P1394.1
Draft Standard for
High Performance Serial Bus Bridges

Sponsor

Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society

Not yet Approved by

IEEE Standards Board

Not yet Approved by

American National Standards Institute

Abstract:
Keywords: bridge, bus, computer, high-speed serial bus, interconnect

The Institute of Electrical And Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN x-xxxxx-xxx-x

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

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

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

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

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

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

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

Introduction

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

This standards effort started in 1996 at the request of...

Patent notice

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

The patent holder has, however, filed a statement of assurance that it will grant a license under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to all applicants desiring to obtain such a license. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreement offered by the patent holder. Contact information may be obtained from the IEEE Standards Department.

Committee membership

The following is a list of voting members of the IEEE P1394.1 working group at the time of publication.

Richard K. Scheel, *Chair*
Peter Johansson, *Editor and Secretary*

The following is a list of other major participants in the IEEE P1394.1 working group (those that attended at least three working group meetings in the last four years).

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

The following persons served on the ballot response committee:

The following persons were members of the balloting group:

If the IEEE Standards Board approves this draft standard, it might have the following membership:

E. G. "Al" Kiener, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

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

*Member Emeritus

Other candiates for inclusion might be the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Steve Sharkey
Robert E. Hebner
Chester C. Taylor

Mary Lynne Nielsen
IEEE Standards Project Editor

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 connects to High Performance Serial Bus as a transaction-capable node.

The project is intended to standardize a model for, the definition of and behaviors of High Performance Serial Bus bridges that may be used to interconnect two or more separately enumerable High Performance Serial Buses. This project extends IEEE Std 1394-1995 and is to be based upon that standard as well as upon ISO/IEC 13213:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses.

1.2 Purpose

IEEE Std 1394-1995, High Performance Serial Bus, 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 standard. This project proposes to adequately specify bridge requirements in order to enable a larger consumer and computer market for High Performance Serial Bus products.

2. References

This standard shall be used in conjunction with the following publications:

ISO/IEC 13213:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses

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

3. Definitions

This clause contains key terms as they are used in this standard.

3.1 Conformance glossary

Several keywords are used to differentiate levels of requirements and optionality, as follows:

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 in cases where either the object or the code value is set aside for future standardization. Usage and interpretation may be specified by future extensions to this or other standards. A reserved object shall be zeroed or, upon development of a future standard, set to a value specified by such a standard. The recipient of a reserved object shall not check its value. The recipient of a defined object shall check its value and reject reserved code values.

3.1.5 shall: A keyword indicating a mandatory requirement. Designers are *required* to implement all such mandatory requirements to ensure interoperability 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."*

0.1. Technical glossary

The following are terms that are used in this standard:

3.1.7 bridge: A Serial Bus node capable of connecting two or more buses into a Serial Bus net. A Serial Bus bridge implements two or more portals and forwards asynchronous and isochronous packets according to route information initialized by other entities, the bridge manager (responsible for asynchronous traffic) or applications at Serial Bus nodes (responsible for isochronous traffic).

3.1.8 bridge manager: A Serial Bus node responsible for the enumeration of buses within a Serial Bus net and, in the process, to initialize all the bridges on the net. A Serial Bus net shall have at most one bridge manager; a procedure is defined to select one from possibly many bridge manager capable nodes.

3.1.9 bus_ID: A 10-bit identifier that shall be unique for each bus within a Serial Bus net. After a power reset or a bus reset, Serial Bus nodes have a *bus_ID* value of $3FF_{16}$, the local bus, specified by the NODE_IDS register. The bridge manager and the bridges are responsible to update this register with the unique ID enumerated for the bus.

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

3.1.11 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 enumerated and assigned a *bus_ID*.

3.1.12 net: A collection of Serial Buses, joined by Serial Bus bridge nodes. Each bus within the net is uniquely identified by its *bus_ID*. Although loops are not permitted within the topology of an individual bus, loops are permitted in the topology of a net.

3.1.13 portal: A connection from a Serial Bus bridge to a bus. Each portal presents a full set of Serial Bus CSR's, as defined in IEEE Std 1394–1995 and in this document, to the connected bus. There may be multiple PHY's for each portal. Serial Bus bridges shall implement at least two portals and may implement up to 255 portals. Portals are identified by a monotonically increasing sequence of ordinals, zero to *n* - 1 where *n* is the number of portals implemented by the bridge.

3.1.14 plug control registers: A set of registers defined in IEC-*nnnn*, proposed standard for Digital Interface for Consumer Electronic Audio/Video Equipment, that are used to manage the isochronous connections into and out of a Serial Bus bridge.

3.1.15 remote node: A Serial Bus node is remote with respect to another node if the nodes are connected to buses that have differing *bus_ID*'s or if one or more Serial Bus bridges lie on the path between the two nodes.

3.1.16 remote-transaction capable: A transaction capable Serial Bus node that is additionally capable of initiating transaction requests directed to a remote node. Since Serial Bus bridges do not propagate bus resets but only transmit reset notifications, a remote-transaction capable node shall implement register(s) to receive reset notifications and shall follow a defined procedure to redetermine the *destination_ID* of the remote node subsequent to a bus reset on the remote bus.

4. Bridge model (informative)

A Serial Bus bridge consists of two or more *bridge portals*, an implementation-specific *switching fabric* and a *cycle clock* together with a method to distribute the clock to all portals. figure 0-1 below illustrates this model.

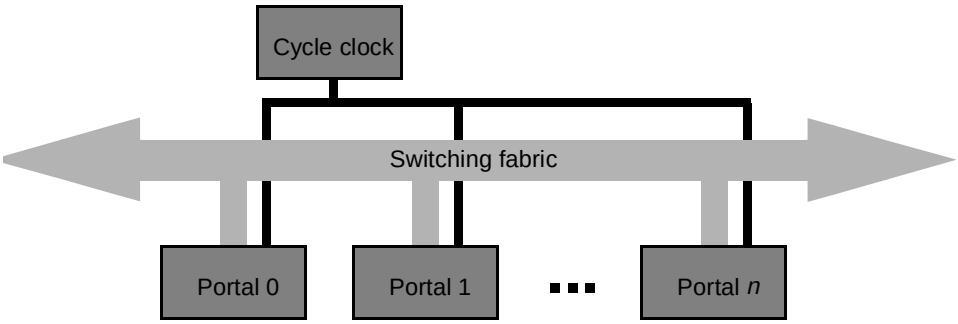

Figure 0-1 — Bridge model

Each bridge portal is a separate Serial Bus node, with 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 packets, asynchronous and isochronous, in order to determine which packets, if any, are to be routed through the bridge's switching fabric to another portal.

The bridge portals are interconnected by means of an switching fabric that is capable of transferring any Serial Bus packet from one portal to any other portal. The details of the switching fabric implementation are not addressed by this standard. The switching fabric may be modest in geographical extent, as when all of the bridge portals and switching fabric are located within a single enclosure. Conversely, the switching fabric may 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: the bridge is the collection of the portals connected by the fabric.

The cycle clock is a common resource to which all bridge portals shall be synchronized. The cycle clock is optional but is required if the bridge supports isochronous routing. The propagation of the cycle clock to all bridge portals is implementation-specific and beyond the scope of this standard.

5. Bridge facilities

Serial Bus bridges are implemented as a unit architecture within a Serial Bus node. This clause describes the facilities that a bridge shall support in order to interoperate with other bridges and the bridge manager. These facilities are configuration ROM entries (which are used to identify the presence of the bridge within a Serial Bus node) and control and status registers, CSR's (which are used to control the operations of and obtain status from the bridge).

5.1 Bridge portal configuration ROM

Each bridge portal shall implement configuration ROM in the general format defined by IEEE Std 1394-1995. Appendix X contains a sample of a valid configuration ROM for a bridge portal and illustrates the usage of the entries defined below.

5.1.1 Bus_Info_Block

A bridge portal's configuration ROM shall contain a Bus_Info_Block, as defined by IEEE Std 1394-1995 and repeated for convenience in figure 5-1 below.

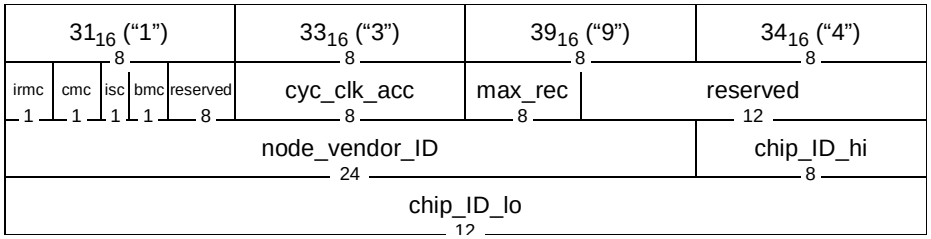

Figure 5-1 — Bus_Info_Block format

The usage of all fields is as defined by IEEE Std 1394-1995. Some fields have particular applicability to Serial Bus bridges, as defined below.

The isochronous resource manager (***irmc***), cycle master (***cmc***), isochronous (***isc***) and bus manager (***bmc***) capability bits specify interrelated abilities of the bridge portal. Bus manager capabilities are orthogonal to bridge requirements; the *bmc* bit shall be set according to the presence or absence of these optional capabilities. A bridge portal with no capability to forward isochronous data shall report values of zero for each of the *irmc*, *cmc* and *isc* bits. A bridge portal that can propagate isochronous data shall report values of one for each of the *irmc*, *cmc* and *isc* bits.

The ***max_rec*** field specifies the maximum payload that the bridge portal can accept in any of an asynchronous block write request, lock request, block read response or lock response packet. This usage is an extension to that defined by IEEE Std 1394-1995, which addresses only the size of an asynchronous block write request. The maximum payload size is defined as 2^{max_rec+1} and is bounded by the maximum payload size permitted by the data transfer speed supported by the bridge portal.

