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

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

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."*

3.2 Technical glossary

The following are terms that are used in this standard:

3.2.1 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).

3.2.2 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.

3.2.3 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.

3.2.4 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 $3FF_{16}$.

3.2.5 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*.

3.2.6 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.

3.2.7 net cycle master: The cycle master for one Serial Bus in the net selected by the bridge manager to be the clock source for the entire net.

3.2.8 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.

~~**3.2.9 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.~~

3.2.10 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.

3.2.11 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 both portals. Figure 4-1 below illustrates this model.

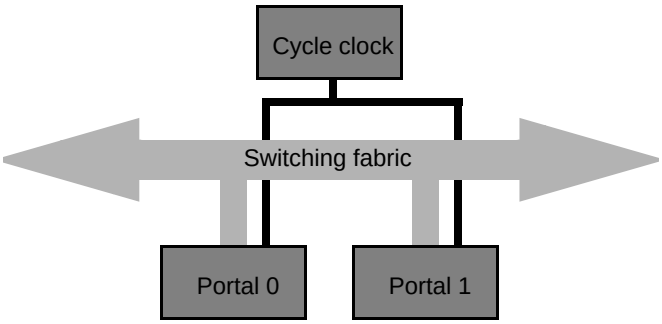

Figure 4-1 — Bridge model

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 the other 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 the 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 both 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 both 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 the 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, CSRs (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.

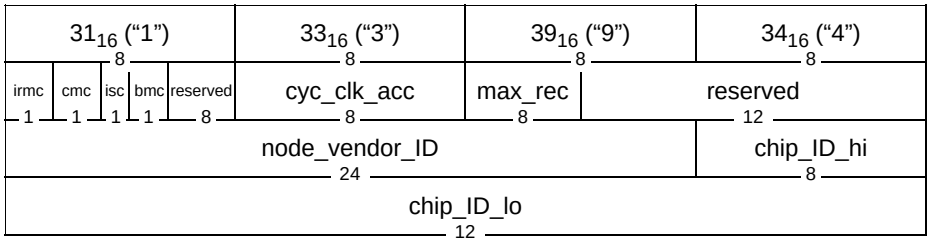

Figure 5-1 — Bus_Info_Block format

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 the net cycle clock and 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, lock response or asynchronous stream packet. This usage is an extension to that defined by IEEE Std 1394-1995, which specifies only the size of an asynchronous block write request. When *max_rec* is zero the bridge's maximum payload is not specified. Otherwise, the maximum payload ~~size~~ is defined as 2^{max_rec+1} and shall not be greater than the maximum asynchronous data payload for the speed implemented by the bridge portals ~~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.

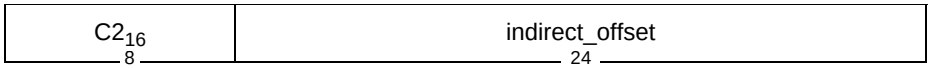

Figure 5-2 — Bus_Dependent_Info entry format

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.

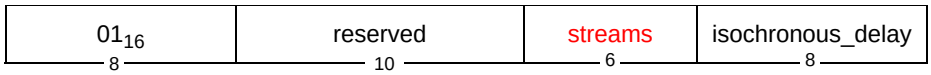

Figure 5-3 — Bridge_Capabilities entry format

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 ***streams*** field specifies the total of STREAM_CONTROL registers implemented by each portal. The value represents the number of concurrently active streams supported by the bridge.

The ***isochronous_delay*** field specifies the constant delay that isochronous packets incur when they are transferred from one bridge portal to another. If the bridge implements no isochronous capabilities, the value of *isochronous_delay* shall be zero. Otherwise, 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

The control and status registers (CSR's) implemented by a bridge portal shall conform to the requirements defined by this standard and its normative references. The CSR's belong to three groups:

— core registers required by ISO/IEC 13213:1994;
— bus-dependent registers required by IEEE Std 1394-1995; and
— registers required by this standard.

The addresses of all registers are specified in terms of offsets, in bytes, within initial register space, where the base address of initial register space is FFFF F000 0000_{16} relative to initial node space. IEEE Std 1394-1995 should be consulted for detailed descriptions of both the core and Serial Bus-dependent registers; the table below lists those that are mandatory for bridges.

Table 5-1 — Core and Serial Bus-dependent registers for all bridges

Offset	Name	Description
0	STATE_CLEAR	State and control information.
4	STATE_SET	Sets STATE_CLEAR bits.
8	NODE_IDS	Contains the 16-bit *node_ID* value used to address the bridge portal.
C_{16}	RESET_START	Resets the bridge's state. TBD—What is the effect of a write to this register, if any?
18_{16} — $1C_{16}$	SPLIT_TIMEOUT	Time limit for split transactions on the connected bus.
210_{16}	BUSY_TIMEOUT	Controls the bridge's retry protocols.

Isochronous capabilities are optional. If a bridge supports isochronous operations, it shall be cycle master capable and isochronous resource manager capable as well as isochronous capable. These capabilities require implementation of the Serial Bus-dependent registers listed in table 5-2.

Table 5-2 — Serial Bus-dependent registers for isochronous bridges

Offset	Name	Description
200_{16}	CYCLE_TIME	24.576 MHz clock required for isochronous operation.
204_{16}	BUS_TIME	System (net) time in seconds.
$21C_{16}$	BUS_MANAGER_ID	Contains the 16-bit *node_ID* 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.

In addition to the preceding requirements of IEEE Std 1394-1995, this standard specifies registers within the Serial Bus-dependent portion of initial units space. These bridge portal registers are summarized in table 5-3.

Table 5-3 — Bridge portal registers

Offset	Name	Description
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*.
$240C_{16}$	BRIDGE_RESET	A write of any value to this location is equivalent to a power reset.
2410_{16}	PORTAL_CONTROL	Configures the portal.
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 requests through the bridge to confirm their receipt of a reset notification.

