

IEEE Std 1212™-2001
(Revision of IEEE Std 1212-1994)

1212™

IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses

IEEE Computer Society

Sponsored by the
Microcomputer and Microprocessor Standards Committee



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

6 September 2002

Print: SH94977
PDF: SS94977

IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses

Sponsor

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

Approved 6 December 2001

IEEE-SA Standards Board

Abstract: A common bus architecture (which includes functional components—modules, nodes, and units—and their address space, transaction set, CSRs, and configuration information) suitable for both parallel and serial buses is provided in this standard. Bus bridges are enabled by the architecture, but their details are beyond its scope. Configuration information is self-administered by vendors and organizations based upon IEEE Registration Authority company_id.

Keywords: address space, architecture, bus, computer, CSR, interconnect, microprocessor, register, transaction set

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

Copyright © 2002 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 6 September 2002. Printed in the United States of America.

Print: ISBN 0-7381-3100-8 SH94977
PDF: ISBN 0-7381-3101-6 SS94977

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

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

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

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

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

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

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

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

Secretary, IEEE-SA Standards Board
445 Hoes Lane
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 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 IEEE and its designees are the sole entities that may authorize the use of the IEEE-owned certification marks and/or trademarks to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of IEEE Std 1212-2001, IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses.)

The IEEE approved project P1212 on December 10, 1996, at the request of Dr. David James, initial Chair of the working group. Participants met informally for almost a year, during which time contact with other industry groups revealed new work (beyond simple reaffirmation of the CSR architecture) whose most natural home would be the P1212 working group. Two industry initiatives, HAVi and the Printer Working Group's (PWG) 1394 task force, were designing enhancements to the configuration ROM facilities of the CSR architecture. The former had been concerned with "self-describing devices (SDD)" and the latter with device discovery and "function discovery services (FDS)." They jointly proposed a new project to the IEEE Microcomputer and Microprocessor Standards Committee who in turn suggested that the work be subsumed within IEEE P1212. As a consequence, the first official meeting of the P1212 working group was convened 11–12 November 1997 in San Jose, CA, with an expanded scope:

- reexamination and revision of the CSR architecture in light of the experience gained with FutureBus, Scalable Coherent Interface (SCI) and High Performance Serial Bus—all buses compliant with the CSR architecture;
- coordination of the revised standard with other standards, notably ANSI NCITS 325-1998, Serial Bus Protocol 2, which had exploited ambiguities in the original CSR architecture in order to incorporate desirable features; and
- enhancement of configuration ROM facilities to support device discovery independent of a device's software architecture, to permit the accurate characterization of physical instances of particular functions within a device and to enable self-administration of these features by organizations and vendors.

The working group met once again in December of the same year and continued to meet the next year. At the October, 1998 meeting in Ka'anapali, HI, there was a change in the Chair; Brian Batchelder indicated his willingness to serve and was elected without dissent. In subsequent meetings, the working draft was assembled from separate document submissions and refined to its present state. At the working group's final meeting, 13 October 1999 in Rennes, France, this draft was approved for initial sponsor ballot.

Significant changes between ISO/IEC 13213:1994 and this standard are summarized as follows:

- The extended address model described in the original standard has never been implemented and is no longer considered useful. It has been deleted from this standard and only the 64-bit fixed address model remains.
- The transaction set (read, write and lock requests and their responses) is fundamentally unchanged from the prior standard, except that the requirements for various data lengths (byte, quadlet, octlet, etc.) have been trimmed to match implementation experiences. A new lock function, allocate, is defined to permit efficient management of shared resources.
- The majority of CSRs previously defined have been deleted from the architecture. Experience with actual implementations, both SCI and Serial Bus, demonstrated that either the registers were bus-dependent or else unused. The smaller set of fundamental CSRs that remains is the minimum common to different bus standards.
- With respect to a pair of registers, MESSAGE_REQUEST and MESSAGE_RESPONSE, a common message format has been specified; prior to this change, multiple users were unable to share the message passing facility.
- Although the CSR architecture retains unit architectures as a fundamental building block, the descriptive material about interrupts, message passing, globally synchronized clocks and memory has been removed. These topics are application-dependent and their inclusion in the CSR architecture adds little value.

- Most of the changes affect configuration ROM. Key values unused in contemporary implementations were deleted in order to make them available for future standardization. New fields are defined for the bus information block, one to advertise dynamic changes in configuration ROM (the *generation* field) and the other to describe support for read requests larger than a quadlet (the *max_ROM* field). Textual descriptors are enhanced to provide support for multiple character sets and languages and a new type of descriptor was created to describe graphic images (icons). Both of these descriptors may be modifiable (a seeming oxymoron in the context of read-only memory!); this permits users to attach nicknames to devices. A variety of new configuration ROM entries has been defined in response to the needs of applications under development for Serial Bus.
- The single most significant enhancement to configuration ROM was the introduction of a hierarchical structure of instance and feature directories. These are complementary to the existing unit directories, which characterize the software protocol used to access a device. The instance directories describe physical instantiations of a particular device function; the capability to separately describe physical instances and software protocols is particularly important when one instance is controllable by more than one protocol. Feature directories assist device discovery by permitting functional capabilities to be specified independently of the software protocol.

Accurate and unambiguous specification of the expanded configuration ROM facilities in normative sections of the standard is not sufficient, by itself, to adequately educate designers and promote interoperability. An informative annex is included with examples of good practice in the design of configuration ROM, but this, too, may fall short. The reason is that practices are continually evolving. The working group hopes that trade associations or similar forums will continue the ongoing effort necessary to maintain the value of the foundations laid down by this standard.