NOTE—The maximum payload for an isochronous packet has no relationship to the *max_rec* field. The method by which the maximum isochronous payload shall be specified has yet to be determined.

5.1.2 Node_Capabilities entry

The mandatory Node_Capabilities in the root directory contains subfields defined by ISO/IEC 13213:1994. All Serial Bus nodes shall implement the ***spt***, ***64***, ***fix***, ***lst*** and ***drq*** bits.

Bridge portals shall set the *spt* bit to one to indicate that the SPLIT_TIMEOUT register is implemented.

Bridge portals shall set the *drq* bit to one to indicate that the STATE_CLEAR.*dreq* bit is implemented.

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 5-2 shows the format of this entry.

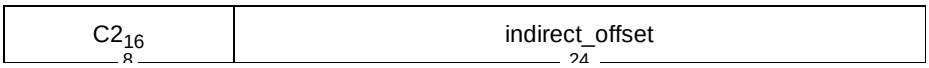

Figure 5-2 — Bus_Dependent_Info entry format

The entry is identified by the *key_type* and *key_value* fields which together have a value of $C2_{16}$.

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 specifies the capabilities of the bridge. Figure 5-3 shows the format of this entry.

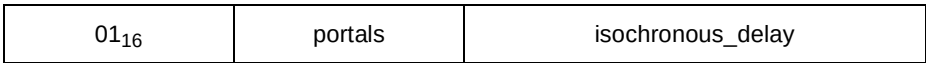

Figure 5-3 — Bridge_Capabilities entry format

The entry is identified by the *key_type* and *key_value* fields which together have a value of 01_{16}.

The ***portals*** field specifies the total count of bridge portals that collectively comprise the Serial Bus bridge. The *portals* field shall have a minimum value of two and a maximum value of 255.

The ***isochronous_delay*** field specifies the constant delay that isochronous packets incur when they are transferred from one bridge portal to another. The *isochronous_delay* field shall specify the delay in units of 125 µs and shall have a minimum value of two.

5.2 Bridge portal control and status registers

In addition to the control and status register (CSR) requirements defined by IEEE Std 1394-1995 for transaction-capable nodes, Serial Bus bridges define common registers within the Serial Bus-dependent portion of initial units space. Initial units space occupies the addresses at FFFF F000 0800_{16} and above. The locations of bridge portal registers, summarized in table 5-1, are specified in terms of offsets within initial register space, where the base of initial register space (from the beginning of initial node space) is FFFF F000 0000_{16}.

Table 5-1 — Bridge portal register locations

Offset	Name	Description
900 — $9FC_{16}$	Plug control registers	As described in draft supplement P1394a.
2400_{16}	OWNER_EUI_64	Compare and swap register used by bridge managers to resolve bridge portal ownership.
2408_{16}	BRIDGE_MANAGER_ID	Contains the value of the bridge manager's *node_ID*.

Table 5-1 — Bridge portal register locations

Offset	Name	Description
$240C_{16}$	BRIDGE_RESET	A write of any value to this location is equivalent to a power reset.
2410_{16}	PORTAL_CONTROL	Configures the *bus_ID* and other parameters associated with each portal.
2414_{16}	PORTAL_SELECT	Selects which portal's registers are accessible through the portal specific window.
2418_{16}	RESET_NOTIFICATION	Bus resets that occur on remote buses are indicated by the a quadlet write to this register with the value of the *bus_ID* of the reset bus and a *generation_number*.
$241C_{16}$	RESET_ACKNOWLEDGE	Used by nodes that forward asynchronous transactions through the bridge to confirm their receipt of a reset notification.
2420_{16}	ROUTING_BOUNDS	Defines the outbound routing for asynchronous packets forwarded by the bridge.
2624_{16}	REMOTE_REQUEST	Specifies the transaction type for asynchronous requests initiated on another portal.
2628_{16}	REMOTE_DESTINATION	Specifies the *destination_ID* and *destination_offset* for asynchronous requests initiated on another portal.
2630_{16}	NODE_ENABLE	A bit mask that indicates, by node *physical_ID*, which local nodes on a portal's bus are enabled to pass transaction requests through the bridge.
2638_{16} — $267C_{16}$		Reserved for future standardization by Serial Bus.
2680_{16} — $26FC_{16}$	CHANNEL_SWITCH	These registers redirect input isochronous channels to output channels on one or more portals.
2700_{16} — 2800_{16}	Portal-specific window	All bridge registers that have an instance for each of the other portals are accessible within this address range.
2800_{16} — 2900_{16}	REMOTE_PAYLOAD	Contains the data value(s) sent as part of a remote write request or obtained as the result of a remote read request.

The following sections provide detailed definitions of the registers implemented by Serial Bus bridges.

5.2.1 OWNER_EUI_64 register

The OWNER_EUI_64 register is a register that supports compare and swap access so that bridge managers may resolve contention for the ownership of the bridge portal. Figure 5-4 below shows the format of this register.

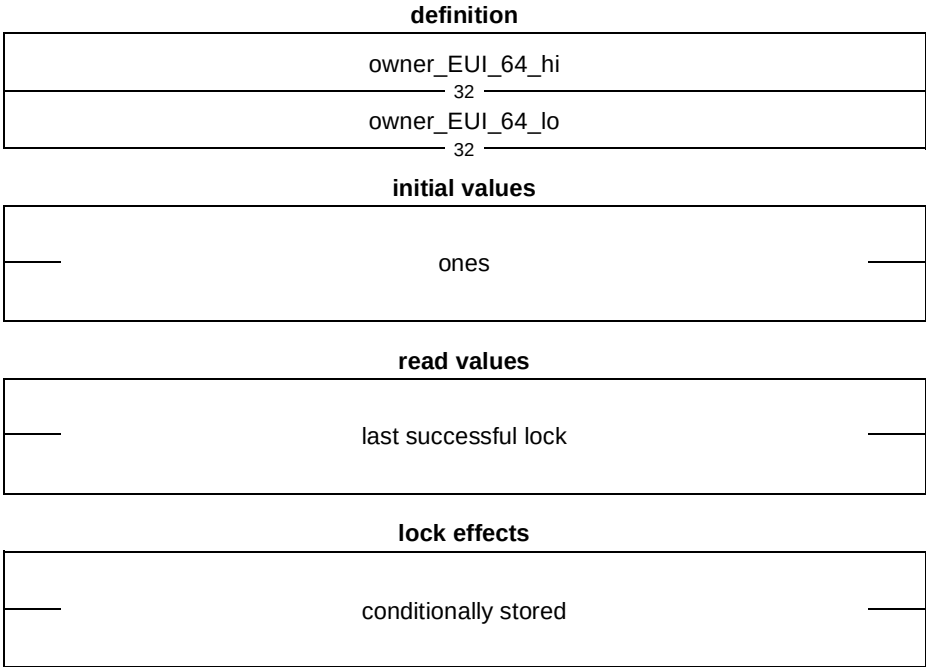

Figure 5-4 — OWNER_EUI_64 format

5.2.2 BRIDGE_MANAGER_ID register

The optional BRIDGE_MANAGER_ID register provides a common location at which remote-transaction capable nodes may obtain the *node_ID* of the bridge manager. If a bridge manager provides advanced functionality (not yet specified by the bridge architecture), such as net topology information, the BRIDGE_MANAGER_ID register provides a convenient way to locate the bridge manager without enumerating all buses and querying all nodes. The format of the BRIDGE_MANAGER_ID register is illustrated by figure 5-5 below.

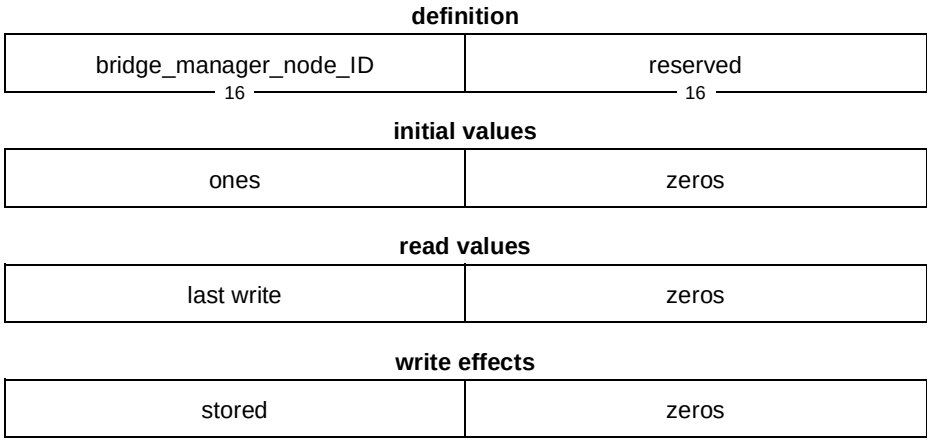

Figure 5-5 — BRIDGE_MANAGER_ID format

The ***bridge_manager_node_ID*** is normally initialized by a write from the bridge manager, once the identity of the bridge manager has been determined as described in X. The value written shall be equal to the content of the bridge manager's NODE_IDS register.

5.2.3 BRIDGE_RESET register

The BRIDGE_RESET register provides a means to atomically reset the entire bridge, *i.e.*, to erase all route information and to erase all *bus_ID*'s associated with the bridge's portals. Figure 5-6 illustrates the format of this register.