Table 5-3 — Bridge portal registers (Continued)

Offset	Name	Description
2420_{16}	ROUTING_BOUNDS	Defines the 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 request and response subactions through the bridge.
2640_{16} — $267C_{16}$	OUTBOUND_SPEED_MAP	Controls the speed at which packets are transmitted by an outbound portal.
2680_{16} — $277C_{16}$	STREAM_CONTROL	These registers control the inbound or outbound routing for asynchronous and isochronous stream packets.
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.

Unless otherwise specified, all registers shall support quadlet read and quadlet write transactions.

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.

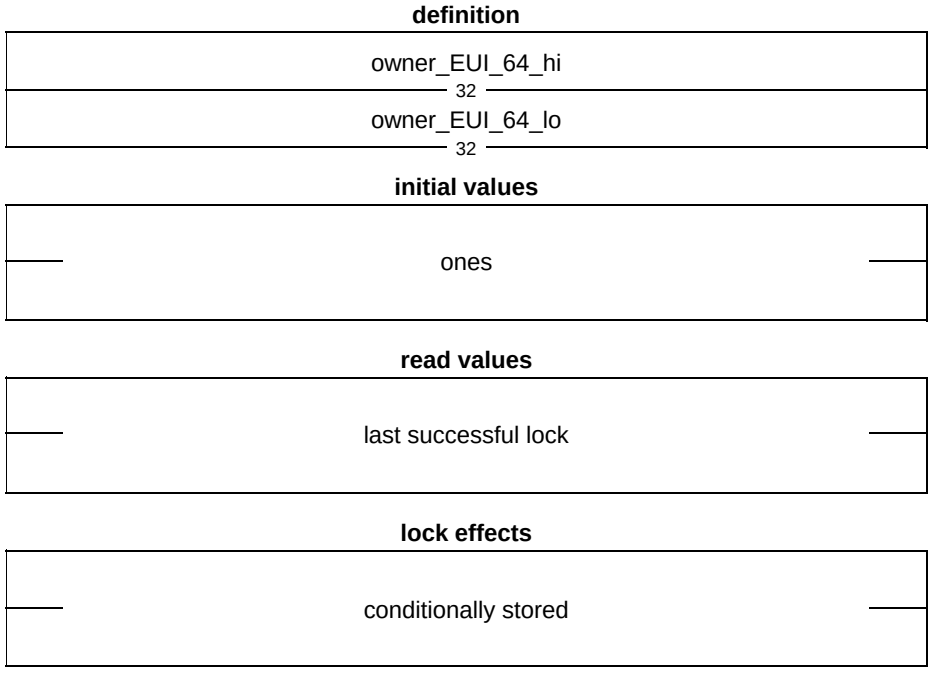

Figure 5-4 — OWNER_EUI_64 format

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.

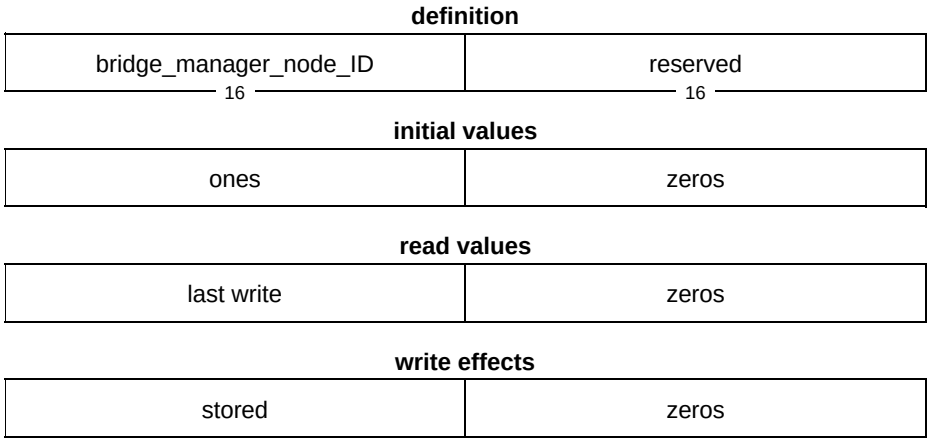

Figure 5-5 — BRIDGE_MANAGER_ID format

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; the effect of a quadlet write to any portal's BRIDGE_RESET register is equivalent to a power reset of the bridge. Figure 5-6 illustrates the format of this register.

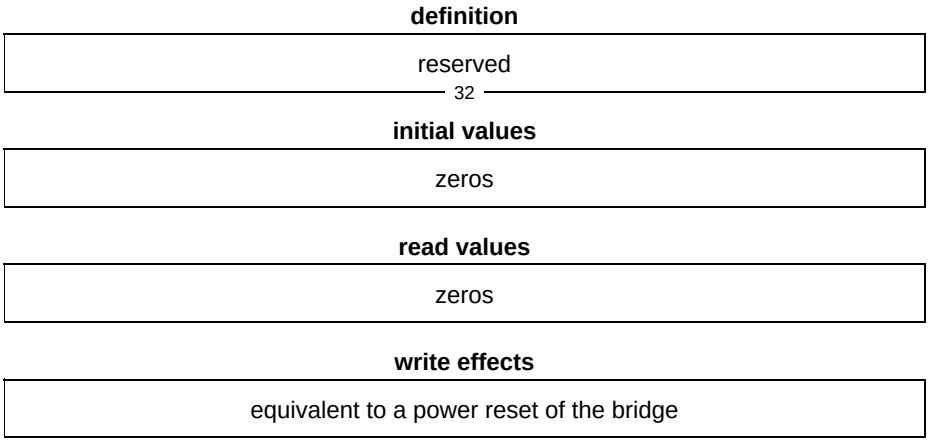

Figure 5-6 — BRIDGE_RESET format

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

A Serial Bus bridge shall implement a PORTAL_CONTROL register for both of the bridge's portals; the register configures the operations of each portal. Figure 5-7 illustrates the format of this register.

