronous resources). Subsequent to each bus reset, one isochronous resource manager is selected from all nodes capable of this function. All bridge portals are required to be isochronous resource manager-capable.

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

3.2.29 listener: A node, or an application at a node, that receives a stream packet.

3.2.30 local ID: The least significant six bits of the *NODE_IDS* register; each node on a local bus is assigned a different local ID as a consequence of bus reset. Also known as physical ID.

3.2.31 local node: A Serial Bus node is local with respect to another node if they are both connected to the same bus. This is true whether the bus does not yet have a unique *bus_ID* and is addressable only as the local bus, $3FF_{16}$, or if the bus has been assigned a *bus_ID*.

3.2.32 local node ID: A 16-bit address usable as the *destination_ID* in an asynchronous primary packet so long as both the sender and the recipients are on the same bus. The local node ID is the concatenation of $3FF_{16}$ and the node's local ID.

3.2.33 local subaction: Any of a) an asynchronous subaction whose *source_ID* and *destination_ID* both contain a local node ID, b) an asynchronous stream subaction that is not GASP, c) a GASP subaction whose *sy* value is less than eight or d) an isochronous subaction.

3.2.34 maximum forward time: The maximum time a bridge is permitted to persist in its attempts to forward a snooped asynchronous subaction to the next recipient, either an intermediate bridge portal on the route to the node identified by *destination_ID* or the destination node itself. The period to be timed commences when the entry portal transmits *ack_pending* or *ack_complete* for a snooped subaction and concludes when the exit portal receives *ack_pending* or *ack_complete* for the retransmitted subaction.

3.2.35 mute bridge: A bridge whose routing functions are disabled; neither asynchronous nor isochronous subactions are forwarded from either co-portal to the other. Each of the mute bridge's portals continues to respond to request subactions addressed to its local node ID and may initiate local or remote request subactions.

3.2.36 net: A collection of buses interconnected by bridges. Each bus within the net is uniquely identified by its *bus_ID*.

3.2.37 net allocation map: An allocation map derived from all of a net's clan allocation maps (see also clan allocation map). When network topology is stable, there is only one clan allocation map, which is identical to the net allocation map.

3.2.38 net cycle master: The cycle master for one Serial Bus in the net selected to be the cycle time source for the entire net.

3.2.39 net update: A process triggered by the addition or removal of one or more buses (or bus IDs) from a net; net topology and all bridge portal route maps are updated to a consistent state.

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

3.2.41 node ID: A 16-bit number that uniquely identifies a node, either within the context of a particular bus (local node ID) or within a net of interconnected buses (global node ID). The ten most significant bits of node ID, the bus ID, identify the bus to which the node is connected. The six least significant bits of node ID identify the node on that bus; within a local node ID they are a physical ID but within a global node ID they are a virtual ID.

3.2.42 physical ID: Synonymous with local ID, a 6-bit address assigned to each node on a bus by the self-identification process that follows a bus reset (see IEEE 1394).

3.2.43 portal: A connection from a Serial Bus bridge to a bus. Each portal presents a set of Serial Bus CSRs, as defined by IEEE 1394 and this document, to the connected bus. There may be multiple PHY ports for each portal. Serial Bus bridges shall implement two portals.

3.2.44 prime portal: The singular portal within a net of interconnected buses that enumerates unique bus IDs and manages their distribution. When two or more nets are joined, one prime portal is selected from the incumbents. Since the location of the prime portal also determines the location of the net cycle master, the selection process is designed to minimize disruption to isochronous streams (which rely upon the stability of the net cycle master). Ties in the selection process between prime portal candidates are resolved by their EUI-64s.

3.2.45 quarantine: A period of time during which a bridge-aware node or bridge-portal shall not originate global subactions. Quarantine commences whenever the self-ID packets that occur subsequent to a bus reset indicate net topology changes; it ends upon a signal from the coordinator or the alpha portal. Multiple bus resets can occur during quarantine.

3.2.46 reallocation proxy: A bridge portal responsible to reallocate channel numbers and bandwidth after a bus reset. The reallocation proxy also keeps track of listeners on the local bus; if the count of listeners drops to zero, either as a result of explicit connection teardown or because of node removal, the reallocation proxy releases the pertinent channel number and bandwidth.

3.2.47 remote node: A Serial Bus node is remote with respect to another node if the nodes are connected to buses that have differing bus IDs or if one or more Serial Bus bridges lie on the path between the two nodes. Nodes connected to the same bus are considered remote with respect to each other if global node IDs are used in request and response subactions exchanged between them.

3.2.48 remote transaction-capable: A transaction-capable Serial Bus node that is additionally capable of initiating transaction requests directed to a remote node.

3.2.49 route map: A data structure, whose contents are different for each portal, that specifies, for each of the 1023 possible bus IDs, whether transaction subactions are forwarded by the portal. The four possible states for a bus ID are: in use but not

forwarded by the portal (VALID), in use and forwarded by the portal (FORWARD), not in use and therefore not forwarded by any portal in the same clan (CLEAN), or a transitional state (DIRTY).

3.2.50 snarf: Jargon that came into use in the Unix community in the 1960s. The following definition is obtained from the Jargon File 4.0.0: “To acquire, with little concern for legal forms or politesse (but not quite by stealing).” In the context of this standard, snarf refers to the interception of some asynchronous subactions while en route to their *destination_ID*. The snarfed packet might or might not be retransmitted by the intercepting bridge.

3.2.51 stream: Either asynchronous or isochronous data originated by a talker and received by zero or more listeners. An isochronous stream is uniquely identified by the talker’s EUI-64 and an index locally unique at the talker. An isochronous stream’s parameters include the payload, arbitration overhead and speed.

3.2.52 stream controller: A node that uses net management messages to set up or tear down routing for an isochronous stream between a talker and one or more listeners.

3.2.53 subordinate portal: A portal that is neither an alpha portal nor the prime portal; there may be multiple subordinate portals on a bus.

3.2.54 terminal acknowledgement: An acknowledge packet other than *ack_pending*, *ack_busy_X*, *ack_busy_A* or *ack_busy_B*. If received in connection with a request subaction, the acknowledgement completes the transaction; no response subaction will follow.

3.2.55 terminal exit portal: The last bridge portal that transmits (or would transmit) a transaction request or response subaction en route to its destination; the portal that normally transforms the subaction’s *destination_ID* from a global node ID to a local node ID.

3.2.56 upstream: An adjective used to describe the relationship of a node to a particular location within the net. When used in the context of cycle time synchronization with respect to the net cycle master, an upstream cycle master or an upstream portal has fewer bridge portals between itself and the net cycle master than the portal to which it is compared. Alternately, in the context of a particular isochronous stream, within a bridge the upstream portal is the one with fewer bridge portals between itself and the talker.

3.2.57 virtual ID: A 6-bit address assigned by the coordinator to all of the nodes present on the portal’s local bus. For any bus, all the bridge portals share the same mapping from 6-bit physical ID to virtual ID.

3.3 Document notation

3.3.1 Size notation

This document avoids the terms *word*, *half-word*, and *double-word*, which have widely different definitions depending on the word size of the processor. In their place, processor-independent terms established by previous IEEE bus standards are used. These terms are illustrated in Table 3-1.

Table 3-1 — Size notation examples

Size (in bits)	IEEE standard notation (used in this standard)	Pseudocode type
4	nibble	
8	byte	BYTE
16	doublet	DOUBLET
32	quadlet	QUADLET
64	octlet	OCTLET

Serial Bus uses big-endian ordering for byte addresses within a quadlet and quadlet addresses within an octlet. For 32-bit quadlet registers, byte 0 is always the most significant byte of the register. For a 64-bit quadlet-register pair, the first quadlet is always the most significant. The field on the left (most significant) is transmitted first; within a field the most significant (leftmost) bit is also transmitted first. This ordering convention is illustrated in Figure 3-1.

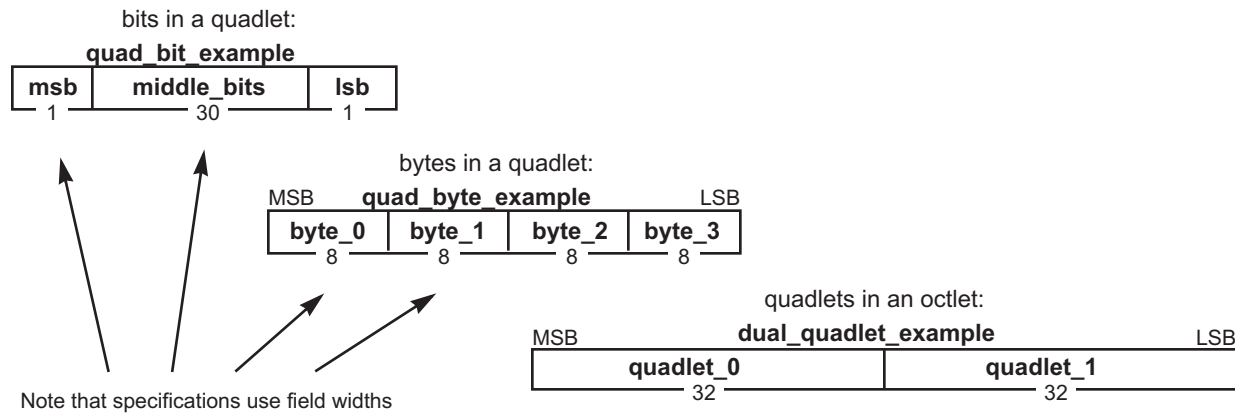


Figure 3-1 — Bit and byte ordering

Although Serial Bus addresses are defined as big-endian, their data values can 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 3-1.

When specific bit fields must be used, the CSR Architecture convention of consistent big-endian numbering is used. Hence, the most significant bit of a quadlet (“msb” in figure 1-2) will be labeled “quad_bit_example[0],” the most significant byte of a quadlet (“byte_0”) will be labeled “quad_byte_example[0:7],” and the most significant quadlet in an octlet (“quadlet_high”) will be labeled “dual_quadlet_example[0:31].”

The most significant bit shall be transmitted first for all fields and values defined by this standard, including the data values read or written to control and status registers (CSRs).

3.3.2 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 or 1) digits, followed by the subscript 2. Thus the decimal number “26” can also be represented as “1A₁₆” or “11010₂”. In C pseudocode 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.”

3.3.3 Packet formats

Most Serial Bus packets consist of a sequence of quadlets. Packet formats are shown using the style given in Figure 3-2.

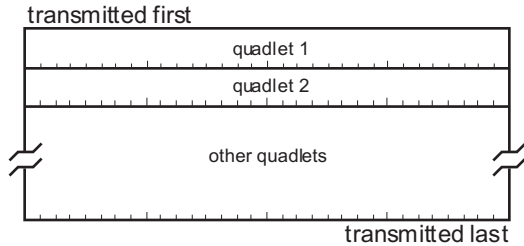


Figure 3-2 — Example packet format

Fields appear in packet formats with their correct position and widths. Bits in a packet are transmitted starting with the upper leftmost bit and finishing with the bottom rightmost bit. Given the rules in 3.3.1, this means that all fields defined in this standard are sent most significant bit first.

3.3.4 C pseudocode notation

This standard normatively describes the operations of a bridge by means of C language code and expository text. The C pseudocode is not a normative description of an implementation; it is a normative description of externally observable bridge behaviors. Different implementations are possible. In case of conflict between the C pseudocode and other text, precedence shall be given first to the C pseudocode and second to the expository text.

Although familiar to software engineers, C pseudocode operators are not necessarily obvious to all readers. The meanings of C pseudocode operators, arithmetic, relational logical and bitwise, both unary and binary, are summarized in Table 3-2.