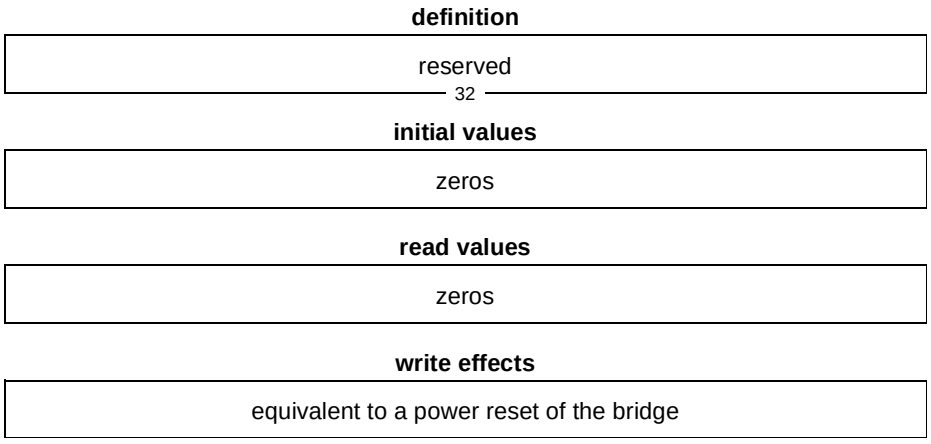

Figure 5-6 — BRIDGE_RESET format

The data value in the broadcast quadlet write to the BRIDGE_RESET register is ignored by the bridge; the write transaction itself is sufficient to cause a reset.

5.2.4 PORTAL CONTROL register

The PORTAL_CONTROL register establishes a mapping between a Serial Bus bridge portal and the *bus_ID* assigned to the Serial Bus connected to that portal. A Serial Bus bridge shall implement a PORTAL_CONTROL register for each of the bridge's *n* portals. Figure 5-7 below illustrates the format of this register.

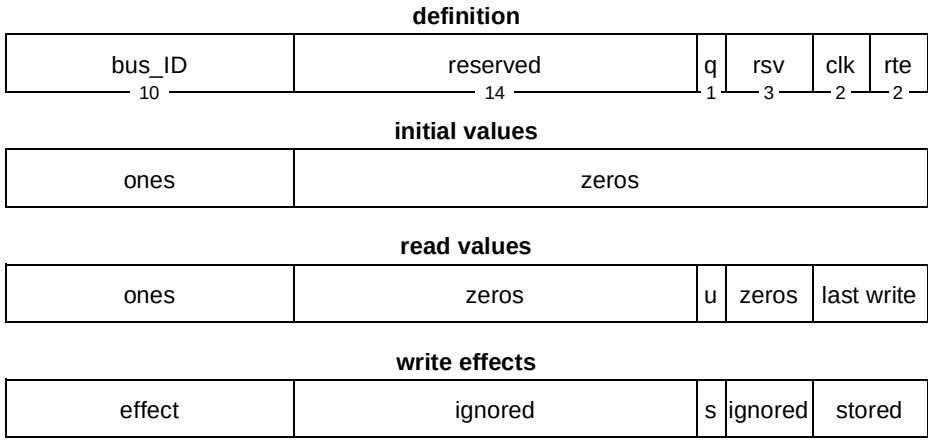

Figure 5-7 — PORTAL_CONTROL format

The ***bus_ID*** field identifies the Serial Bus connected to the portal. In addition to this function, a write to the PORTAL_CONTROL register has a special side-effect. When the *bus_ID* field is updated by a write, the Serial Bus bridge broadcasts a quadlet write transaction to the NODE_IDS register of all nodes on the Serial Bus connected to the portal. The data value broadcast is the *bus_ID* in the most significant ten bits and zeros for the remaining bits. This has the effect of establishing the *bus_ID* for all nodes on the connected bus.

The ***q***, or ***quarantine***, bit, if set, has the property of disabling the propagation of all asynchronous request packets adddressed to *bus_ID* that originate from other portals of the bridge. When a bus reset is detected by a bridge portal on its connected Serial Bus, the bridge shall set the *quarantine* bit to one. The bridge manager is expected to clear the *quarantine* bit after other steps have been taken to insure the integrity of asynchronous transactions subsequent to a bus reset.

The ***clk*** field defines the behavior of the portal with respect to the isochronous cycle clock, as defined below.

Value	Description
0	The portal is disabled for both reception and broadcast of cycle start packets.
1	The portal is configured to synchronize the bridge's cycle clock to cycle start packets received from its local bus.
2	The portal is configured to operate as the cycle master for its local bus. The bridge's CYCLE_TIME register triggers isochronous cycles.
3	Reserved for future standardization by Serial Bus.

The ***rte*** field controls the behavior of the portal with respect to asynchronous transaction routing, as specified in the following table.

Value	Description
0	Disabled; no asynchronous request or response packets are forwarded by the portal
1	Reserved for future standardization by Serial Bus
2	Enabled; forward asynchronous request or response packets if *bus_ID_lower_bound* <= *destination_bus_ID* <= *bus_ID_upper_bound*
3	Enabled; forward asynchronous request or response packets if *destination_bus_ID* < *bus_ID_lower_bound* or *destination_bus_ID* > *bus_ID_upper_bound*

See the subsequent section, “Asynchronous operations and routing”, for the details of asynchronous packet routing by bridges.

5.2.5 PORTAL_SELECT register

The contents of the PORTAL_SELECT register determine which portal’s portal-specific registers are accessible through the portal-specific window in the address range FFFF F000 2480_{16} through FFFF F000 $25FC_{16}$. The format of this register is shown in figure 5-8 below.

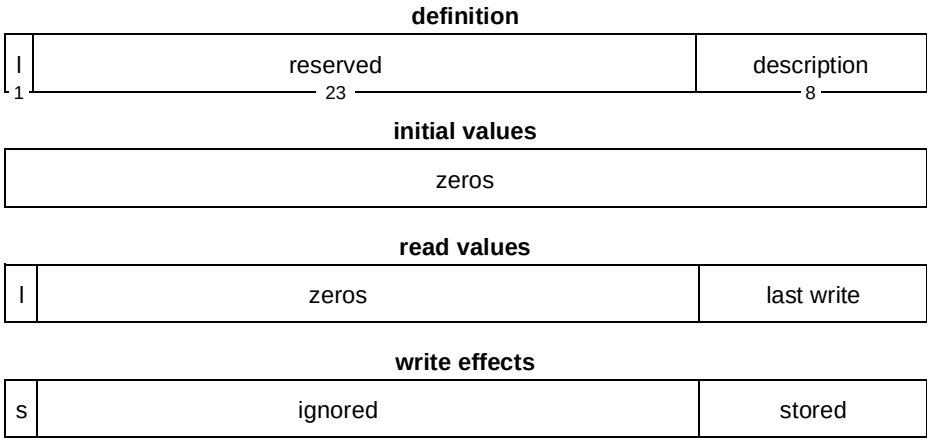

Figure 5-8 — PORTAL SELECT format

The ***l*** bit indicates whether or not the portal-specific window is enabled for the local portal, *i.e.*, the same portal on which a transaction request is physically received. When the *l* bit is set to zero, the portal registers accessible through the portal-specific window are associated with one of the bridge’s other portals. If the *l* bit is set to one, the local portal’s registers are accessible.

The ***portal*** field selects which portal’s registers are accessible through the portal-specific window. If the *l* bit is written with a value of one, the bridge ignores the rest of the data written to the PORTAL_SELECT register and updates the *portal* field to the value that indexes the local portal.

NOTE—If the *portal* field is set to the ordinal of a portal not implemented by the bridge, any accesses to registers within the range of the portal-specific window receive an address error response.

5.2.6 RESET_NOTIFICATION register

The RESET_NOTIFICATION register provides a common location for nodes to receive an indication of a bus reset on another bus within the Serial Bus net. All transaction capable nodes that initiate requests to nodes on other buses shall implement the RESET_NOTIFICATION register with the format shown in figure 5-9 below. This class of nodes includes bridges, bridge managers, any node that initiates asynchronous requests to remote nodes and any node that establishes ownership of isochronous resources on other buses. It does not necessarily include either isochronous talkers or listeners.

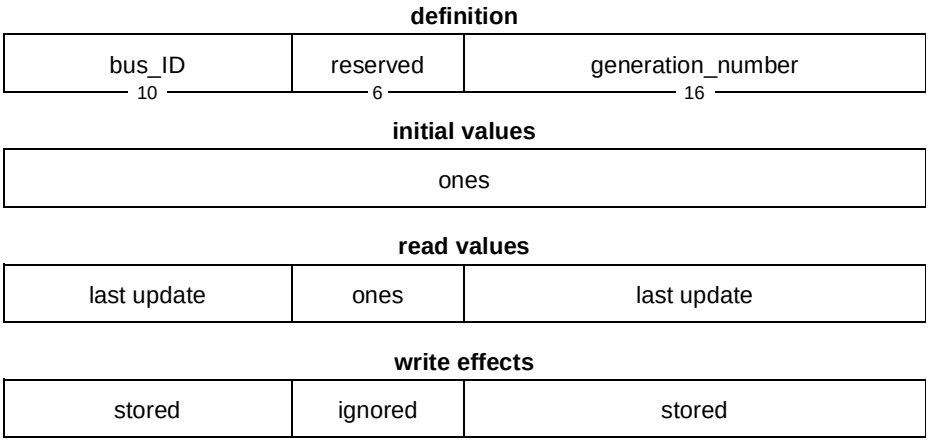

Figure 5-9 — RESET_NOTIFICATION format

The ***bus_ID*** field indicates which bus has been reset. When a bridge portal detects a bus reset on its connected Serial Bus, the bridge shall update the contents of the RESET_NOTIFICATION register by setting *bus_ID* to the value of the field of the same name in the PORTAL_CONTROL register and by incrementing the *generation_number*. The updated value of the RESET_NOTIFICATION shall then be written to the bridge manager. The bridge manager is expected to propagate the bus reset notification by writing the same value to the RESET_NOTIFICATION register(s) of all other bridge(s) previously enumerated by the bridge manager.