Participants

The following is a list of active participants in the IEEE P1212 working group (those who attended two or more meetings from inception to the time of publication):

Brian Batchelder, *Chair*

Peter Johansson, *Editor*

Lee Farrell, *Secretary*

Helmut Bürklin
Navin Chandler
Richard Churchill
Firooz Farhoomand
John Nels Fuller

David V. James
Gregory A LeClair
Aaron Ludtke
Daniel Meirsman
Atsushi Nakamura

Fritz Nordby
Earl Rydell
Hisato Shima
David Smith
Frank Zhao

The following people were members of the balloting group:

Steven R Bard
Brian Batchelder
Edmund H. Baulsir
Janos Biri
Lon Canaday
Keith Chow
Alec Dara-Abrams
James R. Davis
Dante Del Corso
Chris S. Dorsey
Sourav K. Dutta
Firooz Farhoomand
Lee Farrell

John Nels Fuller
David B. Gustavson
Burke Henahan
Ron Hochsprung
Clay E. Hudgins
David V. James
Peter Johansson
Thomas M. Kurihara
Conrad A. Laurvick
Gregory A. LeClair
Joseph R. Marshall
Gene E. Milligan
Atsushi Nakamura

Farrell Ostler
Charles A. Rice
Gary S. Robinson
Jaideep Roy
Thomas J. Schaal
Marius Seritan
Hisato Shima
Akihiro Shimura
Michael D. Teener
Jerry A. Thrasher
Forrest D. Wright
Oren Yuen
Janusz Zalewski

The following people served on the ballot response committee:

Brian Batchelder
Lee Farrell

John Nels Fuller
David V. James
Peter Johansson

Gregory A. LeClair
Atsushi Nakamura

When the IEEE-SA Standards Board approved this standard on 6 December 2001, it had the following membership:

James T. Carlo, *Chair*
James H. Gurney, *Vice Chair*
Judith Gorman, *Secretary*

Sid Bennett
H. Stephen Berger
Clyde R. Camp
Richard DeBlasio
Harold E. Epstein
Julian Forster*
Howard M. Frazier

Toshio Fukuda
Arnold M. Greenspan
Raymond Hapeman
Donald M. Heirman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Peter H. Lips

Nader Mehravari
Daleep C. Mohla
William J. Moylan
Malcolm V. Thaden
Geoffrey O. Thompson
Howard L. Wolfman
Don Wright

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Satish K. Aggarwal, *NRC Representative*

Andrew Ickowicz
IEEE Standards Project Editor

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	2
2. References	2
3. Definitions and notation	3
3.1 Definitions	3
3.2 Notation	6
4. Architectural framework	7
4.1 Modules, nodes, and units	7
4.2 Addressing	8
5. Transaction set	10
5.1 Read and write transactions	10
5.2 Lock transactions	11
5.3 Bus-dependent transactions	13
5.4 Split transactions	13
5.5 Completion status	14
6. CSR definitions	14
6.1 STATE_CLEAR / STATE_SET registers	17
6.2 NODE_IDS register	17
6.3 RESET_START register	18
6.4 SPLIT_TIMEOUT register	19
6.5 MESSAGE_REQUEST / MESSAGE_RESPONSE registers	20
7. Configuration ROM	21
7.1 IEEE Registration Authority	22
7.2 ROM formats	22
7.3 CRC calculation	27
7.4 Minimal ASCII	28
7.5 Data structures	29
7.6 Required and optional usage	44
7.7 Directory entries	49
Annex A (informative) Configuration ROM examples	60
Annex B (informative) Keyword examples	66
Annex C (informative) Bibliography	67

IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses

1. Overview

1.1 Scope

This is a full-use standard, a revision of ISO/IEC 13213:1994 [B11]¹; its scope reflects accumulated experience with the control and status registers (CSR) architecture since it was first promulgated as a standard in 1991. In the intervening years, two bus standards, IEEE Std 1596-1992TM [B5] on Scalable Coherent Interface (SCI), and IEEE Std 1394-1995TM [B3] on Serial Bus, as amended by IEEE Std 1394a-2000TM [B4], have been the source of most practical implementation experience. The revised scope of the CSR architecture is given below:

- a) The overall architectural framework partitions the total available address space into equal spaces available to individual nodes. A node's address space is in turn partitioned into regions which have different usage models, e.g., memory space, private space for vendor uses, configuration ROM, and an I/O space (units space) where transactions may have side effects;
- b) A minimal transaction set (read, write, and lock requests and their associated completion responses) required for compliant bus standards. Bus bridges compliant with this architecture, whether in a homogeneous or heterogeneous environment, are also expected to transport this transaction set;
- c) Fundamental CSRs are defined to provide a common infrastructure for all compliant buses. In some cases the details of the registers are entirely bus-dependent but the function is common to all compliant buses;
- d) Message request and response CSRs are specified to enable directed delivery or broadcast of messages to multiple nodes. The message format permits organizations or vendors to define the meaning of the data payload without the need for a centralized registry of all possible formats; and
- e) Configuration ROM provides self-descriptive data structures that permit nodes to uniformly characterize the device services available. This is critical for buses that permit live insertion and removal of nodes; each newly inserted node contains sufficient information for it to be uniquely identified and for the requisite device drivers to be loaded.

Although the original CSR architecture anticipated widespread development of bridges between heterogeneous bus standards and a diversity of addressing modes, both fixed and variable, no such implementations have been made. As a consequence, the most significant changes in scope between the earlier CSR architecture and this standard are the adoption of a single, fixed addressing model and the removal of tutorial material pertaining to the design of bridges.

¹The numbers in brackets correspond to those of the bibliography in Annex C.