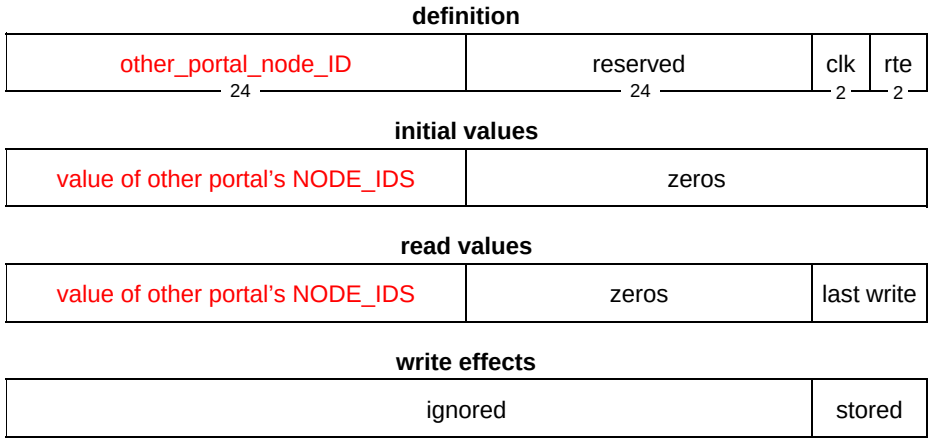

Figure 5-7 — PORTAL_CONTROL format

The *other_portal_node_ID* field shall be equal to the most significant 16 bits of the NODE_IDS register for the other portal.

~~The ***q***, or ***quarantine***, bit, if set, has the property of disabling the propagation of all asynchronous request packets addressed to *bus_ID* that originate from the bridge's other portal. 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

Value	Description
2	Enabled; forward asynchronous request or response packets if ROUTING_BOUNDS.*bus_ID_lower_bound* <= *destination_bus_ID* and *destination_bus_ID* <= ROUTING_BOUNDS.*bus_ID_upper_bound*
3	Enabled; forward asynchronous request or response packets if *destination_bus_ID* < ROUTING_BOUNDS.*bus_ID_lower_bound* or ROUTING_BOUNDS.*bus_ID_upper_bound* < *destination_bus_ID*

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

5.2.5 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 5-8 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.

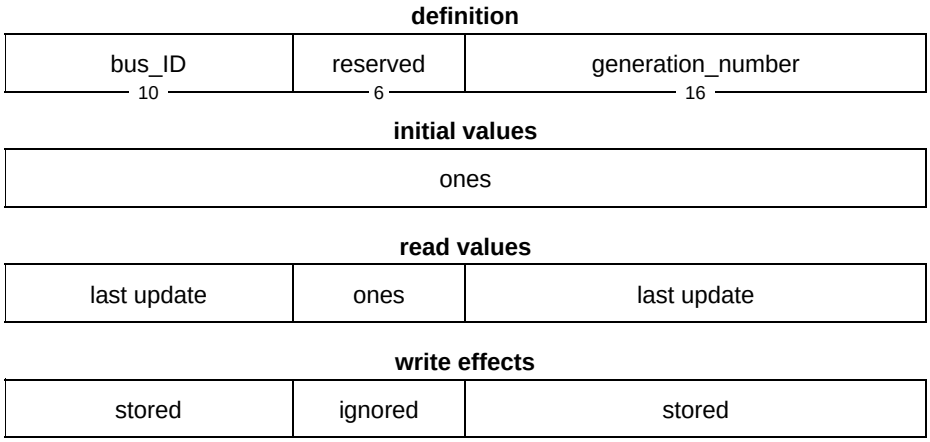

Figure 5-8 — RESET_NOTIFICATION format

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 connected Serial Bus. 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 both bridge portals.

5.2.6 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 5-9 below.

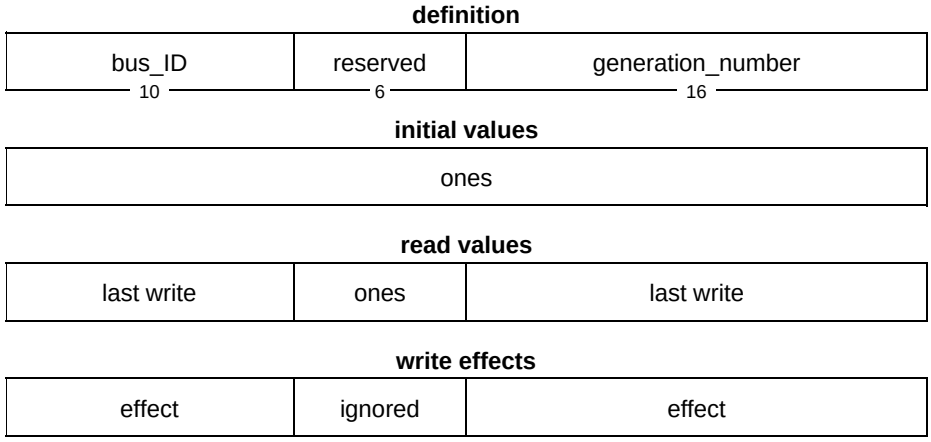

Figure 5-9 — RESET_ACKNOWLEDGE format

A quadlet write addressed to the RESET_ACKNOWLEDGE register shall cause the portal's NODE_ENABLE register to be updated if the following are true:

— The most significant 10 bits of the *source_ID* field of the quadlet write request are equal to either $3FF_{16}$ or the most significant 10 bits of the portal's NODE_IDS register; and
— The *quadlet_data* field is equal to the present value of the RESET_ACKNOWLEDGE register.