Table 3-2 — C pseudocode operators summary

Operator	Description
+, -, * and /	Arithmetic operators for addition, subtraction, multiplication and integer division
%	Modulus; $x \% y$ produces the remainder when x is divided by y
>, >=, < and <=	Relational operators for greater than, greater than or equal, less than and less than or equal
== and !=	Relational operators for equal and not equal; the assignment operator, =, should not be confused with ==
++	Increment; $i++$ increments the value of the operand after it is used in the expression while $++i$ increments it before it is used in the expression
--	Decrement; post-decrement, $i--$, and pre-decrement, $--i$, are permitted
&&	Logical AND
	Logical OR
!	Unary negation; converts a nonzero operand into 0 and a zero operand into 1
&	Bitwise AND
	Bitwise inclusive OR
^	Bitwise exclusive OR
<<	Left shift; $x << 2$ shifts the value of x left by two bit positions and fills the vacated positions with zero

Table 3-2 — C pseudocode operators summary (*continued*)

Operator	Description
>>	Right shift; vacated bit positions are filled with zero or one according to the data type of the operand but in this amendment are always filled with zero
~	One's complement (unary)

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

The descriptions above are casual; if in doubt, the reader is encouraged to consult ISO/IEC 9899:1990 [B3].

The C pseudocode examples assume the data types listed in Table 3-3 are defined.

Table 3-3 — Additional C data types

Data type	Description
timer	A real number, in units of seconds, that autonomously increments at a defined rate
Boolean	A single bit, where 0 encodes FALSE and 1 encodes TRUE

All C pseudocode is to be interpreted as if it could be executed instantaneously, although time can elapse when conditional expressions are evaluated as part of iterated C pseudocode. Time elapses unconditionally only when the following function is called:

```
void wait_time(float time); // Wait for time, in seconds, to elapse
```

3.3.5 CSR, ROM, and field notation

This standard describes CSRs and fields within them. All CSRs are documented in the style used by IEEE Std 1212-2001. To distinguish register and field names from node states or descriptive text, the register name is always capitalized. For example, the notation `STATE_CLEAR.lost` is used to identify the *lost* bit within the `STATE_CLEAR` register.

All CSRs are one or more contiguous quadlets that are quadlet aligned. The address of a register is specified as the byte offset from the beginning of the register space and is always a multiple of four. When a range of register addresses is described, the ending address is the address of the register's last quadlet.

This standard describes 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.

Other data structures (such as packets, timers, and counters) are shown in lowercase and formatted in a fixed-pitch typeface.

3.3.6 Register specification format

This document defines the format and function of Serial Bus-specific CSRs. Registers can be read only, write only, or both readable and writable. The same distinctions may apply to any field within a register. A CSR specification includes the format (the sizes and names of bit field locations), the initial value of the register, the value returned when the register is read, and the effects when the register is written. An example register is illustrated in Figure 3-3.

definition					
unit-dependent	vendor-dependent	sig	resv	why	not
12	16				
initial value					
unit-dependent	zeros	1	0	0	0
read value					
last write	last update	w	0	u	u
write effect					
stored	ignored	s	i	i	e

Figure 3-3 — CSR format specification (example)

The register definition lists the names of register fields. These names are descriptive, but the fields are defined in the text; their function should not be inferred solely from their names. However, the register definition fields in Figure 3-3 have the meanings specified by Table 3-4.

Table 3-4 — Register definition and initial values fields

Name	Abbreviation	Definition
unit-dependent	u	The meaning of this field shall be defined by the pertinent unit architecture.
vendor-dependent	v	When used to name a field, the meaning of the field shall be defined by the vendor. When used to specify an initial value, the value shall be determined by the vendor but might be subject to constraints imposed by this or other standards. Within a unit architecture, the unit-dependent fields may be defined to be vendor-dependent.

A node’s CSRs shall be initialized when power is restored (power reset) and might be initialized when a bus reset occurs or a quadlet is written to the node’s RESET_START register (command reset). If a CSRs bus reset or command reset values differ from its initial values, they shall be explicitly specified.

The read value fields in Figure 3-3 have the meanings specified by Table 3-5.

Table 3-5 — Read value fields

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

The write-effect fields in Figure 3-3 have the meanings specified by Table 3-6.

Table 3-6 — Write value fields

Name	Abbreviation	Definition
stored	s	The value of the written data field shall be immediately visible to reads of the same register.
ignored	i	The value of the written data field shall be ignored; it shall have no effect on the state of the node.
effect	e	The value of the written data field shall have an effect on the state of the node, but might not be immediately visible to reads of the same register.

3.3.7 Reserved CSR fields

Reserved fields within a CSR conform to the requirements of the conformance glossary in this standard (see 3.1). Within a CSR, such a field is labeled *reserved* (sometimes abbreviated as lowercase *r* or *resv*). Reserved fields behave as specified by Figure 3-4: they shall be zero and any attempt to write to them shall be ignored.

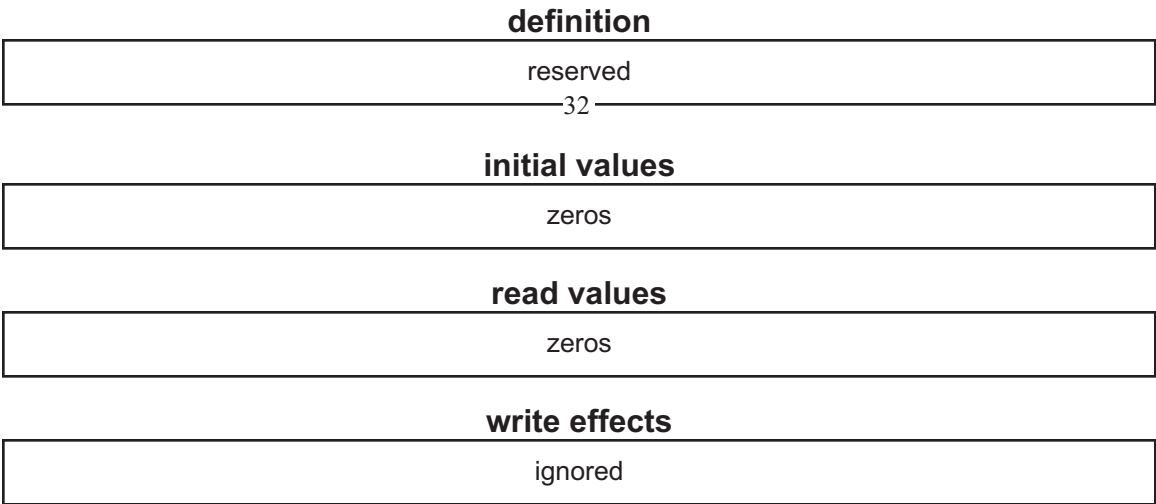


Figure 3-4 — Reserved CSR field behavior

This is straightforward as it applies to read and write requests. The same rules apply to lock requests, but the behaviors are less obvious; see IEEE Std 1394a-2000 for details.

4. Bridge model (informative)

A Serial Bus bridge consists of two *bridge portals* (each with its associated PHY and link), *queues* (FIFOs) for asynchronous and isochronous subactions (which collectively form an implementation-dependent *fabric* between the two portals), *cycle timers*, *route maps*, and *configuration ROM*. Figure 4-1 illustrates this model.

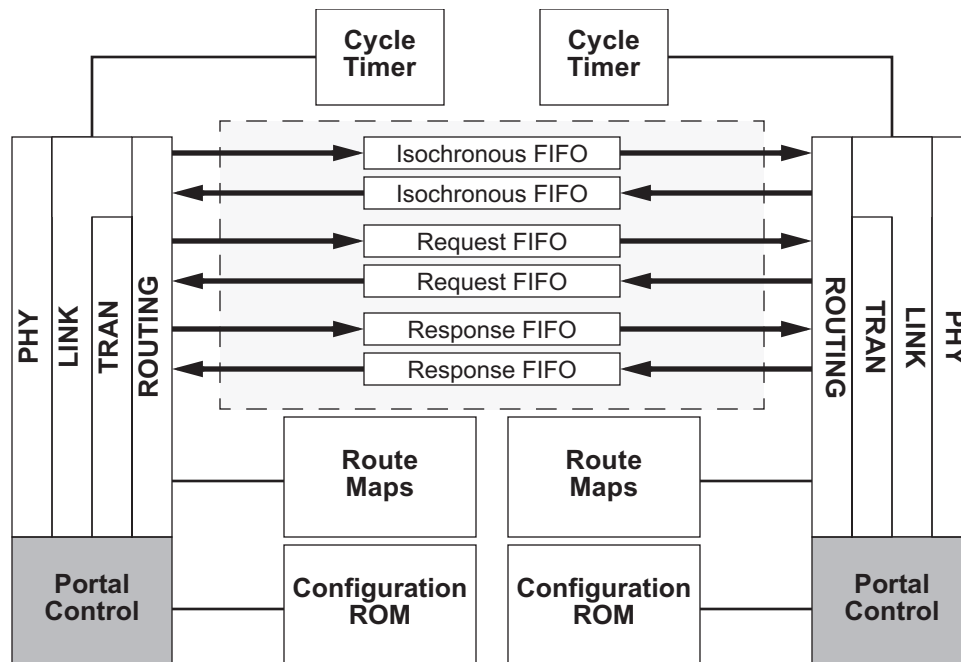


Figure 4-1 — Bridge model

Each bridge portal is a separate Serial Bus node, with a different EUI-64 and its own address space on the bus to which it is connected. A bridge portal responds to Serial Bus read, write, and lock requests from its connected bus as described in this standard. A bridge portal also monitors all Serial Bus subactions, asynchronous and isochronous, and uses route maps to determine which subactions, if any, are to be routed through the bridge's fabric to the co-portal.

The bridge portals are interconnected by means of a fabric that is capable of transferring any Serial Bus subaction from one portal to its co-portal. The fabric is conceptualized as a set of FIFOs (shown enclosed with dashed lines in Figure 4-1) that support bidirectional, non-blocking transfer of asynchronous request subactions, asynchronous response subactions, and isochronous subactions. Although this standard mandates some behavior for these queues, the details of the fabric implementation are not addressed by this standard. The fabric could be modest in geographical extent, as when both of the bridge portals and fabric are located within a single enclosure. Conversely, the fabric could be physically extensive, as could be the case if a bridge's portals were located in separate rooms. In both cases, the model remains the same.

Each portal has a cycle timer driven by a 24.576 MHz oscillator. The propagation of cycle time from one bridge portal to the other is implementation-dependent and beyond the scope of this standard.

The route maps contain information that permits the selective forwarding of both asynchronous and isochronous subactions through the fabric to the co-portal.

4.1 Global node IDs

A global node ID is a 16-bit object that consists of two parts, a 10-bit bus ID and a 6-bit virtual ID; the latter is analogous to the physical ID specified by IEEE 1394, but is not assigned by the self-ID process and does not necessarily change upon each bus reset. A global node ID may be present as a destination or source address in an asynchronous subaction—so long as at least one bridge portal is on the path between the source and the destination nodes.⁶ Global node IDs supplement local node IDs, but they do not replace them.

The usage of global node IDs and the role that bridge portals play in the conversion from local to global node IDs and vice versa can be understood in the context of the simple net topology illustrated by Figure 4-2. Bridges are shown as circles with a line to divide each portal from the other, while bus IDs are shown above the lines that represent different arbitration domains (the buses themselves). The requester and responder nodes are squares with the designation Rq and Rsp, respectively. Other nodes that might be present on a bus are omitted from the diagram.