The ***generation_number*** field is a counter maintained by the bridge portal that observed the bus reset on its Together, the *bus_ID* and *generation_number* fields uniquely identify a bus reset event within a Serial Bus net. Serial Bus nodes that initiate transaction requests to remote nodes shall use the *bus_ID* and *generation_number* fields as a key in a procedure to reestablish pathways to remote nodes, described in X.

A Serial Bus bridge that receives a write transaction to its RESET_NOTIFICATION register shall clear the NODE_ENABLE registers for all bridge portals.

5.2.7 RESET_ACKNOWLEDGE register

The RESET_ACKNOWLEDGE register provides a means for Serial Bus nodes to communicate to adjacent bridge portal(s) that they have received notification of a particular bus reset event. All bridge portals shall implement this register in the format shown by figure 5-10 below.

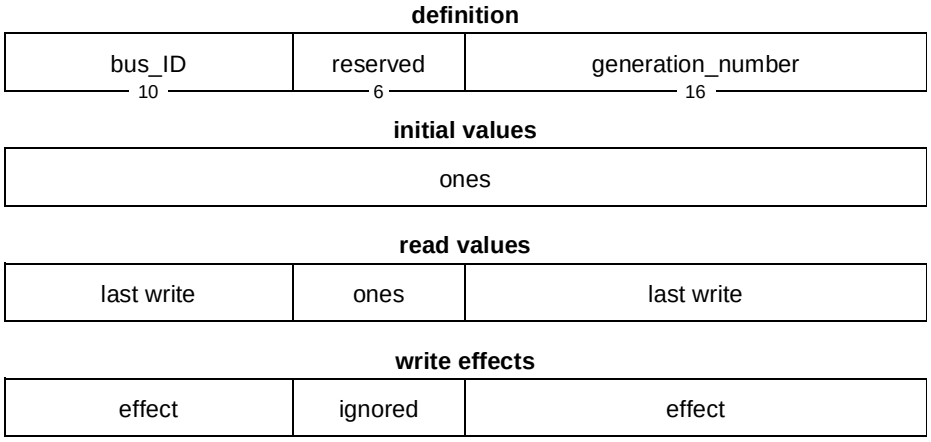

Figure 5-10 — RESET_ACKNOWLEDGE format

A bus reset event on a bus potentially causes the *physical_ID*'s of nodes on the bus to change. Before any Serial Bus node on any bus of the net initiates a new transaction request directed to a node on the reset bus, it shall redetermine the correct *physical_ID* of the intended destination node. The RESET_ACKNOWLEDGE register provides an interlock mechanism to prevent Serial Bus bridges from propagating asynchronous transactions that may have "stale" *bus_ID* values in their *destination_ID*'s. See clause 5.2.6 for a description of how the receipt of a broadcast write transaction causes the bridge to disable the transmission of all asynchronous transaction requests from all nodes on all local buses connected to the bridge's portals. A write to the RESET_ACKNOWLEDGE register is necessary to reenable the forwarding of asynchronous transaction requests. If a Serial Bus bridge receives a quadlet write request addressed to the RESET_ACKNOWLEDGE register and the *bus_ID* and *generation_number* values in the data payload exactly match the current contents of the RESET_NOTIFICATION register, the bridge shall reenable forwarding of asynchronous transactions for the node identified by *source_ID* in the write request.

5.2.8 ROUTING_BOUNDS register

The ROUTING_BOUNDS register provides, for each portal, parameters used by the bridge to determine asynchronous transaction routing. The format of this register is given by figure 5-11 below.

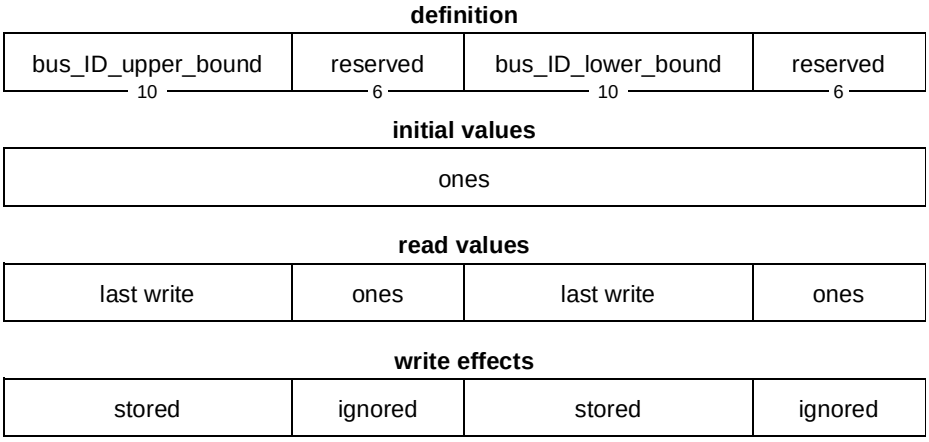

Figure 5-11 — ROUTING_BOUNDS format

The ***bus_ID_upper_bound*** and ***bus_ID_lower_bound*** fields shall specify, respectively, the upper and lower bounds for the *destination_bus_ID*'s of asynchronous request and response packets that are to be forwarded by a bridge portal The value of the two bounds fields is used in conjunction with the *rte* field in the PORTAL_CONTROL register to determine which asynchronous packets shall be forwarded

5.2.9 REMOTE_REQUEST register

The REMOTE_REQUEST register, in conjunction with the REMOTE_DESTINATION and REMOTE_PAYLOAD registers, enables an application to specify the generation of an asynchronous transaction request by a non-local bridge portal. This register, whose format is shown below in figure 5-12, is typically used by the bridge manager during initialization and bus enumeration.

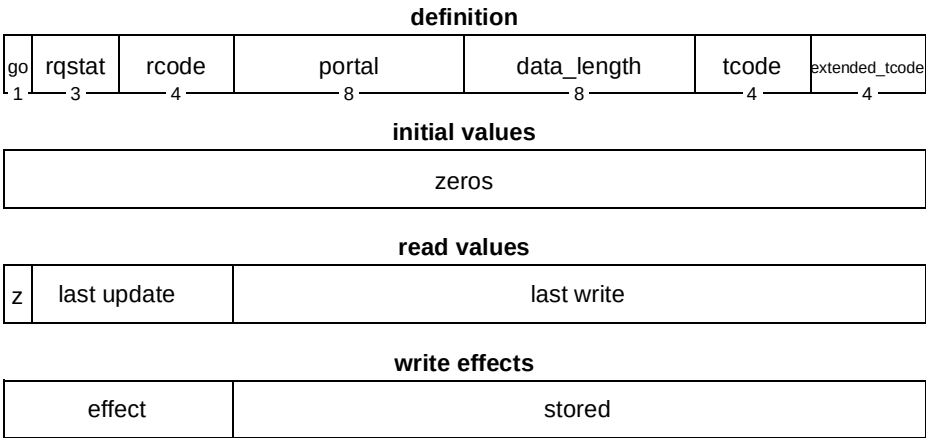

Figure 5-12 — REMOTE_REQUEST format

The ***go*** bit is used to signal the bridge to initiate a remote request with the parameters stored in the REMOTE_DESTINATION and REMOTE_PAYLOAD registers. A write of zero to the *go* bit shall have no effect. A write of one shall cause the bridge to create and transmit a request. When the bridge creates a request packet, the *destination_ID* and *destination_offset* fields shall be obtained from the REMOTE_DESTINATION register. The *tl*, *rt* and *pri* fields of the request packet shall be set to vendor-dependent values. The *source_ID* field of the request packet shall be equal to the most significant 16 bits of the NODE_IDS register for the portal that is to transmit the request. If required by the transaction type, the *data* field (for a write request) or the *arg_value* and *data_value* fields (for a lock request) shall be obtained from the REMOTE_PAYLOAD register. The remaining request packet fields, *tcode*, *data_length* and *extended_tcode* shall be obtained from the REMOTE_REQUEST register. The bridge shall calculate the header and data CRC fields as necessary for the packet format.

The ***rqstat*** field specifies the result of the remote transaction, as encoded by the values defined in the table below. Upon a write to the REMOTE_REQUEST register with the *go* bit set, the bridge shall set the *rqstat* field to a value of REQUEST PENDING. When the remote request completes, either as a result of receipt of a completion acknowledgment or response or because of an unrecoverable error, the bridge shall update *rqstat* to a value other than REQUEST_PENDING.

Value	Request status
0	COMPLETE
1	TIMEOUT
2	ACKNOWLEDGE MISSING
3	RETRY LIMIT
4	INVALID REQUEST
5	DATA ERROR
6	Reserved for future standardization by Serial Bus
7	REQUEST PENDING

The application that initiated a remote request may poll for completion status by reading the REMOTE_REQUEST register and examining *rqstat*.

The ***rcode*** field shall contain the response code returned as part of the remote transaction. The *rcode* field is valid only if *rqstat* is either COMPLETE or DATA ERROR. The values for *rcode* are defined by IEEE Std 1394-1995.

The ***portal*** field shall specify which of the bridge's portals is to generate the remote request. If *portal* specifies an unimplemented bridge portal at the time the REMOTE_REQUEST *go* bit is set, the bridge shall set the value of *rqstat* to INVALID REQUEST.

The ***data_length*** field is used by the bridge to construct the remote request (see X). When the REMOTE_REQUEST register specifies a remote write or lock request, the application is expected to store *data_length* bytes in the REMOTE_PAYLOAD register before setting the *go* bit in the REMOTE_REQUEST register. The *data_length* field is ignored by the bridge if *tcode* specifies a value of zero or four.

The ***tcode*** field shall specify the type of remote request that the bridge shall initiate on the specified portal. The values of *tcode* are specified by IEEE Std 1304-1995; the subset of *tcode* values supported by bridges for remote requests is defined by the table below.