Quadlet write requests that do not meet these criteria shall be rejected with either an *ack_type_error* or a response of *resp_type_error*. Otherwise, the bit in the NODE_ENABLE register that corresponds to the sender's physical ID shall be set to one. See clause 5.2.10 for the mapping from physical ID to particular bits in the register.

The result of a successful write to RESET_ACKNOWLEDGE that updates the NODE_ENABLE register is to enable forwarding of asynchronous request and response packets originated by the node identified by *source_ID*.

5.2.7 ROUTING_BOUNDS register

The ROUTING_BOUNDS register provides parameters used by the bridge to determine asynchronous transaction routing. The format of this register is given by figure 5-10 below.

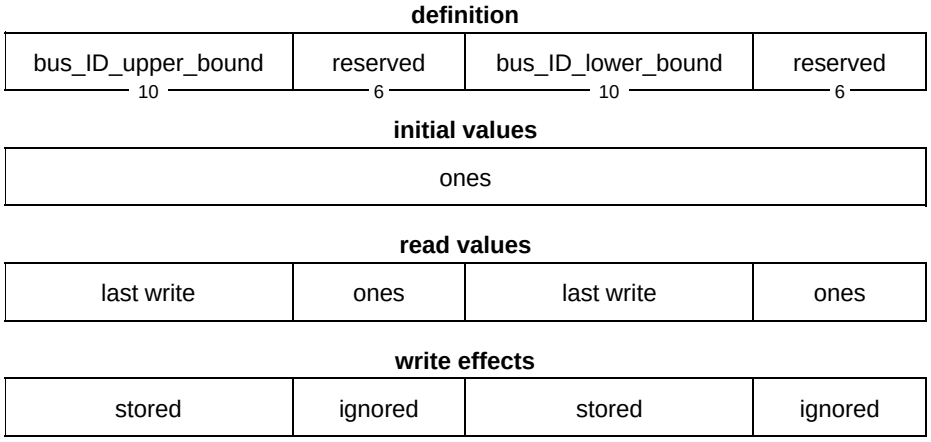

Figure 5-10 — ROUTING_BOUNDS format

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

5.2.8 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 the bridge's other portal. This register, whose format is shown below in figure 5-11, is typically used by the bridge manager during initialization and bus enumeration.

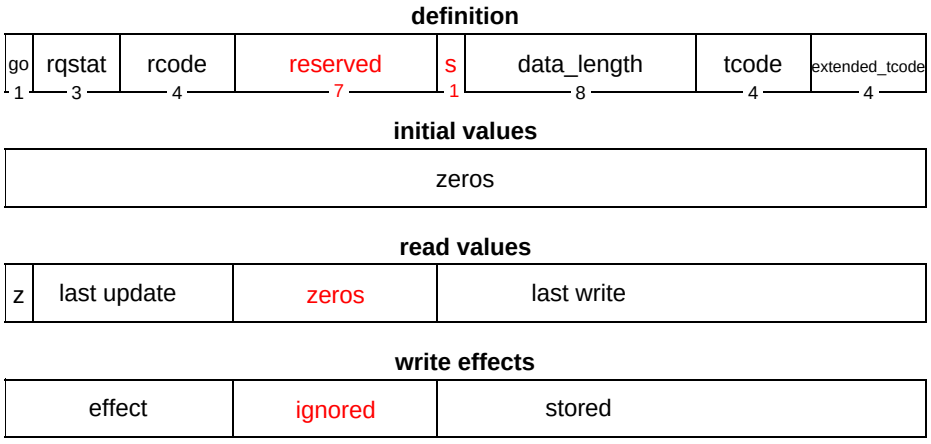

Figure 5-11 — REMOTE_REQUEST format

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 determined by the *source* field (see below). 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 ***source*** bit (abbreviated as *s* in the illustration above) shall specify the value of *source_ID* in the transmitted request. If *source* is zero, the value shall be equal to the most significant 16 bits of the other portal's NODE_IDS register. Otherwise, when *source* is one, the transmitted *source_ID* shall be equal to the concatenation of $3FF_{16}$ and the least significant 6 bits of the other portal's NODE_IDS register.

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 other portal. The values of *tcode* are specified by IEEE Std 1394-1995; 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.

5.2.9 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 5-12 below illustrates the format of this register.