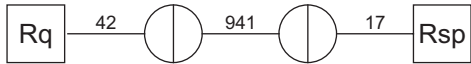


Figure 4-2 — Simple net topology (reference model)

When Rq originates a remote request subaction intended for Rsp, the *destination_ID* field routes the subaction toward bus 17. The initial entry portal (the bridge portal connected to bus 42) transforms the *source_ID* field from the local node ID transmitted by Rq to the corresponding global node ID. When the subaction is transmitted on bus 941, no transformation of either *source_ID* or *destination_ID* is necessary; both of these fields contain global node IDs. Once the request subaction arrives at the destination bus, the terminal exit portal (the bridge portal connected to bus 17) transforms the *destination_ID* field from a global node ID to a local node ID recognizable by Rsp. If Rsp responds, the global node ID present in the request subaction’s *source_ID* field is usable as the destination for the response subaction—and the process just described operates in reverse as the response subaction is transmitted to the requester’s bus. The table below summarizes the observable values for the subaction’s *destination_ID* and *source_ID* fields on each of the three buses.

Subaction	Bus	<i>destination_ID</i>		<i>source_ID</i>	
		Upper 10 bits	Lower 6 bits	Upper 10 bits	Lower 6 bits
Request	42	17	Virtual ID	3FF ₁₆	Physical ID
	941	17	Virtual ID	42	Virtual ID
	17	3FF ₁₆	Physical ID	42	Virtual ID
Response	17	42	Virtual ID	3FF ₁₆	Physical ID
	941	42	Virtual ID	17	Virtual ID
	42	3FF ₁₆	Physical ID	17	Virtual ID

The features of global node IDs that make them useful in a bridged environment are as follows:

- At any point in time, a one-to-one correlation exists between a node’s EUI-64 and its global node ID.
- A node’s virtual ID does not change solely because its physical ID changes after a bus reset.
- On each bus, one and only one bridge portal (the coordinator) is responsible to synchronize virtual ID assignments to the local nodes so that all bridge portals have identical mappings.

⁶The commonplace situation is for one or more bridges to separate nodes on different buses, but a bridge portal may also intercept and transform subactions between nodes on the same bus. These “echo” subactions use global node IDs.

- Global node IDs do not vary with the path to the node. On a bus with multiple bridge portals, each portal maps the same global node ID to the same local node ID.
- Global node IDs do not change unless asynchronous routes within the net change or unless no unallocated virtual ID is available when a new node is inserted on a bus.
- Whenever one or more global node IDs within a net might have changed, this information is reliably signaled on each bus within the net.

As already mentioned, a global node ID is composed of two parts, a 10-bit bus ID and a 6-bit virtual ID. 6-bit virtual IDs are managed locally on each bus by the coordinator, while 10-bit bus IDs are managed centrally for the entire net by the prime portal (see 4.3). This insures that within the scope of each there are no duplicated bus IDs or virtual IDs.

When a previously unrecognized node is inserted on a local bus, it is necessary for the coordinator to assign it a virtual ID and to communicate the virtual ID to all the other bridge portals on the bus. The coordinator maintains information, for each of the 63 possible virtual IDs, that indicates whether it is unallocated and, if allocated, the EUI-64 of the node to which it corresponds. When a node is inserted and its EUI-64 does not match one of those associated with a virtual ID, the coordinator chooses an unallocated virtual ID and assigns it to the node.

When a node is removed from a local bus, it does not immediately lose its virtual ID. The coordinator and all the other bridge portals retain the correlation between its EUI-64 and assigned virtual ID. Should the node subsequently be reinserted, it usually retains the same virtual ID. A node loses its virtual ID assignment if the coordinator requires a virtual ID for a previously unrecognized node and none of the 63 possible virtual IDs are available. In this case, the coordinator releases the current bus ID, discards all virtual ID assignments, and reassigns virtual IDs only for those nodes currently connected to the bus.

NOTE—When the coordinator releases the current bus ID, net update is initiated (see 4.3). This causes the alpha portal to request a new bus ID from the prime portal. Although the coordinator may reassign virtual IDs as soon as the bus ID has been released, the nodes with new virtual IDs are not addressable by global subactions until a bus ID is received from the prime portal.

In many cases subsequent to a bus reset, the coherency of the virtual ID mapping shared by all bridge portals can be maintained without communication between the portals and without intervention by the coordinator. Since the only event that causes the coordinator to assign a virtual ID is the insertion of a previously unrecognized node, bridge portals can rely on a topological analysis of self-ID packets to independently reestablish EUI-64 identity of all previously allocated virtual IDs (see Annex G). Analysis of self-ID packets establishes a correlation between EUI-64 and physical ID for all currently connected nodes, which in turn can be used to derive a tuple of physical ID, virtual ID, and EUI-64 for all nodes connected just prior to bus reset. When a previously unrecognized node is discovered, bridge portals may continue to use existing virtual ID mappings but must await the coordinator's assignment of a virtual ID to the new node.

4.2 Remote time-out

Within the context of the reference model illustrated by Figure 4-2, it is clear that a split transaction originated by Rq and responded by Rsp cannot be guaranteed to complete in the period used by Rq to time split transactions on the local bus. After the initial entry portal on bus 42 transmits *ack_pending* for the request subaction, the subaction will spend some time in the bridge's queues before it is forwarded across bus 941. The time a bridge persists in its attempts to forward a subaction to the next intermediate bridge portal is implementation-dependent; it might be longer than the requester's local bus split time-out. Even if the maximum forward time for a particular bridge is shorter than the requester's split time-out, the sum of all such times might exceed the requester's local split time-out. Finally, there is the split time-out on the destination bus. When the terminal exit portal transmits the request to Rsp, that node will assume that it has a local bus split time-out period within which to respond. When the response subaction returns by the reverse path to Rq, similar transit times are added to the total time that Rq has been awaiting a response.

The naturally longer time necessary for remote transactions leads to the definition of a **remote time-out** period analogous to local bus split time-out. Remote time-out is the sum of all the maximum forward times for all the bridges on the path between requester and responder plus the value of SPLIT_TIMEOUT register on the destination bus. The maximum forward times must be accumulated both for the request subaction on its outbound journey and for the response subaction

on its return trip. In the case of the response subaction, this includes the time the terminal exit portal (which was the initial entry portal from the perspective of the request subaction) persists in its attempt to deliver the response to the original requester.

Remote time-out is path dependent; a net management message exists to permit a requester to determine remote time-out for any valid global node ID. This message is processed by all intervening bridge portals, both in the outbound and return direction, and provides a vehicle within which the various maximum times can be accumulated. Once remote time-out value is obtained for a particular global node ID, it remains valid so long as the global node ID remains valid. From the perspective of a particular requester, the remote time-out for all nodes on a remote bus is identical to the remote time-out for any node on that bus.

NOTE—With reference to Figure 4-2, the remote time-out value obtained by Rq for Rsp might not be equal to the remote time-out value obtained by Rsp for Rq, since the local bus split time-out periods might be different for bus 42 and bus 17.

4.3 Clan affinity and net update

When both portals of a bridge are powered, it constitutes a net formed of the two buses connected to each of its portals. Even if no other devices are connected to those buses, a simple net exists. The bridge's manufacturer may make arbitrary design choices, such as the assignment of a bus ID to each of the buses, without the necessity for coordination with another bridge. However, once two bridge portals are connected to the same bus, coordination is required. A new concept, *clan affinity*, is used to describe how coordination between bridge portals is effected.

A net topology that could be found in a home network is illustrated by Figure 4-3. The figure assumes a star topology with the central bus, 42, located within a wiring closet; the clusters of equipment in other rooms (not shown) are connected through bridges whose cabling radiates from the wiring closet.

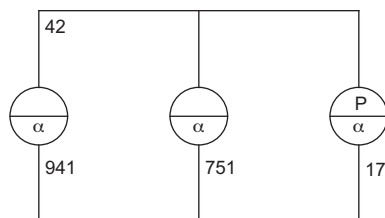


Figure 4-3 — Example home network topology

Under stable operating conditions, each of the bridge portals replicates the routing database for the net—but with slight variations that result from the portal's location within the net. The copies of the routing database are mutually consistent: no bus ID is duplicated within the net and valid routes to each bus ID exist from any point in the net. All of the bridge portals that share a consistent distributed database form a *clan*. A clan is identified by the EUI-64 of a single portal within the clan, called the *prime portal*. In Figure 4-3, the letter *P* designates the prime portal. A prime portal has three functions within a clan. The first is simply to provide the clan with an identity: the prime portal's EUI-64. This identity permits clans to be distinguished from each other when more than one clan is simultaneously connected to the same bus.⁷ The second is to maintain a central registry of allocated bus IDs; whenever a bridge portal requires a new bus ID for its attached bus, it requests one from the prime portal. The central registry of bus IDs prevents duplicate assignment of bus IDs to geographically separate buses. The third is to identify the net cycle master (see 4.4).

Although any portal within a clan may assume the role of prime portal and perform its functions as capably as any other portal, some geographic locations within a clan are superior to others because the identity of the prime portal tends to be stable over time. For example, in Figure 4-3 the prime portal is shown on the central bus (assumed to be located within

⁷The coexistence of two or more clans on a single bus is a temporary state of affairs that is resolved by merging one clan into the other, but until this process completes, it is essential to distinguish between the different clans.

the wiring closet). Net topology changes could occur in the outlying rooms of the home network, caused by the insertion or removal of bridges, without the necessity to relocate the prime portal. A portal within the net may be configured to indicate a preference that it be selected as the prime portal.

Also shown in Figure 4-3 are **alpha portals**, designated by the letter α ; these are portals that are on the route from the local bus towards the prime portal. Alpha portals perform a useful function when it is necessary to route an inter-bridge message to the prime portal without knowledge of the prime portal's global ID. It suffices to send the message to the alpha portal on the bus of origin. The alpha portal transfers the message to its co-portal—which either is the prime portal or forwards the message to the next alpha portal.

The net topology illustrated by Figure 4-3 would change if a bridge portal were removed from or added to any bus in the net. The removal of a bridge portal is straightforward from the perspective of clan affinity. The clan loses a member and the remaining bridge portals are informed of the changes in the routing database; there is no change to the prime portal or to the clan affinity of the remaining portals. When a portal is inserted, the process is more complicated, since two clans must be merged into one. The clan that retains its prime portal is the **survivor clan** while those that lose their former prime portal and transfer their affinity to the survivor clan are called **victim clans**. A hypothetical personal computer (PC), illustrated by Figure 4-4, is used to demonstrate the process that occurs upon the addition of a bridge to a bus that already includes one or more bridge portals.

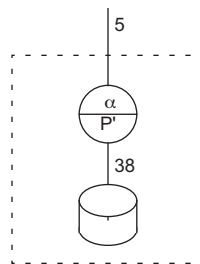


Figure 4-4 — Example PC with internal bridge

The PC is shown with a built-in bridge that separates an external connector from a private, internal bus. A mass storage device is connected to the internal bus; in this case, it is assumed to be advantageous to locate the system disk here so that it could have greater access to bus bandwidth than it would have if it competed with other devices external to the PC. So long as the PC remains disconnected from an external bus that contains other bridges, it forms a small net comprised of buses 5 and 38. The net has a prime portal, P' and an alpha portal. However, if a connection were made between bus 5 in Figure 4-4 and bus 17 in Figure 4-3, the topology illustrated by Figure 4-5 would result.