1.2 Purpose

The cardinal purpose of any revised standard is to correct or resolve ambiguities or deficiencies in the prior standard that have been revealed by applications (or attempted applications) of the earlier standard. This standard also preserves the following objectives of the original standard:

- a) *Scalability.* The CSR architecture is intended to be applicable to a wide range of bus standards and furthermore, to a wide range of device capabilities implemented by particular products;
- b) *Extensibility.* The architecture supports contemporary processor and bus designs and, to the extent that one can anticipate the future, should be a useful framework for similar new designs. In particular, the addressing model is well suited to 32- and 64-bit processors. The CSR architecture creates no obstacles for the designers of adapters that bridge between common system memory and I/O buses;
- c) *Open systems interoperability.* The configuration ROM data structures, as well as the fundamental CSRs, provide tools which vendors may use to guarantee that their devices will work with each other. The segregation of one unit architecture from another tends to restrict possible adverse affects by one module, node, or unit upon another;
- d) *Manageability.* The CSR architecture permits the design of systems that require little or no administration and are simple to support. Automatic configuration based upon configuration ROM is a key feature; indeed, it is essential when the applicable bus standard supports live insertion and removal of devices; and
- e) *Parallelism.* Although not directly addressed by this standard, the architecture contains no features that might preclude its use in a multiprocessor configuration.

Items a) through e) are a smaller set of objectives than described in ISO/IEC 13213:1994 [B11]. The working group responsible for the development of this revised standard believes that its usefulness has been enhanced by restricting some of the unrealized goals of the earlier work.

2. References

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

ISO/IEC 639:1988, Code for the representation of names of languages²

ISO/IEC 639-2:1998, Codes for the representation of names of languages—Part 2: Alpha-3 code

ISO/IEC 646:1991, Information Technology—ISO 7-bit coded character set for information interchange

ISO/IEC 9899:1999, Programming Languages—C

ISO/IEC 10646-1:2000, Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane

²ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (<http://global.ihs.com/>). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

3. Definitions and notation

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

3.1 Definitions

3.1.1 Conformance

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

3.1.1.1 bus-dependent: 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 definition by the applicable bus standard. In cases where the bus standard does not define an object or code value it shall be considered reserved.

3.1.1.2 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.

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

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

3.1.1.5 reserved: A keyword used to describe either objects (bits, bytes, quadlets, octlets, and fields) or the values assigned to these objects in cases where either the object or the value is set aside for future standardization. A reserved object shall be zeroed; the recipient of a reserved object shall ignore it. The recipient of an object whose values are defined by this standard shall inspect its value and reject reserved values. Usage and interpretation of either reserved objects or reserved values may be specified by future extensions to this or other standards.

3.1.1.6 shall: A keyword that indicates a mandatory requirement. Designers are required to implement all such mandatory requirements to assure interoperability with other products conforming to this standard.

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

3.1.2 Technical glossary

The following terms are used in this standard:

3.1.2.1 bridge: An active component that forwards request and response subactions between buses.

3.1.2.2 broadcast request: A request that is distributed to all nodes on a bus.

3.1.2.3 bus ID: The most significant bits of the 16-bit node ID, which uniquely identify a particular bus within a group of interconnected buses. The number of bits used to represent bus ID shall be specified by the applicable bus standard; the remaining less significant bits of node ID constitute the local ID.

3.1.2.4 byte: Eight bits of data.

3.1.2.5 command reset: An initialization event that is initiated by a write to the RESET_START register.

3.1.2.6 directory: A contiguous collection of one or more entries, which is contained within the node's configuration ROM.

3.1.2.7 doublet: Two bytes, or 16 bits, of data.

3.1.2.8 immediate effect: An effect of a transaction that appears to occur between the time the request subaction is accepted and the response subaction is returned. If a bus standard allows CSR transactions to be split, and sufficient time is allowed between the acceptance of a request subaction and the return of a response subaction, an immediate effect can be emulated by a processor on the node.

3.1.2.9 local ID: The least significant bits of the 16-bit node ID, which uniquely identify a particular node on a bus. The number of bits used to represent local ID shall be specified by the applicable bus standard; the remaining more significant bits of node ID constitute the bus ID.

3.1.2.10 leaf: A contiguous information field that is pointed to by a configuration ROM directory entry. A leaf contains a header (length and CRC specification) and other information fields.

3.1.2.11 memory space: A portion of a node's address space, which may provide access to a memory controller unit. Other types of units may also be mapped to non-overlapping portions of the memory space.

3.1.2.12 module: A physical package (printed circuit board or enclosure for multiple boards) that consists of one or more nodes.

3.1.2.13 node: An addressable device attached to a bus. Although multiple nodes may be present within the same physical enclosure (module), each has its own bus interface and address space and may be reset independently of the others.

3.1.2.14 node ID: A 16-bit number that uniquely differentiates a node from all other nodes within a group of interconnected buses. Although the structure of the node ID is bus-dependent, it usually consists of a bus ID portion and a local ID portion. The most significant bits of node ID are the same for all nodes on the same bus; this is the bus ID. The least-significant bits of node ID are unique for each node on the same bus; this is called the local ID. The local ID may be assigned as a consequence of bus initialization.

3.1.2.15 node space: The $256 * 2^{40}$ bytes of address space that may be available to each node. Addresses within node space are 48 bits and are based at zero. Node space includes memory space, private space, register space and units space (a subset of register space).

3.1.2.16 octlet: Eight bytes, or 64 bits, of data.

3.1.2.17 power reset: An initialization event triggered by the restoration of primary power.

3.1.2.18 private space: A $256 * 2^{20}$ byte portion of node space, with a base address of $FFFF\ E000\ 0000_{16}$, allocated for vendor-dependent uses local to the node.

3.1.2.19 quadlet: Four bytes, or 32 bits, of data.