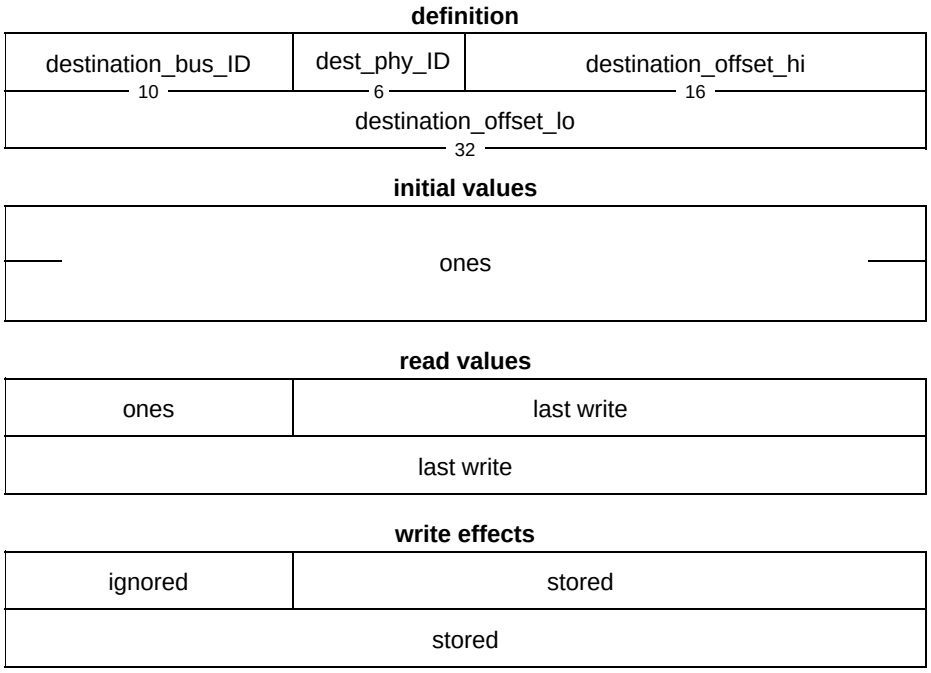

Figure 5-12 — REMOTE_DESTINATION format

The ***destination_bus_ID*** and ***dest_phy_ID*** fields shall be set to a value that identifies the remote node to which the request is to be addressed. Registers within the other portal are addressable when *destination_bus_ID* and *dest_phy_ID* are set to the value of *other_portal_node_ID* reported in the PORTAL_CONTROL register.

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.

5.2.10 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 5-12 below shows the format of this register.

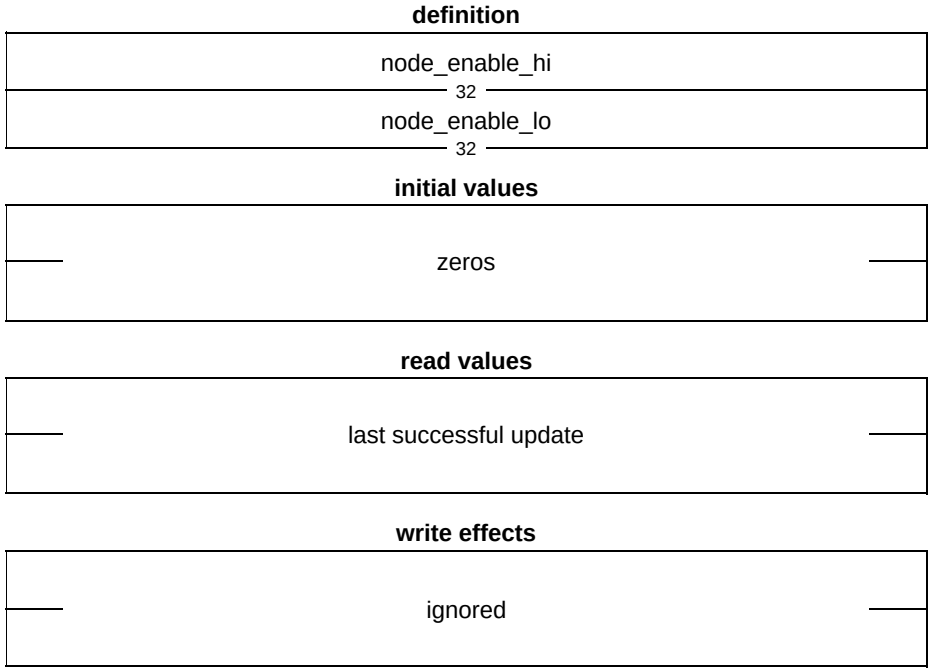

Figure 5-13 — NODE_ENABLE format

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 register.

The NODE_ENABLE register shall be cleared 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 in the NODE_ENABLE register may be modified by means of a write to the RESET_ACKNOWLEDGE register. When a bridge portal receives a quadlet write transaction addressed to the RESET_ACKNOWLEDGE register for which the *source_bus_ID* field is either $3FF_{16}$ or equal to the most significant 10 bits of the portal's NODE_IDS 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.

5.2.11 OUTBOUND_SPEED_MAP register

This register has nearly the same format as a row from the bus manager SPEED_MAP register. The 16 quadlets of the portal's OUTBOUND_SPEED_MAP register represent SPEED_MAP.*speed_code*[*i*] through *speed_code*[*i* + 63], inclusive, where *i* is equal to 64 times the portal's NODE_IDS.*phy_ID*. When the *destination_bus_ID* of an outbound asyn-

chronous packet is equal to the portal's bus ID, the *destination_physical_ID* shall be used as an index to OUTBOUND_SPEED_MAP to obtain the transmission speed for the packet. Otherwise the packet is in transit to another bus and the transmission speed shall be obtained from OUTBOUND_SPEED_MAP.*speed_code*[63].

Subsequent to a power reset or bus reset, the all entries in the OUTBOUND_SPEED_MAP shall be set to indicate S100. Either the bridge portal or the bridge manager may update this register with values that more accurately represent the speed characteristics and topology of the local nodes, including other bridge portals. The value of OUTBOUND_SPEED_MAP.*speed_code*[63] shall not exceed the speed at which the bridge portal may transmit to any other bridge portal on the connected bus.

5.2.12 STREAM_CONTROL registers

The STREAM_CONTROL registers are an array of quadlet registers, each of which may control the inbound or outbound routing of a stream by a bridge portal. The number of registers implemented shall be specified by the *streams* field in the Bridge_Capabilities configuration ROM entry. The registers are addressed as STREAM_CONTROL[0] (the lowest address) through STREAM_CONTROL[*streams* - 1], inclusive. Corresponding STREAM_CONTROL registers from each bridge portal function as pairs; that is, a stream received according to the parameters of STREAM_CONTROL[*n*] on one portal is retransmitted according to the parameters of STREAM_CONTROL[*n*] on the other portal. The format of the STREAM_CONTROL register is illustrated by figure 5-14 below.