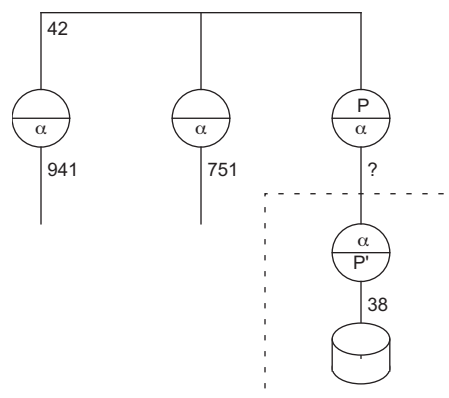


Figure 4-5 — Addition of a PC to the home network

When the connection is made between the PC and the home network, a bus reset is generated. After each bus reset, one bridge portal is selected to act as the **coordinator** to manage **net update**, a process that updates net configuration if its topology has changed. The role of coordinator is retained until the next bus reset, at which time it might shift to a different bridge portal. In this example, the coordinator detects that two clans, P and P' , are present on the bus: net topology has changed. Since there were two bus IDs previously assigned to what is now a single bus, its bus ID is temporarily unresolved.⁸ First, the coordinator determines which clan is to be the victim and which is to be the survivor. The algorithm employed is specified in Clause 10.. The coordinator combines the routing information from all clans into a new routing database and communicates this information to all the local bridge portals. The routing database in the example would include only bus IDs 17, 38, 42, 751, and 941—bus ID 5 is deleted because the just-reset bus retains its survivor clan bus ID assignment. At the same time that the routing database is transferred to each bridge portal, the coordinator informs it of its clan affinity. Within the victim clan, a portal's role as an alpha portal changes to that of a **subordinate portal** (any portal that is neither an alpha nor a prime portal) as it is adopted into the survivor clan. When these operations complete, the net is once again stable, as shown by Figure 4-6.

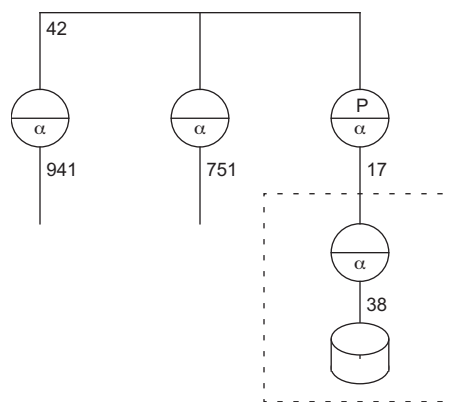


Figure 4-6 — Survivor clan with adopted PC

In Figure 4-6, the bridge within the PC has changed its clan affiliation and become part of the survivor clan, P .

4.4 Cycle time distribution and synchronization

Just as each bus has a single cycle master that provides uniform cycle time to that bus, a net requires a single cycle master to provide cycle offset for the entire net. This singular cycle master is named the **net cycle master**; its location is determined by the location of the prime portal. The cycle master connected to the same bus as the prime portal is the net cycle master.⁹

Net cycle offset originates at the net cycle master and bridges distribute it throughout the net. On the bus that contains the prime portal, all portals obtain cycle time from the net cycle master and communicate its cycle offset to their co-portals, which in turn regulate cycle start events on their own buses. On all buses except the prime bus, the alpha portal receives cycle offset synchronization information from its co-portal and the subordinate portals receive cycle time information from the cycle master—and in turn pass its cycle offset to their co-portals.

For bridges to reliably transport isochronous streams, it is necessary for all cycle masters within the net to maintain frequency synchronization with each other. Without frequency synchronization, cycle time drift might grow large enough to cause isochronous data overrun or underrun. In practice, the simplest method to obtain frequency synchronization is to

⁸Any bus IDs duplicated in different clans (none are shown in this example) are discarded by the coordinator. Once net update completes on a particular bus, if the alpha portal discovers it lacks an assigned bus ID, a request is made to the prime portal for a new bus ID.

⁹A node other than the prime portal can be the net cycle master. For example, a gateway node with access to an accurate external time source, such as a network interface to a satellite or terrestrial cable system, could be the net cycle master.

maintain an effective cycle offset phase difference of zero (phase synchronization) between cycle timers on adjacent buses. Cycle offset phase synchronization exists when cycle start events are simultaneous within tolerances inherent to Serial Bus cycle time distribution and measurement.

The distribution of cycle offset synchronization from the net cycle master requires that any two adjacent buses be synchronized to the upstream cycle master, *i.e.*, the one with fewer intervening bridges between itself and the net cycle master. All alpha portals (except the prime portal) are downstream of the cycle master on their co-portal's bus (which is referred to as the **upstream portal**) and therefore are responsible to synchronize cycle offset on their own bus to that on the upstream portal's bus. An alpha portal may regulate cycle offset on its own bus if either a) it is the cycle master or b) the cycle master is adjustable by means of "go fast" and "go slow" packets. These cycle master adjustment packets (see 6.3) permit the alpha portal to instruct a cycle master to delay or hasten the advent of the next cycle synchronization event by one tick of its 24.576 MHz cycle timer. Four items of data are sufficient for an alpha portal to determine whether a phase adjustment is required: simultaneous samples of `CYCLE_TIME.cycle_offset` from both of the bridge's portals, the propagation delay between the alpha portal and the downstream cycle master, and the propagation delay between the upstream portal and the upstream cycle master. Figure 4-7 illustrates the components of the feedback loop employed to maintain cycle offset phase synchronization between two buses in a net.

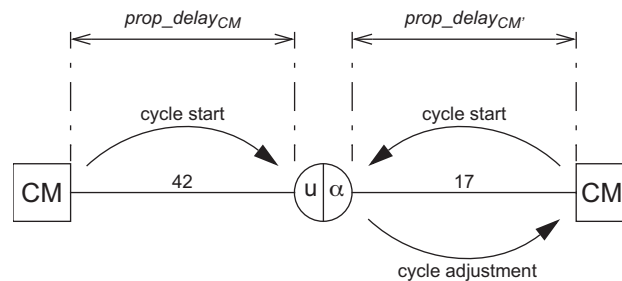


Figure 4-7 — Phase synchronization between two buses

The upstream and downstream (with respect to the net cycle master) cycle masters are marked CM and CM' ; within the bridge the upstream portal is labeled u while the alpha portal is identified by α . The phase difference between the two buses can be calculated by the following formula:

$$(\text{CYCLE_TIME.cycle_offset}_{\alpha} + \text{prop_delay}_{CM'}) - (\text{CYCLE_TIME.cycle_offset}_{\text{upstream}} + \text{prop_delay}_{CM})$$

Each bridge portal measures the propagation delay between itself and the cycle master on its bus by means of PHY ping packets (see IEEE Std 1394a-2000). The propagation delay is assumed to remain constant over time, within tolerances, unless the path between the portal and the cycle master alters. The propagation delay is half the round-trip delay measured by a ping packet. Both bridge portals have an independent 24.576 MHz cycle timer whose value can be sampled *via* the `CYCLE_TIME` register. Once each isochronous period, the phase of both cycle timers (visible as `CYCLE_TIME.cycle_offset`) is sampled and the result combined with the propagation delays according to the formula above. If the resultant phase difference is negative, the downstream cycle master is running slower than the upstream cycle master and the alpha portal instructs it to reduce the threshold value for the impending cycle start by one cycle timer tick. Otherwise, when a positive phase difference is observed, the downstream cycle master is too fast and the alpha portal instructs it to increase the threshold value by one cycle timer tick. Any adjustments made by the alpha portal to the cycle master's threshold value are in effect only for the next `CYCLE_TIME.cycle_count` increment, whereupon the threshold reverts to the nominal 3071 cycle timer ticks. An example of data that could be observed by a bridge, along with the calculated phase difference and the resultant "go fast" adjustment packet transmitted to the downstream cycle master, CM' , is captured in the table that follows.

Observed data	CYCLE_TIME.cycle_offset _{upstream}	25 ticks
	prop_delay _{CM}	7 ticks
	CYCLE_TIME.cycle_offset _{alpha}	15 ticks
	prop_delay _{CM'}	5 ticks
	Phase difference	−12 ticks
Result	Alpha portal transmits a cycle master adjustment packet to temporarily decrease the CYCLE_TIME.cycle_count increment threshold by one cycle timer tick ("go fast").	

NOTE—Although the preceding discussion assumes that the alpha portal and the downstream cycle master are distinct nodes, the principles are the same if the alpha portal is also the cycle master for its bus. In this case, the value of *prop_delay_{CM}* is zero.

4.5 Universal time

The preceding subclause explained the necessity for a single source of cycle time synchronization within the net (called the net cycle master) and described how bridge portals distribute synchronized cycle offset. Bridge portals, however, do not distribute a constant time reference throughout the net. There is no directly accessible universal time pervasive across the net: BUS_TIME and CYCLE_TIME.cycle_count may vary (and probably do vary) from bus to bus even though CYCLE_TIME.cycle_offset is maintained in phase lock synchronization on all buses.

A common time reference is valuable for many applications, whether the need for coordinated action on separate buses is coarse-grained (*e.g.*, a video source, such as a set-top box, becomes active at close to the same time that a destination, such as a DVCR, seeks to record the program) or more exacting, as when editing or mixing is performed. Although a universal time reference is not directly maintained within the net, facilities are provided that permit applications to derive a common sense of time. All that is required is a common reference bus mutually agreed by the applications.

Derivation of universal time is based upon a net management message that permits the time difference, in cycles, to be measured between any two buses within the net. Consider the example topology illustrated by Figure 4-3. Assume that it is possible to simultaneously sample BUS_TIME and CYCLE_TIME.cycle_count on all the buses,¹⁰ with the results summarized in the table below.

Bus ID	BUS_TIME	CYCLE_TIME	Cycles
17	3	0690 Cxxx ₁₆	26 316
42	1	022B 2xxx ₁₆	8690
751	5	0B56 1xxx ₁₆	45 473
941	2	04F3 1xxx ₁₆	19 889

The rightmost column represents the bus time in terms of cycles, that is, BUS_TIME seconds multiplied by 8000 plus CYCLE_TIME.cycle_count. From a simultaneous sample of bus time on all the buses, the time difference offset between any pair of buses can be derived, as shown in the table that follows. Note that the signed time offset in cycles is computed for ordered pairs of buses; at a given moment T_{source_bus} , the simultaneous time $T_{destination_bus}$ is obtained by adding the time offset to T_{source_bus} .

¹⁰ In fact, it is not possible to sample bus time simultaneously on different buses, but it is possible to obtain equivalent information by means of a net management message whose operation is described later in this subclause.

		Destination bus ID			
		17	42	751	941
Source Bus ID	17	0	−17 626	19 157	−6427
	42	17 626	0	36 783	11 199
	751	−19 157	−36 783	0	−25 584
	941	6427	−11 199	25 584	0

Equipped with knowledge of relative time offsets between arbitrary pairs of buses, applications located on buses remote with respect to each other may mutually agree upon a common time; all that is required is a shared point of reference. One such reference is the net cycle master itself. In order to use time on the prime bus, T_{prime_bus} , as the common time reference, applications need only convert their local time to the equivalent time on the prime bus and communicate it to the remote participant, which can then reconvert it to local time. Consider a set-top box located on bus 17 and a DVCR located on bus 941 in Figure 4-3. A user, *via* some unspecified interface such as a remote control, instructs the set-top box to transmit the program starting at a cycle count sometime in the future with respect to current local time on bus 17, T_{bus_17} . The user also wishes to program the DVCR on bus 941 to commence recording at the same time (this is to be accomplished indirectly, *via* the set-top box user interface). The set-top box converts T_{bus_17} to its equivalent time on the prime bus by subtracting 17 626 cycles and then communicates this T_{prime_bus} (along with channel number, bandwidth characteristics and other command information) to the DVCR, which then adds 11 199 cycles to obtain equivalent T_{bus_941} on its local bus.

This example shows how a common sense of time can be shared by applications on different buses even if there is no absolute sense of calendar date and time. All that is necessary is a common point of reference, relative to which “now” can be derived in terms of local bus time. The obvious candidate for shared reference is the prime bus—but there might instances for which another point of reference is superior. For example, if a network interface connected to a terrestrial cable system were located on bus 751 and accurate calendar date and time were available from the service provider, applications could share calendar date and time by determining the relative time offset, in cycles, between their local bus time and the time provided by the network interface. Determination of this relative time offset is a two-step process. First, the time difference between the local bus and bus 751 is obtained as described above. Second, the time difference, if any, between T_{bus_751} and the calendar date and time provided by the network interface is separately obtained and arithmetically combined with the relative time difference between the buses.