3.1.2.20 register: A term used to describe quadlet-aligned addresses that may be read or written by bus transactions. In the context of this standard, the use of the term register does not imply a specific hardware implementation. For example, in the case of split transactions that permit sufficient time between the request and response subactions, the behavior of the register may be emulated by a processor.

3.1.2.21 register space: A $256 * 2^{20}$ byte portion of node space with a base address of $FFFF\ F000\ 0000_{16}$. Registers defined by this standard are located within register space as are bus-dependent registers defined by the applicable bus standard.

3.1.2.22 request subaction: A subaction that initiates an action by the recipient. A request subaction contains an address, request type and sometimes data. For example, a read request specifies an address and a length but no data (it is returned in the response subaction) while a write request specifies an address and data.

3.1.2.23 requester: A node that initiates a transaction request by means of a request subaction addressed to one or more nodes.

3.1.2.24 responder: A node that completes a transaction by returning a response subaction.

3.1.2.25 response subaction: A subaction returned by a responder to complete a transaction initiated by a requester. A response subaction contains completion status and sometimes data.

3.1.2.26 root directory: A term used to describe the directory at the top level of the hierarchical configuration ROM directory structure.

3.1.2.27 split transaction: A transaction that consists of a request subaction followed by a separate response subaction. Subactions are considered separate if ownership of the bus is relinquished between the two.

3.1.2.28 subaction: One of the two components in a transaction; a transaction consists of request and response subactions.

3.1.2.29 transaction: A request subaction and the corresponding response subaction. The request subaction transmits a transaction code (such as read, write, or lock) and an address; some request subactions include data as well as transaction codes. The response subaction is null for request subactions with broadcast destination addresses or broadcast request codes; otherwise it returns completion status and possibly data.

3.1.2.30 unified transaction: A transaction for which the request subaction and its corresponding subaction are completed as an indivisible sequence. Transactions with null response subactions are by definition unified.

3.1.2.31 unit: A component of a node that provides processing, memory, I/O, or some other functionality. A node may have multiple units, which may operate independently of each other.

3.1.2.32 unit architecture: The specification of the interface to and the services provided by a unit implemented within a node.

3.1.2.33 units space: A portion of node space with a base address of FFFF F000 0800₁₆; units space is a subset of register space. The CSRs and other facilities defined by unit architectures are expected to lie within this space.

3.1.3 Abbreviations

The following abbreviations are used in this standard:

CSR	Control and status register
CRC	Cyclical redundancy checksum
EUI-64	Extended Unique Identifier, 64-bits
ROM	Read-only memory, in particular configuration ROM

3.2 Notation

The following conventions should be understood by the reader in order to comprehend this standard.

3.2.1 Numeric values

Decimal and hexadecimal numbers are used within this standard. By editorial convention, decimal numbers are most frequently used to represent quantities or counts. Addresses are uniformly represented by hexadecimal numbers, which are also used when the value represented has an underlying structure that is more apparent in a hexadecimal format than in a decimal format.

Decimal numbers are represented by Arabic numerals without subscripts or by their English names. Hexadecimal numbers are represented by digits from the character set 0–9 and A–F followed by the subscript 16. When the subscript is unnecessary to disambiguate the base of the number it may be omitted. For the sake of legibility, hexadecimal numbers are separated into groups of four digits separated by spaces.

As an example, 42 and $2A_{16}$ both represent the same numeric value.

3.2.2 Bit, byte, and quadlet ordering

CSRs and configuration ROM structures defined by this standard use the ordering conventions specified in this clause, which defines the order and significance of bytes within quadlets and quadlets within octlets in terms of their relative addresses—not their physical position on either a parallel or serial bus.

Within a quadlet, the most significant byte is that which has the smallest address and the least significant byte is that which has the largest address, as shown in Figure 1.

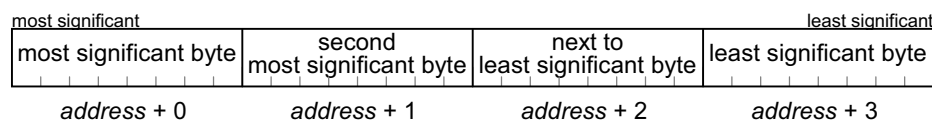


Figure 1—Byte ordering within a quadlet

Within an octlet the most significant quadlet is that which has the smallest address and the least significant quadlet is that which has the largest address, as Figure 2 indicates.

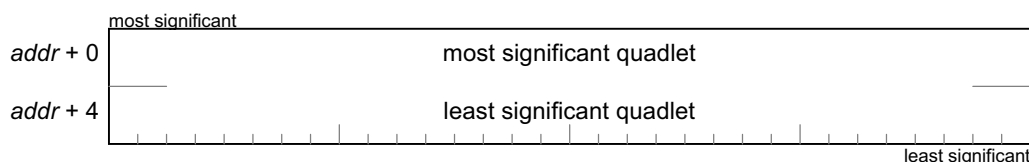


Figure 2—Quadlet ordering within an octlet

4. Architectural framework

This document describes a generic architecture useful to other standards that specify memory-mapped microcomputer buses. This clause establishes the basic components of the architecture (modules, nodes, and units) as well as their common address model. The architecture enables groups of buses interconnected by bridges (a net), but the details necessary to use such a net are beyond the scope of this standard.

4.1 Modules, nodes, and units

The most fundamental component of the CSR architecture is a *node*—an addressable entity with its own bus interface. *Modules* and *units* are other components of the CSR architecture; they have a hierarchical relationship with a node. More than one node may reside in the same physical enclosure (module) and more than one unit may be incorporated within a node, as illustrated by Figure 3.