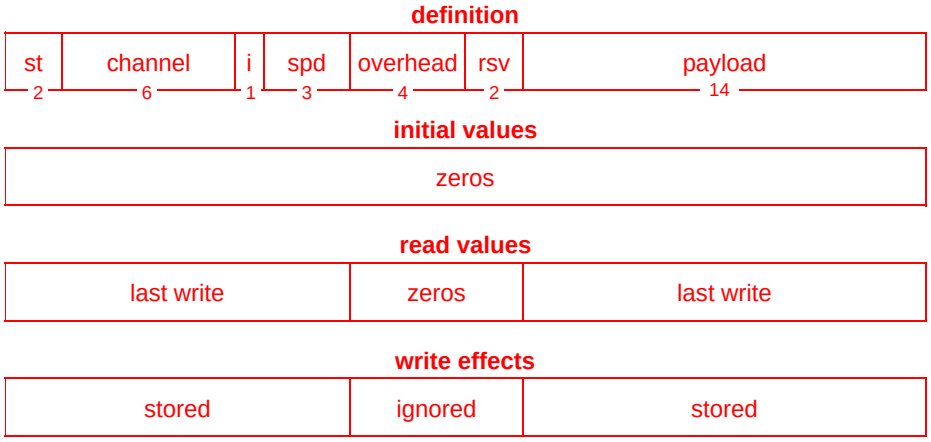

Figure 5-14 — STREAM_CONTROL format

The ***st*** field shall specify the state of the bridge portal with respect to the stream, as encoded by the table below.

Value of *st*	Stream status
0	Inactive
1	Listener
2	Talker
3	Talker (reallocation proxy)

When *st* equals three and the *isochronous* bit is one, the bridge portal initiates Serial Bus requests to reallocate isochronous bandwidth after a bus reset; these operations are described in more detail in X.

The ***channel*** field shall specify the channel number in the stream packet. This field is valid only when *st* is nonzero. When the portal is configured to listen, *channel* specifies which stream packets to receive. Otherwise *channel* specifies the channel number to be transmitted in the stream packet. A stream packet's channel number may be modified as it passes through a bridge.

The ***i***, or ***isochronous*** bit, shall be zero for asynchronous stream operations and one for isochronous stream operations.

When the value of *st* is two or three, the ***spd*** field shall specify the speed at which the stream packets are transmitted, as encoded by the table below.

Value of *spd*	Data rate
0	S100
1	S200
2	S400
3	S800
4	S1600
5	S3200
6 — 7	Reserved

The ***overhead*** field shall encode a value that indicates the amount of isochronous bandwidth allocated in addition to that allocated for the isochronous stream packet. Isochronous bandwidth is expressed in terms of bandwidth allocation units, defined by IEEE Std 1394-1995. One bandwidth allocation unit represents the time required to transmit one quadlet of data at a future S1600 data rate, roughly 20 nanoseconds. Isochronous overhead includes both arbitration time and the gap that follows an isochronous stream packet. When *overhead* is zero, the additional bandwidth is 512 bandwidth allocation units. Otherwise, the additional bandwidth is *overhead* * 32 bandwidth allocation units.

The ***payload*** field shall specify the maximum number of quadlets that may be transmitted in a single isochronous packet for this stream. The value of *payload* does not include the isochronous header, header CRC or data CRC required as part of an isochronous stream packet; it counts only those quadlets that are part of the data payload.

The values of *spd*, *overhead* and *payload* in the STREAM_CONTROL register shall describe the bandwidth allocated for the isochronous stream. If the bandwidth allocation is modified, these fields shall be updated accordingly. When *overhead* is zero, the bandwidth allocation is $512 + (payload + 3) * 2^{4 - spd}$ bandwidth allocation units. Otherwise the bandwidth allocation is $overhead * 32 + (payload + 3) * 2^{4 - spd}$ bandwidth allocation units.

NOTE—In the formulae above there is a negative exponent at the S3200 data rate. When dividing by two at this data rate the result shall be rounded up to the next larger integer value.

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 5-12 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 larger, shall be a integral number of quadlets. The maximum size shall be less than or equal to 64 quadlets.

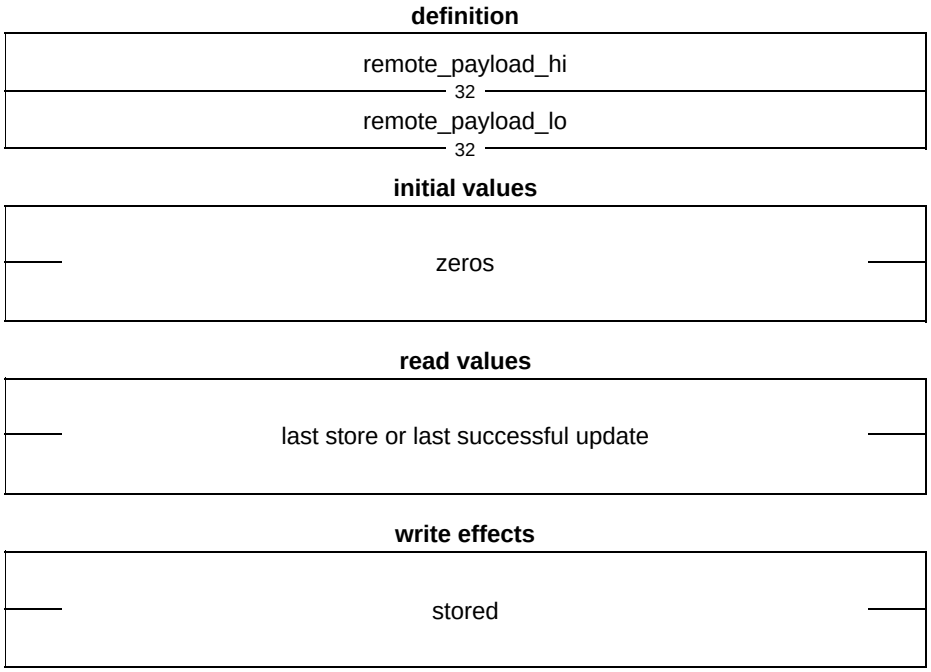

Figure 5-15 — REMOTE_PAYLOAD format

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.