As already noted in a footnote (Footnote 10), it is not possible for devices (except bridges) to take simultaneous samples of bus time on different buses. In the examples given, a hypothetical simultaneous sample was used to derive the relative time offsets between any ordered pair of buses in the net. The same information can be obtained by other means: a net management request and its corresponding response. This method relies upon each bridge on the route between two buses to intercept the message, update the cumulative time difference between the two buses by adding the time difference measured between its own bridge portals, and forward it to the next bridge portal on the route. After the terminal bridge intercepts and processes the request message, it converts it to a response and transmits it directly to the originator without further processing by the intervening bridges.

Once the relative time difference between two buses has been measured, it does not change unless the value of `BUS_TIME` or `CYCLE_TIME.cycle_count` on either bus changes other than as a result of normal increment—in which case it is necessary to measure the relative time offset anew. Bridge portals do not change `BUS_TIME` or discontinuously alter `CYCLE_TIME.cycle_count` except during net update, which invalidates all cached relative time difference measurements. In addition, a bridge portal that observes a change, other than as a result of normal increment, in bus time or cycle time should initiate net update.

4.6 Stream connection management

Although bridges configure themselves with respect to asynchronous subaction routing and GASP transmitted on the default broadcast channel, isochronous stream communication between two nodes located on different buses requires explicit connection management. Messages are used to set up, tear down, and query the status of stream routing within the net.

This subclause introduces the reader to the basic principles of stream connection management; it uses the model net topology illustrated by Figure 4-8 as a reference. The drawing is simplified by the omission of most of the devices that would populate a real-world example—including the cycle masters and isochronous resource managers. Neither the prime portal nor the alpha portals are illustrated, since they are not relevant to the scenarios discussed.

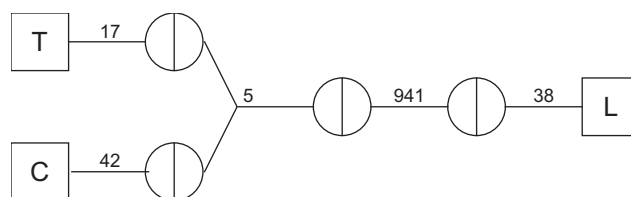


Figure 4-8 — Stream connection management (reference model)

Bridges are shown as circles with a line to divide each portal from the other. The bus IDs are shown above the lines that represent different arbitration domains. The controller, talker, and listeners are squares with the designation C, T, and L respectively. The following definitions are essential to understand the examples:

controller: A node that uses net management messages to set up or tear down routing for a stream between a talker and one or more listeners.

initial entry portal: The first bridge portal that snoops a subaction and transforms the subaction's *source_ID* from a local node ID to a global node ID.

listener: A node (other than a bridge portal) that is the recipient of a stream.

listening portal: A bridge portal that listens to a set of channel numbers so that their packets may be retransmitted by its co-portal.

reallocation proxy: A bridge portal responsible to reallocate channel numbers and bandwidth after a bus reset. The reallocation proxy also keeps track of listeners on the local bus; if the count of listeners ever drops to zero, either because of explicit connection teardown or node removal, the reallocation proxy releases the pertinent channel number and bandwidth.

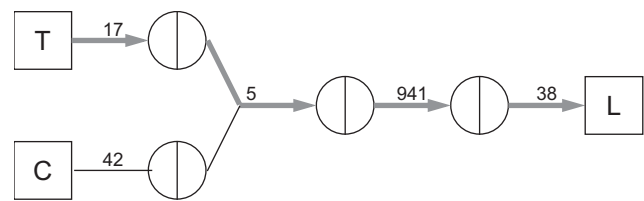
stream: Isochronous data originated by a talker and received by zero or more listeners. A stream is uniquely identified by the talker's EUI-64 and an index unique within the context of the talker.

talker: A node (other than a bridge portal) that is the source of a stream within the net.

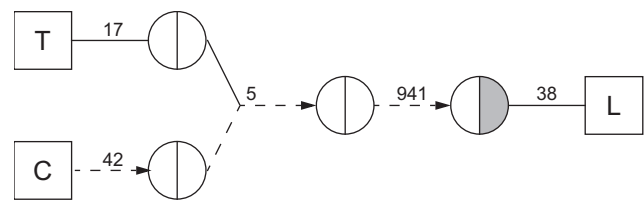
talking portal: A bridge portal that transmits stream packets for a set of channel numbers.

terminal exit portal: The last bridge portal that transmits (or could transmit) a subaction en route to its destination; the portal that normally transforms the subaction's *destination_ID* from a global node ID to a local node ID.

The first example is a simple connection setup that does not overlay an existing connection. The controller on bus 42 wishes to establish a path from the talker on bus 17 to the listener on bus 38 (shown by gray arrows in the figure below). Once the controller has received confirmation that the path has been created, it may instruct the talker to commence transmission of isochronous data.



The controller initiates the connection by transmitting a JOIN request as if it were intended for the listener. The JOIN request's *snarf* field¹¹ is one, which causes it to be intercepted by the terminal exit portal. The path of the JOIN request is indicated by dashed lines below; the portal that intercepts the JOIN request is shown shaded.



The JOIN request contains parameters sufficient to permit incremental set-up of the entire connection and eventual provision of completion notification to the originating controller. The parameters pertinent to this discussion are listed in the table below.

JOIN request parameters	
Parameter	Description
<i>talker_EUI_64</i>	Unique ID from talker's bus information block.
<i>talker_index</i> <i>listener_index</i>	Values unique within the context of the talker or listener that identify the stream.
<i>controller_ID</i> <i>talker_ID</i> <i>listener_ID</i>	Each is used during different phases of the stream setup process to route the JOIN request or other requests towards the appropriate bridge portal.
<i>max_payload</i> <i>aggregate_payload</i>	In conjunction with speed, permits calculation of bandwidth units to be allocated from BANDWIDTH_AVAILABLE.
<i>tspd</i> <i>lspd</i>	The maximum speed at which stream packets are to be transmitted by the talker or are to be transmitted to the listener (by the terminal exit portal), respectively.

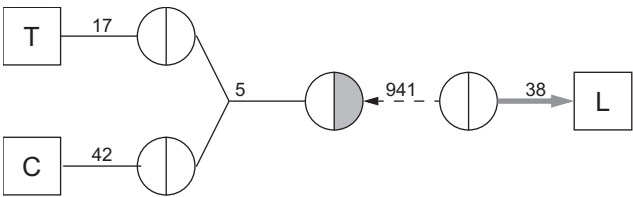
In addition to the parameters above, the composite of *talker_EUI_64* and *talker_index* form another parameter, *stream_ID*, which uniquely identifies a stream within a net of interconnected buses. A stream ID remains valid across bus resets and net topology changes; it persists until the stream, explicitly or automatically, is torn down.

The initial recipient of the JOIN request, a terminal exit portal, becomes the reallocation proxy on its local bus for the stream identified by *stream_ID* only if it is the talking portal for the stream. A reallocation proxy maintains a database for all active isochronous streams, keyed upon *stream_ID*. The database includes the information summarized by the following table.

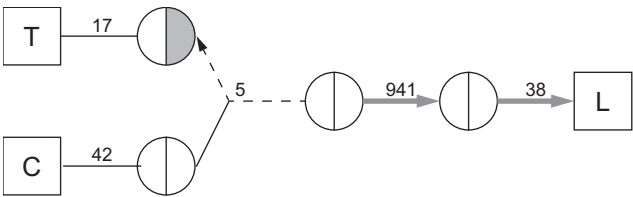
¹¹ A field in the block write packet header, newly defined by this standard. It may cause some or all of the bridge portals to intercept and process the request subaction en route to its destination. See 6.5 for details.

Stream context information maintained by a reallocation proxy	
Parameter	Description
<i>stream_ID</i>	Uniquely identifies a stream within a net of interconnected buses.
<i>controller_ID</i>	Provide error or other status information to the controller if the reallocation proxy takes action that affects the stream.
<i>channel</i>	The channel number allocated by the reallocation proxy from CHANNELS_AVAILABLE for the stream's use on the local bus
<i>payload</i>	If nonzero, the maximum payload permitted in a stream packet transmitted on the local bus. When zero, there is no payload limit.
<i>speed</i>	The speed at which stream packets are to be transmitted.
<i>bandwidth</i>	The number of bandwidth units allocated for the stream's use on the local bus. It is calculated from payload and speed but also includes an allowance for isochronous arbitration overhead.
<i>listeners</i>	A bit map that represents each node on the local bus for which a JOIN request has been successfully processed. The use of a bit map in place of a count permits the reallocation proxy to monitor the removal of listeners.

Upon receipt of a JOIN request, the reallocation proxy first reserves the necessary resources, channel number and bandwidth on its local bus. If successful in its attempted reservations, the reallocation proxy's local bus is configured for the stream (indicated by the gray arrow in the next figure below) and an iterative process is started that extends a stream path backward from the listener's bus to the talker's bus. This process uses a JOIN request sent to either a) the talking portal (or what will be the talking portal) for the stream on the immediately upstream bus or b) a portal selected to be a reallocation proxy on the talker's bus. This JOIN request is essentially the same as the initial JOIN request originated by the controller except that its *destination_ID* is the talker's (not the listener's) global node ID and the *snarf* field has a value of three, which causes it to be intercepted by all portals on its route. The path of this JOIN request, originated by the terminal exit portal, is shown by a dashed line below:



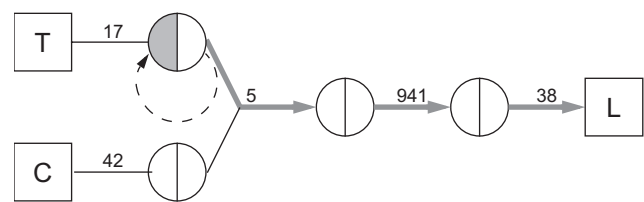
If the recipient of this JOIN request is not already a reallocation proxy for the stream identified by *stream_ID* it becomes one, as described above, and allocates the necessary resources from the isochronous resource manager on its bus. If the recipient is already the talking portal and reallocation proxy for the stream, it simply adds the new listener (the bridge portal identified by *source_ID* in the JOIN request's packet header) to its bit map of members on the local bus. In either case, if resources are successfully allocated, the process is iterated as shown below:



In this iteration, the JOIN request is once again addressed to the talker's *destination_ID*; the *snarf* field causes it to be intercepted by the next entry portal on the route back to the talker. Resources were successfully allocated on bus 941; the new JOIN will cause resources to be allocated on bus five and the bit map of local listeners to be updated.

Eventually the connection’s path will be established on all intermediate buses and the terminal bus that contains the listener. At this point, the behavior required of a JOIN request changes. All of the previous JOIN requests have nominated a bridge portal as the reallocation proxy for a particular bus—and each of these bridge portals is also the talking portal that transmits the stream on the bus. However, now that the JOIN message has arrived at the talker’s bus, there is no need for a talking portal: the stream’s source is the talker itself. Yet, there is still a requirement for a reallocation proxy on the talker’s bus, since the talker is not necessarily a controller and is not necessarily responsible to reallocate stream resources after bus reset.

In this case, the listening portal on the talker’s bus is designated as a reallocation proxy. The last step in resource allocation is shown by the figure below. No Serial Bus transaction is involved—the listening portal is the recipient of the JOIN request from its co-portal, as is indicated by the dashed line. More than one bridge on the talker’s bus could be listening to the same stream. In this case, all the listening portals are potential reallocation proxies; a cooperative connection management protocol is used to avoid duplicate allocation of resources.