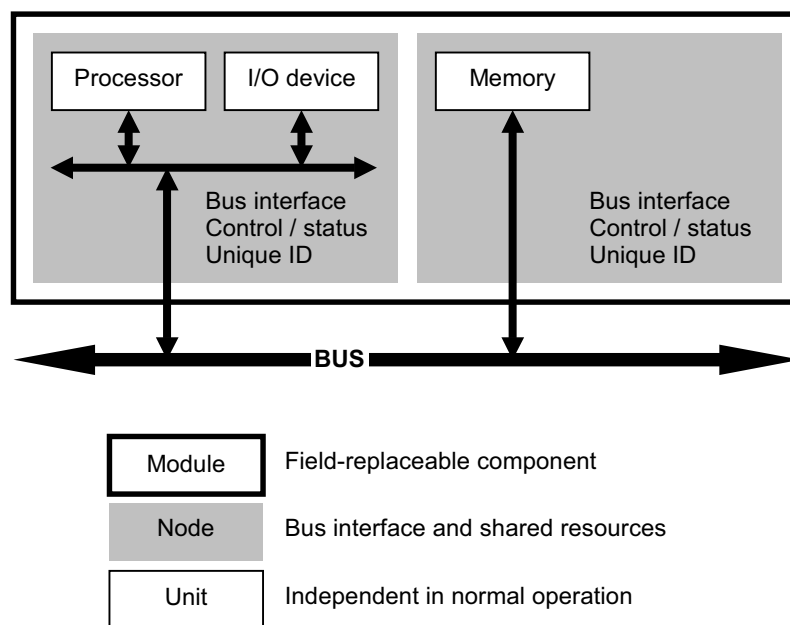


Figure 3—Component architecture

Each module consists of one or more nodes, which may be independently initialized and configured. A module is a physical packaging concept while a node is a logical addressing concept. When a module contains more than a single node, the physical affinity of its nodes may be described by unique ID information contained within each node. For example, if one node is in some sense “primary,” the other nodes in the module may establish their affinity by a reference to the primary node’s unique ID. Single-node modules have no need for this additional information.

Each node is independently addressable and contains a dedicated bus interface and other resources common to the node. Nodes operate independently and may be inserted or removed from the bus without necessarily affecting the operations of other nodes. A node may contain one or more independently controllable units, each of which may provide processing, memory, I/O, or some other functionality. The means by which software may control a unit are specified by a unit architecture, a document that specifies the programming interface for the device. A unit may be further subdivided into multiple logical units or subunits; their definition is beyond the scope of this document.

4.2 Addressing

This standard utilizes a 64-bit fixed addressing scheme, where the most significant 16 bits of each address are the node ID and the least significant 48 bits are a byte offset into a particular node’s address space. This provides address space for up to 65536 nodes, each of which has an address space of $256 * 2^{40}$ bytes. Some buses compliant with this standard reserve portions of the address space for broadcast. Compliant buses may also subdivide the node ID into a bus ID portion and a local ID portion. The hierarchy of the address space is illustrated by Figure 4.

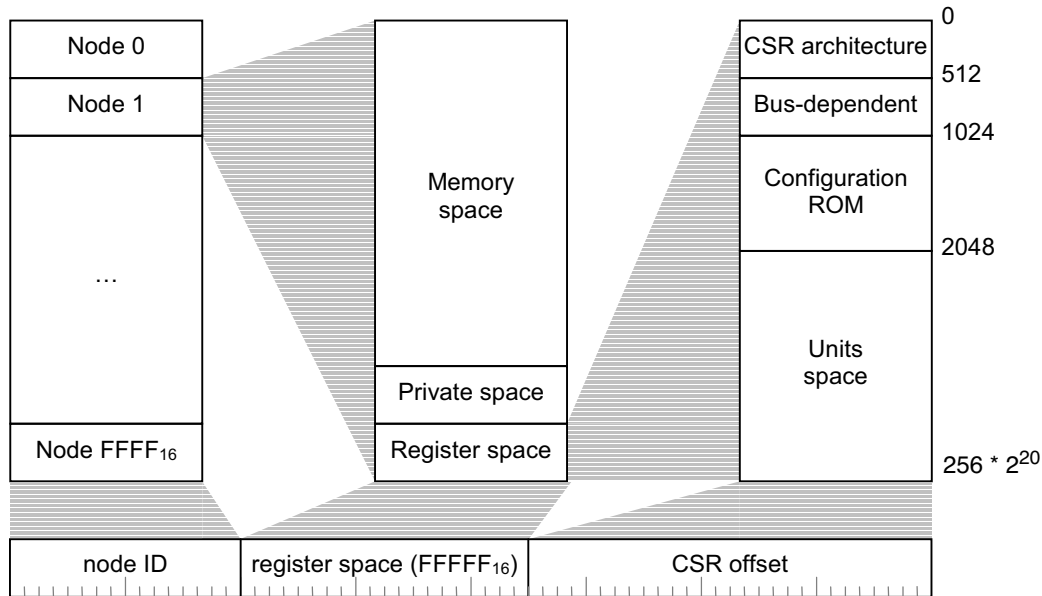


Figure 4—64-bit fixed addressing hierarchy

The most significant 16 bits of the address, the node ID, selects a particular node. Within that node’s $256 * 2^{40}$ byte address space, the next most significant 20 bits select an address space: memory, private, or register. The register space is further subdivided into regions for standard CSR architecture registers, bus-dependent registers, configuration ROM and units space. The base address (relative to the node’s address space) and size of these address spaces are summarized in Table 1.