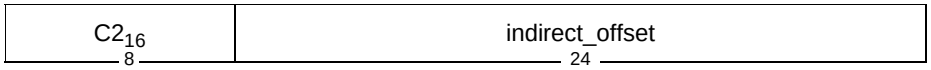

Figure 6-1 — Bus_Dependent_Info entry format

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.

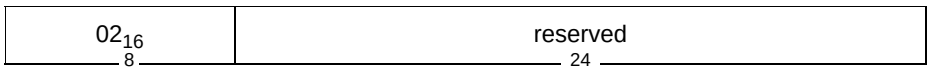

Figure 6-2 — Bridge_Manager_Capabilities entry format

The entry is identified by the *key_type* and *key_value* fields which together have a value of 02_{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.

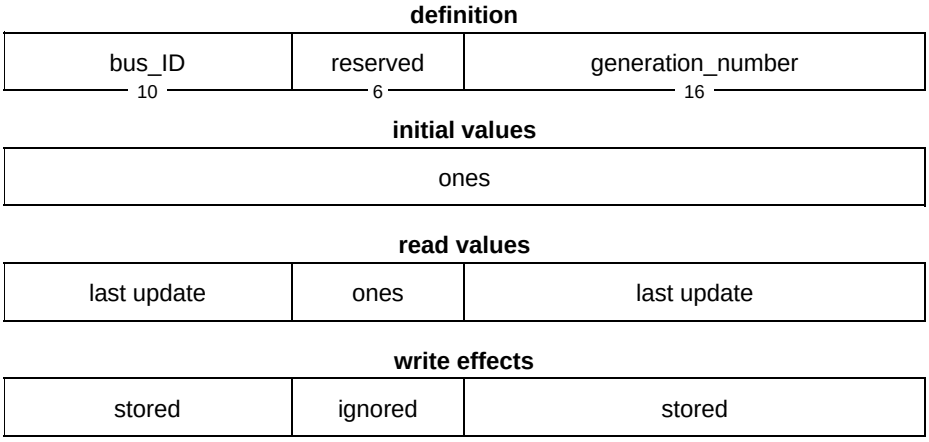

Figure 6-3 — RESET_NOTIFICATION format

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. Remote requests

The REMOTE_REQUEST, REMOTE_DESTINATION and REMOTE_PAYLOAD registers provide a facility for a read, write or lock request subaction to be originated by the bridge's other portal and for the response to be returned. This facility is available whether or not the bridge is configured to route asynchronous or isochronous packets.

8. 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 initialization 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 transmits 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 8-1 — 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

8.1 Inbound portal operations

Each bridge portal shall eavesdrop on all asynchronous primary packets on its connected Serial Bus in order to determine whether the destination is local or remote and, if remote, whether or not it shall be forwarded to the outbound portal. Asynchronous packet routing is determined by a combination of the source and destination ID fields in the packet, the PORTAL_CONTROL.*rte* value, the portal's bus ID and the state of the NODE_ENABLE bits. These interrelationships are summarized by table 8-2.

Table 8-2 — Asynchronous packet routing

rte	Destination Bus ID	Destination PHY ID	Source Bus ID	Action
—	$3FF_{16}$ or NODE_IDS.*bus_ID*	—	—	Local packet. Acknowledge *per* IEEE Std 1394-1995 and draft standard P1394a.
0	Neither $3FF_{16}$ nor NODE_IDS.*bus_ID*	—	—	Routing disabled. Ignore nonlocal packet and do not transmit an acknowledgment.
Nonzero	—	—	$3FF_{16}$	Unable to route. Ignore packet and do not transmit an acknowledgment.
	$3FF_{16}$	$3F_{16}$	Not $3FF_{16}$	Global broadcast. Forward packet to outbound portal but do not transmit an acknowledgment.

Table 8-2 — Asynchronous packet routing (Continued)

rte	Destination Bus ID	Destination PHY ID	Source Bus ID	Action
2	>= *bus_ID_lower_bound* and <= *bus_ID_upper_bound*	—	NODE_IDS.*bus_ID*	Remote packet origination. Forward packet to outbound portal according to NODE_ENABLE and (if forwarded) transmit an *ack_pending* or *ack_complete*.
			Neither $3FF_{16}$ nor NODE_IDS.*bus_ID*	Remote packet in transit. Forward packet to outbound portal and transmit an *ack_pending* or *ack_complete*.
3	< *bus_ID_lower_bound* or > *bus_ID_upper_bound*	—	NODE_IDS.*bus_ID*	Remote packet origination. Forward packet to outbound portal according to NODE_ENABLE and (if forwarded) transmit an *ack_pending* or *ack_complete*.
			Neither $3FF_{16}$ nor NODE_IDS.*bus_ID*	Remote packet in transit. Forward packet to outbound portal and transmit an *ack_pending* or *ack_complete*.
All other combinations not described above				Unable to route. Ignore packet and do not transmit an acknowledgment.

When the portal's NODE_ENABLE register determines routing eligibility, the packet's 6-bit source physical ID is used as an index into the register's bits. The most significant bit of the first quadlet of NODE_ENABLE corresponds to a physical ID of 63 and the least significant bit in the same quadlet represents a physical ID of 32. Physical ID's 31 through zero, inclusive, map to the most significant to the least significant bits, respectively, of the second quadlet of NODE_ENABLE. When an asynchronous packet is otherwise eligible to be forwarded to the outbound portal, the action taken is specified by the value of the NODE_ENABLE bit mapped from the packet's 6-bit source physical ID. If the bit is zero, the packet shall not be forwarded and the portal shall return an address error response. Otherwise the portal transmits an *ack_pending* and forwards the packet to the outbound portal.