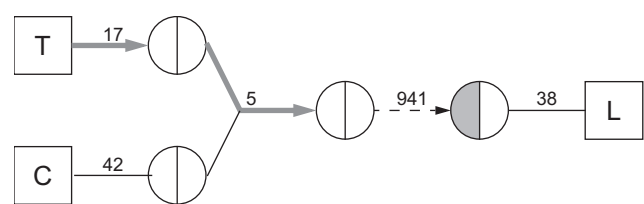


Once the final set of resources are allocated (on the talker’s bus), it is necessary to communicate to the various listening portals on the route from talker to listener the channel number they should associate with a particular stream ID. This is accomplished by a LISTEN request propagated and modified by the listening portals. The parameters included within a LISTEN request are shown in the table below.

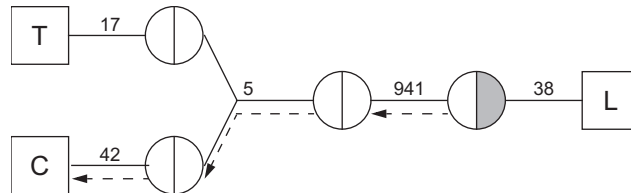
LISTEN request parameters	
Parameter	Description
<i>stream_ID</i>	Uniquely identifies a stream within a net of interconnected buses.
<i>channel</i>	The channel number on which the talking portal for the bus will transmit the stream.

The process starts after the listening portal on the talker’s bus has either confirmed that resources are already allocated or else allocated resources in its role as reallocation proxy for the stream. The listening portal transmits a LISTEN request that identifies the channel number allocated and associates it with the stream. The request’s *destination_ID* is set to the value of the listener’s global node ID (saved from the JOIN request originated by the controller) and the *snarf* field is set to three so the packet will be intercepted by all listening portals on the way towards the listener.

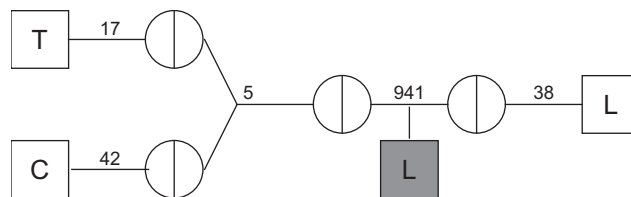
When a listening portal receives a LISTEN request, it configures itself to listen to the indicated *channel* and forward stream packets to its co-portal. The listening portal also forwards the LISTEN request to its co-portal, which modifies the request to identify the channel number used when the co-portal (talking portal) transmits the stream on its bus. The talking portal retransmits the LISTEN request, which is still addressed to the listener on the terminal bus. This process iterates for all intermediate buses until the terminal bridge intercepts the LISTEN request. The figure below shows the arrival of the request at the terminal bridge; at this point, all of the upstream buses are fully configured and able to transport the stream.



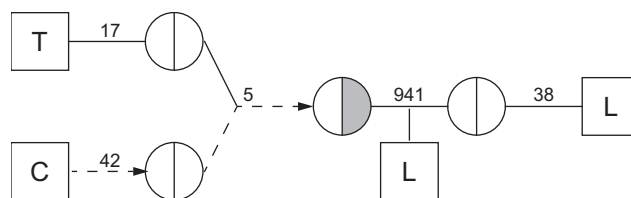
The terminal bridge has several jobs. First, it configures itself to listen on the indicated channel (as have all of the upstream listening portals). Next, it passes the request to its co-portal—but instead of further propagation of the request to another listening portal, the co-portal configures the listener to receive the stream on the allocated channel. Once this step is complete, the terminal exit portal transmits a STREAM STATUS message to the controller that originated the JOIN request. This confirmation contains the *stream_ID* supplied by the controller in the original request (the path of the confirmation message, transmitted by the portal that intercepted the original JOIN request from the controller, is shown by a dashed line).



What happens in the case that a new listener is to be added to an existing multicast group? The operations are similar, even though the route from the talker to the new listener might completely or partially overlay the path already in use. Consider the addition of a new listener (shown shaded).

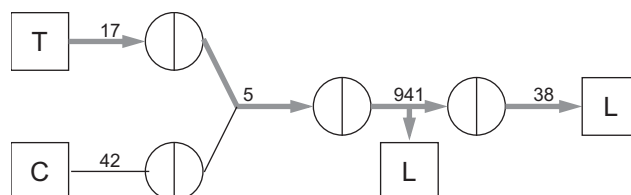


The controller transmits a JOIN request to the listener to be added; the *snarf* field is one to cause the terminal exit portal to intercept the request.

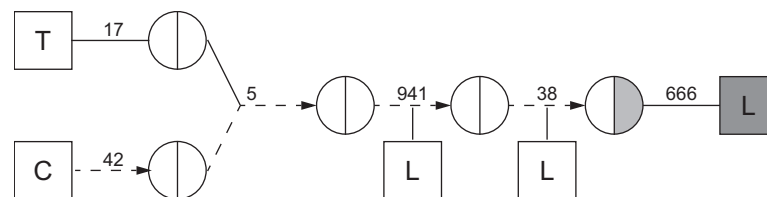


The sequence of JOIN, LISTEN, and STREAM STATUS messages generated by the bridge portals is exactly analogous to that already described, except that no new resources are allocated. The shaded bridge portal, in its role as reallocation proxy, updates its bit map of members to indicate the presence of the new listener but otherwise has no additional work to perform.

The gray arrows in the figure below show the resultant stream path once the operations have completed.



If the overlaid connection involved a new listener beyond the current extent of the stream's flow, the stream path is extended to accommodate the added listener. This is illustrated below.



The variations on this theme are endless (complex multicast routes with multiple branches could be created) but the basic operations of JOIN, LISTEN, and STREAM STATUS remain the same.

When a stream connection to a particular listener is no longer needed, it is usually torn down in an orderly fashion by a LEAVE message. Just as is the case for the JOIN message, the stream controller transmits a LEAVE message toward the listener. When it reaches the talking portal on the listener's bus, the connection with the listener is deleted and, if no other listeners remain on the bus, the stream's isochronous resources (channel and bandwidth) are released. Once the resources are released, the talking portal transmits the LEAVE message toward the talker; its receipt by intervening portals on the path between talker and listener causes resources to be released. After all the resources on the path have been released, the last portal process the LEAVE message transmits a STREAM STATUS message to the stream controller.

An isochronous connection can also be torn down in response to an unexpected disconnection anywhere along the path from talker to listener. For example, if the talker is disconnected, all of the listening portals on the talker's former bus detect this event and initiate stream teardown. If a listener is disconnected, its talking portal detects the event and initiates stream teardown for that listener, only. If the connection between an intermediate talking portal and an intermediate listening portal is severed, two actions take place: the talking portal initiates stream teardown for the disconnected listening portal, while the listening portal initiates stream teardown for all of its downstream listeners.

5. Bridge portal and bridge-aware node facilities

This clause describes the facilities that a bridge portal or a bridge-aware node shall support in order to interoperate with other bridge portals. These facilities are configuration ROM entries (which identify the presence of a bridge within a Serial Bus node) and control and status registers (CSRs—which can be used to control the operations of or obtain information from a bridge portal or bridge-aware node).

5.1 Configuration ROM

Bridge-aware nodes and bridge portals shall implement configuration ROM in the general format defined by IEEE 1394. Annex H contains a sample of valid configuration ROM for a bridge portal and illustrates the usage of the entries defined below.

5.1.1 Bus information block

A bridge-aware node’s or bridge portal’s configuration ROM shall contain a bus information block, as defined by this standard in Figure 5-1.

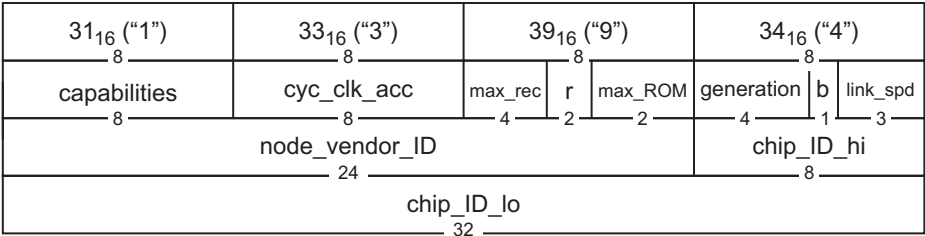


Figure 5-1 — Bus information block format

The definition and usage of all fields specified by IEEE 1394, with the exception of *max_rec*, are unchanged. Other fields are new while some have particular applicability to Serial Bus bridges, as defined below.

The *capabilities* field consists of individual bits shown in Figure 5-2.



Figure 5-2 — Bus information block *capabilities* field

A bridge portal shall report values of one for the *irmc*, *cmc*, and *adjustable* bits; bridge portals and bridge-aware nodes whose *irmc* bit is one shall implement the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers with the lock (*compare_swap*) algorithms specified by IEEE Std 1394a-2000. The *isc* bit does not describe any bridge capabilities, only the node’s isochronous capabilities on the local bus. If the node is a bridge portal, consult the Bridge_Capabilities entry in configuration ROM to determine whether the portal is capable of forwarding isochronous data to its co-portal. When the *adjustable* bit is one, the node implements support for the cycle master adjustment packets defined in 6.2 and interprets them as specified by 8.1.2. If the *adjustable* bit is one, the *cmc* bit shall also be one and the node shall implement support for the PHY ping packet specified by IEEE Std 1394a-2000. Bus manager and power manager capabilities are orthogonal to the other capabilities; the *bmc* and *pmc* bits shall be set according to the presence or absence of these optional capabilities.

NOTE—Nodes other than bridge portals may implement an adjustable cycle master and report this capability by means of the *adjustable* bit.

The *max_rec* field specifies the maximum payload that a bridge portal accepts in any of an asynchronous block write request, lock request, block read response, lock response, or asynchronous stream packet. This usage is an extension to that defined by IEEE 1394, which specifies only the size of an asynchronous block write request. When *max_rec* is zero the portal's maximum payload is not specified. Otherwise, the maximum payload is defined as 2^{max_rec+1} bytes and shall be the maximum asynchronous data payload permitted by the slower of each portal's fastest PHY port.

NOTE—The maximum payload for an isochronous packet has no relationship to the *max_rec* field; see the *max_isoch* field defined in 5.1.4.

The *bridge_aware* bit (abbreviated as *b* in the Figure 5-1), when set to one indicates the node complies with the requirements for a bridge-aware device specified in 9.2 and elsewhere within this standard. A bridge portal shall be bridge-aware and report a value of one for this bit.

The *node_vendor_ID*, *chip_ID_hi*, and *chip_ID_lo* fields collectively form the bridge portal's EUI-64, whose value shall not be equal to the value of the co-portal's EUI-64.

5.1.2 Node_Capabilities entry

The Node_Capabilities entry is obsolete. Conforming devices may include the entry in their root directory, as specified by IEEE 1394, but this is not recommended.

5.1.3 Bus_Dependent_Info entry

The Bus_Dependent_Info entry is a directory entry in the root directory that specifies the location of the Bus_Dependent_Info directory within configuration ROM. Figure 5-3 shows the format of this entry.

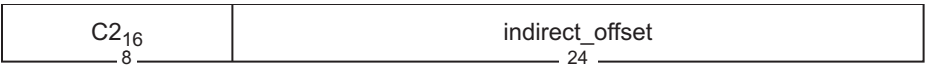


Figure 5-3 — Bus_Dependent_Info entry format

The entry is identified by the *key* field, which has a value of C2₁₆.

The *indirect_offset* field specifies the number of quadlets from the address of the Bus_Dependent_Info entry to the address of the Bus_Dependent_Info directory within configuration ROM.