Table 1— Address spaces

Name	Base address	Size (bytes)	Description
Memory space	zero	$256 * 2^{40}$ less $512 * 2^{20}$	The name reflects a historical expectation that node resources located here exhibit behaviors usually associated with memory. That is, read and write transactions do not have side effects. In fact, any node resources may be located in this address space—even registers with associated side effects.
Private space	FFFF E000 0000 ₁₆	$256 * 2^{20}$	Allocated for vendor-dependent uses local to the node. Some implementations may preclude the reuse of an internal address as an externally accessible bus address.
Register space	FFFF F000 0000 ₁₆	$256 * 2^{20}$	Register space is allocated for the definition of registers that may have side effects when accessed; it includes standard CSR architecture registers, bus-dependent registers, configuration ROM, and units space.
CSR architecture registers	FFFF F000 0000 ₁₆	512	Registers defined by this or future CSR architecture standards.
Bus-dependent registers	FFFF F000 0200 ₁₆	512	Allocated for definition by the applicable bus standard.
Configuration ROM	FFFF F000 0400 ₁₆	1024	Descriptive information for the node (which includes a 64-bit unique identifier) and its units.
Units space	FFFF F000 0800 ₁₆	$256 * 2^{20}$ less 2048	Allocated for cooperative, shared definition by the applicable bus standard or the organizations or vendors responsible for unit architectures.

The recipient of a read request addressed to the first 2048 bytes of register space, FFFF F000 0000₁₆ to FFFF F000 07FF₁₆ inclusive, shall transmit a response; the request shall have no side effects.

5. Transaction set

Buses compliant with this standard shall define a transaction set that permits the exchange of data between a requester (initiator of a transaction request) and a responder. A common example of a transaction is a read request initiated by a processor and subsequently satisfied by a response from a memory controller.

This standard defines common facilities for transaction sets to be specified by bus standards and additionally mandates implementation of a minimum transaction set.

The underlying model is one of the transactions in which the request and response may be separable from each other. A transaction consists of one or more subactions: indivisible operations unaffected by other subactions on the bus. A request subaction conveys the address, request type, and sometimes data from the requester to one or more recipients. When there is a single recipient, a response subaction returns request completion status and sometimes data to the requester.

Transactions may be split or unified. In a split transaction, other subactions may intervene between a request subaction and its corresponding response subaction. The request and response subactions of a unified transaction complete indivisibly; other subactions are inhibited between the initiation of the request subaction and the completion of its associated response subaction. A transaction that consists solely of a request subaction is by definition unified.

Buses compliant with this standard may support split or unified transactions—or both. If a bus implements split transactions it shall support some method to label transactions so that each response may be matched with its corresponding request.

5.1 Read and write transactions

All buses compliant with this standard shall implement read and write transactions that transfer the data payloads, specified by the following table, to or from addresses that are a multiple of the size of the data payload:

Data payload (bytes)	Comment
1	Provides accessibility to the smallest unit of data at an arbitrarily aligned address
4	Commonly referred to as quadlet reads and writes
8	Permits locations targeted by lock operations to be read with a single transaction
16	
64	Required for the sake of efficiency. This is the size of the MESSAGE_REQUEST and MESSAGE_RESPONSE registers and also the largest payload guaranteed to be forwarded by bridges.

Bus standards may also define read and write transactions that transfer arbitrary amounts of data from or to arbitrarily aligned addresses—and these transactions may be used to satisfy the above requirements. These are hereafter referred to as block read and block write transactions.

A bus that specifies broadcast address capabilities may permit both quadlet write and block write requests to be broadcast. Although the lack of a transaction response prevents delivery confirmation of the broadcast write request, the expected effect is equivalent to the simultaneous receipt of individual addressed write requests.

Read and write requests shall be processed indivisibly whether completed as a split or unified transaction. For example, after the successful completion of two write requests addressed to the same location, the resultant contents of the location shall be equal to the data contained in one of the two requests (as opposed to a mixture of data present in both). This shall apply to quadlet and block transactions equally.

5.2 Lock transactions

Some operations, e.g., test and set, have been implemented atomically when a bus permits the examination of a data value and its subsequent conditional update while the bus remains locked for other access. Because buses compliant with this standard may implement split transactions, another method is necessary to provide equivalent, atomic behavior. For this purpose, a set of *lock* transactions is defined. Locks are transactions that communicate the intended operation to the responder; the responder performs the lock operation indivisibly and returns the result.

Lock requests consist of an address, an optional *arg* value, a *data* value and a *lock function* that specifies the operation to be performed. Lock responses consist of a response status and the *old* value. The relationship between the parameters supplied by the lock request, the current value of the addressed location (in particular the way in which it is both returned as an argument of the lock response and provides input to the lock operation) and the updated value of the addressed location are illustrated by Figure 5.

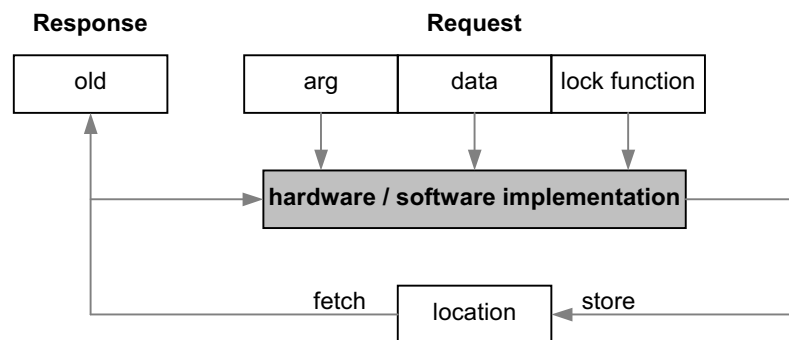


Figure 5—Simplified lock model

The three operands, *arg*, *data* and *old*, are all the same size and are either four or eight bytes. The lock function, specified by Table 2, selects the operation. The current value at the addressed location is fetched and eventually returned as *old* in the lock response. In the meantime, it serves as an input to the lock operation. The *arg* value, if present, may be compared with the *old* value to determine conditional outcome of the lock operation. At the end, the result of the operation may also be masked and the *new* value is stored at the original location.