Value	Description
0	Write request for data quadlet
1	Write request for data block
2 – 3	Not supported for remote requests
4	Read request for data quadlet

Value	Description
5	Read request for data block
6 – 8	Not supported for remote requests
9	Lock request
$A_{16} – B_{16}$	Not supported for remote requests
$C_{16} – F_{16}$	Reserved for future standardization by Serial Bus

If *tcode* specifies an unsupported transaction code at the time the REMOTE_REQUEST *go* bit is set, the bridge shall set the value of *rqstat* to INVALID REQUEST.

The ***extended_tcode*** field shall specify the extended transaction code used for lock requests. The meaning of *extended_tcode* values are defined by IEEE Std 1394-1995.

5.2.10 REMOTE_DESTINATION register

The REMOTE_DESTINATION register is used to specify the 64-bit Serial Bus address to which a remote request is addressed. figure 5-13 below illustrates the format of this register.

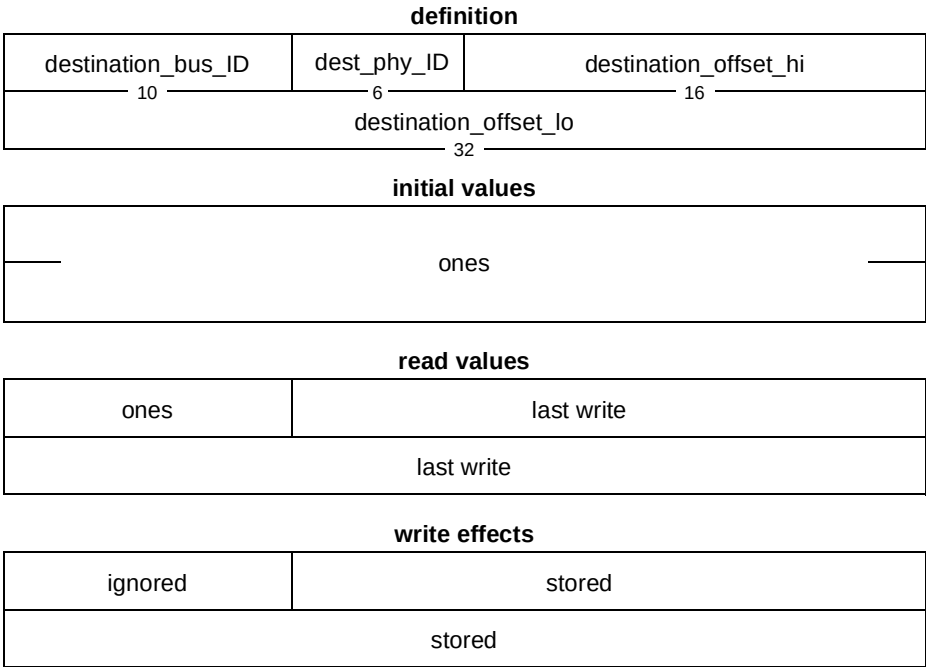

Figure 5-13 — REMOTE_DESTINATION format

The ***destination_bus_ID*** field shall have a value of $3FF_{16}$, the local bus ID.

The ***dest_phy_ID*** shall be set to the value of the 6-bit physical ID of the node to which the remote request is to be addressed.

The ***destination_offset_hi*** and ***destination_offset_lo*** fields shall be set, respectively, to the most- and least-significant portions of the 48-bit *destination_offset* to which the remote request is to be addressed.

5.2.11 NODE_ENABLE register

The NODE_ENABLE register is a read-only register that provides information about the current state of the Serial Bus bridge. This register is a bit map that represents, for each of the 63 possible *physical_ID*'s of nodes locally connected to a bridge's portal, whether or not asynchronous transaction requests and responses are forwarded by the bridge. figure 5-13 below shows the format of this register.

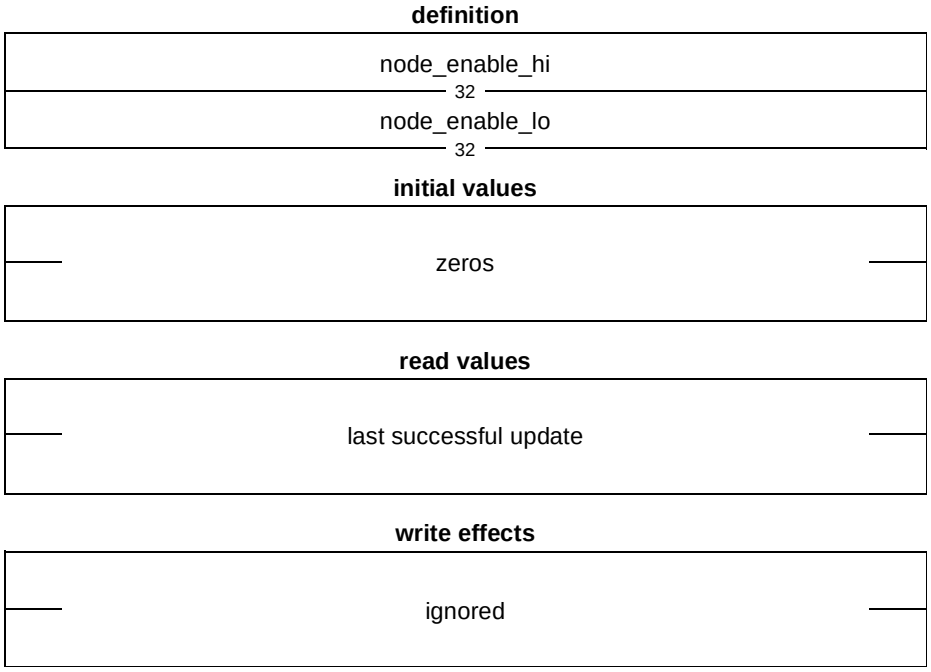

Figure 5-14 — NODE_ENABLE format

The NODE_ENABLE register is a bit mask whose 64 bits represent all the possible *physical_ID*'s of nodes connected to a portal of the Serial Bus bridge. If the corresponding bit is zero, transaction requests from the specified node are refused with an address error. If transaction forwarding is enabled for a particular node, *i.e.*, the corresponding bit is one, then the request packet received on the portal is retransmitted according to the information in the ROUTING_BOUNDS and ROUTE_UPPER_BOUND registers.

The NODE_ENABLE register shall be reset to zero by the bridge upon either of two events: a) a bus reset is observed on the Serial Bus connected to the bridge portal, or b) the receipt of a quadlet write transaction addressed to the bridge portal's RESET_NOTIFICATION register.

Individual bits within the NODE_ENABLE are updated indirectly, by means of a write to the RESET_ACKNOWLEDGE register. When a bridge receives a quadlet write transaction directed to the RESET_ACKNOWLEDGE register in which the *source_bus_ID* field is either $3FF_{16}$ (the local bus) or equal to the *bus_ID* in the relevant PORTAL_CONTROL register and both the *bus_ID* and *generation_number* fields in the data payload exactly match the current contents of the RESET_NOTIFICATION register, the bridge shall set the bit in the NODE_ENABLE register that corresponds to *source_physical_ID* in the write transaction. The most significant bit in the NODE_ENABLE register corresponds to *physical_ID* 63 and the least significant bit corresponds to *physical_ID* zero.

5.2.12 CHANNEL_SWITCH register

The CHANNEL_SWITCH, together with the INPUT_PLUG register of the source portal and the OUTPUT_PLUG register of the output portal, provides a method to control the broadcast of isochronous traffic by the Serial Bus bridge. There is a set of CHANNEL_SWITCH registers implemented for each portal. On a given portal, if a Serial Bus bridge implements *n* OUTPUT_PLUG registers it shall also implement *n* CHANNEL_SWITCH registers. There is an implicit relationship between the CHANNEL_SWITCH and the OUTPUT_PLUG registers. The channel number and speed information necessary to retransmit the source isochronous channel identified by the CHANNEL_SWITCH[*n*] register shall be obtained from the corresponding OUTPUT_PLUG[*n*] register for the same portal. The format of the CHANNEL_SWITCH register is illustrated by figure 5-15 below.

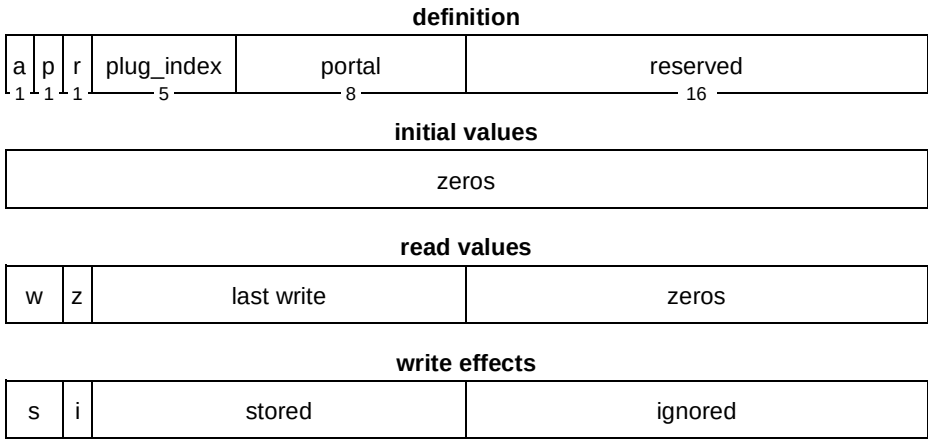

Figure 5-15 — CHANNEL_SWITCH format

The ***a***, or ***active***, bit indicates whether or not the channel switch is active. An application normally sets the *active* bit to one to enable the switch between the input and output isochronous channels. Other events may require the Serial Bus bridge to disable the connection, in which case the *active* bit shall be cleared.