NOTE—The timing requirements of IEEE Std 1394-1995 for the transmission of an acknowledge packet preclude most firmware implementations of 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 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.

8.2 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 8.1. As a result, outbound bridge portals simply replicate all packets from the internal switching fabric to the portal's connected Serial Bus. The speed at which an outbound asynchronous packet shall be transmitted is determined by both the destination bus ID and physical ID, as specified by table 8-3.

Table 8-3 — Outbound packet speed

Destination Bus ID	Destination PHY ID	Speed
NODE_IDS.*bus_ID*	Not $3F_{16}$	OUTBOUND_SPEED_MAP.*speed_code*[PHY ID]
	$3F_{16}$	S100
$3FF_{16}$	$3F_{16}$	S100
Neither $3FF_{16}$ nor NODE_IDS.*bus_ID*	—	OUTBOUND_SPEED_MAP.*speed_code*[63]

After transmitting an outbound asynchronous packet, the bridge portal expects to observe an acknowledge packet. The action taken by the bridge portal depends upon the acknowledgment, as defined in table 8-4.

Table 8-4—Outbound portal actions in response to acknowledgment

Name	Action
ack_complete	If the packet was a request subaction, the bridge shall transmit the appropriate response packet (with *resp_code* that indicates RESPONSE COMPLETE) to the node identified by *source_node_ID* in the request. If the packet was a response subaction no further action is required.
ack_pending	If the packet was a request subaction no further action is required. A pending acknowledgment to any other primary packet is an error but requires no action by the bridge.
ack_busy_X ack_busy_A ack_busy_B	The bridge may retransmit the packet in the next fairness interval in accordance with BUSY_TIMEOUT. If the bridge abandons retry attempts and the packet was a request subaction the bridge shall transmit the appropriate response packet (with *resp_code* that indicates TBD) to the node identified by *source_node_ID* in the request. If the bridge abandons retry attempts and the packet was a response subaction no further action is required.
ack_tardy	To be determined...
ack_conflict_error ack_data_error ack_type_error ack_address_error	If the packet was a request subaction, the bridge shall transmit the appropriate response packet (with *resp_code* that indicates RESPONSE COMPLETE) to the node identified by *source_node_ID* in the request. If the packet was a response subaction no further action is required.

9. Stream operations and routing

This section describes the normal operations of a bridge to route stream 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 asynchronous streams shall be able to recognize stream packets (identified by their channel number) on an inbound portal and retransmit the stream packet (with a remapped channel number) on an outbound portal. Bridges that support isochronous streams in addition shall support the distribution of a synchronized cycle clock throughout the Serial Bus net.

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

9.1 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. This singular cycle master is name the net cycle master.

~~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 net cycle master 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 net cycle master is selected and enabled by the bridge manager.

Once a net cycle master 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 9.2 and 9.3.

9.2 Inbound portal operations

During an isochronous cycle, inbound portals eavesdrop on all stream 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 channels are to be buffered and, after a constant isochronous delay across the bridge's internal switching fabric, subsequently retransmitted by the other portal.

The information necessary to determine whether or not a particular channel is to be retransmitted resides in the inbound portal's STREAM_CONTROL register(s). If the *st* field has a value of one, the portal is enabled to listen to and forward packets identified by the *channel* field in the STREAM_CONTROL register.

An inbound portal shall forward a stream packet by making it available to the bridge's internal switching fabric, with the expectation that the other portal subsequently retransmits the packet.

9.3 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. Isochronous stream 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.

Stream packets distributed *via* the bridge's internal switching fabric shall be identified to the outbound portal by vendor-dependent methods.

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

9.4 Common Isochronous Packet format time stamps

The bridge is responsible to both filter and transform packets with respect to channel numbers, as described in the preceding clauses. Isochronous stream packets, if they are in the Common Isochronous Packet (CIP) format specified by IEC 61883/FDIS require additional transformations of optional time stamp data contained within their data payloads. The time stamp transformations are explained below.

CIP stream packets may contain isochronous time stamp information that is absolute rather than relative. That is, the time stamps contained within header data are a fixed offset ahead of, in the most significant bits, the isochronous cycle times contained in the cycle start packet that indicates the cycle in which the packets are transmitted. Time stamps may be found in one of two places in packets that conform to CIP format:

— the *syt* field of the second quadlet of a two-quadlet CIP header if the *fmt* field in that quadlet has a value between zero and $1F_{16}$, inclusive; and
— the *cycle_count* and *cycle_offset* fields of the isochronous source packet header.

Both of these time stamps are specified in absolute values that reference a future cycle time. Since isochronous packets experience a constant delay when routed through a bridge it is sufficient to transform the time stamp(s) by the addition of this constant. The value, in units of cycles, shall be obtained from the *isochronous_delay* field in the Bridge_Capabilities entry in configuration ROM.

For the *syt* field, the transformation shall be performed by applying the following formula (shown in C code notation):

$$syt_{transmitted} = (syt_{observed} + (isochronous\_delay << 12)) \& 0x0000FFFF;$$

Because IEEE Std 1394-1995 constrains *cycle_count* to the range zero to 7999, inclusive, the transformation of the *cycle_count* component of the source packet header differs. The addition of the constant isochronous delay to the *cycle_count* shall be performed modulus 8000, as shown below:

$$cycle\_count_{transmitted} = (cycle\_count_{observed} + isochronous\_delay) \% 8000;$$

10. Reset notification

To be determined...

11. 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***.

NOTE—Should a possible algorithm by which the bridge manager configures the net be described in an informative annex? There are probably many ways to configure the net and we should not specify one of them as normative.