Table 2—Lock operations

Value	Name	Definition
0		Reserved; not to be used
1	<i>mask_swap</i>	$\text{new} = (\text{data} \& \text{arg}) \mid (\text{old} \& \sim\text{arg});$
2	<i>compare_swap</i>	$\text{if } (\text{old} == \text{arg}) \text{ new} = \text{data};$
3	<i>add_big</i>	$\text{new} = \text{old} + \text{data};$
4	<i>add_little</i>	
5	<i>bounded_add_big</i>	$\text{if } (\text{old} != \text{arg}) \text{ new} = \text{old} + \text{data};$
6	<i>wrap_add_big</i>	$\text{new} = (\text{old} != \text{arg}) ? \text{old} + \text{data} : \text{data};$
7	bus-dependent	Allocated for definition by the applicable bus standard
8	<i>allocate_big</i>	$\text{if } (\text{old} \geq \text{arg}) \text{ new} = \text{old} - \text{data};$
9	<i>allocate_little</i>	
A ₁₆ –F ₁₆		Reserved for definition by future CSR architecture standards

NOTE—A swap command is not explicitly defined, since a mask_swap operation that specifies an *arg* value of all ones provides equivalent functionality.

Buses compliant with this standard shall define an encoding for the lock functions specified by Table 2 and sufficient to accommodate 16 values; the specification of other lock functions numbered 10₁₆ and higher is a bus-dependent option.

Some of the lock operations perform arithmetic operations; their operands are unsigned integers. Correct implementation of these operations requires knowledge of the relative significance of the bytes within the addressed data; these are denoted by the suffixes *_big* and *_little*. For operations suffixed with *_big*, the byte with the smallest offset within the addressed region is deemed the most significant while for operations suffixed with *_little* it is the least significant. Correct implementation of arithmetic operations also requires the applicable bus standard to specify an invariant mapping from bytes transferred over the bus to bytes in the node's address space. The mapping is independent of the relative arithmetic significance of the bytes. The recommended mapping is for the first byte transferred over the bus to correspond to a zero byte offset within the addressed region and subsequent bytes to correspond to incrementing offsets within the region.

When a lock transaction addresses a location that has one or more reserved bits or fields, the results are not necessarily obvious. Unless otherwise specified by the applicable bus standard, the behavior of a particular lock function shall be determined by applying rules for reserved fields in order, as follows:

- The addressed location's *old* value shall be obtained as if via a read request and shall be returned in the lock response; reserved fields are read as zeros;
- An intermediate value shall be calculated according to the C code in Table 2 (this is not explicitly shown but is the right-hand part of each of the assignment statements in the table); and
- The intermediate value shall be stored in the addressed location as if via a write request; reserved fields shall be ignored and remain zero in the location. The contents of the location after this operation are the *new* value.

5.3 Bus-dependent transactions

A bus standard may define additional transactions whose behavior is beyond the scope of this standard. Examples include (but are not limited to) read and write transactions that selectively update bytes within a quadlet, large block transfers, either aligned or unaligned, that promote efficient use of the bus, move transactions (unacknowledged writes) or bus-dependent operations pertaining to initialization, clock synchronization and cache coherency.

5.4 Split transactions

When split transactions are supported, a node may accept more than one request subaction before any are processed. Similarly, several request subactions may be processed before any of the associated response subactions are transmitted. Implementation-dependent request and response queues mediate the flow of these subactions between the node and the bus, as illustrated by Figure 6.

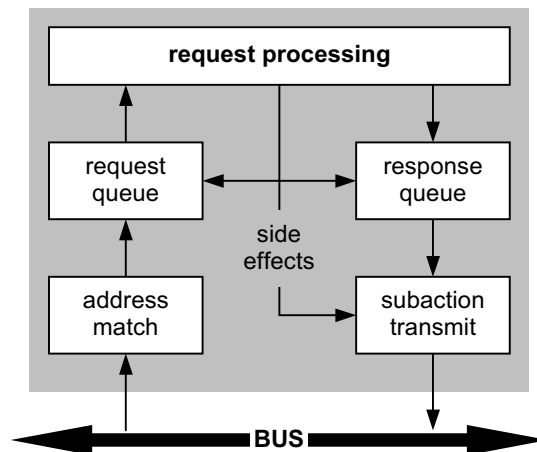


Figure 6—Request processing mode

In Figure 6, a request subaction is accepted by a node if queue space is available; the request processing may be deferred. If a response is generated as a result of request processing, the response subaction may be temporarily queued before it is returned to the original requester. The CSR architecture imposes no ordering constraints, either within the individual queues or in relationship to each other.

Although the relative order of transactions is independent with respect to each other, all transaction effects of an individual request shall occur simultaneously after the request subaction has been dequeued (the creation of a response subaction is one of these effects).

Write requests addressed to certain CSRs, e.g., the RESET_START register, may cause queued request or response subactions to be discarded; see Clause 6 for detailed information. Other write requests may change the addresses recognized by a node. The latter may have undesirable side effects if the request or response queues are not empty.

5.5 Completion status

Bus compliant with this standard shall define a method to specify the completion status of the request. The minimum set for completion status is specified by Table 3.

Table 3—Transaction status

Name	Description
<i>complete</i>	The responder has successfully completed the transaction request.
<i>type_error</i>	The transaction request specified an invalid or unsupported operand (e.g., a write to a read-only address).
<i>address_error</i>	The address specified by the transaction request is not accessible or not implemented by the responder.
<i>conflict_error</i>	A lack of resources (e.g., queue space for the receipt of the transaction request or a transient busy condition of some resource needed to service the request) prevents completion of the request. A subsequent retry may succeed.
<i>timeout</i>	The requester has received no completion status for the transaction request.