The ***p***, or ***proxy***, bit indicates whether or not the Serial Bus bridge is to act as a proxy for the application and attempt to reallocate isochronous resources, bandwidth and channel, in the event of a bus reset on the bus connected to the portal. If a bridge does not implement this capability, it shall ignore any attempt to set the *proxy* bit to one and reject the transaction with a type error. When the *proxy* bit is set to one and the bridge observes a bus reset on the portal's bus, the bridge acts as a proxy owner of the isochronous resources as follows:

a) The OUTPUT_PLUG[*n*] register for the bridge portal, where *n* is the same index used to reference the CHANNEL_SWITCH[*n*] register, is consulted to obtain the necessary speed, channel number, packet overhead and data payload values.

b) These are converted to the corresponding values needed to update the BANDWIDTH_ALLOCATE and CHANNELS_AVAILABLE registers at the isochronous resource manager on the portal's bus.

c) If the compare and swap lock transactions fail because the channel number is in use or there is insufficient bandwidth, the bridge clears both the *active* and the *proxy* bits in the CHANNEL_SWITCH register and no longer retransmits the affected isochronous channel. No indication, other than the fact that the *active* and *proxy* bits are clear, is given to the application that programmed the CHANNEL_SWITCH register.

d) If the isochronous resource reallocation is successful, both the *active* and *proxy* bits retain their present values.

The ***plug_index*** and the ***portal*** fields together specify which INPUT_PLUG register describes the source of the isochronous data to be retransmitted over this portal. Within the set of INPUT_PLUG registers for *portal*, the INPUT_PLUG[*plug_index*] register contains the channel number field that identifies the input isochronous channel.

5.2.13 REMOTE_PAYLOAD register

The REMOTE_PAYLOAD register is used obtain data returned by a response to a remote read request, to provide data for a remote write request or first to specify the argument and data values for a remote lock request and subsequently to obtain the old data value returned by the lock response. Although figure 5-13 below illustrates a 64-bit REMOTE_PAYLOAD register, the size of the REMOTE_PAYLOAD register is vendor-dependent and specified in configuration ROM. The REMOTE_PAYLOAD register shall be at least an octlet and, if greater, shall be a integral number of quadlets.

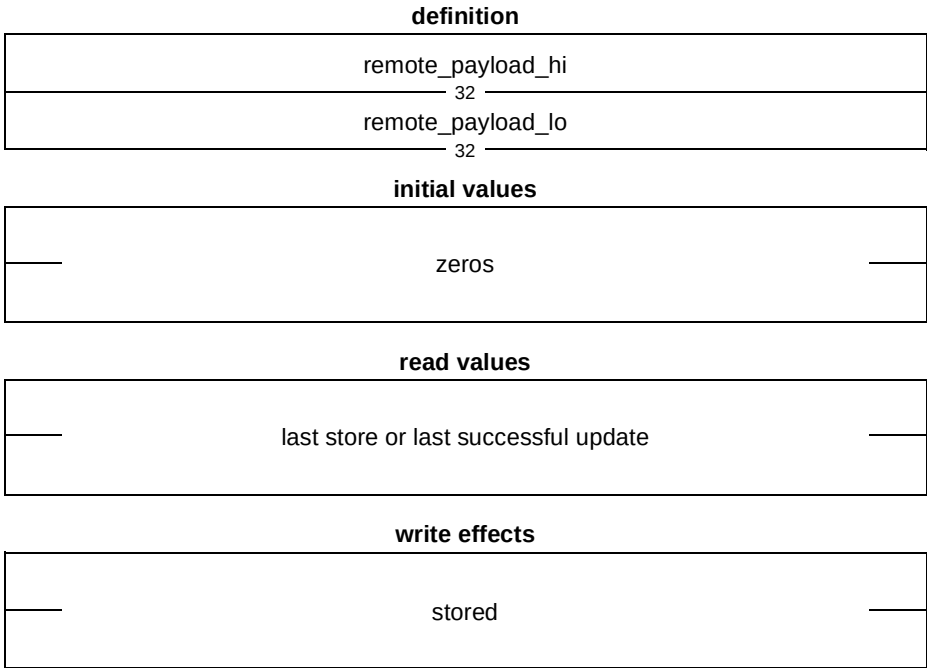

Figure 5-16 — REMOTE_PAYLOAD format

If a bridge portal implements a REMOTE_PAYLOAD register larger than an octlet, the entire register shall be accessible by both block read or block write transactions whose *data_length* field is equal to the size of the REMOTE_PAYLOAD register reported by configuration ROM

When the REMOTE_REQUEST register is used to initiate a read request, the bridge shall update the REMOTE_PAYLOAD register with the data value(s) received in the read response packet. When the REMOTE_REQUEST register is used to initiate a write request, the bridge shall obtain the data value(s) for the request from the REMOTE_PAYLOAD register at the time the *go* bit is set. When an application initiates a lock request *via* the REMOTE_REQUEST register, the REMOTE_PAYLOAD register shall be used for both purposes: the *arg_value* and *data_value* fields are obtained when the *go* bit is set and the register is updated with the *old_value* when the lock response packet is received.

6. Bridge manager facilities

Bridge managers are implemented as a unit architecture within a Serial Bus node. This clause describes the facilities that a bridge manager shall support in order to interoperate with Serial Bus bridge(s) and other bridge manager(s). These facilities are configuration ROM entries (which are used to identify the presence of the bridge manager within a Serial Bus node) and control and status registers, CSR's (which are used to control the operations of and obtain status from the bridge manager).

6.1 Bridge manager configuration ROM

Each bridge manager shall implement configuration ROM in the general format defined by IEEE Std 1394-1995. Appendix X contains a sample of a valid configuration ROM for a bridge manager and illustrates the usage of the entries defined below.

6.1.1 Bus_Info_Block

A bridge portal's configuration ROM shall contain a Bus_Info_Block, as defined by IEEE Std 1394-1995.

6.1.2 Node_Capabilities entry

The mandatory Node_Capabilities in the root directory contains subfields defined by ISO/IEC 13213:1994. All Serial Bus nodes shall implement the ***spt***, ***64***, ***fix***, ***lst*** and ***drq*** bits.

Bridge managers shall set the *drq* bit to one to indicate that the STATE_CLEAR.*dreq* bit is implemented.

6.1.3 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 6-1 shows the format of this entry.

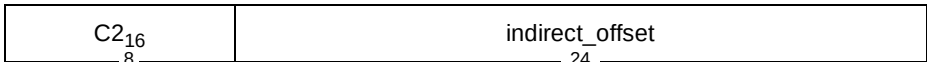

Figure 6-1 — Bus_Dependent_Info entry format

The entry is identified by the *key_type* and *key_value* fields which together have a value of $C2_{16}$.

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.

NOTE—If a node implements both bridge and bridge manager capabilities, only one Bus_Dependent_Info entry is required in the root directory.

6.1.4 Bridge_Manager_Capabilities entry

The Bridge_Capabilities entry is an immediate entry in the Bus_Dependent_Info directory that specifies the capabilities of the bridge manager. Figure 6-2 shows the format of this entry.

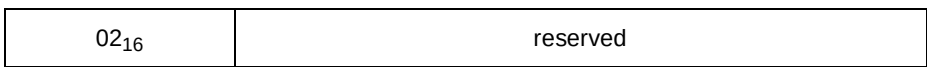

Figure 6-2 — Bridge_Manager_Capabilities entry format

The entry is identified by the *key_type* and *key_value* fields which together have a value of 01_{16}.

6.2 Bridge manager control and status registers

In addition to the control and status register (CSR) requirements defined by IEEE Std 1394-1995 for transaction-capable nodes, Serial Bus bridge managers define common registers within the Serial Bus-dependent portion of initial units space. Initial units space occupies the addresses at FFFF F000 0800_{16} and above. The locations of bridge portal registers, summarized in table 6-1, are specified in terms of offsets within initial register space, where the base of initial register space (from the beginning of initial node space) is FFFF F000 0000_{16}.

Table 6-1 — Bridge portal register locations

Offset	Name	Description
2418_{16}	RESET_NOTIFICATION	Bus resets that occur on remote buses are indicated by the a quadlet write to this register with the value of the *bus_ID* of the reset bus and a *generation_number.*

The following sections provide detailed definitions of the registers implemented by Serial Bus bridges.

6.2.1 RESET_NOTIFICATION register

The RESET_NOTIFICATION register provides a common location for nodes to receive an indication of a bus reset on another bus within the Serial Bus net. All transaction capable nodes that initiate requests to nodes on other buses shall implement the RESET_NOTIFICATION register with the format shown in figure 6-3 below. This class of nodes includes bridges, bridge managers, any node that initiates asynchronous requests to remote nodes and any node that establishes ownership of isochronous resources on other buses. It does not necessarily include either isochronous talkers or listeners.

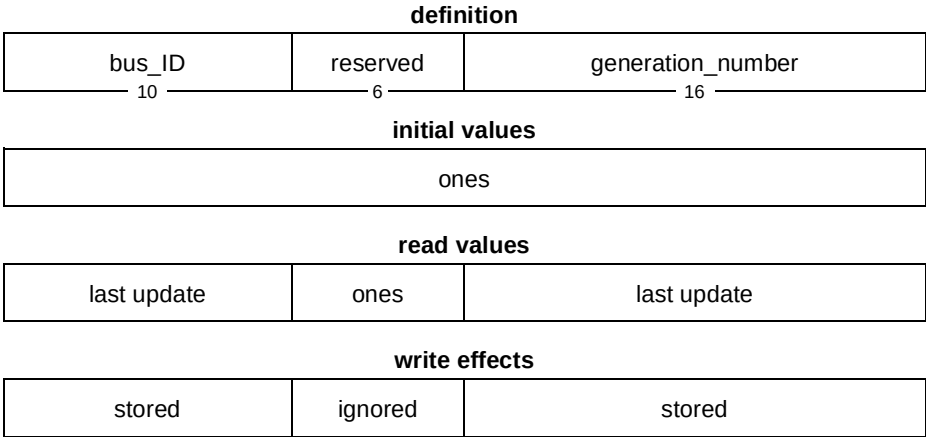

Figure 6-3 — RESET_NOTIFICATION format

The ***bus_ID*** field indicates which bus has been reset. The bridge manager is expected to propagate the bus reset notification by writing the same value to the RESET_NOTIFICATION register(s) of all bridge(s) previously enumerated by the bridge manager.

The ***generation_number*** field is a counter maintained by the bridge portal that observed the bus reset on its Together, the *bus_ID* and *generation_number* fields uniquely identify a bus reset event within a Serial Bus net. Serial Bus nodes that initiate transaction requests to remote nodes shall use the *bus_ID* and *generation_number* fields as a key in a procedure to reestablish pathways to remote nodes, described in X.

7. Asynchronous operations and routing

This section describes the normal operations of a bridge to route asynchronous transactions once the bridge has been configured by a bridge manager. See a later section, “Serial Bus net configuration”, for details.

Bridge portals function as ***inbound portals*** when they eavesdrop on their bus to detect asynchronous primary packets to be forwarded to another bus. A portal that repeats a primary packet on its bus is acting as an ***outbound portal***. Unless a bridge portal is deactivated for asynchronous traffic, it is capable of acting as either an inbound or outbound portal.

The table below enumerates the transaction codes recognized by inbound and outbound portals in their forwarding of asynchronous primary packets.

Table 7-1 — Asynchronous primary packet transaction codes

Code	Subaction	Payload
0	Write request	Quadlet
1	Write request	Block
2	Write response	—
4	Read request	Quadlet
5	Read request	Block
6	Read response	Quadlet
7	Read response	Block
9	Lock request	Block
B_{16}	Lock request	Block

The clauses that follow describe the algorithms that govern the operation for each mode of portal behavior.

7.1 Inbound portal operations

All active bridge portals shall eavesdrop all primary packets on the bus in order to determine whether or not the packet is to be forwarded. The algorithm for inbound portal operations is:

a) The *destination_bus_ID* field of all asynchronous packets (as defined by table 7-1) shall be examined to determine if the packet is local to the Serial Bus connected to the portal. If the *destination_bus_ID* field indicates the local bus ID, $3FF_{16}$, or if the *destination_bus_ID* field is equal to *bus_ID* in the PORTAL_CONTROL register, the packet is local. In this case, the portal’s acknowledgment and eventual response shall be as specified by IEEE Std 1394-1995.

b) If *rte* is zero, any non-local asynchronous packets (*i.e.*, those that do not meet the criteria defined above) shall be ignored by the bridge portal; no acknowledge or response packet shall be transmitted.

c) If *rte* is nonzero, all asynchronous response packets may be forwarded by the portal, according to the value of *destination_bus_ID* as described in e) below.

d) If *rte* is nonzero, the *source_ID* field of all asynchronous request packets is examined to determine if the portal is to forward the packet. If the *source_bus_ID* component of the field is equal to $3FF_{16}$, the portal shall not forward the request packet. If the *source_bus_ID* indicates that the request did not originate from a local node, the request may be forwarded by the portal, according to the value of *destination_bus_ID* as described in e) below. Otherwise, the *source_phy_ID* field is examined. If the corresponding bit in the NODE_ENABLE register is clear, the bridge shall not forward the request and shall generate a response of address error. When the corresponding bit in the NODE_ENABLE register is set, the request may be forwarded by the portal, according to the value of *destination_bus_ID* as described in e) below.

e) The *destination_bus_ID* field shall be compared against the portal’s routing information according to the Boolean expression:

ROUTING_BOUNDS.*bus_ID_lower_bound* <= *destination_bus_ID* <= ROUTING_BOUNDS.*bus_ID_upper_bound*.

If *rte* has a value of two, the packet is forwarded to the bridge's internal switching fabric if the expression evaluates to TRUE. Otherwise, if *rte* has a value of three, the packet is forwarded if the expression evaluates to FALSE.

f) If an asynchronous primary packet is forwarded by the portal, an ack_pending shall be transmitted on the local bus.

NOTE—The timing requirements of IEEE Std 1394-1995 for the transmission of an acknowledge packet effectively preclude the use of firmware to implement the above algorithm.

Once a portal has determined that an asynchronous packet is to be forwarded, that packet shall be replicated on the bridge's internal switching fabric and thus made available to the intended outbound portal. The implementation details of the internal switching fabric, in particular the buffering requirements (if any) and the timing of the exchange of asynchronous primary packets between portals, are beyond the scope of this standard.

7.2 Outbound portal operations

Asynchronous primary packets available on the bridge's internal switching fabric have already been screened according to the algorithm described in clause 7.1. As a result, outbound bridge portals have a simpler algorithm to determine whether or not a particular asynchronous primary packet shall be replicated on the portal's local bus. The procedure is described below:

a) If *rte* is zero, the portal is disabled and no asynchronous packets shall be replicated from the internal switching fabric to the portal's local bus;

b) Otherwise, the *destination_bus_ID* field of any asynchronous primary packet on the internal switching fabric shall be compared against the portal's routing information according to the Boolean expression:

ROUTING_BOUNDS.*bus_ID_lower_bound* <= *destination_bus_ID* <= ROUTING_BOUNDS.*bus_ID_upper_bound*.

If *rte* has a value of two, the packet is forwarded to the bridge's internal switching fabric if the expression evaluates to FALSE. Otherwise, if *rte* has a value of three, the packet is forwarded if the expression evaluates to TRUE.

c) If the acknowledgment received for the replicated packet is ack_pending, the outbound portal is finished with the forwarded packet; else

d) When an acknowledgment other than ack_pending is received, the outbound portal shall behave in accordance with X.

NOTE—The routing criteria in b) above are the inverse of the rules used to select inbound asynchronous packets. This permits the single ROUTING_BOUNDS register to control both inbound and outbound routing.

8. Isochronous operations and routing

This section describes the normal operations of a bridge to route isochronous data once an application has configured intervening bridge(s) to support an end-to-end path between a talker and a listener.

Bridges that support isochronous data transfers have two functions to perform:

a) The replication of a synchronized clock throughout the Serial Bus net; and
b) The replication of particular streams of isochronous data (identified by their channel number) throughout a portion of the Serial Bus net.

Bridge portals function as ***inbound portals*** when they eavesdrop on their bus to detect isochronous packets to be forwarded to one or more other buses. A portal that repeats an isochronous packet on its bus is acting as an ***outbound portal***.

The clauses that follow describe cycle clock replication and the algorithms that govern the operation for each mode of portal behavior.

8.1 Cycle clock replication

Just as there is a single cycle master node that provides uniform system time to a Serial Bus, a net of buses interconnected by bridges requires a single cycle master as the source of system time for the entire net.

NOTE—A usable name is needed for this singular cycle master. The affectionate term “cycle monster” is undoubtedly too informal for a standard but it pleases the editor to use this nickname until the working group selects a more formal appellation.

The cycle monster may be a cycle master-capable node or it may be one of the portals of one of the bridge(s) that connect the Serial Bus net. In either case, the cycle monster is selected and enabled by the bridge manager.

Once a cycle monster is active, the bridges propagate cycle start packets in accordance with the value of the *clk* field in each portal’s PORTAL_CONTROL register. A bridge that propagates the cycle time observes cycle start packets on only one portal; the remaining bridge portals shall be cycle masters or they shall be inactive with respect to isochronous traffic.

The value of the *clk* field is set by the bridge manager as it enumerates buses in the Serial Bus net and determines the routing for isochronous data. The bridge manager shall set the value of each portal’s *clk* field subject to the following restrictions:

a) Any number of portals may have a *clk* value of zero.
b) At most one portal may have a *clk* value of one.
c) No portal shall have a *clk* value of two unless at least one other portal has a *clk* value of one. If there is at least one such portal, any number of portals may have a *clk* value of two.

Note that the value of *clk* determines the portal’s behavior for all other isochronous data. The reception or transmission of isochronous data is inhibited on all portals with a *clk* value of three, regardless of the state of the portals’ CHANNEL_SWITCH registers. This is an important consideration when the physical topology of the Serial Bus net includes loops, since it permits the bridge manager to parse the Serial Bus net into a tree.

A bridge shall have a single, free-running cycle timer shared by all portals[1]. The cycle timer shall be resynchronized in accordance with cycle start packets observed by one of the portals. This portal is identified by a value of one for the *clk* field in its PORTAL_CONTROL register.

[1] This is a logical requirement, not an implementation. A bridge design may have separate cycle timers for the portals so long as the implementation provides a method to retain synchronization between the timers.

When the bridge's cycle timer generates a cycle synchronization event (*i.e.*, the *cycle_count* portion increments when a 125 µs period elapses), the PHY's of all portals whose *clk* field has a value of two shall arbitrate for the bus and transmit a cycle start packet as specified by IEEE Std 1394-1995.

During an isochronous cycle, any portal whose PORTAL_CONTROL.*clk* field is nonzero shall forward isochronous data packets in accordance with the algorithms described in clauses 8.3 and 8.4.

8.2 Plug control registers

The plug control registers, specified in P1394a, draft supplement to the Serial Bus standard, provide a uniform method to manage isochronous connections. The bridge architecture makes use of the plug information in conjunction with bridge portal's own registers to determine the routing of isochronous data streams.

Plug control registers are not a global bridge resource; they are replicated for each portal.

P1394a should be consulted in order to understand the descriptions that follow.

8.3 Inbound portal operations

During an isochronous cycle, inbound portals eavesdrop on all isochronous packets in order to examine the *channel* field in the packet header. Although the details are dependent upon the implementation, it is assumed that each portal has a bit mask that identifies which of the 64 isochronous channels are to be buffered and, after a constant isochronous delay across the bridge's internal switching fabric, subsequently retransmitted by one or more outbound portals.

The information necessary to determine whether or not a particular isochronous channel is to be retransmitted resides in the inbound portal's INPUT_PLUG register(s). If either the *broadcast* bit or the *point_to_point* field is nonzero, the portal is enabled to listen to and forward packets identified by the *channel* field in the INPUT_PLUG register.

An inbound portal shall forward an isochronous packet by making it available to the bridge's internal switching fabric, with the expectation that one or more outbound portals subsequently retransmit the packet.

8.4 Outbound portal operations

Outbound portals shall maintain queue(s) of isochronous packets observed on the bridge's internal switching fabric that match the portal's criteria for retransmission. The isochronous packets shall be retransmitted on a particular isochronous cycle number that is a fixed number of cycles later than the cycle during which the packet(s) were observed by the inbound portal.

Isochronous packets distributed *via* the bridge's internal switching fabric are uniquely identified by the combination of their channel number and the ordinal of their inbound portal. The isochronous channel number alone is insufficient since the same channel number may be in use on buses connected by a single Serial Bus bridge.

The information necessary to determine whether or not a particular isochronous packet is to be retransmitted resides in the outbound portal's OUTPUT_PLUG register(s) and the corresponding CHANNEL_SWITCH registers. If either the *broadcast* bit or the *point_to_point* field in an OUTPUT_PLUG register is nonzero, the outbound portal is configured to retransmit packets for an isochronous channel. The packet to retransmit is identified by information in the CHANNEL_SWITCH[*n*] register that corresponds to the OUTPUT_PLUG[*n*] register with the same ordinal, *n*. The CHANNEL_SWITCH register specifies the originating *portal* and *channel* number that uniquely identify the isochronous packet within the internal switching fabric.

Before the packet may be retransmitted, the outbound portal shall transform the packet header according to the information in the OUTPUT_PLUG register. The OUTPUT_PLUG register specifies the *channel* number for the retransmitted isochronous packet header. This register also specifies the speed at which the isochronous packet shall be transmitted.

9. Serial Bus net configuration

The bridge manager, once selected as described in the preceding section, is responsible to: a) enumerate all the connected buses within the Serial Bus net and assign each a unique *bus_ID*; b) configure each Serial Bus bridge so that it properly routes asynchronous and isochronous data from one bus to another; and c) arbitrarily parse the Serial Bus net topology into (logically) a tree topology even if it is (physically) connected with loops. The sections that follow describe the procedures the bridge manager uses to accomplish these goals.

In the discussion of the procedures that follow, a few definitions are necessary. The global variable ***max_bus_ID***, which is initialized to zero at the start, represents the highest *bus_ID* assigned to any bus in the Serial Bus net. A bus is ***enumerated*** if a broadcast write has occurred to the NODE_IDS register of all nodes connected to the bus. At this point, the state of the bridges is indeterminate and the bus is ***unconfigured***. An unconfigured bridge on an enumerated bus that is to be configured is the ***target bridge***. Each target bridge becomes ***outbound configured*** when all of its PORTAL_CONTROL[*n*] registers are initialized and when those portions of its ROUTING_BOUNDS registers necessary to forward transactions from the bridge manager are initialized. When all bridges on an unconfigured bus are outbound configured, the bus is outbound configured as well. During the traversal of the Serial Bus net, the information needed to complete configuration of the outbound configured bridges accumulates each time *max_bus_ID* is incremented. When the last bus is enumerated, it is possible to update all the remaining portions of the bridge ROUTING_BOUNDS registers. Once this update is complete, the bridges and the buses are ***fully configured***.

The bridge manager follows the steps described below to fully configure the Serial Bus net:

a) The global variable *max_bus_ID* is initialized to zero;
b) The bridge manager broadcasts a quadlet write transaction with a destination address of BRIDGE_RESET register. The data value of the write is ignored. A bridge that receives a quadlet write to its BRIDGE_RESET register shall reset its PORTAL_CONTROL, NODE_ENABLE and ROUTING_BOUNDS registers to their initial values.
c) The bridge manager broadcasts a quadlet write transaction with a data value of zero to the NODE_IDS registers of all nodes on the local bus. This write has side effects at any bridges connected to the local bus: the *bus_ID* field in each bridge's local PORTAL_CONTROL[*n*] register, where *n* is the ordinal of the portal connected to the bridge manager's local bus, is updated to the broadcast value.
d) The bridge manager invokes the bus configuration procedure, described in the next section. This procedure, together with recursive invocations of itself and of the bridge configuration procedure, parses the Serial Bus net into a tree and effects a traversal of the tree to enumerate all buses and configure all bridges. Upon return from the invocation of the bus configuration procedure, the Serial Bus net is outbound configured and the global variable *max_bus_ID* is set to the highest numbered bus on the net.
e) In order to fully configure the route information at all bridges, the bridge manager must update any unconfigured *out_portal*[*j*] fields at each bridge, where *j* is less than or equal to the final value of *max_bus_ID*. The updates are accomplished by propagating the value of the *out_portal*[*k*] field, where *k* is the largest index to a configured *out_portal* field at a bridge, to all the unconfigured *out_portal*[*j*] fields at the same bridge. Once this is complete for all bridges, the Serial Bus net is fully configured.

9.1 Bus configuration procedure

The only argument to the bus configuration procedure is the global variable *max_bus_ID*. The value of *max_bus_ID* indicates the current bus to be configured. Configuration consists of identifying all the bridges present on the bus and configuring each one in turn. Any buses that lie beyond the bridges are recursively configured by additional invocations of this same procedure. The steps to be followed are outlined below:

a) Examine the 63 possible nodes connected to the bus identified by *max_bus_ID* and construct a list of all bridge nodes. Serial Bus bridges have a standard unit architecture that may be identified by configuration ROM entries. The order in which bridges are configured is arbitrary.
b) If there are no unconfigured bridges on the bus, return to the invoker of this procedure. Note that at this point *max_bus_ID* contains the ID of the most recently enumerated and configured bus.

c) Otherwise, select one of the unconfigured bridges and invoke the bus configuration procedure described in the following section. This procedure is invoked with one argument, *target_bridge_ID*, which is set to the unique node ID of the bridge to be configured.

d) Upon return from the bridge configuration procedure, all buses and bridges connected to the bridge have been enumerated and outbound configured. The most recently enumerated bus is indicated by the value of *max_bus_ID*. Mark the bridge outbound configured and continue with step b).

The information necessary to complete the configuration of the bridges on this bus, that is, to move them from the outbound configured state to the fully configured state, is not available until the last bus in the Serial Bus net is enumerated. See the preceding section for a description of how the bridge manager revisits each bridge in order to fully update the ROUTING_BOUNDS registers.

9.2 Bridge configuration procedure

Once an unconfigured bridge is encountered, it is necessary to configure all of its portals and to configure as much as possible of its route map. Note that at least one of the PORTAL_CONTROL[*n*] registers is configured at the outset. This is the result of a broadcast write to the NODE_IDS register of all nodes on the local bus connected to that portal. The other, unconfigured portals may be identified by a value of $3FF_{16}$ in the *bus_ID* field of the PORTAL_CONTROL[*n*] registers. Portals are configured by means of the following procedure:

a) First, identify the portal connected to the current bus identified by *max_bus_ID*. That is, set the *l* bit in the PORTAL_SELECT register to one and note the value of the *portal* field in a subsequent read of the same register.examine the *n* PORTAL_CONTROL registers and note the value of *n* for which the *bus_ID* field of the PORTAL_CONTROL[*n*] register is equal to *max_bus_ID*. Update the ROUTING_BOUNDS registers by writing *portaln* to the *out_portal*[0] through *out_portal*[*max_bus_ID*] fields, inclusive.

b) Next, locate an unconfigured PORTAL_CONTROL register, identified by its ordinal *n*. PORTAL_CONTROL registers may be examined by altering the value of the *portal* field in the PORTAL_SELECT register to *n* before reading the PORTAL_CONTROL register. A PORTAL_CONTROL register is unconfigured if its *bus_ID* field is equal to $3FF_{16}$. If no unconfigured portals remain, the bridge configuration procedure is complete; return control to the invoker of this procedure.

c) Portal *n* connects to a not yet enumerated bus. Increment the value of *max_bus_ID* by one and write the new value to the *bus_ID* field of the PORTAL_CONTROL[*n*] register. This also causes the bridge to broadcast a quadlet write transaction to the NODE_IDS registers on that bus. The value of the *bus_ID* field of the broadcast is *max_bus_ID*.

d) Update the route information for the just enumerated bus by writing *n* to the *out_portal*[*max_bus_ID*] field in the target bridge's ROUTING_BOUNDS registers.

e) Configure the just enumerated bus that lies beyond portal *n* by means of the bus configuration procedure described in the preceding section.

f) Upon return from the bus configuration procedure, all buses and bridges connected through portal *n* are outbound configured. Update the current bridge's route information by writing the portal identifier, *n*, to any unconfigured *out_portal*[*j*] fields in the ROUTING_BOUNDS registers where *j* is less than or equal to *max_bus_ID*. A value of FF_{16} in an *out_portal*[*j*] field indicates an unconfigured portion of the ROUTING_BOUNDS.

g) Configure any remaining unconfigured portals by continuing with step b).

NOTE—When all the portals of a bridge (and the buses and other bridges that lie beyond them) have been configured by this procedure, the bridge is outbound configured. It is not yet possible to configure the remainder of the ROUTING_BOUNDS registers *out_portal* fields, since the ultimate value of *max_bus_ID* has not yet been determined. Once all buses are enumerated, the bridge manager can complete the configuration of the bridges.