5.1.4 Bridge_Capabilities entry

The Bridge_Capabilities entry is an immediate entry in the Bus_Dependent_Info directory that identifies the presence of a bridge portal within the node and specifies its capabilities. Bridge portal configuration ROM shall contain a Bridge_Capabilities entry even at times when the node's self-ID packets report a *brdg* value of zero. Figure 5-4 shows the format of this entry.

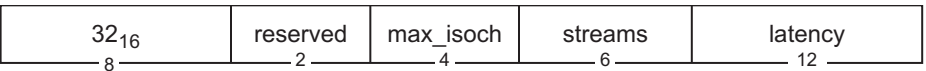


Figure 5-4 — Bridge_Capabilities entry format

The entry is identified by the *key* field, which has a value of 32₁₆.

The **max_isoch** field specifies the maximum isochronous subaction data payload the bridge is capable of transporting from the fastest PHY port of one portal to the fastest PHY port of the other. The bridge's internal fabric shall be capable of transporting isochronous subactions at least as large as those permitted by the slower of each portal's link speed. When **max_isoch** is zero, the portal's maximum isochronous payload is not specified. Otherwise, the maximum isochronous payload is defined as $2^{\text{max_isoch}+1}$ bytes. The value of **max_isoch** shall be the same for both portals of a bridge.

The **streams** field specifies the maximum number of concurrently active isochronous streams supported by the bridge. A value of zero indicates that the bridge does not forward isochronous streams. The value of **streams** shall be the same for both portals of a bridge. With the exception of heterogeneous bridges, **streams** shall be nonzero.

NOTE—A bridge might not be able to support as many concurrently active isochronous streams as the maximum reported in the **streams** field if the aggregate bandwidth of the streams exceeds the bridge's resources.

The **latency** field specifies the constant delay that isochronous packets incur when they are transferred from a bridge's entry portal to a listener on the bus connected to the bridge's exit portal. If the bridge implements no capability to forward isochronous streams (*i.e.*, the **streams** field is zero) or if different isochronous streams can incur different constant delays, the value of **latency** shall be zero. Otherwise, the **latency** field shall specify the delay in units of 125 μs and shall have a minimum value of two. The value of **latency** may be different for each of a bridge's portals.

5.2 Control and status registers

The control and status registers (CSRs) implemented by a bridge portal or bridge-aware node shall conform to the requirements defined by this standard and its normative references. The CSRs belong to three groups:

- Core registers specified by IEEE Std 1212-2001
- Bus-dependent registers specified by IEEE 1394
- Registers specified by this standard

The addresses of all registers are specified in terms of offsets, in bytes, within register space, where the base address of register space is FFFF F000 0000₁₆ relative to node space. IEEE 1394 shall be consulted for detailed descriptions of both core and Serial Bus-dependent registers; Table 5-1 lists those that are mandatory for bridge-aware nodes.

Table 5-1 — Bridge-aware node control and status registers

Offset	Name	Description
0	STATE_CLEAR	State and control information.
4	STATE_SET	Sets STATE_CLEAR bits.
8	NODE_IDS	Contains the 16-bit <i>node_ID</i> value used to address the bridge portal locally.
C ₁₆	RESET_START	Initiates a command reset; see individual register definitions for the effects (if any).
18 ₁₆ – 1C ₁₆	SPLIT_TIMEOUT	Time limit for split transactions on the connected bus.
80 ₁₆ – BC ₁₆	MESSAGE_REQUEST	Well-known addresses for receipt and protocol demultiplex of 64-byte messages in a standard format.
C0 ₁₆ – FC ₁₆	MESSAGE_RESPONSE	
214 ₁₆	QUARANTINE	Permits bridge-aware nodes (which include bridge portals) to manage their quarantine periods.

In addition to those registers required of bridge-aware nodes, bridge portals shall implement the registers listed in Table 5-2.

Table 5-2 — Bridge portal control and status registers

Offset	Name	Description
200 ₁₆	CYCLE_TIME	24.576 MHz cycle timer required for isochronous operation.
204 ₁₆	BUS_TIME	System time in seconds.
210 ₁₆	BUSY_TIMEOUT	Controls the bridge portal's retry protocols.
21C ₁₆	BUS_MANAGER_ID	Contains the 16-bit <i>node_ID</i> of the bus manager on the connected bus, if one is present.
220 ₁₆	BANDWIDTH_AVAILABLE	Known location for Serial Bus bandwidth allocation.
224 ₁₆ – 228 ₁₆	CHANNELS_AVAILABLE	Known location for Serial Bus channel allocation.
234 ₁₆	BROADCAST_CHANNEL	Identifies the channel number used for asynchronous broadcast.
1C00 ₁₆ – 1DFC ₁₆	VIRTUAL_ID_MAP	Correlates EUI-64s of nodes on the local bus with virtual IDs assigned by the coordinator.
1E00 ₁₆ – 1EFC ₁₆	ROUTE_MAP	Asynchronous routing information for each bridge portal.
1F00 ₁₆ – 1F04 ₁₆	CLAN_EUI_64	Identifies the clan of which a bridge portal is a member.
1F08 ₁₆	CLAN_INFO	Clan information (bus ID, bridge hops to prime portal, <i>etc.</i>) used during net update.

The CYCLE_TIME, BUS_TIME, BUS_MANAGER_ID, BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE, and BROADCAST_CHANNEL registers are required because a bridge portal might need to function as an isochronous resource manager.

The QUARANTINE, VIRTUAL_ID_MAP, ROUTE_MAP, CLAN_EUI_64, and CLAN_INFO registers are particular to bridge functions and are specified by this standard. If *brdg* was zero in a bridge portal's most recent set of self-ID packets, it may respond to requests addressed to these registers even though their contents might not be meaningful.

5.2.1 QUARANTINE register

This register permits bridge-aware nodes and bridge-portals to manage their own quarantine periods. Both bridge-aware nodes and bridge portals shall implement the QUARANTINE register. The contents of the QUARANTINE register are updated in response to analysis of self-ID packets observed by the node and are affected by write requests.

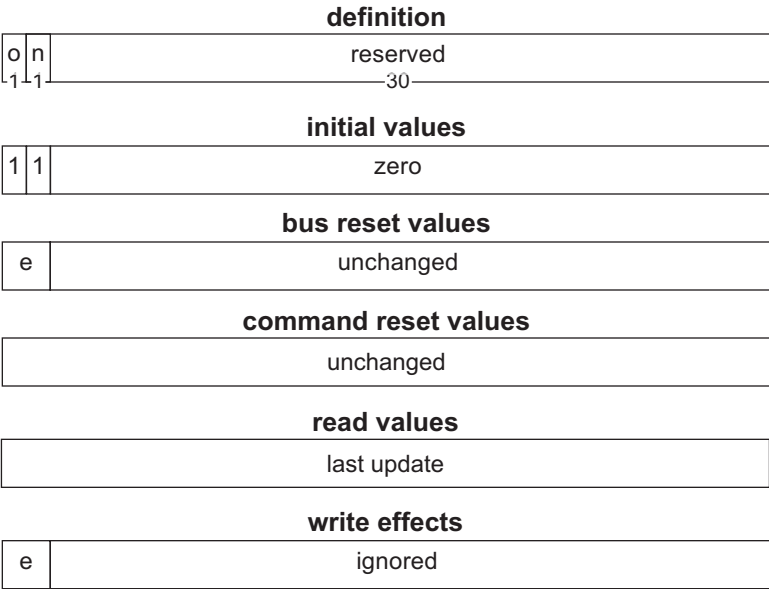


Figure 5-5 — QUARANTINE format

The *orphan* bit (abbreviated as *o* in Figure 5-5) indicates whether a global node ID is assigned to the node. A zero value indicates that the node is assigned a valid global node ID, while a value of one indicates uncertainty. When this bit is zero, the node may originate both global and local subactions. Otherwise, when *orphan* is one, the node shall not originate global subactions.

The *net_update* bit (abbreviated as *n* in Figure 5-5) indicates whether net update is in progress. Bridge-aware nodes that are not also bridge portals shall reserve this bit; its value shall be zero. Bridge portals shall implement the *net_update* bit with an initial value of one. When this bit is zero, the bridge portal may transmit both global and local subactions. Otherwise, when *net_update* is one, the bridge portal shall not transmit global subactions.

NOTE—“Originate” is a subset of “transmit.” If a subaction’s *source_ID* contains a local node ID, the subaction was originated by the node that transmitted it; however, if it contains a global node ID, the subaction was transmitted—but not originated—by a bridge portal.

A bridge-aware node or bridge portal shall set its *orphan* and *net_update* bits (when implemented) to one if, subsequent to a bus reset, any of the following conditions exist:

- One or more self-ID packets (including the bridge portal’s own self-ID packet) are observed whose *brdg* field is equal to three.
- Topology analysis of the self-ID packets (see Annex G) reveals that one or more bridge portals, not present on the bus prior to the bus reset, have been connected to the bus.
- Topology analysis of the self-ID packets reveals that one or more bridge portals, present on the bus prior to the bus reset, have been removed from the bus.
- Topology analysis of the self-ID packets reveals that one or more nodes, present on the bus prior to the bus reset, has reported a *brdg* field whose value changed from zero to nonzero (or vice versa) relative to the value reported prior to the bus reset. This last condition indicates the virtual insertion or removal of a bridge portal.

Any of these conditions indicates that either the node does not have an assigned global node ID or that net update is underway. When the state of either the *orphan* or *net_update* bit changes from zero to one, the node has recognized that quarantine has begun. Once either bit has been set to one, it shall retain its value through subsequent bus resets until zeroed. When the *net_update* bit changes from one to zero, a bridge portal shall set the value of *brdg* to two.

NOTE—The values of the *net_update* and *orphan* bits might not be identical for all nodes on a bus. For example, when a bridge-aware node is connected to the bus, its *orphan* bit is one but all the other node’s *orphan* bits remain zero. The already connected nodes do not perceive a net topology change but the newly connected node observes a bridge portal not present before the bus reset.

Write requests addressed to the QUARANTINE register affect the value of the *orphan* and *net_update* bits according to the value written. A zero written to either bit shall zero the bit while a one written to either bit shall not affect its value. Whenever zero is written to QUARANTINE.*orphan*, the bridge portal shall clear its *transmit_virtual_ID_map* flag (see 11.1) to FALSE. A bridge portal ignores writes to its QUARANTINE register during certain critical periods that can occur during net update; see 10.5 for details.

5.2.2 VIRTUAL_ID_MAP register

This 512-byte register correlates an EUI-64 with each of the 63 possible virtual IDs assignable to nodes on the local bus. It also identifies the alpha portal. The format of the register is specified by Figure 5-6.

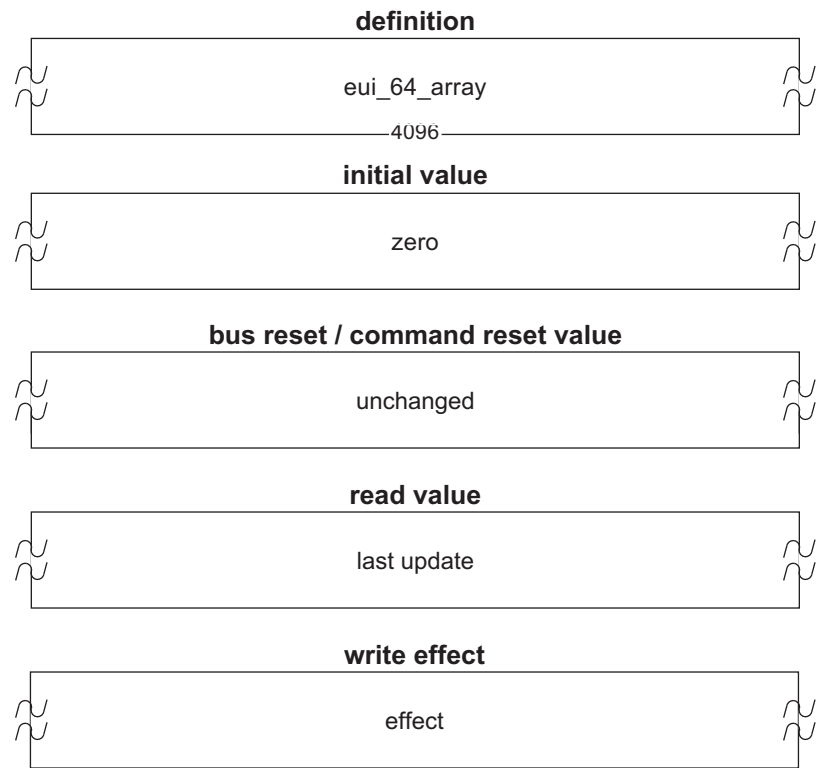


Figure 5-6 — VIRTUAL_ID_MAP format

The *eui_64_array* consists of 64 eight-byte entries, each of which contains an EUI-64. With the exception of the last entry, the value of VIRTUAL_ID_MAP.*eui_64_array*[*n*] shall specify the EUI-64 of the node assigned virtual ID *n*. An entry whose value is zero indicates an unassigned virtual ID. Since 3F₁₆ is invalid as a virtual ID, its corresponding entry in the virtual ID map is used for a different purpose. VIRTUAL_ID_MAP.*eui_64_array*[63] shall contain the EUI-64 of the alpha portal; in no other case shall a nonzero EUI-64 be present more than once in the array.

Bridge portals shall reject write requests addressed to any portion of the VIRTUAL_ID_MAP register except block write requests originated by the coordinator with a *data_length* of 512 bytes addressed to the start of the register’s address space. The coordinator maintains its own VIRTUAL_ID_MAP autonomously, without need of block write requests. A bridge portal that receives a 512-byte write addressed to its VIRTUAL_ID_MAP register shall set its *transmit_virtual_id_map* flag (see 11.1) to TRUE.

5.2.3 ROUTE_MAP register

This is a 256-byte register that contains the bridge portal’s asynchronous routing information for all 1023 possible buses within a net. The format of the register is specified by Figure 5-7.

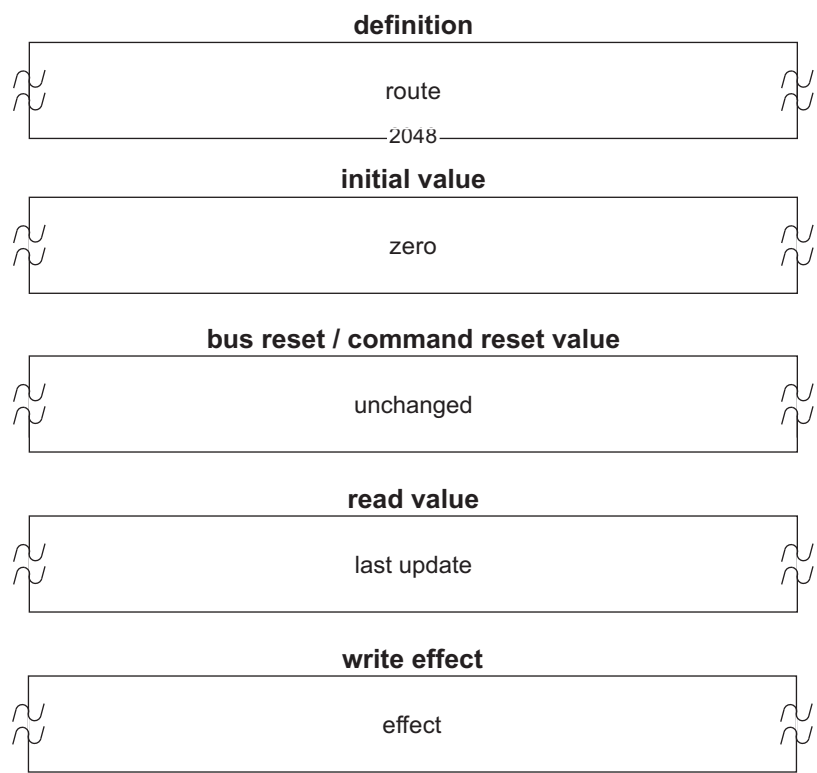


Figure 5-7 — ROUTE_MAP format

The *route* data is an array of 1024 two-bit entries, each of which represents the asynchronous subaction routing state for the corresponding bus ID. Although the local bus ID, 3FF₁₆, is not routable, its space in the array is allocated for the sake of simplicity. For a particular *bus_ID*, reference is made to byte *n*, relative to zero, within the *route* data, where *n* is equal to *bus_ID* / 4. Figure 5-8 illustrates the arrangement of two-bit *route* entries within a byte.

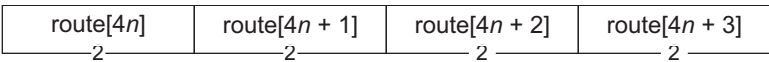


Figure 5-8 — ROUTE_MAP entries format (within one byte)

NOTE—Throughout this standard, the notation ROUTE_MAP[*bus_ID*] represents the *route* entry that corresponds to *bus_ID*.

Each *route* entry shall have a value specified by Table 5-3. The *route* entry for the unused local bus ID, 3FF₁₆, shall be CLEAN.

Table 5-3 — ROUTE_MAP entries encoding

Value	Name	Comment
0	CLEAN	The bus ID is not in use within the clan; subactions addressed to this bus ID are not forwarded by any portal.
1	DIRTY	A valid route to the bus ID no longer exists; subactions addressed to this bus ID are not forwarded by any portal. This is a transient state that persists until net update completes.
2	VALID	The bus ID is in use within the clan but subactions addressed to this bus ID are not forwarded by this portal.
3	FORWARD	The bus ID is in use within the clan and subactions addressed to this bus ID are forwarded by this portal.

5.2.4 CLAN_EUI_64 register

The CLAN_EUI_64 register identifies the clan to which the bridge portal belongs. A clan is identified by the EUI-64 of one of the clan’s bridge portals, designated the prime portal. The register is specified by Figure 5-9.

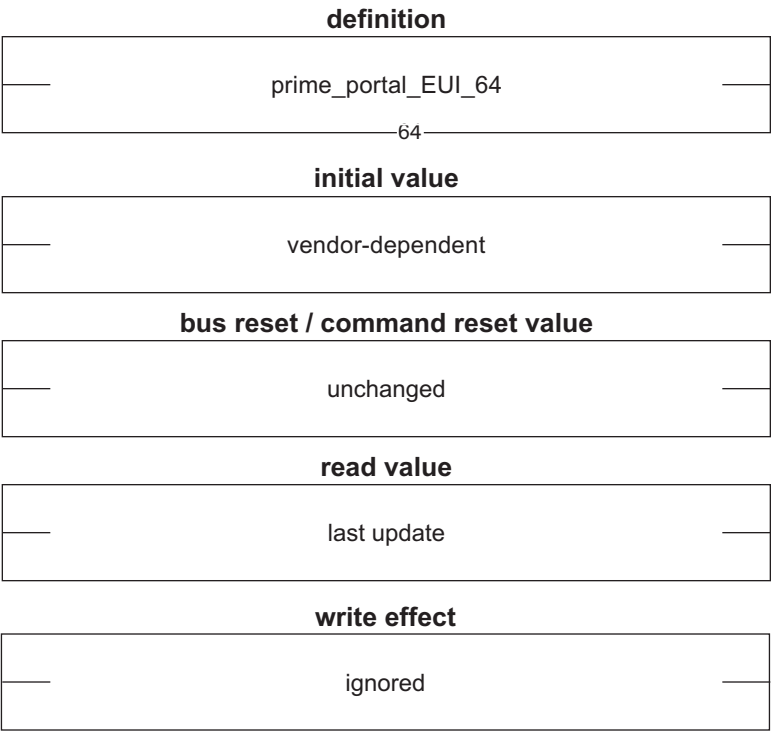


Figure 5-9—CLAN_EUI_64 format

The *prime_portal_EUI_64* field shall contain the same value as the *eui_64* field in the prime portal’s bus information block. The initial value shall be equal for both portals of a bridge, shall be the EUI-64 of either the bridge portal or its co-portal, and shall be the larger byte-wise reversed EUI-64. The comparison of the two EUI-64 values shall be performed in

byte-wise reverse order, *i.e.*, for the purpose of the unsigned comparison, the least significant byte shall be treated as if it were most significant, the most significant byte shall be treated as if it were least significant, and the interior bytes shall be compared in reverse order of their normal significance.

5.2.5 CLAN_INFO register

The CLAN_INFO register contains state information for the bridge portal and its clan, as specified by Figure 5-10.

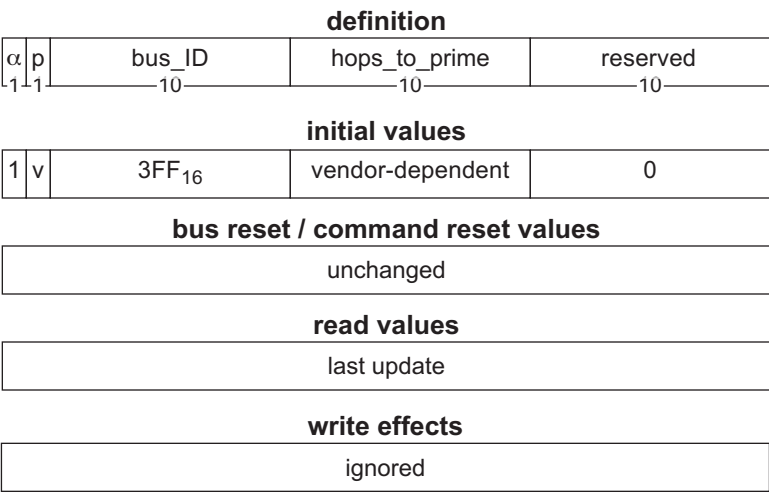


Figure 5-10—CLAN_INFO format

The *alpha* bit (abbreviated by the letter α in Figure 5-10) shall be zero if the bridge portal is a subordinate portal. If the portal is an alpha portal, the *alpha* bit shall be one. An alpha portal whose EUI-64 in its bus information block is equal to the value of its CLAN_EUI_64 register is a prime portal.

NOTE—The initial value of *alpha* is one since upon completion of power reset initialization a bridge consists of a prime portal and its alpha co-portal.

The *preferred_clan* bit (abbreviated by the letter p in Figure 5-10) indicates whether the clan shall be granted preference in the survivor clan selection algorithm (see 10.2). The value of the *preferred_clan* bit shall be identical for all of a clan’s portals and shall be inherited from the prime portal. The prime portal’s *preferred_clan* bit shall be zero unless it is field-configurable. Within a net, only one portal should be configured to set *preferred_clan* to one if and when it becomes the prime portal.

The *bus_ID* field shall be 3FF₁₆ if the local bus connected to the portal has no bus ID assigned and otherwise shall specify the assigned bus ID.

The *hops_to_prime* field shall be zero if the bridge portal is the prime portal or on the same bus as the prime portal. Otherwise, the field shall contain the count of bridges that are on the routing path between the portal and the prime portal. For alpha portals other than the prime portal, the count shall include the bridge that contains the alpha portal. On a particular bus, the value of *hops_to_prime* shall be equal for all of a clan’s portals. Although the initial value of *hops_to_prime* is vendor-dependent, it shall be either zero or one dependent upon whether the bridge portal is prime or not, respectively.