When resources are temporarily unavailable to complete a transaction, there may be some bus-dependent information that hints at the duration of the condition. For fleeting busy conditions, bus standards may design retry protocols below the transaction layer. Presumably there would be less overhead to hardware automated retry than to the end-to-end retry engendered by the return of *conflict_error* status.

6. CSR definitions

Clause 6 specifies the format and usage of fundamental control and status registers (CSRs); these CSRs provide a common point of departure for the definition of additional registers by compliant bus standards or unit architectures. The necessity for standardization within these CSRs is motivated by:

- uniform software access for essential features such as configuration of the bus and node address space;
- common definition of facilities generally useful to a wide range of bus standards; and
- consistent use of well-known addresses, in particular where different applications or protocols might use the same address for message passing.

CSRs are conventionally located with a node's register space, the range of addresses at or above FFFF F000 0000₁₆, although there are no prohibitions that forbid the definition of CSRs at lower addresses. Within register space, address ranges are reserved to different standards documents, organizations or vendors for definition as specified by Table 1.

The following table locates the CSRs defined by this standard in terms of a byte offset from the start of register space, FFFF F000 0000₁₆; all unallocated addresses within the first 512 bytes of register space are reserved for definition by future CSR architecture standards:

Offset	Size (bytes)	Register name	Required	Description
0	4	STATE_CLEAR		Bus-dependent state and control information
4	4	STATE_SET		Sets STATE_CLEAR bits
8	4	NODE_IDS	Yes	Contains the 16-bit <i>node_ID</i> value used to address the node
0C ₁₆	4	RESET_START	Yes	Used to initiate a command reset
18 ₁₆	8	SPLIT_TIMEOUT		Time limit for split transactions
80 ₁₆ C0 ₁₆	64	MESSAGE_REQUEST MESSAGE_RESPONSE		Well-known addresses for receipt and protocol demultiplex of 64-byte messages in a standard format

Except as noted in the individual register definitions that follow, all of the above CSRs shall support quadlet read and quadlet write requests.

In the clauses that follow, the conventions illustrated by Figure 7 are used to define CSR formats, initial values, and the effects of read, write or other transactions. Bus standards compliant with the CSR architecture should use the same conventions.

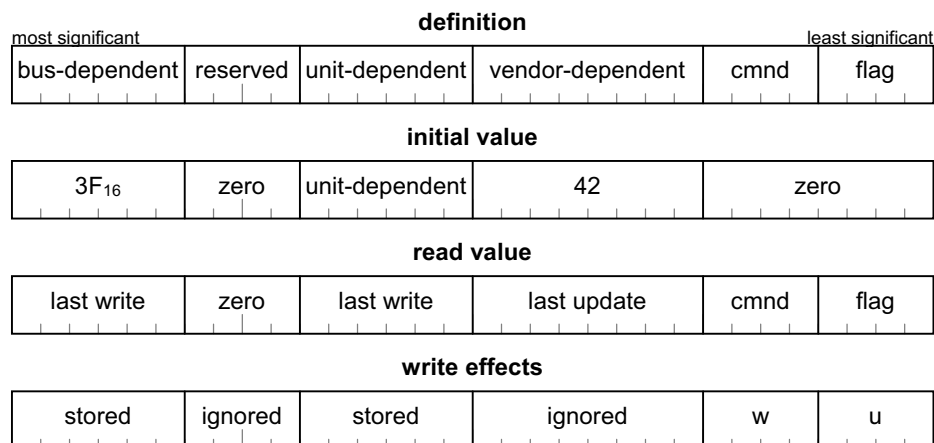


Figure 7—CSR specification example

The register definition contains the names of register fields. The names are intended to be descriptive, but the fields are defined in the text; their function should not be inferred solely from their names. However, the following field names have defined meanings:

Name	Abbreviation	Definition
bus-dependent	bus-depend	The meaning of the field is defined by the applicable bus standard
reserved	r	The field is reserved for future standardization (see 3.1.1)
unit-dependent	unit-depend	The meaning of the field shall be defined by the organization or vendor responsible for the unit architecture
vendor-dependent	vendor-depend (or v)	The meaning of the field shall be defined by the node's vendor

CSRs shall assume initial values upon the restoration of power (a power reset), upon a bus reset (if the applicable bus standard defines such a reset) or upon a write to the node's RESET_START register (a command reset). If the power reset values differ from the command reset values, they are separately and explicitly defined. Initial values for register fields may be described as numeric constants or with one of the terms defined for the register definition. Values for register fields subsequent to a reset may be described in the same terms or as defined below.

Name	Abbreviation	Definition
unchanged	x	The field retains whatever value it had just prior to the power reset, bus reset, or command reset.

In addition to numeric values for constant fields, the read values returned in response to a quadlet read request may be specified by the terms in the following table:

Name	Abbreviation	Definition
last write	w	The value of the field shall be either the initial value or, if a write or lock transaction addressed to the register has successfully completed, the value most recently stored in the field. ^a
last update	u	The value of the field shall be that most recently updated by the node hardware or software. An updated field value may be the result of a write effect to the same register address, a different register address or some other change of condition within the node.

^aFor clarity, read values for a field in a register that accepts lock transactions may be described as *last successful lock* rather than *last write*. However, the abbreviation in both cases remains *w*. Similar liberties may be taken with the use of *conditionally stored* in place of *stored* when the action occurs as the result of a lock transaction, but the corresponding one-letter abbreviation, *s*, is also unchanged.

The effects of data written to the register are specified by the terms in the following table: